

FEDERATED LEARNING

From Algorithms to
System Implementation

Liefeng Bo
Heng Huang
Songxiang Gu
Yanqing Chen



机械工业出版社
CHINA MACHINE PRESS



World Scientific

FEDERATED LEARNING

From Algorithms to
System Implementation

This page intentionally left blank

FEDERATED LEARNING

From Algorithms to
System Implementation

Liefeng Bo
Heng Huang
Songxiang Gu
Yanqing Chen

JD Technology, China

Published by

World Scientific Publishing Co. Pte. Ltd.

5 Toh Tuck Link, Singapore 596224

USA office: 27 Warren Street, Suite 401-402, Hackensack, NJ 07601

UK office: 57 Shelton Street, Covent Garden, London WC2H 9HE

Library of Congress Cataloging-in-Publication Data

Names: Bo, Liefeng, author.

Title: Federated learning : from algorithms to system implementation /

Liefeng Bo, Heng Huang, Songxiang Gu, Yanqing Chen, JD Technology, China.

Description: New Jersey : World Scientific, [2025] | Includes bibliographical references and index.

Identifiers: LCCN 2024018824 | ISBN 9789811292545 (hardcover) |

ISBN 9789811292552 (ebook for institutions) | ISBN 9789811292569 (ebook for individuals)

Subjects: LCSH: Federated learning (Machine learning)

Classification: LCC Q325.65 .B63 2025 | DDC 006.3/1--dc23/eng20240612

LC record available at <https://lcn.loc.gov/2024018824>

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library.

联邦学习：算法详解与系统实现

Originally published in Chinese by China Machine Press Co., Ltd.

Copyright © China Machine Press Co., Ltd. 2022

Copyright © 2025 by World Scientific Publishing Co. Pte. Ltd.

All rights reserved. This book, or parts thereof, may not be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording or any information storage and retrieval system now known or to be invented, without written permission from the publisher.

For photocopying of material in this volume, please pay a copying fee through the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, USA. In this case permission to photocopy is not required from the publisher.

For any available supplementary material, please visit

<https://www.worldscientific.com/worldscibooks/10.1142/13823#t=suppl>

Desk Editors: Nambirajan Karupiah/Julio Hong

Typeset by Stallion Press

Email: enquiries@stallionpress.com

Printed in Singapore

Preface

Machine learning technology developed rapidly in the past two decades. Mathematical concepts gradually evolved to algorithm implementations, bringing innovations in various areas to both industry and academia. In recent years, data privacy has drawn more and more attention from the public, and researchers have begun to develop various technologies to protect private data. As one of these important technologies, federated learning is one of these important technologies; federated learning is expected to have optimistic prospects.

This book conducts a comprehensive review and summary of federated learning technology ranging from basic concepts, application scenarios, advanced algorithms, and finally system implementation, hoping to inspire our audiences. Please note that this book majorly reviews federated learning from a technical perspective. It is necessary to meet the legal and regulatory requirements of corresponding countries regarding real-world data collection and authorization.

The content of this book is divided into three parts to introduce federated learning. The first part contains the basic knowledge of federated learning, including the definition, challenges, application scenarios, and main technologies of federated learning. Corresponding chapters are as follows: Chapter 1: Introduction to Federated Learning, Chapter 2: Federated Learning Application Scenarios, and Chapter 3 Common Privacy Protection Technologies. The second part contains details of many vertical and horizontal federated learning algorithms for different application scenarios. Corresponding chapters are Chapter 4 Tree-Based Models in Vertical

Federated Learning, Chapter 5 Vertical Federated Linear Regression Algorithm, Chapter 6 Vertically Federated Kernel Learning, Chapter 7 Asynchronous Vertical Federated Learning Algorithm, Chapter 8 Secure Bilevel Asynchronous Vertical Federated Learning with Backward Updating, Chapter 9 Vertical Federated Deep Learning Algorithms, Chapter 10 Faster Secure Data Mining Framework via Homomorphic Encryption, Chapter 11 Horizontal Federated Learning Algorithms, Chapter 12 Mixed Federated Learning Algorithms, and Chapter 13 Federated Reinforcement Learning. The third part contains related knowledge on building a robust federated learning system, including the architecture of federated learning system designed by JD.com as well as engineering optimization and algorithm tricks to improve runtime performances. Corresponding chapters are Chapter 14 Detailed Exploration of the FedLearn Federated Learning System, Chapter 15 Application of gRPC in FedLearn, Chapter 16 Performance Optimization Practices in Real-World Scenarios, and Chapter 17 Federated Learning Based on Blockchain.

About the Authors

Liefeng Bo is the Vice President of JD Technology and the Head of the Silicon Valley R&D Department. He served as committee member of several top artificial intelligence conferences, including NeurIPS, CVPR, ICCV, ECCV, AACL, and SDM. He has published more than 80 papers in top international conferences and journals, with his papers being cited 10,186 times and having an H-index of 44. His doctoral dissertation was awarded the National Excellent Doctoral Dissertation Award, and his paper on RGB-D object recognition won the Best Computer Vision Paper Award at the prestigious academic conference ICRA.

Heng Huang is AIMBE Fellow, an international academic leader in the fields of big data, machine learning, and artificial intelligence. He is a Distinguished Lifetime Professor in the Department of Electrical and Computer Engineering at the University of Pittsburgh. As a conference program chair or member of the chairing committee, he has organized more than 20 international academic conferences. He has published over 220 articles in top international conferences and journals, with his articles being cited more than 18,000 times. As a project leader, he has led more than 20 internationally leading scientific research projects.

Songxiang Gu is a Direct at JD Technology. He received Ph.D. in Computer Science and conducted in-depth research on large-scale parallel systems. He has served as a senior machine learning and statistical scientist at the U.S. FDA, where he developed an evaluation system for radiographic medical imaging equipment. He subsequently

joined WalmartLabs and LinkedIn, where he was responsible for the architectural design of machine learning platforms. In 2018, he joined JD Technology and has since led multiple teams in developing intelligent customer service, knowledge graphs, and federated learning systems.

Yanqing Chen is a Director at JD Technology. He receives Ph.D. in Computer Science at Stony Brook University. He is a pioneer and conducts in-depth research in the field of federated learning. He designs and develops multiple novel federated learning algorithms, publishes multiple papers regarding algorithms, optimizations and evaluations under privacy-preserving conditions.

Contents

<i>Preface</i>	v
<i>About the Authors</i>	vii
Part I: Federated Learning Knowledge	1
Chapter 1 Introduction to Federated Learning	3
1.1 What Is Federated Learning	3
1.1.1 History of Federated Learning	6
1.1.2 Federated Learning Overview	7
1.1.3 Classification of Federated Learning	11
1.2 Current Application State and Development Trends of Federated Learning	12
1.2.1 Current Applications of Federated Learning	13
1.2.2 Development Trends of Federated Learning . . .	14
1.3 Distributed Machine Learning and Federated Learning .	16
1.3.1 History of Distributed Machine Learning	16
1.3.2 Overview of Distributed Machine Learning . . .	18
1.3.3 The Co-development of Distributed Machine Learning and Federated Learning	21
1.4 Conclusion	22
Chapter 2 Federated Learning Application Scenarios	23
2.1 Federated Learning and Finance	23
2.2 Federated Learning and Biomedicine	26

2.3	Federated Learning and Computer Vision	30
2.4	Federated Learning and Natural Language Processing	35
2.5	Federated Learning, Cloud Computing, and Edge Computing	40
2.6	Federated Learning and Computer Hardware	43
2.7	Conclusion	46
Chapter 3 Common Privacy Protection Technologies		49
3.1	Privacy Preserving Machine Learning	49
3.1.1	Overview of Privacy Preserving Machine Learning	49
3.1.2	Developments in Privacy-Preserving Machine Learning	54
3.2	Commonly Used Privacy Protection Techniques	56
3.2.1	Differential Privacy	56
3.2.2	Secure Multi-Party Computation	68
3.2.3	Homomorphic Encryption	80

Part II: Federated Learning Algorithms **107**

Chapter 4 Tree-Based Models in Vertical Federated Learning		109
4.1	Introduction to Tree-Based Models	109
4.2	Vertical Federated Random Forest Algorithm	111
4.2.1	Algorithm Structure	111
4.2.2	Algorithm Description	112
4.2.3	Security Analysis	114
4.3	Vertical Federated Gradient Boosting Algorithm	120
4.3.1	Review of XGBoost Algorithm	121
4.3.2	Review of SecureBoost Algorithm	122
4.3.3	Vertical Federated Gradient Boosting Algorithm	123
Chapter 5 Vertical Federated Linear Regression Algorithm		127
5.1	Vertical Federated Linear Regression	128
5.1.1	Training Process	130
5.1.2	Inference Process	131
5.1.3	A Dilemma of Vertical Federated Learning	132

5.2	Federal Multi-View Linear Regression	132
5.2.1	Second-Order Optimization Method Based on BFGS	136
5.2.2	Secure Computation Protocol	139
5.2.3	Conclusion	148
Chapter 6 Vertically Federated Kernel Learning		151
6.1	Introduction	152
6.2	Brief Review of Doubly Stochastic Kernel Methods . . .	154
6.2.1	Problem Statement	154
6.2.2	A Brief Introduction to Kernel Methods	156
6.2.3	Random Fourier Features Approximation	159
6.2.4	Doubly Stochastic Gradient	160
6.3	Vertical Federated Kernel Learning Approach	161
6.3.1	Problem Statement	162
6.3.2	Algorithm Structure	162
6.3.3	Algorithm Design	164
6.3.4	Scenario Case Study	167
6.4	Theoretical Analysis	169
6.4.1	Convergence Analysis	169
6.4.2	Security Analysis	170
6.4.3	Complexity Analysis	172
6.5	Experiments	172
6.5.1	Experimental Setup	172
6.5.2	Results and Discussions	173
6.6	Conclusion	178
Chapter 7 Asynchronous Vertical Federated Learning Algorithm		179
7.1	Introduction	180
7.2	Related Work	182
7.2.1	Overview of Existing Work	182
7.2.2	Review of SGD-Type Algorithms	183
7.3	Problem Statement	184
7.4	Asynchronous Vertical Federated Learning Algorithm	185
7.4.1	Algorithm Framework	186
7.4.2	Algorithm Description	189
7.4.3	Scenario Case Study	192

7.5	Theoretical Analysis	194
7.5.1	Convergence Analyses	194
7.5.2	Security Analysis	199
7.5.3	Complexity Analysis	200
7.6	Experimental Results	201
7.6.1	Experimental Setup	201
7.6.2	Results and Discussions	205
7.7	Conclusion	210
Chapter 8 Secure Bilevel Asynchronous Vertical Federated Learning with Backward Updating		211
8.1	Introduction	212
8.2	Problem Formulation	213
8.3	Understanding Algorithms	214
8.3.1	VFB ² Framework	215
8.3.2	Details of Algorithms	217
8.3.3	Scenario Case Study	218
8.4	Theoretical Analysis	221
8.4.1	Convergence Analysis for Strongly Convex Problem	223
8.4.2	Convergence Analysis for Non-convex Problem	225
8.4.3	Security Analysis	226
8.5	Experiments	228
8.5.1	Experiment Settings	228
8.5.2	Evaluations of Asynchronous Efficiency and Scalability	229
8.6	Conclusion	232
Chapter 9 Vertical Federated Deep Learning Algorithms		233
9.1	Introduction	234
9.2	Vertical Federated Deep Learning Algorithms	234
9.2.1	Algorithm Framework	235
9.2.2	Algorithm Details	237
9.2.3	Scenario Case Study	238
9.3	Theoretical Analysis	241
9.3.1	Complexity Analysis	241
9.3.2	Security Analysis	241

9.4	Experiments	242
9.4.1	Experimental Setup	242
9.4.2	Results and Discussion	242
9.5	Conclusion	243
Chapter 10 Faster Secure Data Mining Framework via Homomorphic Encryption		245
10.1	Introduction	246
10.2	Related Work	248
10.3	Faster Secure Data Mining Framework via Homomorphic Encryption	250
10.3.1	Algorithm Framework	250
10.3.2	Algorithm Description	252
10.4	Experimental Results	256
10.4.1	Distributed Learning	257
10.4.2	Federated Learning	262
10.5	Conclusion	264
Chapter 11 Horizontal Federated Learning Algorithms		267
11.1	Introduction to Horizontal Federated Learning	267
11.2	Common Distributed Optimization Algorithms	270
11.2.1	Synchronous Parallel Algorithms	272
11.2.2	Asynchronous Parallel Algorithms	275
11.3	Synchronous Horizontal Federated Learning Algorithms	278
11.4	Asynchronous Horizontal Federated Learning Algorithms	283
11.5	Rapid Communication in Horizontal Federated Learning Algorithms	289
11.6	Conclusion	292
Chapter 12 Mixed Federated Learning Algorithms		295
12.1	Scenario Requirements for Mixed Federated Algorithms	295
12.2	Algorithm Details	298
12.2.1	Algorithm Design	299
12.2.2	Model Storage and Mixed Inference	306

12.3	Background of the Mixed Federated Linear Regression Algorithm	309
12.4	Objective Function of Mixed Linear Regression	311
12.5	Improvements in First and Second Derivative Calculations	313
12.5.1	For Private Data	315
12.5.2	For ID-Overlapping Data	315
12.6	Secure Computation Protocol	317
12.6.1	Key Generation Based on Secret Sharing	318
12.6.2	Homomorphic Encryption with Threshold Decryption	318
12.6.3	Secure Federated Training Method	319
Chapter 13 Federated Reinforcement Learning		321
13.1	Overview of Reinforcement Learning	322
13.1.1	Markov Property	322
13.1.2	Various Types of Policies	323
13.1.3	Expected Returns	324
13.1.4	Different Components and Setups for Learning Policies	325
13.2	Introduction to Reinforcement Learning Algorithms	328
13.2.1	Value-Based RL	328
13.2.2	Fitted Q -Learning Algorithm	329
13.2.3	Deep Q -Networks	330
13.2.4	Double DQN	331
13.2.5	Policy-Based RL	333
13.2.6	Stochastic Policy Gradient	333
13.2.7	Deterministic Policy Gradient	335
13.2.8	Actor–Critic Method	336
13.2.9	Model-Based RL	338
13.3	Distributed and Federated Reinforcement Learning	343
13.3.1	Distributed Reinforcement Learning	343
13.3.2	Federated Reinforcement Learning	345
13.4	Conclusion	348

Part III: Federated Learning Systems 351

Chapter 14	Detailed Exploration of the FedLearn Federated Learning System	353
14.1	Open-Source Federated Learning Systems and Their Pain Points	353
14.1.1	Programming Language and Environment . . .	353
14.1.2	Big Data and Computational Efficiency	355
14.2	Advantages of the FedLearn Federated Learning System	356
14.3	FedLearn System Architecture Design	358
14.3.1	Common Federated Learning System Architectures	358
14.3.2	Overview of FedLearn Architecture	359
14.3.3	FedLearn Standard Architecture Features	361
14.3.4	Distributed Federated Learning	364
14.3.5	Blockchain Federated Learning Architecture	367
14.4	Cross-Language Support in FedLearn	368
14.4.1	Introduction to Remote Procedure Call Technology	370
14.5	High-Performance Open-Source RPC Framework: gRPC	376
14.5.1	Unique Advantages of gRPC	376
14.5.2	Key Concepts of gRPC	379
14.6	Decoupling of FedLearn System Services and Algorithms	385
14.6.1	Phase Concept and Automata	385
14.6.2	Componentization	385
14.6.3	Other System-Level Optimizations	387
14.7	FedLearn Deployment and Use	387
14.7.1	System Components and Functionality	387
14.7.2	Standard Deployment	388
14.7.3	Deployment for Distributed Version	392
14.7.4	Container-Based Deployment	394
14.7.5	User Interface Operations and API	394

Chapter 15	Application of gRPC in FedLearn	401
15.1	Application Example I: Vertical Federated Random Forest Algorithm	401
15.1.1	Overview of the Algorithm Process	403
15.2	Application Example II: Horizontal Federated Learning Scenario	407
15.2.1	Brief on Horizontal Federated Learning	407
15.2.2	Framework Design and Implementation of Horizontal Federated Learning in FedLearn . . .	408
15.2.3	Utilizing gRPC to Support Diverse Model Types	413
Chapter 16	Performance Optimization Practices in Real-World Scenarios	417
16.1	Introduction to FedLearn Business Scenarios	417
16.1.1	Precise Marketing Monitoring of Financial Products	417
16.1.2	Intelligent Credit Scoring	418
16.2	From 0-1: Practical Optimization of Federated Learning Algorithms	419
16.3	Performance Optimization	421
16.3.1	GMP Computing Library	423
16.3.2	Accelerating Homomorphic Encryption Computations with the GMP Library in Python	423
16.3.3	Homomorphic Encryption Computation Protocol Optimization	428
16.4	Optimization of Engineering Service Performance	431
16.4.1	Parallel Optimization	431
16.5	Optimizing Information Transfer across Multiple Machines	434
16.5.1	Streamlining Data Transfer	434
16.5.2	Communication Message Compression and Asynchronous Communication	436
16.6	Real-Time Inference Optimization	438
16.6.1	Conclusion	439

Chapter 17	Federated Learning Based on Blockchain	441
17.1	Introduction to Blockchain	441
17.1.1	Overview of Blockchain	441
17.1.2	Technical Features of Blockchain	443
17.1.3	Integration of Blockchain Technology with Modern Technologies	450
17.2	Innovative Integration of Federated Learning and Blockchain	452
17.2.1	Innovations in Architecture	453
17.2.2	Innovations in Process	454
17.2.3	Data Support	455
17.2.4	Incentive Support	456
17.2.5	Regulatory and Audit Support	457
17.3	Blockchain-Based Federated Learning Incentive Algorithms	457
17.3.1	Model Quality Assessment Incentive Algorithm	458
17.3.2	Weighted Incentive Algorithm	462
17.3.3	Incentive Allocation Algorithm	468
17.4	Implementation of a Blockchain-Based Federated Learning System	471
17.4.1	System Model	471
17.4.2	System Architecture	476
	<i>Bibliography</i>	483
	<i>Index</i>	525

This page intentionally left blank

Part I

Federated Learning Knowledge

This page intentionally left blank

Chapter 1

Introduction to Federated Learning

As public concern over personal privacy leaks grows and related laws and regulations are enacted, traditional AI technologies urgently need to adapt to the new circumstances. Federated Learning (FL), as one of the technological solutions, has received a great deal of attention from researchers and industry professionals. This chapter aims to provide readers with a comprehensive introduction to FL, in the hope of serving the public and the readership.

1.1 What Is Federated Learning

The year 2016 marked the maturity of Artificial Intelligence (AI). With AlphaGo defeating top human Go players, we truly witnessed the immense potential of AI and began to look forward to more complex and cutting-edge AI technologies being applied in various fields, including autonomous driving, biomedicine, finance, and more. Today, AI technology demonstrates its advantages in almost all industries and professions. The latest AlphaFold 2 technology can even predict the structures of 350,000 proteins, virtually covering 98.5% of all human proteins. However, the success of these technologies largely relies on vast amounts of data. For instance, the development of various techniques in computer vision such as image classification and object detection, depends on large-scale image datasets like ImageNet, COCO, and PASCAL VOC.

In the field of autonomous driving, numerous domestic and foreign manufacturers have accumulated hundreds of thousands of kilometers of road test data. In 2016, AlphaGo used a total of 300,000 games as training data, which contributed to its outstanding performance.

With the success of AlphaGo, people naturally hope that AI technologies driven by big data will soon be applied in all aspects of our lives. However, the reality is somewhat disappointing: Except for a few industries, most fields only have limited or poor quality data, making the implementation of AI technology more difficult than we imagine. Can we overcome this by transferring data across organizations and merging them at a common site? In fact, breaking down barriers between data sources (data owners) is often very difficult, if not impossible. Generally, any AI project requires data of various types. For example, in AI-driven product recommendation services, product sellers have product information and user purchase data but lack data describing user purchasing power and payment habits. In most industries, data exist in isolated silos. Due to industry competition, privacy and security, complex administrative procedures, and other reasons, even data integration between different departments of the same company faces significant resistance. Consolidating data scattered across the country and various institutions is virtually impossible or economically infeasible.

Meanwhile, as more and more large companies realize the severity of damaging data security and user privacy, emphasizing data privacy and security has become a global major issue. News related to public data leaks has attracted significant attention from public media and governments. For instance, a recent data leakage incident involving a foreign social media giant sparked widespread protest. In response, countries worldwide are strengthening laws to protect data security and privacy. For example, the General Data Protection Regulation (GDPR), enforced by the EU on May 25, 2018 (see Figure 1.1), is aimed at protecting users' personal privacy and data security. It requires companies to use clear language in user agreements and grants users the "right to be forgotten", i.e., personal data can be deleted or withdrawn. Companies that violate the law face hefty fines. The US and China are also implementing similar privacy and security measures. For instance, China's Cybersecurity



Figure 1.1. The General Data Protection Regulation (GDPR).

Law and General Provisions of the Civil Law, enacted in 2017, stipulate that internet companies should not disclose or tamper with the personal information they collect and must ensure that proposed contracts comply with legal data protection obligations when trading data with third parties. The establishment of these regulations certainly contributes to a more civilized society but also poses new challenges to the data trading procedures commonly used in current AI.

Specifically, traditional data processing models in AI often involve a simple data transaction model where one party collects user data and transmits it to another. The receiving party is responsible for cleaning and integrating the data. Ultimately, a third party will use the integrated data to build models for other parties' use. The model is usually the final product sold as a service. This traditional procedure faces challenges from the aforementioned new data regulations. Moreover, these transactions may violate laws and regulations like GDPR because users may not be aware of the future uses of these models. As a result, data users are confronted with a dilemma — data exist in the form of islands, but in many cases, data users are prohibited from collecting, integrating, or transmitting data to other organizations or individuals for AI processing. Therefore, how to legally and compliantly address the issues of data fragmentation and isolation will be an important challenge for future AI researchers and practitioners.

1.1.1 History of Federated Learning

The term federated learning (FL) was introduced by McMahan *et al.* (2016):

We term our approach FL, since the learning task is solved by a loose federation of participating devices (which we refer to as clients) which are coordinated by a central server.

Imbalanced, non-Identically and Independently Distributed (non-IID) data divisions across a large number of unreliable devices with limited communication bandwidth have been introduced as a series of decisive challenges. Significant related work had already been carried out before the term FL was coined. A long-term goal pursued by many research groups (from various fields like cryptography, databases, and machine learning (ML)) is to analyze and learn data distributed across many owners without exposing or leaking these data. Cryptographic methods for computing on encrypted data began in the early 1980s (see the paper by Rivest *et al.* in (1982)) and examples of early attempts to learn from local data while preserving privacy using a centralized server include the works of Agrawal and Srikant (2000) and Vaidya *et al.* (2000, 2008). On the contrary, even with the introduction of the term FL, we have not found any single work that directly addresses all of FL’s challenges. Therefore, the term FL provides a convenient shorthand for a set of characteristics, constraints, and challenges that frequently co-occur in ML applications on decentralized data, where privacy is of paramount importance.

A key attribute of many open challenges in the field of FL is that they are inherently interdisciplinary. Addressing these problems may require not only ML but also technologies from distributed optimization, cryptography, security, differential privacy, fairness, compressed sensing, systems, information theory, statistics, and other areas. Many of the most difficult problems are at the intersections of these fields, so we believe that cross-disciplinary collaboration will be crucial for the continued development of FL. When FL was first proposed, it attracted attention from application scenarios, such as mobile and edge devices. Since then, the use cases for FL have been growing, for example, multiple organizations collaborating to train a

model. The above-mentioned changes related to FL led to its broader definition.

Definition 1.1. FL is a ML framework where multiple entities (clients) collaborate to solve a ML problem under the coordination of a central server or service provider. The original data for each client remains stored locally and is not exchanged or directly transmitted; instead, targeted updates intended for immediate aggregation are utilized to achieve the learning objective.

Targeted updates refer to updates in a narrower sense, containing the minimum information required for a specific learning task at hand; in the service of data minimization, aggregation operations are executed as early as possible. While privacy-preserving data analysis research has spanned over 50 years, it's only within the last decade that large-scale solutions have been widely deployed, such as Rappor. Cross-device FL and federated data analysis are being applied to consumer digital products. For instance, FL is widely used in the Gboard mobile keyboard, Pixel phones, and Android Messages. In iOS 13, cross-device FL is used for applications like the QuickType keyboard and the “Hey Siri” voice classifier. Applications across data silos have been proposed or described in countless fields, including financial risk prediction in reinsurance, drug discovery, mining electronic health records, medical data partitioning, and intelligent manufacturing. The growing demand for FL technology has resulted in the emergence of various tools and frameworks, including TensorFlow Federated, Federated AI Technology Enabler (FATE), PySyft, Leaf, PaddleFL, and the Clara Training Framework. More detailed information on the similarities and differences between various frameworks can be found in the review by Kairouz *et al.* (2019). Established tech companies and smaller startups are also developing commercial data platforms utilizing FL technology.

1.1.2 Federated Learning Overview

Before introducing the training process of FL, let's first provide a sketch of an FL model's lifecycle. The FL process is typically driven by engineers who develop models for specific applications. For example, experts in the field of natural language processing may develop a

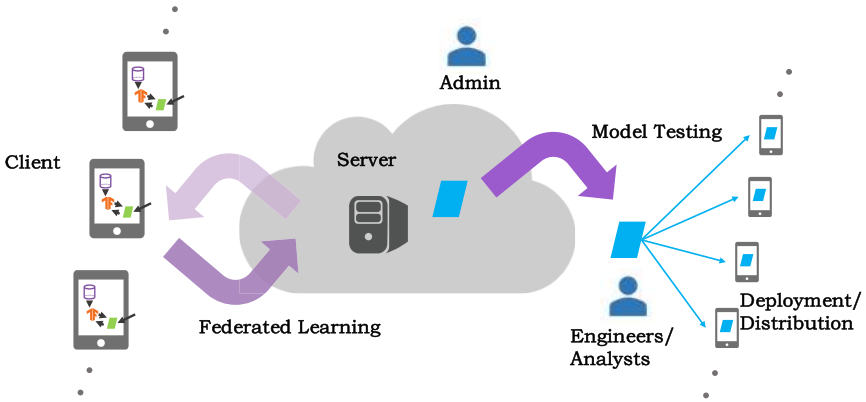


Figure 1.2. Vertical federated learning (FL split by feature).

next-word prediction model for a virtual keyboard. Figure 1.2 shows the main components and participants. At a higher level, the typical workflow includes the following:

- ❑ **Problem Identification:** The model engineer identifies a problem to be solved with FL.
- ❑ **Client Instrumentation:** If necessary, the client (such as an application running on a mobile phone) is instrumented to locally store necessary training data (subject to time and quantity limits). In many cases, the application has already stored these data (e.g., a messaging app must store text messages and a photo management app has photos stored). However, in some cases, it may be necessary to maintain additional data or metadata, such as user interaction data, to provide labels for supervised learning tasks.
- ❑ **Simulation Prototyping (Optional):** The model engineer may prototype the model architecture and test learning hyperparameters in an FL simulation using a proxy dataset.
- ❑ **Federated Model Training:** Multiple federated training tasks are initiated to train different variants of the model or with different optimization hyperparameters.
- ❑ **(Federated) Model Evaluation:** After the task has been sufficiently trained (usually after several days), the models are analyzed and good candidates are chosen. Analysis may include metrics computed on standard datasets in the data center, or federated

evaluations, where the model is pushed to a reserved set of clients for evaluation on local client data.

- ❑ **Deployment:** Finally, once a good model is chosen, it undergoes a standard model release process, including manual quality assurance, live A/B testing (usually with the new model on some devices and the previous model on others to compare *in vivo* performance), and staged rollouts (so that detrimental behaviors can be discovered and rolled back before affecting too many users). The specific release process for a model is determined by the application's owner and is generally unrelated to how the model was trained. In other words, this step applies to models trained via either FL or traditional data center methods.

One of the main practical challenges faced by FL systems is making the above workflow as simple as possible, ideally approaching the usability of ML systems that implement centralized training.

We now introduce an FL training template, which works for the federated averaging algorithm proposed by McMahan *et al.* (2016a), and many other algorithms. Likewise, there are some possible variations today, but this provides a common starting point.

The server (service provider) organizes the training process by repeating the following steps until training stops (as determined by the model engineer monitoring the training process):

- ❑ **Client Selection:** The server selects a group of eligible clients. For example, to avoid affecting the user of a device, a mobile phone might only check in with the server when it's plugged in, has an uncapped Wi-Fi connection, and is idle.
- ❑ **Broadcast:** The selected clients download the current model weights and a training program (such as a TensorFlow Graph) from the server.
- ❑ **Client Computation:** Each selected device computes an update to the model locally by executing the training program, for example, the training program might run Stochastic Gradient Descent (SGD) on local data (as in the FederatedAveraging algorithm).
- ❑ **Aggregation:** The server aggregates the updates from the devices. For efficiency, straggling devices may be dropped once enough devices have reported results. This stage is also an

integration point for many other technologies, which is discussed later, and may include secure aggregation for increased privacy, lossy compression for communication efficiency, and the addition of noise and update clipping for differential privacy.

- **Model Update:** The server updates the shared model locally based on the aggregate update computed from the clients participating in the current round.

Separating the client-side computation, aggregation, and model update stages is not a strict requirement for FL, but it does exclude certain algorithm classes, such as Asynchronous Stochastic Gradient Descent (ASGD), where updates from each client are applied to the model immediately before any aggregation with updates from other clients. This asynchronous approach might simplify aspects of system design and can be beneficial from an optimization perspective. However, the aforementioned method has significant advantages in distinguishing the focus of different research directions: Advancements in compression, differential privacy, and secure multi-party computation can be applied to standard primitives, such as calculating sums or means via decentralized updates, which can then be composed by any optimization or analysis algorithm, as long as these algorithms are expressed in terms of aggregation primitives.

It is worth emphasizing that the FL training process should not impact user experience. First, as mentioned above, even though model parameters are usually sent to some devices during the broadcast phase of each round of federated training, these models are just transient parts of the training process and are not used to display real-time predictions to the user. This is critically important as training ML models is challenging, and a misconfiguration of a single hyperparameter can produce a model that makes incorrect predictions. Instead, the user-visible models are deployed during a phased rollout process described in detail in Step 6 of the model lifecycle. Second, the purpose of the training itself is to remain unseen by the user, as in client selection, training does not slow down the device or drain the battery, as it only occurs when the device is idle and plugged in. However, these limitations on availability lead directly to open research challenges, such as semi-cyclic data availability and potential biases in client selection.

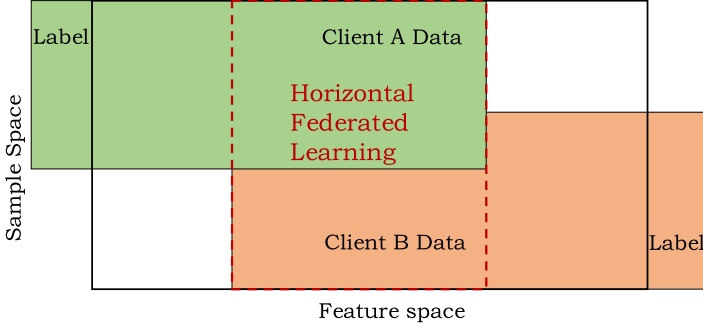


Figure 1.3. Horizontal federated learning (federated learning partitioned by sample).

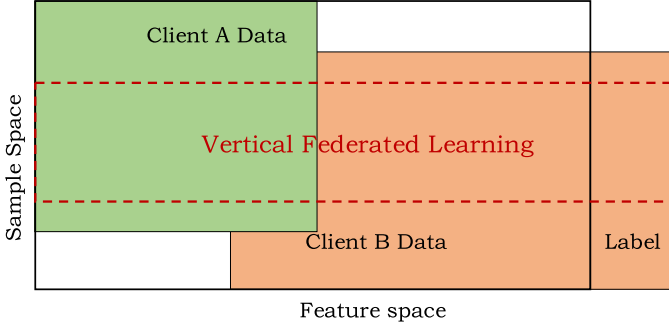


Figure 1.4. Vertical federated learning (FL partitioned by feature).

1.1.3 Classification of Federated Learning

Based on the different distributions of data and features, we can divide FL into two categories: Horizontal Federated Learning (HFL) and Vertical Federated Learning (VFL). Suppose we have a FL scenario with two participants. Figures 1.3 and 1.4 illustrate the definitions of the two types of federated learning.

HFL is suitable when participants in FL have overlapping data features, i.e., data features are aligned among participants, but the data samples they own are different. This type of HFL is akin to horizontally (or sample-wise) partitioning data in a matrix (table), hence we also refer to HFL as sample-partitioned FL or sample-distributed FL. Unlike HFL, VFL applies when participants in FL

have overlapping data samples, i.e., data samples are aligned among participants, but the data features they own are different. This VFL is similar to vertically (or feature-wise) partitioning data in a matrix (table), hence we also refer to VFL as feature-partitioned FL or feature-distributed FL. Recently proposed split learning, which involves partitioning the computation of deep neural network models between client and server, and learning on vertically partitioned data, can be considered a special form of VFL.

For example, consider two commercial banks serving different regions participating in FL. They might have very different user groups from their respective regions, with a small intersection between their user populations. However, their businesses are very similar, hence the feature spaces are the same. Therefore, these two banks can cooperatively learn a ML model through HFL. On the contrary, consider two different companies in the same city: one a bank and the other an e-commerce company. Their user sets probably cover most residents in the area, hence a large intersection exists between their user populations. However, since the bank records user income and expenditure behavior as well as credit ratings, and the e-commerce company retains user browsing and purchasing history, the feature spaces are significantly different. Suppose we want both parties to have a product purchase prediction model based on user and product information. VFL aggregates these different features and computes training loss and gradients while preserving privacy, thus collaboratively building a model that includes data from both parties. In other words, HFL typically involves participants from the same industry cooperatively learning, while VFL tends to involve cooperative learning between cross-industry participants. We introduce HFL and VFL and their related technologies in detail in Chapters 11 and 4–10, respectively. Additionally, there is a special type of FL, federated reinforcement learning, which is detailed in Chapter 13.

1.2 Current Application State and Development Trends of Federated Learning

In this section, we introduce the current application state and future development trends of FL.

1.2.1 Current Applications of Federated Learning

FL, as an innovative modeling mechanism, allows unified model training across multi-party data while ensuring data privacy and security (considered only at the technical level). This presents excellent application prospects in sales, finance, and many other industries.

Take intelligent retail as an example. The aim is to use ML technology to provide personalized services to customers, mainly including product recommendations and sales services. The data features involved in intelligent retail mainly include consumer purchasing power, personal preferences, and product characteristics. In practical applications, these three types of data features are likely scattered among three different departments or companies. For example, purchasing power can be inferred from bank deposits and personal preferences can be analyzed from social networks, while e-stores record product characteristics. In this situation, two issues arise: (1) To protect data privacy and security, it's challenging to break down the data barriers between banks, social networks, and e-stores, which makes it impossible to directly aggregate data to train models. (2) The data stored by the three parties are usually heterogeneous, which conventional ML models can't directly handle. At present, traditional ML methods have not effectively solved these problems, impeding the promotion and application of AI in more fields.

FL is key to resolving these issues. First, by leveraging the characteristics of FL, a tripartite ML model can be constructed without exporting enterprise data. This not only technically ensures data privacy and security but also provides personalized, targeted services to customers, achieving a win-win situation. Therefore, FL provides good technical support for us to build a big data and AI ecosystem that spans enterprises, data, and fields. FL frameworks can be used to query multi-party databases without exposing data. For instance, in financial applications, we're interested in detecting multi-party loans, a major risk factor in banking. This occurs when some users maliciously borrow from one bank to pay off loans at another. Multi-party loans pose a threat to financial stability because a large number of such illegal activities could lead to a systemic collapse of the entire financial structure. To identify these users without mutually exposing user lists between Bank A and Bank B, we can utilize the FL

framework. Specifically, we can use the encryption mechanism of FL to encrypt the user lists of each party and then take the intersection of the encrypted lists in the federation. The decryption of the final result provides a list of multi-party borrowers without exposing other “good” users to the other party. As we see in the following, this operation corresponds to the VFL framework. Intelligent healthcare is another field that we expect will benefit significantly from the rise of FL technology. Medical data such as disease symptoms, genetic sequences, and medical reports are very sensitive and private, but they are difficult to collect and exist in isolated medical centers and hospitals. The lack of data sources and labels has led to suboptimal performance of ML models, becoming a bottleneck for current intelligent healthcare. We imagine that if all medical institutions come together to collaboratively train a ML model without leaking data privacy, the model’s performance will significantly improve. Moreover, FL can be combined with specific technologies, such as computer vision, natural language processing, edge computing and cloud computing, and feasible computer hardware, which we elaborate on in Chapter 2.

1.2.2 Development Trends of Federated Learning

Next, we introduce four core challenges in the development trends of FL.

Challenge 1. Expensive communication costs: In a federated network, communication is a key bottleneck, and privacy issues when sending raw data make it necessary to store the data generated on each device locally. In fact, federated networks can consist of a large number of devices, such as millions of smartphones, and communication in the network may be many orders of magnitude slower than local computation. Therefore, it is necessary to develop communication-efficient iterative methods that send small messages or portions of the model updates during the training process rather than sending the entire dataset over the network. In this case, two key aspects need to be considered to further reduce communication: (1) reducing the number of communication rounds or (2) reducing the size of information transferred in each round.

Challenge 2. System heterogeneity: The storage, computation, and communication capabilities of each device in a federated network may vary due to changes in hardware (CPU or memory), network connections (3G, 4G, 5G, or WIFI), and power (battery level). In addition, the network size and system-related constraints on each device usually result in only a small portion of devices being active at the same time. For example, there may be hundreds of active devices in a network with millions of devices. These devices may also be unreliable, as active devices frequently drop out in a given iteration due to connection or energy limitations. These system-level characteristics significantly exacerbate challenges, such as reducing dropouts and fault tolerance. Thus, the FL methods that are developed and analyzed must (1) anticipate low participation, (2) tolerate heterogeneous hardware, and (3) be robust to device losses in the network.

Challenge 3. Statistical heterogeneity: Devices often generate and collect data in the network in non-identical distributions. For example, mobile phone users use different languages in the context of a next-word prediction task. Additionally, the number of data points between devices may vary greatly, and there may be an underlying structure to capture the relationship between devices and their associated distributions. This data generation paradigm violates the independent and identically distributed assumption often used in distributed optimization, increasing the likelihood of scatter and potentially complicating modeling, analysis, and evaluation. In fact, although common FL problems aim to learn a single global model, there are also other alternatives, such as learning different local models simultaneously through a multi-task learning framework (refer to the work of Smith *et al.* (2017)). In this respect, there are also close connections between FL and the main methods of meta-learning (refer to the work of Li *et al.* (2019)). Both multi-task and meta-learning perspectives support personalized or device-specific modeling, which is often a more natural way to handle data statistical heterogeneity.

Challenge 4. Privacy issues: Finally, privacy is often a major concern for FL applications. FL takes a step toward protecting the data generated on each device by sharing model updates such as gradient

information or model parameters rather than raw data (refer to the work of Carlini *et al.*, 2018). However, communicating model updates throughout the training process can still leak sensitive information to third parties or central servers (refer to the work of McMahan *et al.* (2017), on language models). Although recent methods aim to enhance the privacy of FL through tools such as secure multi-party computation or differential privacy, these methods often provide privacy protection at the expense of reducing model performance or system efficiency. Understanding and balancing these trade-offs, both theoretically and empirically, is a considerable challenge for achieving privacy-preserving FL systems.

The above four challenges will also be the main trends and directions for the development of FL practitioners in the future.

1.3 Distributed Machine Learning and Federated Learning

Distributed ML, also known as distributed learning, refers to algorithms and systems that utilize multiple computing nodes (also referred to as workers) for ML or deep learning. These are aimed at improving performance, safeguarding privacy, and scaling to larger training datasets and larger models. FL can be seen as a special type of distributed learning that can further address some difficulties encountered in distributed ML, thereby enabling us to construct applications and products for privacy-conscious AI.

1.3.1 History of Distributed Machine Learning

In recent years, rapid advancements in technology have led to unprecedented growth in data collection. ML algorithms are increasingly used for dataset analysis and decision-making system development. However, due to the complexity of the problems, algorithmic solutions are not always feasible, such as controlling autonomous cars, recognizing speech, or predicting consumer behavior (refer to the work of Khandani *et al.* (2010)). In some cases, the extended run time for model training guides solution designers to use distributed systems to increase parallelization and the total amount of

I/O bandwidth, given that training data required for complex applications can easily reach the terabyte scale. In other cases, where the data themselves are distributed or too large to be stored on a single machine, centralized solutions may not even be desirable, like transaction processing of data stored in different locations in large enterprises, or astronomical data that are too extensive to be moved and centralized. To allow these types of datasets to be accessed as training data for ML problems, one must select and implement algorithms capable of parallel computation, adapting to various data distributions, and possessing fault tolerance capabilities.

ML techniques have seen wide application in increasingly complex scenarios in recent years. While various competing methods and algorithms have emerged, the data representations used are surprisingly similar in structure. Most of the computation in ML work involves basic transformations on vectors, matrices, or tensors, which are common issues in linear algebra. Over the decades, the demand for optimization of these operations has been a highly active research area in the field of High Performance Computing (HPC). As a result, certain technologies and libraries from the HPC community, such as BLAS or MPI, have been successfully adopted and integrated into systems by the ML community. Meanwhile, the HPC community has identified ML as an emerging high-value workload and has started to apply HPC methods to it. Coates *et al.* (2013) were able to train a network with 1 billion parameters in just three days on their commercial off-the-shelf high-performance computing (COTS HPC) systems. You *et al.* proposed in 2017 to optimize the training of neural networks on Intel's Knights Landing, a chip designed for HPC applications. Kurth *et al.* (2017) demonstrated in 2017 how deep learning problems, such as extracting weather patterns, are optimized and scaled on large parallel HPC systems. Yan *et al.* (2016) proposed to model workload demands by borrowing techniques such as lightweight analytics from HPC, solving the challenges of scheduling deep neural network applications on cloud computing infrastructure. Li *et al.* (2017) studied the resilience characteristics of deep neural networks running on accelerators against hardware errors in 2017. Accelerators are often deployed in major HPC systems.

Like other large-scale computing challenges, there are two fundamentally different and complementary ways to accelerate

workloads: adding more resources to a single machine (vertical scaling or scaling up, like the continuous improvement of GPU/TPU computing cores) or adding more nodes to the system (horizontal scaling or scaling out, which is cost-effective).

The boundaries between traditional supercomputers, grids, and clouds are increasingly blurred when it comes to the optimal execution environment for demanding workloads such as ML. For instance, GPUs and accelerators are now more common in major cloud data centers. Therefore, parallelization of ML workloads is crucial for large-scale implementation with acceptable performance. However, when transitioning from centralized solutions to distributed systems, distributed computing faces typical challenges in terms of performance, scalability, fault resilience, or security.

1.3.2 Overview of Distributed Machine Learning

Designing a universal system that can efficiently distribute conventional ML is a challenge due to each algorithm having unique communication patterns. While distributed ML encompasses various concepts and implementations, we have identified a common architectural framework that covers the entire design space. Each section discusses specific areas where decisions must be made by ML solution designers. Generally, the problems of ML can be divided into the training phase and the inference phase (Figure 1.5). The training phase involves training a ML model by inputting a large amount of training data and using common ML algorithms such as Evolutionary Algorithms (EAs), rule-based ML algorithms (like decision trees and association rules), Topic Models (TM), matrix factorization, and algorithms based on SGD to update the model. Besides choosing a suitable algorithm for the given problem, we also need to tune the hyperparameters for the selected algorithm. The end result of the training phase is a trained model that can then be deployed. The inference phase is used to deploy the trained model in practice. The trained model accepts new data as input and generates predictions as outputs. While the model's training phase usually requires a large amount of computation and a large dataset, inference can be performed with less computational power. The training and inference phases are not mutually exclusive. Incremental learning combines

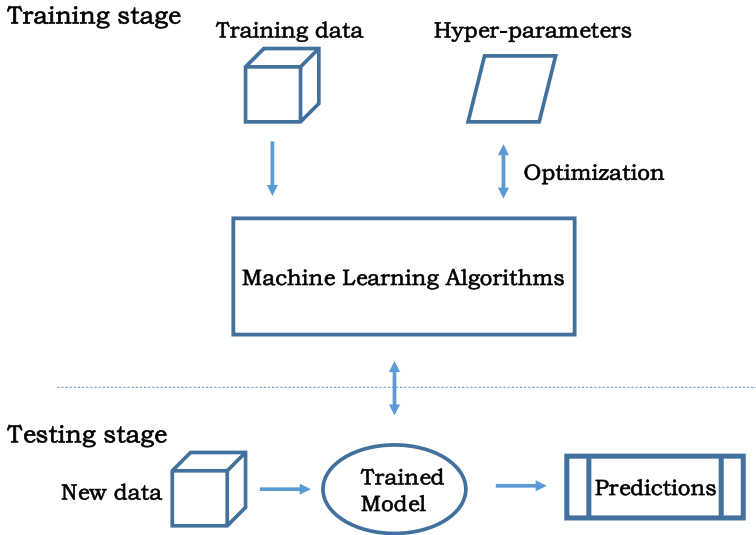


Figure 1.5. Overview of ML. In the training phase, the ML model is optimized using training data and adjusting hyperparameters. Then, the trained model is deployed in the system to provide predictions for new data inputs.

the training and inference phases, using new data from the inference phase for continuous training of the model.

When it comes to distribution, there are fundamentally two different ways to distribute the problem across all machines, either by data or model parallelization (Figure 1.6). Both methods can also be applied simultaneously. In the data parallel method, data are partitioned as many times as there are worker nodes in the system, and then all worker nodes apply the same algorithm to different datasets. The same model can be used across all worker nodes (either centralized or replicated), thus naturally producing a single consistent output. This technique can be used for each ML algorithm that assumes independent and identically distributed (IID) samples on the data. In the model parallel method, an exact replica of the entire dataset is processed by the worker nodes, with the worker nodes operating different parts of the model. Therefore, the model is an aggregation of all model parts. The model parallel method cannot be automatically applied to every ML algorithm, as model parameters often can't be split.

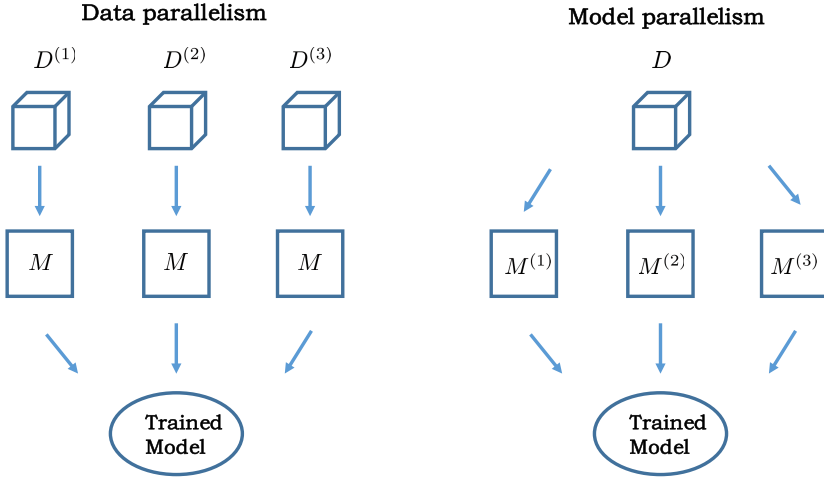


Figure 1.6. Parallelism in distributed ML. Data parallelism is training multiple instances of the same model on different subsets of the training dataset, whereas model parallelism is distributing parallel paths of a single model across multiple nodes.

One option is to train different instances of the same or similar models and aggregate the outputs of all trained models using ensemble techniques, such as bagging or boosting. The final architecture decision is the topology of the distributed ML system. The different nodes that make up the distributed system need to be connected through specific architectural patterns to achieve rich functionality. This is a common task. However, the choice of pattern affects the roles that nodes can play, the degree of communication between nodes, and the fault-tolerance capability of the entire deployment. Figure 1.7 shows four possible topologies that align with Baran’s (1964) general classification of distributed communication networks.

Centralized systems (Figure 1.7(a)) adopt a strict hierarchical method for aggregation, which occurs at a single central location. Decentralized systems allow for intermediate aggregation, with model replicas continuously updated when aggregation is broadcast to all nodes — as in tree topologies (Figure 1.7(b)) — or using a partitioned model sharded across multiple parameter servers (Figure 1.7(c)). Fully distributed systems (Figure 1.7(d)) consist of

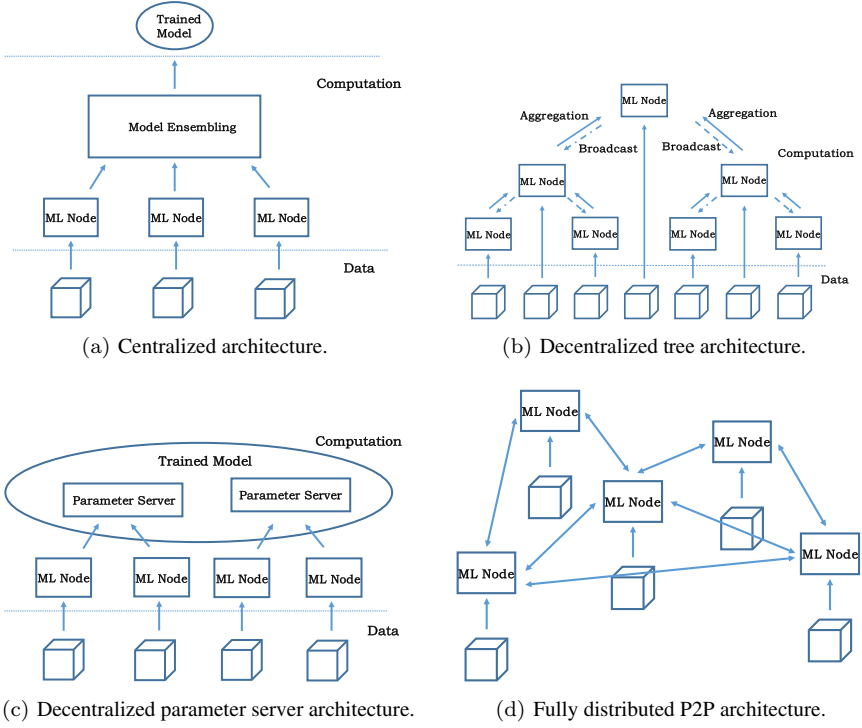


Figure 1.7. Exhibition of distributed ML topologies based on the degree of distribution.

a network composed of independent nodes, which integrate solutions without assigning specific roles to certain nodes.

1.3.3 The Co-development of Distributed Machine Learning and Federated Learning

The evolution of distributed ML has also led to some privacy protection requirements, thus intersecting with FL in some content areas. Common encryption methods like secure multi-party computation, homomorphic encryption, and differential privacy are gradually being employed in distributed ML. Broadly speaking, FL is an effective method of utilizing distributed resources for the collaborative training of ML models. FL is a distributed ML method wherein multiple users cooperatively train a model while keeping the original data dispersed, not moved to a single server or data center. FL not only

leverages distributed resources to efficiently carry out the ML training process but also promises to provide security and privacy assurances for dispersed original data from a technical perspective. In FL, original data or data derived from the original data through secure processing is used as training data. FL allows only the transmission of intermediate data between distributed computing resources while avoiding the transfer of training data. Distributed computing resources refer to end-user mobile devices or servers across multiple organizations. FL brings the code to the data, instead of bringing the data to the code, technically addressing fundamental issues of privacy, ownership, and data locality. In this way, FL enables multiple users to train a model while complying with legal data restrictions.

1.4 Conclusion

This chapter primarily provides an overall introduction to FL, giving readers a general understanding of it. We hope that through the comprehensive explanation in this chapter, readers will develop sufficient interest in FL to contribute their efforts to the advancement of FL technology.

Chapter 2

Federated Learning Application Scenarios

In today's machine learning algorithms, to capture certain relationships (such as those between disease patterns, socio-economic and genetic factors, and complex and rare cases), we need a model adaptable to various situations. This requires us to collect sufficient data that represents the true underlying data distribution of the problem, thereby adequately evaluating our model. However, centralized collection and storage of data from different organizations not only face regulatory, ethical, and legal challenges related to privacy and data protection but also pose technical challenges. Federated learning (FL), by supporting machine learning algorithms from non-coexisting data, can technically address privacy and data governance challenges. This chapter only reviews the application prospects of FL in finance, biomedicine, computer vision (CV), natural language processing (NLP), edge computing and cloud computing, and computer hardware from a technical perspective. If the information involves data collection, please ensure compliance with the laws, regulations, and regulatory requirements of the jurisdiction.

2.1 Federated Learning and Finance

Due to the sensitive nature of financial data, the application of financial data is strictly regulated by customers and regulatory agencies. To effectively unleash the productivity of big data while preserving

privacy, to quickly iterate and innovate business processes, to more agilely adapt to market changes, and to improve business competitiveness, many financial companies and commercial banks are using FL to create a win-win situation for all participants. Common applications now include credit risk control, anti-money laundering, and intelligent ecosystems. For details, please refer to the literature by Zhang *et al.* (2020).

In credit risk control, during the process of commercial bank credit review, multiple external customer credit data interfaces need to be accessed. Most of these data interfaces are not free, and some crucial data cannot be obtained by banks and other financial institutions. As a result, commercial banks, when dealing with customers, especially micro and small customers, lack sufficient credit data support and cannot provide the corresponding credit services, leading to difficulties in financing for small and micro enterprises.

In recent years, artificial intelligence (AI) and other digital financial technologies have been introduced into traditional financial risk control businesses. AI technology comprehensively judges the qualifications and credit status of loan customers, such as the operational status, profitability, income, and future growth potential of small and micro enterprises, serving as a customer admission evaluation. However, the application of AI technology first needs to solve the integration problem of multi-source data required for modeling. Through FL, a new data cooperation model is established to obtain transaction data, tax data, commercial information, and other multi-dimensional feature data from partners for potential bank customers, especially small and micro customers. This helps address the problems of insufficient, incomplete, and inadequate historical data for small and micro customer credit reviews.

Moreover, anti-money laundering, counter-terrorist financing, and compliance related to sanctions are receiving increasing attention globally and are gradually rising to the level of national and regional security, entering the realm of international politics. Both domestic and foreign regulatory bodies are constantly strengthening anti-money laundering supervision, continuously raising supervision standards and requirements. For banks, legal compliance is a basic condition for sustainable and healthy operations, and the construction of a high-standard anti-money laundering system is a threshold that commercial banks must cross for international development.

In bank anti-money laundering operations, traditional practices have issues like utilizing singular data sources, having complex process designs, and requiring extensive manual intervention. Applying FL to bank anti-money laundering operations allows commercial banks and external government agencies, peer banks, and other multi-data owners to utilize FL for joint modeling. This helps address the shortcomings of rule-based models in traditional bank anti-money laundering operations, such as limited data sources and insufficient coverage. At the same time, it allows multiple banking parties to contribute different dimensions of data features for joint modeling, promotes cross-bank anti-money laundering supervision, and can make use of various existing machine learning models to enhance the intelligence level of anti-money laundering operations. It reduces manual judgment and intervention, automatically identifies risk factors, and effectively improves the detection accuracy of the anti-money laundering system, as shown in Figure 2.1.

As economy develops and consumption upgrades continually accelerate, the new retail industry has emerged. In every link of new retail, digitization under lawful and compliant circumstances becomes an important tool for retail companies to cope with change

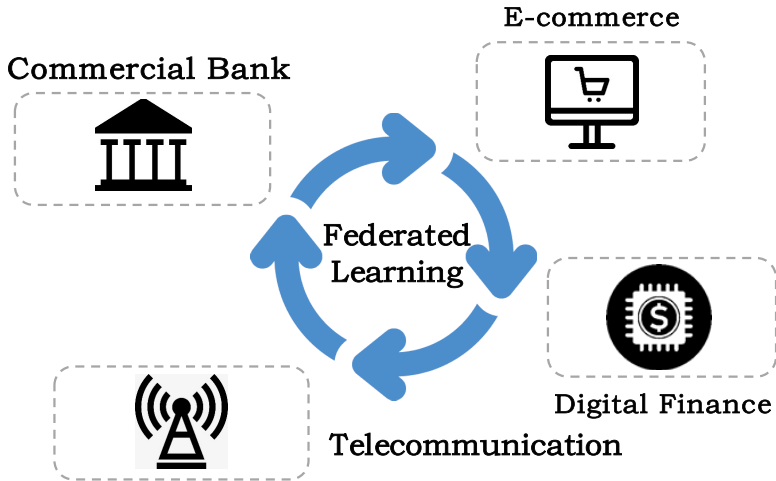


Figure 2.1. Implement FL applications in finance by studying the data characteristics distributed across commercial banks, digital finance, e-commerce, and telecommunications industries.

and challenges. For instance, using fresh products, we can dynamically adjust freshness and discount promotion strategies through dynamic inventory, timely clearance of shelves, and predicting popular products. Simultaneously, we can use intelligent models to predict popular products in advance, providing retail businesses with forecast information, facilitating cooperative marketing strategy optimization, and improving sales conversion. Commercial banks need to plan personalized financial service solutions in this ecosystem for businesses and individuals, enhancing the quality of financial services.

In this smart retail business scenario, the data characteristics mainly include the purchasing power of users, user preferences, and product characteristics. However, in reality, these three data features often involve multiple enterprises, each owning massive user data: Banks hold data on users' purchasing power, social platforms have data on users' personal preferences, and e-commerce platforms possess product characteristic data (which makes up a typical vertical FL scenario). Due to domestic and international data privacy protection laws and regulations, these companies cannot directly merge data from all parties, as the original data providers, i.e., their respective users, also disagree with this approach. But the data volume and data features of each are insufficient to independently build a high-quality model to achieve optimal results.

Through FL, data from each institution doesn't need to be exported and merged. All participants in smart retail can jointly construct machine learning models. On the one hand, this fully protects user privacy and data security; on the other hand, it provides service solutions for each participant in smart retail in financial service design, product configuration, and marketing strategies.

2.2 Federated Learning and Biomedicine

The year 2020 brought numerous changes as we grappled with the effects of the pandemic. One of the most noticeable aspects was the stark shortage of resources in the healthcare sector. As mentioned earlier, if medical datasets come from a limited number of sources, there could be biases in predicting aspects related to demographic information, such as gender and age, or technical imbalances, such as data collection protocols or device manufacturers, adversely affecting

the accuracy for certain groups or locations. However, adapting patterns to different cases is crucial for capturing the subtle relationships between disease patterns, socioeconomic and genetic factors, as well as complex and rare cases.

The demand for large databases for AI training has spurred many initiatives aiming to consolidate data from various institutions. Those data are usually collected into so-called data lakes. Their objective is to capitalize on the commercial value of data, like IBM's consolidated healthcare acquisitions, or as a resource for economic growth and scientific advancement, such as the National Safe Haven of NHS Scotland, the French Health Data Hub, and Health Data Research UK. More modest yet substantial initiatives include the Human Connectome, UK Biobank, The Cancer Imaging Archive (TCIA), CXR8 and DeepLesion of the National Institutes of Health (NIH), The Cancer Genome Atlas (TCGA), the Alzheimer's Disease Neuroimaging Initiative (ADNI), and significant medical competition challenges like the CAMELYON competition, the International Multimodal Brain Tumor Segmentation (BraTS) competition, or the Medical Segmentation Decathlon. Public medical data are usually task or disease-specific and, when published, is often subject to varying degrees of permission restrictions, sometimes limiting its utility.

Consolidating or publishing data not only brings regulatory, ethical, and legal challenges related to privacy and data protection but also introduces technical difficulties. Anonymizing, controlling access to, and securely transmitting medical data is a formidable task, sometimes even an impossible one. Anonymous data in electronic health records might seem harmless and GDPR/PHI compliant, yet only a few data elements can reidentify a patient. The same applies to genomic data and medical images, making them as unique as fingerprints. Therefore, unless the anonymization process compromises the fidelity of the data, rendering it unusable, the potential for patient reidentification or information leakage cannot be ruled out. It is commonly suggested to provide gated access to approved users to tackle this issue. However, besides limiting data availability, this only applies if the data owners unconditionally agree, as retracting data from potential users is practically unfeasible.

FL, by supporting machine learning algorithms from non-cohabitating data, can technically address privacy and data governance challenges. In a FL setting, each data controller not only

defines their own data processing workflow and associated privacy policies but also controls data access and has the ability to revoke it. This applies during both training and validation stages. FL can open up new opportunities, for instance, by allowing extensive intra-institutional validation or by supporting new research on rare diseases, as the incidence rate is low and individual institution datasets are too small. Moving the model to the data, rather than the data to the model, has another significant advantage: High-dimensional, storage-intensive medical data do not have to be copied from local institutions to a central pool and be replicated again by each user training local models on that data. When the model is moved to local institutions, it can naturally scale with the potential growth of global datasets without exponentially increasing data storage demands.

Given that FL is a general learning method, it circumvents the data pool requirements of traditional AI model development, so its applications can span the entire field of AI in healthcare. By providing opportunities to capture larger data variability and analyze patients with different demographic characteristics, FL can bring disruptive innovations to our future lives while being progressively applied across various sectors.

For example, in the context of EHR, FL aids in representing and finding clinically similar patients and in predicting hospitalization due to heart events, mortality rates, and ICU stays, as illustrated in Figure 2.2. The applicability and advantages of FL have also been validated in the field of medical imaging for whole brain segmentation and brain tumor segmentation in Magnetic Resonance Imaging (MRI). Recently, this technique has been used for functional MRI (fMRI) classification to find reliable disease-related biomarkers and is suggested as a promising approach in the context of COVID-19.

It is worth noting that the operation of FL requires a protocol to define the scope, objectives, and techniques used. As it is still a novel technology, it is challenging to determine specifics. In this context, large-scale plans now serve as pioneers in establishing safe, fair, and innovative collaboration standards for healthcare applications tomorrow.

These efforts include alliances aimed to advance academic research, such as the Trustworthy Federated Data Analytics (TFDA)

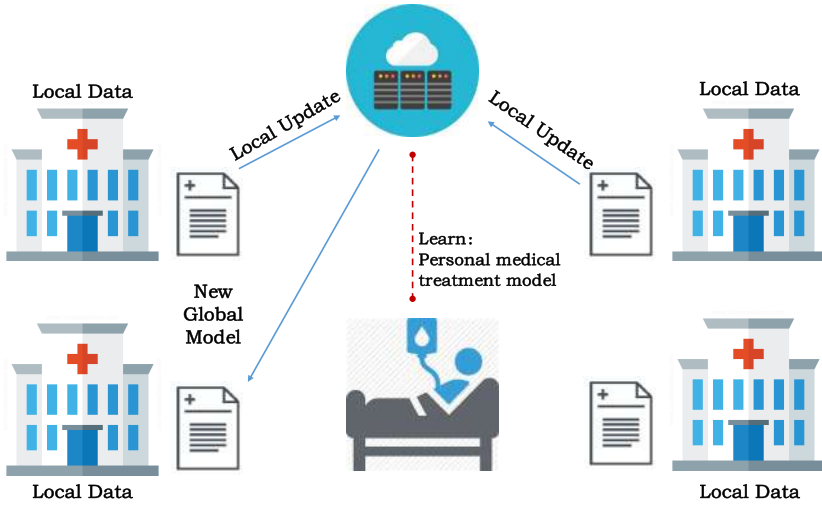


Figure 2.2. Implement FL applications in personalized healthcare through learning from the heterogeneous Electronic Health Records (EHR) spread across multiple hospitals.

project and the German Cancer Consortium’s Joint Imaging Platform, allowing decentralized research across German medical imaging institutions. Another example is an international research collaboration using FL to develop AI models for assessing mammography. Research has shown that models generated by FL outperform those trained on data from a single institution and maintain good performance on data from other institutions, indicating their robust generalizability. However, the use of FL is not confined to academia.

By uniting healthcare institutions (not limited to research centers), FL can have a direct clinical impact, such as in Florida, USA. For instance, the ongoing HealthChain project aims to develop and deploy a FL framework in four hospitals in France. The solution generates general models capable of predicting the treatment responses of breast cancer and melanoma patients, aiding oncologists in determining the most effective treatment approach for each patient from their histological slides or dermatoscopic images. Another large-scale initiative is the Federated Tumour Segmentation (FeTS) project, a consortium composed of 30 committed medical institutions globally,

utilizing an open-source FL framework with a graphical user interface to improve tumor boundary detection, including brain glioma, breast tumor, liver tumor, and skeletal lesions in multiple myeloma patients.

Another domain influenced by FL is industrial research and translation. FL can facilitate collaborative research even among competing companies. One of the most significant initiatives in this regard is the Melloddy project, aiming to deploy multi-task FL across datasets from 10 pharmaceutical companies. By training a general prediction model to infer how compounds bind with proteins, the partners plan to optimize the drug discovery process without revealing their valuable internal data.

2.3 Federated Learning and Computer Vision

Computer Vision (CV) is a scientific discipline that researches how to enable machines to “see”. More precisely, it involves using cameras and computers to replace human eyes in recognizing, tracking, and measuring targets, subsequently performing image processing to transform the images into a format more suitable for human observation or transmission to instruments for further detection.

As a scientific field, CV studies related theories and techniques, aiming to create AI systems capable of extracting “information” from images or multi-dimensional data. Here, the term “information” refers to the concept defined by information theorist Shannon, which can be used to assist in decision-making. Given that perception can be considered as the extraction of information from sensory signals, CV can also be regarded as a science that studies how to enable artificial systems to “perceive” from images or multi-dimensional data.

As an engineering discipline, CV seeks to construct CV systems based on relevant theories and models. The components of these systems include the following:

- process control (e.g., industrial robots and autonomous vehicles),
- event monitoring (e.g., image surveillance),
- information organization (e.g., creating indexes for image databases and image sequences),

- ❑ object and environment modeling (e.g., industrial inspection, medical image analysis, and topological modeling),
- ❑ interactive sensory inputs (e.g., input devices for human–computer interaction).

CV can also be seen as a complement to biological vision. In the field of biological vision, the vision of humans and various animals has been studied, leading to the creation of physical models used in these vision systems' information perception process. On the other hand, in CV, AI systems implemented through software and hardware have been studied and described. The cross-disciplinary exchange between biological vision and CV brings tremendous value to both fields.

CV encompasses many aspects, including image reconstruction, event monitoring, target tracking, target recognition, machine learning, image segmentation, image recovery, and more. Since its first public appearance in the 1980s, CV has rapidly evolved. Its feature extraction methods have gradually shifted from basic image features, such as Histogram of Gradient (HOG) and Scale Invariant Feature Transform (SIFT), to Deep Neural Networks (DNN) and Convolutional Neural Networks (CNN) (compared to DNN, CNN introduces convolution kernels, capturing local features of images and thus enhancing performance). Traditional CV solutions mostly follow a sequence of steps including image preprocessing, feature extraction, model training, and output result. However, as deep learning frameworks gradually develop, more mature frameworks such as Google's TensorFlow, Facebook's PyTorch, Amazon's MXNet, Baidu's PaddlePaddle, and Huawei's MindSpore are available to address CV problems directly through end-to-end solutions. What we need to do is simply input raw data, and the machine and the framework will handle the rest of the complicated engineering tasks automatically.

Thanks to the availability of massive image datasets, such as the classic CIFAR-10/100, STL-10, and the highly popular ImageNet, the field of CV has made substantial progress, pushing the AI industry forward. However, this data-centered development model greatly hinders many small companies from using advanced AI technologies. Although these companies possess limited but highly valuable

data resources, data sharing, a viable method, is not actively pursued due to data privacy concerns, regulatory risks, and business secrecy. Now, with the advent of FL technology, it allows multiple companies to collaboratively construct and train shared AI models using their own data without compromising data privacy. Moreover, FL supports online real-time feedback and dynamic model updating, enabling trained models to meet users’ dynamic demands in real time.

In the field of target recognition and detection, for instance, a paper accepted by the top AI conference AAAI2020¹ introduces an online visual object detection platform based on FL, FedVision. Unlike traditional centralized data aggregation training methods shown in Figure 2.3, this paper proposes a new training method based on FL model aggregation. It allows practitioners without prior knowledge of FL to quickly learn and train this new object detection framework, as shown in Figure 2.4.

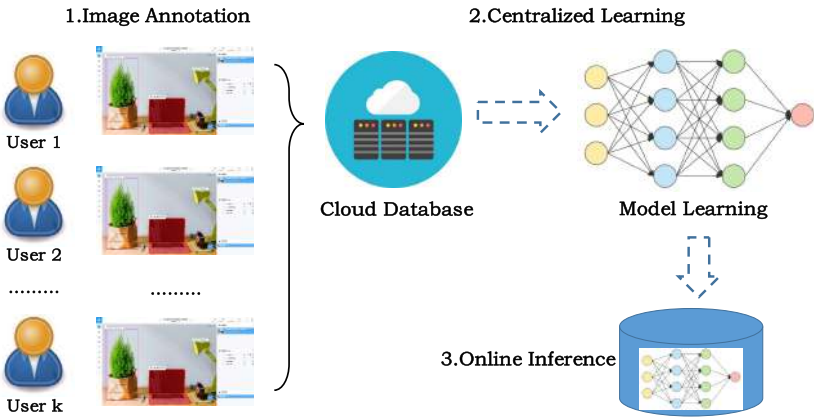


Figure 2.3. Traditional centralized training method.

¹AAAI represents Association for the Advance of AI. It is also recognized as a Class A conference of China Computer Federation (CCF). The top AI conferences also classified as CCF-A include CVPR, IJCAI, ICCV, ACL, ICML, and NeurIPS.

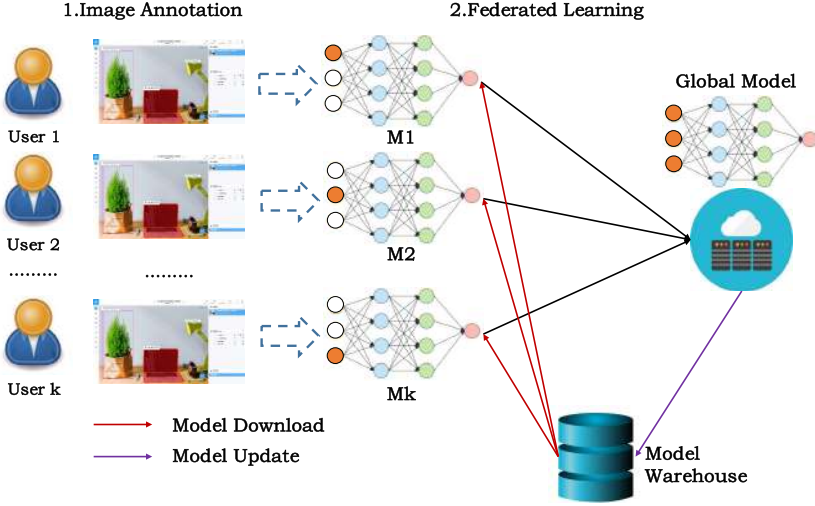


Figure 2.4. The new model-centralized training approach.

The traditional training method requires multiple users to annotate data first² (commonly known as crowdsourced image annotation), then upload the annotated images to a cloud database for storage, and then based on the acquired data, train and optimize the model (the model used here is the classic object detection model YOLO V3). Finally, the trained model can be used for inference evaluation and practical application. As mentioned earlier, traditional training methods cannot adequately accommodate the data protection needs and model usage requirements of small companies, making it difficult to promote in this era of data privacy protection. This hinders many small companies from using advanced AI technologies, and their productivity is not fully unleashed.

²The nature of the annotation depends on the domain of the task. If it is the field of object detection, annotations are generally the bounding boxes of the objects of interest in the image, which are represented by the coordinates (x, y, w, h) of the bounding boxes. Here, x and y are the coordinates of the center of the bounding box and w and h are the width and height of the bounding box. If it is the field of object classification, annotations are generally the categories of the objects in the image.

Now, based on the new FL training method, as shown in Figure 2.4, the annotated data of each user no longer needs to be uploaded to a public server, which greatly facilitates the training process of the model. We now only need to send the model framework from the FL server to each participant and then directly use the data stored locally to train this model. After training convergence, the encrypted model parameters from each party will be sent back to the server, and they will then be aggregated into a global federated model. This global model will finally be distributed to the participants in the federation for subsequent inference evaluation. The specific task in this chapter is flame detection. We can also diverge our thinking and extend to more applications, such as the detection of defects in object products, behavior anomaly detection, security detection, and so on.

Person reidentification is also a hot topic in the CV field. It is a technology that uses CV to determine whether a specific person exists in an image or video sequence. Given an image of a person to be monitored, images of the same person under different devices are retrieved. It aims to compensate for the visual limitations of a single fixed camera, can be combined with pedestrian detection/tracking technology, and can be widely used in fields, such as intelligent video surveillance and smart security. However, the training of current person reidentification models relies on a large amount of centralized personal image data, which brings potential privacy risks to personal information and even leads to the suspension of some countries' person reidentification research projects. Therefore, it is necessary to guide its development under the premise of privacy protection. Recently, Nanyang Technological University and SenseTime jointly proposed a new FL framework for this task, FedReID, and then based on the characteristic that the model structure is not the same in all clients, that is, it may have different identity classifiers, they proposed a performance improvement optimization method Federated Partial Averaging (FedPav), which supports joint training with clients with partially different models. In the entire model training process, it is similar to Federated Averaging (FedAvg), the difference being that each client only sends a part of the updated model to the server.

With the increasing emphasis of various countries on privacy protection, the application of data privacy-sensitive technologies such

as pedestrian reidentification may become increasingly challenging. This could also provide some new opportunities for federated learning.

2.4 Federated Learning and Natural Language Processing

Natural Language Processing (NLP) is a branch of AI and linguistics. This field investigates how to process and utilize natural language. NLP includes many aspects and steps, primarily involving cognition, understanding, and generation.

Natural language cognition and understanding allow computers to transform the input language into meaningful symbols and relationships, which are then processed according to the goal. Natural language generation systems convert computer data into natural language.

With the continuous development of CNN technology in the field of CV, NLP has also begun to adopt similar technologies to process text information. However, convolutional neural networks focus only on local information and struggle with capturing and storing long-distance dependency information within the text. On one hand, researchers want to enlarge the convolution kernel and increase the number of channels to capture long-distance dependency information, while on the other, they fear the curse of dimensionality that comes with increasing model parameters. On another note, Recurrent Neural Networks (RNN), their variants Long Short Term Memory (LSTM), and Gate Recurrent Unit (GRU) were proposed to tackle these issues. Their structure allows information from previous text to be written into the memory module due to the presence of various gate structures. The input gate allows the neural network to selectively record some long-term effective information, the forget gate discards some redundant information specifically, and the update gate allows the network to update its current state based on input in real time. However, this sequential network structure and its dispersed storage of effective information still struggle to store ultra-long text dependency information effectively. Moreover, this structure is not easily parallelized for acceleration.

In 2017, the transformer, which introduced a new self-attention mechanism (self-attention scheme), and its variants, such as BERT and GPT-3, began to dominate the state-of-the-art (SOTA) leaderboard in various NLP tasks. Self-attention technology can update the mapping results of each word in the input text in parallel dynamically, thereby capturing long-distance word sequencing information. Superior model performance requires a massive amount of model parameters, with the latest models reaching tens of billions. As the parameters increase, the model performance seems to continue to rise. There have also been some attempts of transformer models in the CV field, such as the Vision Transformer (ViT). However, these NLP methods require storing users' private data collectively to train models. In reality, users' natural language data are also highly sensitive, therefore, it is necessary to incorporate FL technology.

In the current era of intelligence, smartphones are practically ubiquitous, and the keyboard input method has become a necessary tool for communication. As a standout in the input methods, Gboard supports more than 600 languages and has been installed over a billion times. Gboard provides a variety of input functions, including tap and word gesture input, auto-correction, word completion, and next word prediction. Learning frequently typed words from user-generated data is a crucial part of developing mobile device keyboards. Common uses include incorporating new popular vocabulary (celebrity names, pop culture terms, etc.) or simply compensating for omissions in the initial keyboard implementation, especially for lesser-resourced languages. The word list, often referred to as the vocabulary, might be manually input or not. Words missing from the vocabulary cannot be predicted on the keyboard suggestion bar, nor can they be input via gestures; moreover, even if input correctly, they may be auto-corrected. Besides, mobile keyboard models are limited in many ways. To run on both low-end and high-end devices, the model should be small, and inference time latency should be low. Users typically expect a visible keyboard response within 20 milliseconds of an input event. Considering the frequency of use of mobile keyboard applications, if CPU consumption is unrestricted, the client device's battery might be drained quickly. Therefore, the size of the language model is usually limited to tens of megabytes and the vocabulary to hundreds of thousands of words. This means that the vocabulary supporting the model is essentially limited. Hence, discovering

and including the most useful words in this relatively short vocabulary is crucial. Words not in the vocabulary are often referred to as out-of-vocabulary (OOV) words. Please note that the concept of a vocabulary is not limited to mobile device keyboards. Other natural language applications, such as Neural Machine Translation (NMT), rely on vocabularies to encode words during end-to-end training. Thus, learning OOV words and their ranking is quite a common technical requirement. Given the risk of privacy leakage when transmitting and storing user content on centralized servers, but the poor results from learning language models only from individual user devices, Google proposed using FL technology FederatedAveraging to train models on user devices. When model interaction updates are needed, only the instantaneous model updates are uploaded to the server for model aggregation, while users' original data remain on their own devices, as illustrated in Figure 2.4.

In addition to this, FL technology has been proposed to handle and solve the next word prediction task in input methods. Next word prediction provides a tool for users to input text conveniently. Based on a small amount of user-generated previous text, the language model can predict the next most likely word or phrase. The performance of the neural language model trained using FL surpasses that of the traditional models trained based on server-collected and trained data. Similar tasks include the emoji suggestion function. Due to the inherent similarities of these tasks, similar network designs are used, namely a variant of LSTM called Coupled Input and Forget Gate (CIFG). Like the GRU, CIFG uses a single gate to control the input and self-connections of the recurrent unit. Compared to LSTM, this coupling reduces the number of parameters per unit by 25%. The CIFG architecture has an advantage in the mobile device environment because it reduces the number of computations and the size of the parameter set without affecting the performance of the model.

Certain companies have extended the technology of FL to search query suggestions in input methods. For example, when a user types text, Gboard employs a baseline model (the most fundamental model, which combines knowledge graph and LSTM model, and is trained in a traditional server-based manner) to determine and possibly display search suggestions related to the input, as shown in Figure 2.5. If the user types "Let's eat at Charlie's", for instance, a search query

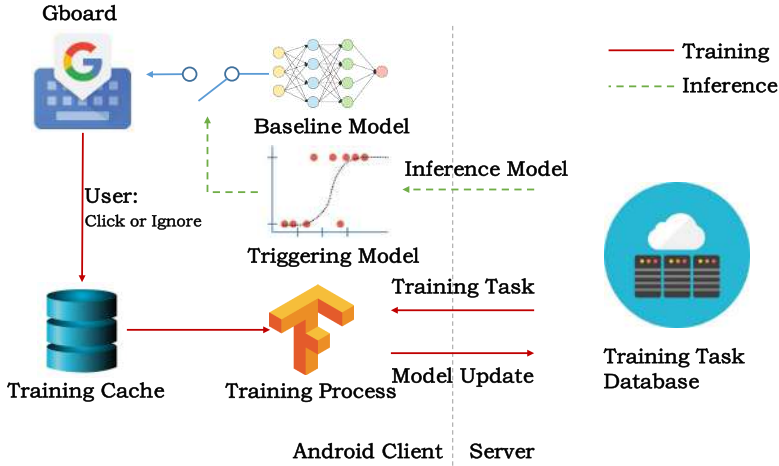


Figure 2.5. Training process for search query suggestions in Google's Gboard. TensorFlow's mobile framework, TensorFlow Lite, is utilized for model training.

suggestion might appear for a webpage to search for restaurants of the same name nearby. Other types of suggestions may include GIFs and emoji images. They improve this function by filtering search suggestions from the baseline model using a triggering model trained with FL. The triggering model, used in experiments, is a trained logistic regression model that predicts the probability of clicks, with its output being a score given to a query. The higher the score, the more trust is put in the suggestion. By integrating search suggestions with FL and on-device inference, they have improved the quality of Gboard's suggestions while protecting user privacy and adhering to mobile device constraints. Similarly, a user interaction model has been trained with FL to improve the ranking performance of input suggestions in the browser's URL bar while protecting privacy. This model successfully replaced the manually made heuristic model, with results showing that users can now find what they want by typing half a character less.

In the smart era, people most frequently interact with the voice assistant feature on smartphones. Wake word detection is used to initiate interactions with the voice assistant. Keyword spotting represents a specific case of keyword recognition, where a predefined keyword or a set of keywords is detected by continuously monitoring an audio stream. Typical examples of wake words include applications

in various scenarios, such as “Hey Siri”, “OK, Google”, and “Xiao Ai Tong Xue”. Once a “wake word” is detected, the voice input is activated and processed by the speech understanding engine, enhancing the perception ability of the voice assistant. Wake word detectors usually run on the device in an always-on mode, presenting two main difficulties. First, it should operate with minimal memory usage and computational cost. The resource constraints for wake word detectors are 200 k parameters and 20 Million Floating-Point Operations Per Second (MFLOPS). Second, wake word detectors should perform consistently in various use cases and be robust against background noise. Given that the model can be triggered at any time, high accuracy is essential: It is expected to capture the majority of commands (high recall) while avoiding unintentional triggers (low false alarm rate). Today, wake word detectors are typically trained on datasets collected in real usage environments, like user homes. As voice data are inherently very sensitive, centralized collection has raised significant privacy concerns. Recently, Snips explored the use of FL in the context of embedded wake word detection. They were the first to propose using an adaptive averaging strategy, inspired by the commonly used deep neural network optimizer Adam, to replace the standard federated weighted average model, significantly reducing the number of communication rounds needed to reach target performance. This algorithm has an upstream communication cost of only 8 MB per user, which is acceptable in the context of smart home voice assistants.

Beyond applications in virtual digital keyboards and voice recognition, Dianbo Liu, a researcher from Boston Children’s Hospital, proposed leveraging a large amount of clinical records to train an NLP model for a medically useful and generalizable context representation model. Furthermore, because clinical doctors vary greatly in their style of writing records across hospitals and healthcare systems, it is crucial to gather data from many sources. However, due to privacy and regulatory issues, sharing clinical data is difficult. Training NLP models with FL is an excellent choice to overcome these challenges. In their paper, they trained the BERT model with FL on clinical records from multiple sites without leaking the content of the records. They also suggested that the BERT model can be fine-tuned with FL for different downstream tasks, such as Named Entity Recognition (NER).

From digital virtual keyboards to voice recognition and clinical records, the application of FL in the NLP field is becoming increasingly diverse and comprehensive. We look forward to forming a larger scale NLP pre-training model in the future, encompassing almost all NLP tasks.

2.5 Federated Learning, Cloud Computing, and Edge Computing

Cloud computing is a mature model that shares computing resources by providing users with network access. In the cloud computing model, users communicate with data centers to obtain resources such as hardware and software and to store data. However, in recent years, the number of client devices, such as smart devices and monitors, has been rapidly increasing. According to reports from Internet of Things (IoT) analysis companies, the number of connected devices reached 17 billion just in 2018. These client devices generate a vast amount of data that needs to be processed. The overhead of data transmission and encryption between devices and data centers has become a significant bottleneck in many IoT scenarios, hence posing severe challenges in terms of throughput, latency, and security. Edge computing has emerged as a promising technological framework to overcome the challenges posed by cloud computing. The edge computing framework adds a new layer to the traditional cloud computing framework, known as edge nodes. In this framework, only real-time data processing is transferred to the edge nodes, while other complex data processing tasks are still executed on cloud servers. Edge computing decomposes large tasks that were entirely handled by central cloud nodes into smaller and more manageable parts, processed at the edge nodes. Edge nodes, being closer to the user terminal devices, can accelerate data processing and transmission speeds, reducing latency. Under this architecture, data processing and knowledge generation are closer to the data source, i.e., the client, making it more suitable for handling big data.

Figure 2.6 shows a generic edge computing framework, including three layers: cloud servers, edge nodes, and client devices. In edge computing scenarios, the distribution of data and workload

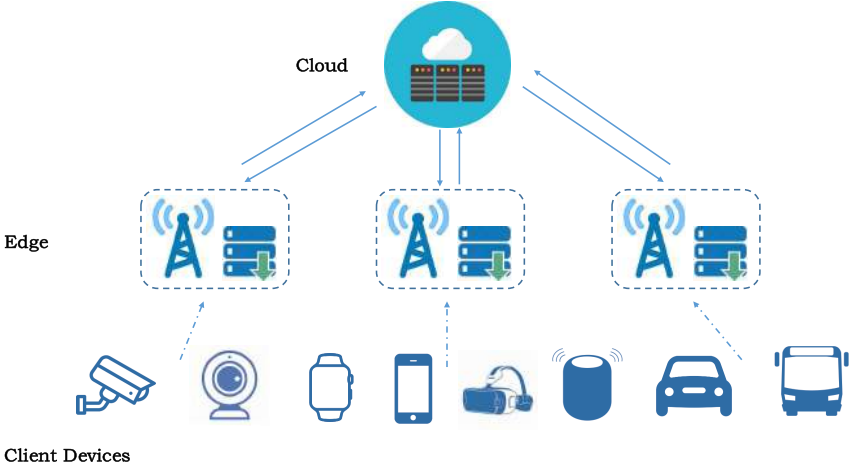


Figure 2.6. A generic edge computing framework. The edge here can typically be mobile signal base stations, routers, and gateways for smart homes.

across different layers and their coordination are vital concerns for performance, security, and privacy issues. Hence, for benchmarking, designing, and implementing edge computing systems or applications, we adopt an end-to-end viewpoint, considering all three layers. In these scenarios, AI technologies are widely used to enhance device, edge, and cloud intelligence, posing the highest demands in terms of computational power, data storage, and networking. Typical application scenarios include smart cities, smart homes, autonomous driving cars, surveillance cameras, intelligent medical care, and wearable devices. These scenarios are complex due to different types of client devices, a large amount of heterogeneous data, privacy, and security issues, and they mostly have high requirements for latency and network bandwidth. Hence, researchers apply FL to the edge computing framework, collaboratively training high-quality shared models by aggregating and averaging local computation updates uploaded by IoT devices. The primary advantage of this approach is that it separates model training from the need for direct access to training data; thus, FL can technically achieve privacy-protected global models.

In IoT applications, FL currently faces three major challenges: (1) device heterogeneity, such as different storage, computation,

and communication capabilities; (2) statistical heterogeneity, such as Non Independent Identically Distributed (Non-IID) characteristics of data from different devices; (3) model heterogeneity, where different devices want to customize their models to adapt to their application environments. Specifically, resource-constrained IoT devices will only allow lightweight models to be trained under certain network conditions, possibly leading to high communication costs, disconnections, and fault-tolerance issues, which traditional FL cannot handle effectively. As FL aims to achieve a high-quality global model by extracting common knowledge from all participating devices, it fails to capture individual information from each device, leading to a decline in inference or classification performance.

To address these issues, researchers at Sun Yat-sen University proposed personalization at the device, data, and model levels to alleviate heterogeneity and obtain high-quality personalized models for each device. This personalization would be useful in scenarios, such as personalized IoT-based intelligent medical care, smart home services and applications, fine-grained location-aware recommendation services, and on-site intelligent video analysis. They proposed a collaborative cloud-edge framework named PerFit for personalized FL, addressing the three major challenges in IoT applications holistically. To solve the problem of high communication and computational costs in device heterogeneity, each IoT device can choose to offload its computation-intensive learning tasks to edge nodes to meet requirements of rapid processing capability and low latency. Furthermore, edge computing can mitigate privacy issues by storing data locally at edge nodes instead of uploading it to a remote cloud. For statistical and model heterogeneity, the framework also allows terminal devices and edge servers to jointly train a global model under the coordination of a central cloud server in a cloud-edge architecture. After training the global model through FL, different personalized FL methods can be used at the device end for deploying personalized models for different devices customized according to their application needs. Researchers at the Hong Kong University of Science and Technology similarly proposed a client-edge-cloud layered training FL system, supporting a HierFAVG algorithm, allowing multiple edge servers to perform partial model aggregation. In this way, models can be trained more quickly, achieving a better communication-computation trade-off.

Autonomous driving cars are classic scenarios for edge computing, where effectiveness is of high demand. This means that even without human intervention, it must take absolutely correct actions. This characteristic represents the needs of some edge computing AI scenarios. A standard vehicle is limited by time and space of driving, and the sensor information obtained is usually limited. By introducing horizontal FL, we can integrate sensor information from different vehicles, such as cameras, ultrasonic sensors, and radars (e.g., millimeter-wave radar and lidar), to establish scene information more rapidly and help improve the robustness of the model. Utilizing data from different vehicles while protecting data privacy, some regular intelligent function computations such as obstacle avoidance can be accomplished by multiple vehicles, reducing bandwidth for communication with cloud servers, lowering operation latency, and greatly enhancing the safety of intelligent systems.

Specifically, the “FL Whitepaper 2.0”, jointly released by many companies and institutions, provides a reasonable outlook on the application of FL technology in autonomous driving. We can boldly imagine that autonomous cars should not only have the ability to learn or replicate human driving skills but also interact with the Vehicle to Everything (V2X), vehicle-road coordination, and even the entire traffic system to create a safer and more comfortable driving environment. The vehicle’s interaction with the environment and learning from it can be supplemented with other city information, such as city security cameras, traffic lights, various traffic signs, and future intelligent roads. By integrating information from different sources, the future experience of autonomous driving technology can be significantly enhanced.

As commercial companies and the governments continue to deploy and develop in fields such as smart cities, smart healthcare, and smart homes, the combination of edge computing and cloud computing with FL technology is bound to become a novel solution.

2.6 Federated Learning and Computer Hardware

Currently, the industry has two main technical routes for addressing data sharing issues related to privacy leaks and data misuse: trusted computing based on Trusted Execution Environment

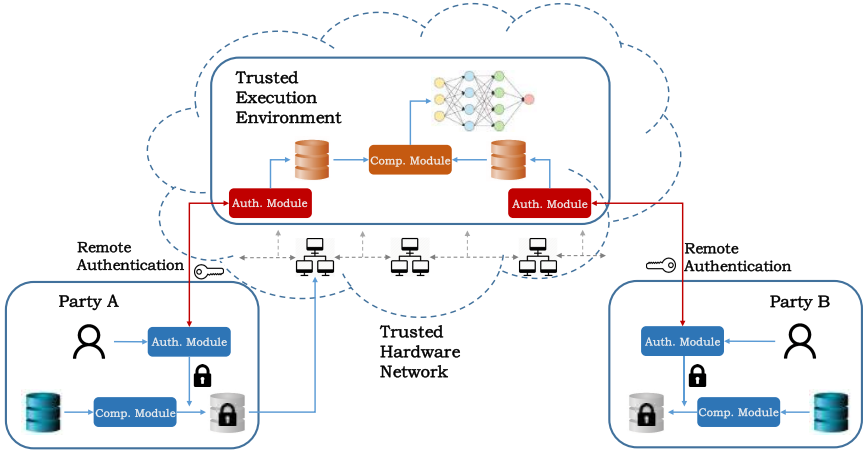


Figure 2.7. Illustration of a general TEE scheme.

(TEE) technology, and cryptography-based homomorphic encryption (HE) and secure multi-party computation (SMC), which provide theoretical/mathematical privacy guarantees.³

TEE provides a secure computing environment that is isolated from the untrusted environment. It is this isolation and trusted verification mechanism that makes trusted computing possible, as shown in Figure 2.7. TEE is usually directly based on hardware implementations, such as Intel Software Guard Extensions (SGX), AMD SEV, ARM TrustZone, and RISC-V Keystone; it can also be built with virtualization technology, such as Microsoft’s VSM and Intel’s Trusty for iKGT and ACRN, but it still cannot match the security of hardware TEE. Among them, Intel’s SGX is the most advanced TEE implementation in commercial CPUs. It provides a new instruction set that allows users to define secure memory areas known as enclaves. The CPU ensures that the enclave is isolated from the outside, thus protecting the confidentiality, integrity, and verifiability of the code and data within it. Apple has designed an independent secure enclave, a security coprocessor separate from the main processor, which includes a hardware-based key manager that provides additional security protection. It also allows Touch ID and Face ID

³This section refers to articles from “Synced” titled “Shared Machine Learning: Ant Financial’s Solution For Data Privacy” and some artworks by Ant Financial.

on Apple devices to provide secure authentication while protecting the privacy and security of user biometric data.⁴ As deep learning implementations at the hardware level (such as tensor processing units or machine-learning-specific instruction sets) become increasingly important, these system-based privacy protections built into edge hardware (like smartphones) are likely to become more common.

Some companies have proposed the concept of shared learning, using Intel's SGX technology at the bottom layer, attempting to create a business model based on a cloud ecosystem. Compared to distributed cryptography-based HE and SMC, shared learning tends more toward centralization. That is, when multiple data providers want to share data, to solve the problem of mutual distrust, they will find a third-party platform with a trusted hardware execution environment (TEE) that they all trust, and all the data will be aggregated on this trusted third-party platform for fusion and computation.

SGX technology involves two core concepts: "Enclave" and "Remote Attestation". The enclave can be thought of as a safe provided by Intel; including the OS, VMM, BIOS, and SMM cannot access the data inside on their own, only the CPU can decrypt it using the hardware on the CPU during computation. The safe solves the problem of data and code within the SGX being attacked externally. Ensuring that the code within the safe is secure and trustworthy involves "Remote Attestation" technology. Remote attestation has a prerequisite assumption that the code inside the safe is open source to the user. Users can confirm the behavior of the code by checking the code, and remote attestation is just to confirm that what is currently running in the safe is indeed the code that the user checked before. Through the cooperation of enclave and remote attestation technologies, it can be ensured that data sent into the SGX via an encrypted channel will definitely be processed as the user expects, ensuring that the information is not leaked.

The enclave brings us unprecedented security guarantees, but it currently faces significant usability issues. These are mainly manifested in several aspects. First, it is necessary to divide the original application into two parts: One part is the untrusted part outside

⁴Refer to "Secure enclave overview. Apple Platform Security", <https://support.apple.com/guide/security>.

the enclave and the other part is the trusted part inside the enclave; second, it is necessary to carefully design the interface between the two parts, plan when to enter the enclave and when to exit the Enclave — this has a certain technical threshold, and it is tedious and easy to make mistakes; third, even if we make a perfect division, the environment inside the enclave is very limited compared to our familiar general Linux operating environment. For example, system calls cannot be made inside the enclave, `libc`, `pthread` are incomplete, there is no `openmp`, and multi-process support is lacking. Another big issue is that in order to cope with the massive amounts of data from large companies and institutions, confidential computing must be expanded from a single node to a cluster.

Addressing the key issues that confidential computing faces, as mentioned above, Ant Financial has innovatively proposed the SOFAEnclave confidential computing middleware. The core of SOFAEnclave consists of three parts: the enclave kernel Occlum, the cloud-native confidential computing cluster KubeTEE, and the security testing and analysis framework. Similar to an operating system kernel, the Occlum LibOS provides complete system services to trusted applications inside the enclave, without requiring them to be divided or modified to receive enclave protection. Currently, Occlum can easily support large AI frameworks, such as XGBoost, and TensorFlow, and also large server applications like Shell, GCC, Web Server, and so on. This greatly reduces the development cost for developing applications for the enclave. KubeTEE combines cloud-native technology to provide confidential computing cluster services more efficiently and concisely. For detailed information, please refer to related articles by Ant Financial.

At present, there have been explorations in the industry about FL based on computer hardware. It is believed that in the near future, TEE technology based on computer hardware will evolve toward miniaturization, lightness, and efficiency, greatly promoting the further practical application of FL.

2.7 Conclusion

In this chapter, we have introduced the applications of FL in finance, biomedical, CV, NLP, edge computing, cloud computing,

and computer hardware and also presented some prospects. As federated technology continues to evolve, it is anticipated that more and more industries will be able to employ FL technology, which enables them to protect data privacy from a technical perspective while performing large-scale model training and learning, thereby obtaining more and better intelligent models.

We hope that the introduction in this chapter provides readers with a comprehensive understanding of the current industry applications of FL. We also hope this chapter inspires our wide audience and further promotes the practical applications of FL technology.

This page intentionally left blank

Chapter 3

Common Privacy Protection Technologies

3.1 Privacy Preserving Machine Learning

Common privacy protection technologies such as various encryption algorithms, differential privacy (DP), obfuscation circuits, multi-party secure computing, and others are developing rapidly. As people place increasing importance on privacy protection, machine learning (ML) combined with these privacy protection technologies, also known as secure ML, is becoming a hot research direction. To give everyone a simple understanding of secure ML algorithms, this section outlines common privacy protection technologies and their development from a technical perspective.

3.1.1 Overview of Privacy Preserving Machine Learning

ML is increasingly used in a variety of applications, from detection (such as object detection), recognition (such as face recognition), and prediction (such as age prediction) to recommendation systems. Some ML applications require personal privacy data for model training. These privacy data are uploaded to a centralized location in plaintext for ML algorithms to extract patterns and learn from the model. The problems that ML faces include not only exposing all these private data to internal threats from these companies but also external threats such as the company holding these datasets being

attacked by hackers. Moreover, even if the data are anonymized, it is still possible to collect additional information about the private data, or if the data and the model itself are inaccessible, external users only know the test results of the model.

In ML tasks, there may be three different roles: input parties (data owners or contributors), computing parties, and result parties. In such a system, data owners send their data to the computing party, which performs the required ML tasks (such as learning models) and sends the output to the result party. This output could be a ML model, which the result party can use to test new samples. In other cases, the computing party might retain the ML model and perform testing tasks on new samples provided by the result party, then return the test results to the result party. If all three roles are undertaken by the same entity, then privacy is naturally protected at the technical level; however, when these roles are distributed across two or more entities, privacy protection technologies are needed. In most cases, the computing party and the result party are the same entity, which is usually separate from the data owner. In fact, all the data collected daily from around the world, data owners may not know how the data they have been collected are being used (or misused). Even in many cases, they do not know that some data are being collected. According to the process of data contribution and utilization, there exist several levels of attack threats, as shown in Figure 3.1.

If the data owner is separate from the computing party, then private data will likely be transmitted to the computing party through

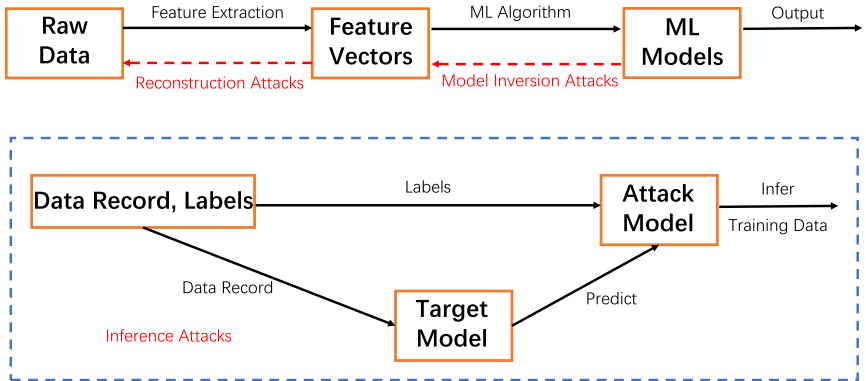


Figure 3.1. Machine learning and the threats it faces.

a secure channel. However, the data are likely to exist in its original form on the computing server, that is, not encrypted or transformed in any way. This is the biggest threat, as private data are vulnerable to both internal and external attacks. These privacy data can be stored as original data or as features extracted from the original data. Naturally, storing data in its original form poses a greater threat, as they can be processed in various ways.

Reconstruction Attacks: Even if only the features (extracted from the original data) are transmitted to the computation server for storage, there remains a threat of reconstruction attacks. In such a case, the attacker's objective is to reconstruct the original private data by utilizing their knowledge of the feature vectors. Reconstruction attacks require white-box access to the ML model, i.e., the feature vectors in the model must be known. This kind of attack could occur if the feature vectors used for model training are not deleted after the desired ML model has been established. In fact, some ML algorithms like support vector machines (SVM) or k -nearest neighbors store the feature vectors within the model itself. Successful instances of reconstruction attacks include the reassembly of fingerprint images, where the fingerprint image (original data) can be reconstructed from the minutiae template (features), and the rebuilding of mobile device touch gestures, where touch event data (original data) can be reconstructed from gesture features (such as speed and direction). In both these cases, the privacy threat (caused by not protecting the private data in its feature form) compromises the security of the authentication system, leading to an inability to safeguard the data owner's privacy (since attackers may gain access to the data owner's device). Although these examples aim to mislead ML systems into believing the reconstructed original data belong to a certain data owner, other reconstruction attacks may directly expose private data, such as location or age.

To resist reconstruction attacks, ML models that directly store explicit feature vectors (like support vectors) should be avoided, or kept confidential and not provided to the result recipient.

Model Inversion Attacks: Some ML algorithms do not explicitly store feature vectors in the models they generate, such as ridge regression or neural networks. Therefore, the attacker's knowledge is limited to (a) a ML model without stored feature vectors

(white-box access) or (b) the response returned by the computation party when the result recipient submits new test samples (black-box access). Here, the attacker's goal is to create feature vectors similar to those used to create the ML model by leveraging the responses received from the ML model. This attack utilizes credibility information (e.g., probabilities or support vector machine decision values), which is sent back as the response to the test samples submitted by the result recipient. These attacks yield an average representative of a specific class. Hence, when a class represents an individual, for instance, in facial recognition, the threat to privacy is most severe. To resist such attacks, the result recipient should be limited to black-box access, and output should be restricted. This minimizes the knowledge a black-box attacker can obtain. For example, the attack success rate decreases when a classification algorithm reports rounded-off confidence values or only predicted class labels. Furthermore, results from multiple sample tests can be aggregated, although this approach is not suitable for all application scenarios.

Membership Inference Attacks: Model inversion attacks do not produce a real sample from the training set nor infer whether a sample is in the training set based on the output of the ML model. Membership inference attacks, however, do infer whether a sample is in the training set. Given a ML model and a sample (knowledge known to the attacker), the goal of a membership inference attack is to determine if the sample is a member of the training set used to build this ML model (the attacker's objective). Attackers can use this attack to determine if the records of a specific individual were used to train a ML model associated with a specific disease. This attack leverages the differences in predictions made by the ML model for samples used in the training set versus those not included. Shokri *et al.* (2017) researched these attacks, using the correct label of samples and the prediction of the target ML model as input to train an attack model that determines if a sample is in the training set. These attack models are trained using shadow models established from data generated in three ways: model inversion attacks, statistically based synthesis, or noisy real data. When training attack models using black-box or white-box access, executing attacks using these models requires only a black-box attacker. The work also tried different attack defense techniques, such as using regularization and coarsening the

prediction vector, and found limiting the output to class labels is the most effective (though not sufficient to completely stop the attack).

Common privacy protection techniques focus on allowing multiple input parties to collaboratively train a ML model rather than releasing the original private data. These techniques are primarily realized through the use of encryption methods or DP. DP techniques are especially effective in preventing membership inference attacks. When a ML application requires data from multiple input parties, encrypted protocols can be utilized to conduct ML training or testing on encrypted data. Among these techniques, to achieve greater efficiency, it is necessary for data owners to contribute their encrypted data to computation servers, reducing the problem to secure two-party or three-party computation scenarios.

The majority of these techniques address the case of feature-distributed data: Each data owner collects the same set of features for different data objects. Facial recognition is an example, as anyone wishing to train a ML model for their own face could submit multiple feature vectors extracted from their own photos. In each case, the same set of features would be extracted by each data owner. Commonly employed techniques for implementing privacy-preserving machine learning (PPML) include homomorphic encryption (HE), garbled circuits, secret sharing, and multi-party computation (MPC), among others. The following offers a brief introduction to some secure technologies. For more specific content and work, please see the following sections.

Homomorphic Encryption: Full HE allows computation on encrypted data. Through operations such as addition and multiplication, it enables the calculation of more complex arbitrary functions. Due to the high cost of frequent ciphertext refreshing (required due to noise accumulation), additive homomorphic schemes are primarily adopted in PPML. These schemes support only addition operations on encrypted data and multiplication operations on plaintext. Paillier encryption is a common example. To extend the functionality of additive HE, efforts have been made to implement the comparison of two encrypted values or perform secure multiplication and decryption operations, primarily through adding an encrypted random value to the ciphertext that needs protection.

Garbled Circuits: Suppose Alice and Bob wish to obtain the result of a function computed on their private data. Alice can convert this function into a garbled circuit and send out this circuit along with her garbled input. Bob then receives a garbled version of his input from Alice, while Alice remains unaware of Bob’s private data, as with oblivious transfer (OT). Bob can now use his garbled input and the garbled circuit to get the desired function result (and choose to share it with Alice if he wishes). Some PPML methods combine additive HE with garbled circuits.

Secret Sharing: This is a method for distributing a secret among multiple parties, each of which holds a “share” of the secret. A single “share” is useless on its own; however, when all “shares” are combined, the secret can be reconstructed. In one setup, multiple input parties can generate “shares” of their private data and send these shares to a set of non-colluding computation servers. Each server can compute a “partial result” from the received “shares”. Finally, the result party (or an intermediary) can receive these partial results and combine them to find the final result. Since these computation servers play similar roles, special attention should be given to where these servers are hosted and which entities control them in order to instill confidence in the data owners.

Differential Privacy: DP technologies resist membership inference attacks by adding random noise to the input data, during an algorithm’s iteration, or in the algorithm’s output. Although most DP methods assume a trusted data aggregator, local differential privacy (LDP) allows each input party to add noise locally; hence, no trusted server is required. Finally, the data can also be perturbed by projecting it onto a low-dimensional hyperplane to prevent the reconstruction of the original data and/or limit inference of sensitive information. Please refer to the subsequent sections for specifics.

3.1.2 Developments in Privacy-Preserving Machine Learning

Despite the use of the aforementioned technologies to protect privacy during ML training and testing, non-privacy-preserving ML algorithms are still widely used, and private data continue to be

uploaded to the cloud or other servers daily. Current laws may compel companies to declare that they are collecting data and may even allow users to opt out of this collection, but this doesn't seem to be an optimal strategy. A better alternative would be to utilize privacy-preserving machine learning technologies to implement secure ML. Numerous secure ML algorithms have already been proposed, and in the following, we introduce a few ML algorithms based on commonly used security techniques.

To improve the efficiency of using additive HE, data packing techniques have been developed to allow multiple plaintext values to be encrypted by the same ciphertext. Some PPML methods have utilized these techniques to achieve efficient and secure PPML systems, such as the collaborative filtering system proposed by Erkin *et al.* (2012). In this system, data owners provide data encrypted with the Privacy Service Provider's (PSP) public key but send the encrypted data to the Service Provider (SP). The PSP provides privacy and computation services, and the SP provides storage and computation services, with the goal of providing private recommendations to their clients (data owners). For the system's security, the SP and PSP cannot collude, as they provide different services. The SP and PSP can be separate companies, making a non-collusion assumption reasonable. In this system, data owners serve as the input and result parties, while the SP and PSP serve as the computation parties.

As previously mentioned, some PPML methods combine additive HE with garbled circuits. Nikolaenko *et al.* (2013) developed a ridge regression system utilizing these two techniques. In this system, encrypted shares from multiple data owners can be added to obtain encrypted intermediate values. These shares are additively homomorphically encrypted using the public key of the Cryptographic Service Provider (CSP, similar to the aforementioned PSP). Then, the CSP creates a garbled circuit and sends it to the evaluator, who also receives garbled versions of the intermediate shares from the CSP. The evaluator can then continue to use the garbled circuit and their garbled inputs to construct the required ML model. Some PPML methods focus only on classification tasks (testing phase instead of both training and testing phases). Bost *et al.* (2015) used HE and garbled circuits to develop cryptographic building blocks that enabled them to construct three popular classification models: hyperplane decisions,

naive Bayes, and decision trees. The objective is to allow the testing of new samples while protecting the ML models and the submitted samples.

ShareMind is another example of secure ML, where Cybernetica combined parallel Principal Component Analysis (PCA) computation methods with a secret sharing scheme, creating a privacy-preserving system for executing PCA calculations. In another setup, the “shares” (from secret sharing) are distributed to other input parties (users) rather than computation parties (servers). Bonawitz *et al.* (2017) developed a protocol for securely aggregating vector sums of model updates provided by users. Each user uses dual masking for their private update vector: a user-specific secret value and a secret value shared with other users. Ohrimenko *et al.* (2016) developed data-oblivious ML algorithms for models, such as neural networks, SVM, k -means clustering, and decision trees.

Currently, secure ML is a popular research direction with much to explore. We hope that everyone will continue to research in directions such as privacy protection, computational efficiency, and model performance, in order to design better secure ML algorithms.

3.2 Commonly Used Privacy Protection Techniques

As discussed in the previous section, many existing encryption technologies and secure multi-party computation (SMPC) techniques can be combined with ML to design secure ML algorithms. This section introduces several common privacy protection technologies used in secure ML.

3.2.1 Differential Privacy

With the rapid development of the internet industry, an increasing amount of user data has been generated. Based on big data and ML technologies, these vast amounts of user data can be analyzed to provide users with higher-quality services. However, extensive statistical analysis of this data can severely threaten users’ privacy. DP offers a technical solution to the potential conflicts of interest between data privacy protection and data analysis. This section introduces the foundational knowledge of DP, such as its definition and categories, and its application in federated learning (FL), aiming

to increase researchers' and practitioners' understanding of FL based on DP.

3.2.1.1 Introduction

The rapid development of wireless communication technology has led to a proliferation of internet access devices, such as smart devices and IoT devices, producing a large amount of data through crowd-sourced sensing. The DP model is a widely recognized stringent privacy protection model. It protects potential user privacy information in the published data by adding disturbing noise, ensuring that attackers cannot infer specific information even if they have access to all other information. This is a method that completely cuts off the possibility of privacy information leakage from the data source. Indeed, big data has brought us abundant knowledge and significant benefits, such as traffic flow control, epidemic prediction, and recommendation systems, greatly facilitating our daily life. Almost all data statistics and analysis tasks fundamentally rely on understanding data distribution. However, the collection and analysis of data, containing a variety of users' sensitive information, pose serious privacy concerns. Worse still, in the era of big data, users' private data are more susceptible to attacks and leaks under the drive of advanced data fusion and analysis technologies, as described in Soria's paper. For instance, adversaries can infer family members' daily habits or behavioral characteristics (e.g., time at home or not at home, watching TV, cooking, and other specific activities), even identity information, social relations, religious attitudes, etc., by analyzing household appliance usage. For example, consider a marriage database where three are single and seven are married, only able to query the number of singles. Initially, it was found that three people were single. After Jane registered his marital status, another check found still three people were single. Therefore, we can tell that Jack is married. Through such simple ways, privacy data can be obtained.

Therefore, preventing personal data leaks while collecting data from various devices is a matter of urgency. DP is a mathematically grounded method to solve this problem technically. It formalizes the concept of privacy, allowing formal proofs that certain data collection methods can protect user privacy. Furthermore, it allows us to quantify privacy leakage by a specific value (making it possible to

compare different techniques) and define the range of permissible privacy leakage. It is also immune to post-processing, meaning no data analyst can exploit the output of a DP algorithm to compromise its corresponding privacy protection capability. If applied correctly, DP allows the revelation of the properties of the entire user group while protecting individual privacy.

The concept of DP originates from the concept of semantic security in cryptography, i.e., an attacker cannot distinguish the encryption results of different plaintexts. In DP, it is required that an attacker cannot guess which result corresponds to which dataset based on the published result. This model ensures that the public output will not show significant changes whether an individual is in the dataset or not by adding random noise and provides a quantitative model for the degree of privacy leakage. The changes of an individual do not have a significant impact on the data query result, and the attacker cannot infer the individual sample's privacy information with a significant advantage through the public result. Thus, the DP model does not depend on how much background knowledge the attacker has and provides a higher level of semantic security for privacy information, so it is widely used as a new privacy protection model.

DP was first mentioned by Dwork in a patent in 2005 and formally proposed in a paper by Dwork in 2006. Since then, a lot of work has been done to describe the nature and mechanism of DP and review its results, as in papers by Dwork in 2007, 2008, 2010, and 2011.

Existing DP is classified in different ways according to different criteria.

The initial work mainly focused on how much noise needs to be added to the data to achieve privacy, introducing the Laplace mechanism. Based on these works, Hardt and Talwar proved the lower bound of the noise level required to achieve DP in 2010. Regarding the issue of adding noise, Lee and Clifton studied in a 2011 paper how to select the appropriate parameter ϵ (this parameter defines the level of privacy protection; the smaller the parameter, the higher the degree of privacy protection).

3.2.1.2 Differential privacy

In this section, we formally introduce the concept of DP. We first present some commonly used terms, then provide a mathematical

definition of DP through a specific example, and finally introduce the classic concepts of ϵ -DP and (ϵ, δ) -DP.

Data Curator: The data curator is responsible for managing the collected data throughout its lifecycle. This can include data cleaning, annotation, publication, and representation. The goal is to ensure that the data can be reliably reused and preserved. In the context of DP, the data curator is responsible for ensuring that the privacy of individuals represented in the data is not violated.

Adversary: An adversary is a data analyst interested in discovering (sensitive) information about individuals in the dataset. In the context of DP, even legitimate users of the database can be referred to as adversaries, as their analysis may infringe upon individual privacy.

Database: The database D can be viewed as a multiset (multiset is a concept in mathematics, which is a generalization of the concept of set. In a set, the same element can only appear once, so only present or absent attributes can be displayed. In a multiset, the same element can appear multiple times), where its entities (elements) come from a finite set $\mathcal{X}$.

ℓ_1 Norm of the Database: The ℓ_1 norm of the database D , denoted as $\|D\|_1$, measures the size of the database, i.e., the number of records d it contains. Its mathematical definition is as follows:

$$\|D\|_1^{|\mathcal{X}|} |d_i| \quad (3.1)$$

ℓ_1 Distance between Databases: The ℓ_1 distance between two databases D_1 and D_2 is $\|D_1 - D_2\|_1$. It measures how many data records are different between the two databases. Its mathematical definition is as follows:

$$\|D\|_{12} \quad (3.2)$$

Neighboring Databases: Two databases are considered “neighboring databases” if they differ by at most one element, i.e., $\|D_1 - D_2\|_1 \leq 1$.

Mechanism: DP itself is an abstract concept and can only be mathematically guaranteed through specific mechanisms. These mechanisms often represent algorithms for publishing statistical information from the dataset.

Let’s illustrate the concept of DP with a simple example. A data curator executes a statistical operation denoted by $\mathcal{K} : \mathbb{N}^{|\mathcal{X}|} \rightarrow \mathbb{R}^k$ for

any k . This statistic could be, for example, the percentage of smokers in the database. An adversary proposes two vector databases D_1 and D_2 , which contain information about the smoking status of the participants, and a set S that contains all possible return values of $\mathcal{K}$. In this example, the datasets D_1 and D_2 represent survey results on whether participants smoke, where “1” denotes smoking and “0” denotes non-smoking. Suppose both datasets contain 100 entries and have the following composition:

$$D_1 = \{0 : 100, 1 : 0\} (100 \text{ “0”s}) \quad (3.3)$$

and

$$D_2 = \{0 : 99, 1 : 1\} (99 \text{ “0”s and one “1”}) \quad (3.4)$$

The adversary could also select a range $S = [T, 1]$ such that $\mathcal{K}(D_i) \geq T$ if and only if $i = 2$. This means the operation $\mathcal{K}$ was performed on database D_2 . The adversary’s goal is to use the operation $\mathcal{K}$ to distinguish between databases D_1 and D_2 to reveal certain privacy information. The data curator, on the other hand, has a completely opposite goal. First, the curator needs to choose a statistical operation $\mathcal{K}$ such that $\mathcal{K}(D_1)$ and $\mathcal{K}(D_2)$ are so close that no feasible T can completely distinguish them, thus ensuring DP. Second, this operation must also be a good estimate of the statistic required for data analysis. If $\mathcal{K}$ is a deterministic statistic, such as the mean of the dataset, then we have $\mathcal{K}(D_1) = 0$ and $\mathcal{K}(D_2) = 0.01$. In this case, if the adversary chooses $T = 0.005$, then he can effectively distinguish between the two datasets. As stated above, it is not possible to achieve privacy protection under deterministic settings (i.e., the statistic is deterministic). Therefore, to achieve DP, a controlled amount of random noise must be added to each query result. This noise can hide the differences between datasets. One possible noise distribution is the Laplace distribution. A simple graphical example is shown in Figure 3.2.

The red area in Figure 3.2 represents the probability that the return value of $\mathcal{K}(D_1)$ is greater than the threshold T . This implies that the adversary will mistake D_1 for D_2 . As shown, the probability of $\mathcal{K}(D_2) \leq T$ is still much greater than that of $\mathcal{K}(D_1) \leq T$. If the noise amplitude is greater, it would lead to the two shaded areas being of similar size.

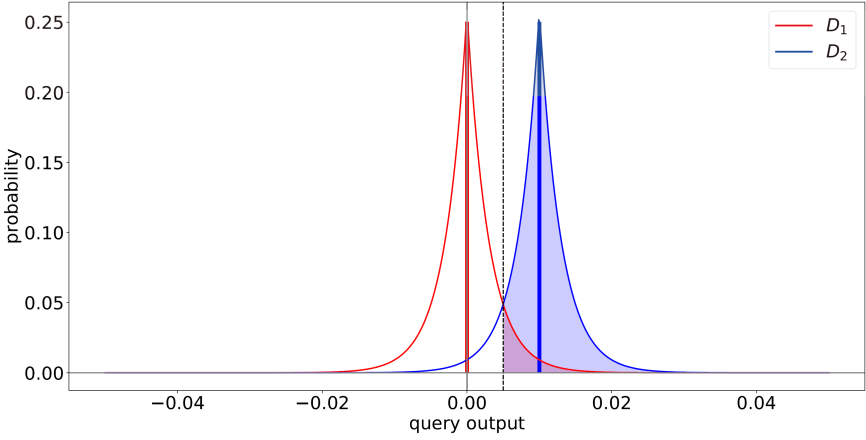


Figure 3.2. Noise from the Laplace distribution (amplitude $b = 0.003$) changes the probability of the adversary misidentifying D_1 and D_2 . The figure is taken from an article by Boenisch.

Through visualization, it can be intuitively seen that the privacy amount of statistics depends on the ratio between the probabilities of D_1 and D_2 . At any given point, when computing statistics on databases D_1 and D_2 , the probabilities of obtaining specific results should be very similar. This similarity can be expressed by a multiplicative factor close to 1, such as $(1 + \epsilon)$.

ϵ -Differential Privacy: The previous examples intuitively demonstrated the necessity of adding noise and what the privacy parameter ϵ signifies. Next, we introduce the formal definition of ϵ -differential privacy proposed by Dwork *et al.* (2006).

Definition 2 (ϵ -Differential Privacy). A randomized algorithm $\mathcal{K}$ on the domain $\mathbb{N}^{|\mathcal{X}|}$ adheres to ϵ -differential privacy if for all neighboring databases $D_1, D_2 \in \mathbb{N}^{|\mathcal{X}|}$ and all $S \in \text{Im}(\mathcal{K})$, it holds that

$$\Pr[\mathcal{K}(D_1) \in S] \leq e^\epsilon \cdot \Pr[\mathcal{K}(D_2) \in S] \quad (3.5)$$

As D_1 and D_2 can be swapped, this definition directly leads to the following lower bound:

$$\Pr[\mathcal{K}(D_1) \in S] \geq e^{-\epsilon} \cdot \Pr[\mathcal{K}(D_2) \in S] \quad (3.6)$$

Combining the two equations above, we get

$$-\epsilon \leq \log \left(\frac{\Pr[\mathcal{K}(D_1) \in S]}{\Pr[\mathcal{K}(D_2) \in S]} \right) \leq \epsilon \quad (3.7)$$

Here, $\log x$ represents $\ln x$.

Based on the multiplicative factor $(1 + \epsilon)$ seen in practical examples, Sarathy adopted a slightly different formula in a paper published in 2011 which defined a mechanism as ϵ -differential privacy if it satisfies

$$\Pr[\mathcal{K}(D_1) \in S] \leq (1 + \epsilon) \cdot \Pr[\mathcal{K}(D_2) \in S] \quad (3.8)$$

and referred to this equation as another definition of e^ϵ -differential privacy. When ϵ is small, the difference between $(1 + \epsilon)$ and e^ϵ is minimal. However, when $\epsilon \geq 0.5$, e^ϵ is at least 10% larger. As ϵ increases, the two diverge more and more. Nonetheless, e^ϵ is a common factor. Bambauer *et al.* argued in his 2013 paper that using e^ϵ has greater advantages because it corresponds to a well-known distribution curve, namely the Laplace distribution curve. This suggests that for DP, the curve must have certain properties: When the curve shifts a fixed amount, the ratio of probabilities between the original curve and the shifted curve stays within a pre-specified boundary. For e^ϵ -differential privacy, there are at least the following advantages:

1. Fast Computation: The product of two probabilities corresponds to an addition in the logarithmic space, and multiplication is more computationally expensive than addition.

2. High Precision: When probabilities are very small, using logarithmic probabilities can increase numerical stability, due to the way computers approximate real numbers. In the normal probability space, they can generate more rounding errors.

3. Simplicity: Many probability distributions, especially those extracted from random noise, have exponential forms. Taking the logarithm of these distributions removes the exponential function, making it possible to perform calculations with just the exponent.

ϵ, δ -Differential Privacy: ϵ -Differential privacy demands high privacy requirements from the mechanism. However, if too much noise is added to the original data, it can result in information loss. Thus, a number of weakened versions of DP mechanisms have been proposed. One of the most popular versions is ϵ, δ -differential privacy.

Definition 3 (ϵ, δ -Differential Privacy). A random algorithm $\mathcal{K}$ on the domain $\mathbb{N}^{|\mathcal{X}|}$ satisfies ϵ, δ -differential privacy if, for all neighboring databases $D_1, D_2 \in \mathbb{N}^{|\mathcal{X}|}$ and all $S \in \text{Im}(\mathcal{K})$, it complies with

$$\Pr[\mathcal{K}(D_1) \in S] \leq e^\epsilon \cdot \Pr[\mathcal{K}(D_2) \in S] + \delta \quad (3.9)$$

ϵ, δ -Differential privacy allows an additive violation of ϵ -differential privacy by δ . From a probabilistic perspective, δ represents the probability that the query output of neighboring datasets exceeds a factor of e^ϵ . Or more formally, the absolute value of privacy loss is bounded by ϵ with probability at least $1 - \delta$. If $\delta = 0$, ϵ, δ -differential privacy degenerates into ϵ -differential privacy. Therefore, more discussions can be conducted around ϵ, δ -differential privacy.

In addition to the aforementioned ϵ -differential privacy and ϵ, δ -differential privacy, there are also concepts such as group privacy studied by Taylor *et al.* (2016) and compositions of DP mechanisms, including sequential composition, parallel composition, and advanced composition. Another way to categorize DP mechanisms is into local and global DP mechanisms. According to Dwork's paper published in 2006, in a local method, data are perturbed at the input stage. As there is no trustworthy data curator, each individual is responsible for adding noise to their data before sharing it. In the global method, data are perturbed at the output stage. Under a global setting, every user requires a trustworthy data curator. The local method is a more conservative and secure model. Under local privacy, each individual data point is very noisy, and thus, it is not very useful in itself. However, for a large number of data points, the noise can be filtered out, allowing meaningful analysis on the dataset. Overall, the global method is more accurate as the analysis is performed on "clean" data, and only a small amount of noise needs to be added at the end of the process.

3.2.1.3 Applications of differential privacy

With the growing awareness of personal information data security, privacy protection has become a major global issue, especially for big data applications and distributed learning systems. One stand-out advantage of FL is that it can perform local training without exchanging personal data between servers and clients, thereby protecting client data from eavesdropping by hidden adversaries.

However, privacy information can still be leaked by analyzing differences in client-uploaded parameters, such as weights in deep neural network training. A natural way to prevent information leakage is to add artificial noise, i.e., employing DP techniques.

NbAFL Algorithm: Wei *et al.* (2020) proposed a new framework based on the concept of DP, known as Noise before Aggregation for Federated Learning (NbAFL). First, the paper demonstrated that by suitably adapting different artificial noise variances, NbAFL can meet DP under different protection levels. Then, a theoretical convergence bound of the loss function of the trained FL model in NbAFL was established. Specifically, the theoretical bound revealed the following three key properties: (1) There's a trade-off between convergence performance and privacy protection level, i.e., the better the convergence performance, the lower the protection level; (2) with a fixed privacy protection level, increasing the total number of clients participating in FL can improve convergence performance; (3) for a given protection level, there is an optimal maximum number of aggregation rounds (communication rounds) in terms of convergence performance. Moreover, the paper proposed a K -random scheduling strategy. This strategy randomly selects $1 < K < N$ users from the N users to participate in each aggregation round. The paper also established the corresponding loss function convergence bound under this strategy, and the K -random scheduling strategy also retains the aforementioned three properties. Furthermore, it was found that there is an optimal K to achieve the best convergence performance at a fixed privacy level. Experimental results show that the theoretical results are consistent with simulations, thereby facilitating the design of FL algorithms with different trade-off requirements between convergence performance and privacy level. The specific steps of NbAFL are shown in Algorithm 3.1.

Algorithm 3.1 outlines the process of training a model using the NbAFL algorithm under ϵ, δ -differential privacy. At the beginning of this algorithm, the server broadcasts the necessary privacy level parameters ϵ, δ and distributes the initial parameters w^0 to each user. At the t th aggregation, N active clients train the parameters separately under predefined termination conditions using their local databases. After completing local training, the i th user adds noise to

Algorithm 3.1 Noise before Aggregation for Federated Learning (NbAFL)

```

1: Input:  $T, w^0, \epsilon$  and  $\delta$ 
2: Output:  $\tilde{w}^T$ 
3: Initialization:  $t = 1$  and  $w_i^0 = w^0$ 
4: while  $t \leq T$  do
5:   Local Training Process:
6:   while  $\mathcal{C}_i \in \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_N\}$  do
7:     Update model parameters  $w_i^t$ 
8:     Clip local parameters  $w_i^t = w_i^t / \max(1, \frac{\|w_i^t\|}{C})$ 
9:     Add noise to local parameters  $\tilde{w}_i^t = w_i^t + n_i^t$ 
10:  end while
11:  Model Aggregation Process:
12:  Update global model parameters  $w^t : w^t = \sum_{i=1}^N p_i \tilde{w}_i^t$ 
13:  Server broadcasts global noisy parameters  $\tilde{w}^t = w^t + n_D^t$ 
14:  Local Testing Process:
15:  while  $\mathcal{C}_i \in \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_N\}$  do
16:    Test aggregated parameters  $\tilde{w}^t$  using local dataset
17:  end while
18:   $t \leftarrow t + 1$ 
19: end while

```

the trained parameters w_i^t , updates the noisy parameters $\tilde{w}_i^t$, and uses them for aggregation. The server then updates the global parameter w^t by aggregating different local parameters' weights.

LDP-Fed Algorithm: Other literature has proposed FL algorithms based on LDP. Truex *et al.* (2020) proposed a new FL system, LDP-Fed, which utilizes LDP. Existing LDP protocols are primarily developed to protect the privacy of individual numeric or categorical value sets in data, such as click counts in web access logs. However, in FL models, parameter updates are iteratively collected from each participant and are composed of high-dimensional, continuous, and precise values (up to 10 decimal places), rendering existing LDP protocols unsuitable. To address this challenge in FL problems based on LDP, two new methods were designed and developed. First, the

LDP-fed's LDP module provides formal differential privacy guarantees for repetitively collecting model training parameters in large-scale neural network joint training on separate participants' private datasets. Second, LDP-fed implements a selection and filtering technique for disturbing (i.e., adding noise) and sharing the selected updated parameters on the server.

The LDP-fed system coordinates FL of deep neural networks with N participants (clients) and a parameter server, integrating LDP into the overall architecture of FL algorithms to protect participant data from inference attacks. Specifically, consider N participants with the same dataset structure and learning tasks who wish to collaboratively train a deep neural network in a FL mode. That is, each participant aims to train locally on their private data and share parameters with the server for model updates. Moreover, participants wish to avoid the risk of privacy leakage while achieving personalized LDP. The following outlines how the algorithm achieves this from the user (participant) and server perspective:

Client Side:

- (1) Participants initialize the local deep neural network with model parameters θ_0 , and the LDP module is initialized with different privacy parameters based on user preferences.
- (2) Each participant computes the gradients required for training based on their private local datasets.
- (3) Each participant adds noise to their gradients according to their LDP module.
- (4) The updated model parameters are anonymously sent to the k -client selection module, which accepts or rejects updates randomly at a uniform probability $q = k/N$.
- (5) Each participant waits to receive the updated aggregated parameters from the parameter server. After receiving the aggregated update, each participant updates their local deep neural network model and starts the next iteration.

Server Side:

- (1) The parameter server initializes model parameters θ_0 and sends them to each participant.

- (2) The server waits to receive k parameters randomly selected by the k -client selection module.
- (3) Once parameter updates are received, the aggregation module aggregates the updates, i.e., it averages the gradient updates to determine the new model parameters.
- (4) The parameter server updates the model parameters and sends the updated values back to the participants for local model updates.

These steps iterate among the N participants and the server until a predetermined convergence condition is met, such as reaching the maximum number of rounds (iterations) or no further improvement in the test set performance. Compared to traditional FL systems, LDP-fed introduces two new components: (1) the LDP module running on N clients and (2) the k -client selection module. The former is responsible for adding a certain amount of noise to the input data to achieve DP, while the latter randomly selects updates from k participants and uploads them to the server in any given round, like traditional FL systems where not every participant is required to share their local training parameter updates in each round. Experimental results on numerous benchmark datasets demonstrate that, from a technical standpoint, LDP-fed significantly outperforms existing state-of-the-art methods in terms of model accuracy and privacy protection.

3.2.1.4 Other algorithms

FL represents the latest advancements in privacy protection. In this scenario, a trusted server-side entity aggregates parameters optimized by multiple clients in a distributed manner. The final model is then distributed back to all the clients, eventually converging to a unified representation model without explicitly sharing data. However, this training protocol is susceptible to differential attacks, which could originate from any party during federated optimization. Specifically, in such attacks, analysis of the distributed model can reveal the contributions of clients during the training process and the information of their datasets. Therefore, Geyer *et al.* (2017) proposed a client-centric, DP-based federated optimization algorithm in their 2017 paper, aiming to address this issue. Its goal is to hide client contributions during training, achieving a balance between privacy loss

and model performance. First, the paper demonstrates that in FL, client participation can be concealed while maintaining high model performance and proves that the proposed algorithm can achieve client-level DP with a relatively small loss in model performance. Second, the algorithm can dynamically adapt the DP preservation mechanism in distributed training. Case studies suggest that model performance can be enhanced this way, contrasting with the latest advancements in centralized training. This performance discrepancy can be linked to a fact: In FL, gradients show different sensitivities to noise and batch size throughout the training process compared to centralized learning.

Some research focuses on specific applications. For instance, McMahan *et al.* (2017) investigated privacy issues on mobile keyboard word prediction and used it as a validation case. Specifically, this work added user-level privacy protection in the federated averaging algorithm, allowing for “large-step” updates from user-level data. Moreover, the study demonstrates that achieving different levels of privacy is at the expense of computational load, rather than degrading model performance as most previous works suggest, given a dataset with a sufficient number of users (even a small-scale internet dataset can easily meet this requirement). Additionally, experiments showed that when trained on large datasets, the privacy-protected LSTM language model demonstrated similar quantitative and qualitative performance to a noiseless model.

Other works have researched the stochastic gradient descent (SGD) algorithm under a DP mechanism. For instance, Koskela *et al.* (2018) proposed an algorithm that performs SGD with an adaptive learning rate, avoiding the use of a validation set. The adaptive idea comes from an extrapolation technique: To estimate the error in the SGD gradient flow, the authors compared the results of a full step and two half steps. The algorithm can be applied to two independent frameworks: FL and privacy learning based on DP. The paper indicated that, unlike common optimization methods, this algorithm exhibits robustness in the context of FL.

3.2.2 Secure Multi-Party Computation

In real-life scenarios, there often exists a situation where multiple parties wish to collaboratively train various models without revealing

their private information. To technically address this issue, the concept of MPC, also known as SMC, has emerged. In this section, we present an overview of the recent advancements related to this issue.

3.2.2.1 Definition of secure multi-party computation

How can we compute a function using data from different parties without violating their privacy? Ideally, we could assume the existence of a Trusted Third Party (TTP). Participants only need to submit their individual inputs to this third party, who then calculates the value of the function on behalf of the participants and informs them of the result, as shown in Figure 3.3(a).

However, in practical situations, a TTP often does not exist. We need to devise an encryption protocol, in which, by exchanging messages among themselves, parties can still learn the function values without revealing their actions or relying on a TTP, as illustrated in Figure 3.3(b). From this, we provide the definition of SMC:

Definition 3.1. In MPC, assume there are n participants: $P_1, P_2, \dots, P_n$ each having their own private data $d_1, d_2, \dots, d_n$. Participants wish to compute the value of a public function $F(d_1, d_2, \dots, d_n)$ on the private data while technically protecting the privacy of their data.

From a technical perspective, MPC has numerous applications in real life, such as electronic voting, private bidding and auctions,

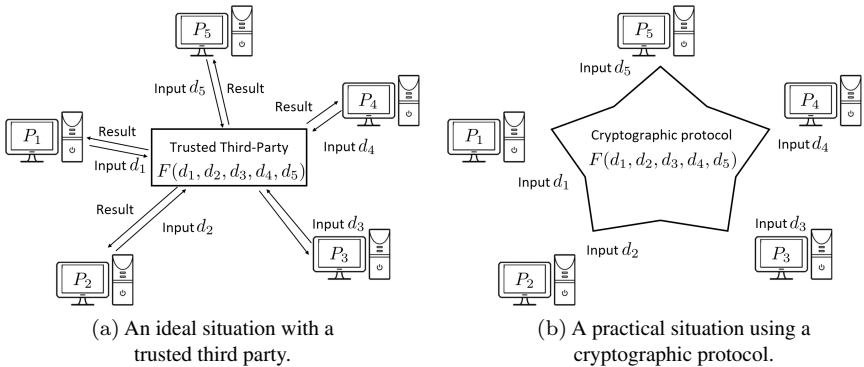


Figure 3.3. Two scenarios of multi-party computation.

shared signing or decryption functions, and proprietary information retrieval. Based on the number of participants, we divide MPC into secure two-party computation (represented by Yao's Garbled Circuit) and SMC (represented by secret sharing) and introduce them in the following sections.

3.2.2.2 Yao's Garbled circuit

Yao's Garbled Circuit (GC), an important technology in MPC, was proposed in 1986 by Turing Award winner Yao Qizhi. Initially, this technology was developed to solve a secure two-party problem, known as Yao's Millionaires' problem. This problem assumes there are two millionaires, Alice and Bob, interested in determining who is wealthier without revealing their actual wealth. In mathematical terms, given two numbers a and b , we want to compare their magnitude without exposing their true values. This problem requires a Garbled circuit protocol for a comparison circuit, which can be quite complex for beginners. Here we start by explaining the Garbled circuit protocol from a simple AND gate circuit problem, which can be easily extended to the entire comparison circuit: Alice and Bob wish to understand whether they should collaborate on a project. This is a sensitive issue for them, so neither wants the other to know their opinion unless the other agrees to cooperate. Here we denote Alice's and Bob's willingness as a and b , respectively. The following section introduces this technology.

To simplify the process, we assume their intentions to be binary digits, Bits, namely $\{0,1\}$. We know that the basic unit of a circuit is a gate, common ones include AND gate, OR gate, NOT gate, and XOR gate. Figure 3.4 shows four common types of gates. Each gate has a corresponding truth table, in the original garbled circuit, encryption and scrambling can be represented by truth tables. Next, we'll use the AND gate as an example to introduce the operation process of the garbled circuit. Let Alice play the role of a garbled circuit generator or "Garbler", who can generate a garbled AND gate and then send it to Bob. Bob, on the other hand, plays the "Evaluator", who can compute and evaluate this garbled gate.

Now we have four random strings, namely $W_A^0, W_B^0, W_A^1, W_B^1$. W_A^0 corresponds to the situation where $a = 0$, and W_A^1 corresponds to the situation where $a = 1$. Similarly, W_B^0 corresponds to the situation

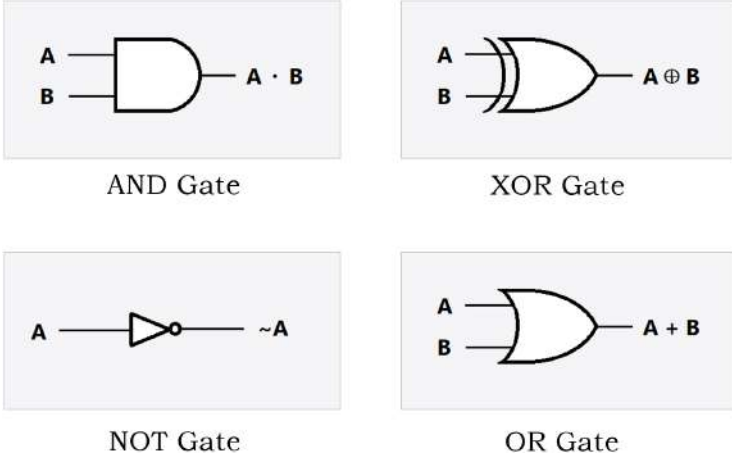


Figure 3.4. Schematic diagram of four common gates.

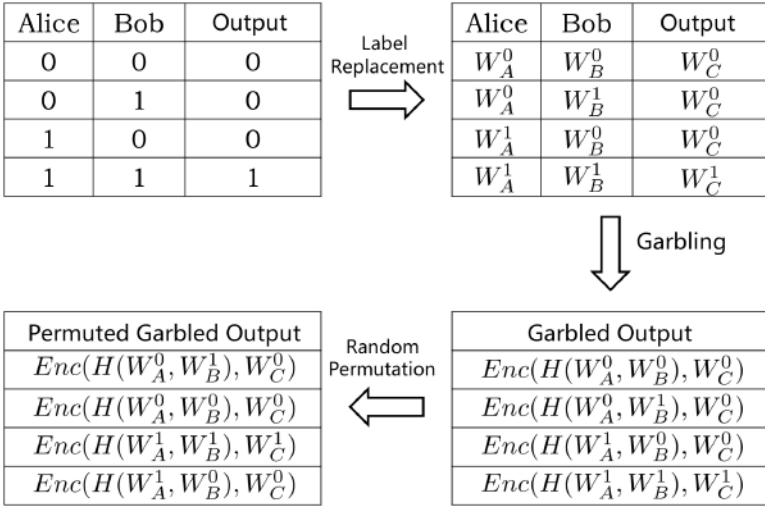


Figure 3.5. Garbling of the AND gate. $H(\cdot, \cdot)$ represents the key generation function H and $Enc(\cdot, \cdot)$ represents the encryption process.

where $b = 0$, and W_B^1 corresponds to the situation where $b = 1$. If we replace the real inputs and outputs with random strings, then a truth table for the input and output of an AND gate would look like Figure 3.5. Alice then uses each pair of strings corresponding to possible scenarios $((a = 0, b = 0), (a = 0, b = 1), (a = 1, b = 0),$ and

($a = 1, b = 1$) to encrypt the output corresponding to that scenario. Two string inputs generate a symmetric encryption key through the key generation function H and use this key to encrypt the AND gate. The final garbled gate will contain four ciphertexts in random order. The whole garbled circuit process is shown in Figure 3.5.

Once Bob receives the garbled gate, he needs to accurately decrypt a ciphertext that corresponds to the real values of a and b and is generated by $H(W_A^a, W_B^b)$. To accomplish this, he needs to receive the values of W_A^a and W_B^b from Alice. Since Alice knows her own value of a , she can send W_A^a . Since W is random, independent, and identically distributed, Bob will not obtain any information about a from W_A^a . However, sending W_B^b to Bob will be difficult. Alice can't send both W_B^0 and W_B^1 to Bob, as that would allow Bob to decrypt two ciphertexts in the garbled gate, which would violate the security definition in MPC. Similarly, Bob can't ask for any value he wants because he doesn't want Alice to know the value of b . Therefore, Alice and Bob can use the OT method, which allows Bob to know only W_B^b without revealing b to Alice. Note that for this method to work, Bob needs to know when decryption is successful and when it fails. Otherwise, he wouldn't know which ciphertext would produce the correct answer.

Since Bob needs to attempt to solve all four ciphertexts here, this introduces a large amount of computation. To solve this problem, researchers propose the use of the point and permute method, allowing Bob to directly locate the ciphertext to be decrypted, thus saving a significant amount of computation. We give a brief introduction to this method: Since each gate has two inputs, we usually refer to the inputs as wires. Now each wire w , besides the strings W^0 and W^1 , will also input select bits p^0 and p^1 . For $v \in \{0, 1\}$, the select bit p^v will equal $v \oplus r$, where $\oplus$ refers to the XOR operation and r is a randomly chosen bit. Therefore, the select bit p^v differs from the two possible values of v but at the same time does not reveal any information about v . Each wire simultaneously inputs the select bit p and the string W (via OT if it's an input wire, otherwise by decryption). When evaluating a gate, Bob uses the two select bits corresponding to the two input wires (for instance, let p_i and p_j correspond to wires w_i and w_j) to determine which ciphertext in gate k needs to be decrypted. With this point and permute optimization, Alice often

places $Enc(H(W_i^{v_i}, W_j^{v_j}), W_k^{g_k(v_i, v_j)} ||| p_k^{g_k(v_i, v_j)})$ at position $2p_i^{v_i} + p_j^{v_j}$ in the truth table (note that there's no need to scramble the truth table).

This method makes use of simpler, more efficient encryption schemes (such as one-time pad) possible. Since the select bit tells Bob which ciphertext to decrypt, there's no longer a need to differentiate between decryption using the correct key and decryption using the wrong key, thereby reducing the computational load by a factor of 4 during evaluation. More optimizations for garbled circuits, such as Free XOR and Row reduction, can be found in the survey written by Sophia Yakubov in 2017.

The binary input example given here is a simple illustration of secure two-party computation. In reality, Alice and Bob may both have multiple binary inputs, and they might want to compute a more complex function or to accomplish more complicated tasks. For instance, they might wish to compute the Hamming distance of the input strings, the size of their set intersection, or the previously mentioned Millionaires' problem. In such a case, Alice would garble the entire circuit (i.e., the comparison circuit), not just a single gate. For gates whose output is the input to other gates, she will not encrypt the output result but rather the string W corresponding to the output result, which can then be used to derive the decryption key for the ciphertext in other gates.

3.2.2.3 Secret sharing

Secret sharing is another common method in MPC. With secret sharing techniques, we can perform MPC. Here, we assume that there are n parties, each with data x_i . They want to compute the value of a function F on data $x_1, x_2, \dots, x_n$, but at the same time, they do not want to disclose their data to other participants. This process consists of three steps: First, each party generates n shares of their own data and distributes them to others. Please note that the shared data must meet a condition: Only by accessing at least t shares can the original data be recovered. Next, each party needs to run a local evaluation algorithm on the shares they receive to get local results. The final step is to recover the function value $F(x_1, x_2, \dots, x_n)$ by combining the local results after using the decoding algorithm. The secret sharing

scheme used in this scenario is called the (t, n) -threshold scheme. The following discusses different cases for different values of t :

- $t = 1$: This situation is meaningless. The data are distributed among all n participants.
- $t = n$: In this case, all shares need to be collected to recover the original data. Although this scheme is relatively secure in terms of data, it is also fragile (if a share cannot be accessed, it leads to the original data never being recoverable).
- $1 < t < n$: This can also be described as a broader interpretation, i.e., any subset of size n . This situation needs to ensure security while not needing n shares to recover the data.

However, this arbitrary enumeration method becomes increasingly unrealistic as the subset size grows. For example, if we want to recover the secret through any 20 out of 50, we will have to design $C_{50}^{20} \approx 4.7 \times 10^{13}$ schemes. Such a design would be inefficient. Next, we introduce the classic secret sharing schemes proposed by Shamir (1979) and Blakley (1979), respectively.

Shamir's Secret Sharing Scheme: Shamir's secret sharing scheme is constructed based on Lagrange's interpolation theorem. which as follows:

Definition 3.2. For a polynomial function, we have given t points: $\{(x_1, y_1), (x_2, y_2), \dots, (x_t, y_t)\}$, where x_i represents the value of the variable and y_i represents the value of the function at the corresponding variable. Then we can construct such a Lagrange interpolation polynomial:

$$L(x) = \sum_{i=1}^t y_i \prod_{j=1, j \neq i}^t \frac{x - x_j}{x_i - x_j}. \quad (3.10)$$

Such a polynomial exists and is unique if the highest degree does not exceed $t - 1$. Here, $l_i(x) = \prod_{j=1, j \neq i}^t \frac{x - x_j}{x_i - x_j}$ is generally referred to as the Lagrange base polynomial (or interpolation base function). It is characterized by taking the value 1 at x_i and 0 at other points $x_j, j \neq i$.

With this theorem, we can first construct a polynomial function of degree $t - 1$: $f(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \dots + a_1x^1 + s$,

where s is the secret to be shared. We first randomly select $t - 1$ positive integers $a_{t-1}, a_{t-2}, \dots, a_1$ to determine the parameters of the $t - 1$ degree polynomial $f(x)$. Then we generate n shares for this function, i.e., randomly select n points, n different positive integers $x_n, x_{n-1}, \dots, x_1$, and compute $y_i = f(x_i)$. Then we distribute the sample point-function value pairs (x_i, y_i) to each participant.

Now, if any t participants want to recover the secret s , they can first collect their own shares (x_i, y_i) , i.e., gather the required t points, and use Formula (3.10) by setting x to 0 to achieve the goal, i.e.,

$$s = \sum_{i=1}^t y_i \prod_{j=1, j \neq i}^t \frac{x_j}{x_j - x_i} \quad (3.11)$$

To facilitate understanding for our readers, we provide a simple example: Let's suppose we have a secret $s = 10$ that we wish to share. We want the secret to be recoverable when $t = 4$. Since the constant term is s , we need to specify $t - 1 = 3$ parameters. For the sake of convenience, let's assume that $a_1 = 1, a_2 = 0, a_3 = 2$. Hence, the polynomial function we construct is $f(x) = 2 \cdot x^3 + 1 \cdot x^1 + s$. We now need to sample n points and distribute them to the n participants (we assume that $x = n$, but in reality, it could be chosen randomly). The shares each participant receives, that is, the sample point-function value pairs (x_i, y_i) , are as follows:

$$D_1 = (1, f(1)) = (1, 2 \cdot 1^3 + 1 \cdot 1^1 + 10) = (1, 13)$$

$$D_2 = (2, f(2)) = (2, 2 \cdot 2^3 + 1 \cdot 2^1 + 10) = (2, 28)$$

$$D_3 = (3, f(3)) = (3, 2 \cdot 3^3 + 1 \cdot 3^1 + 10) = (3, 67)$$

⋮

Let's say participants 1, 2, 5, and 7 decide to group together and recover the secret. They can use the data they received to construct the following system of equations:

$$a_3 \cdot 1^3 + a_2 \cdot 1^2 + a_1 \cdot 1^1 + s = 13$$

$$a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + s = 28$$

$$a_3 \cdot 5^3 + a_2 \cdot 5^2 + a_1 \cdot 5^1 + s = 265$$

$$a_3 \cdot 7^3 + a_2 \cdot 7^2 + a_1 \cdot 7^1 + s = 703$$

Solving this system of equations can yield the secret s and the three parameters a_1, a_2, a_3 . You can also use the known sample point-function value pairs to solve for the secret s alone using Equation (3.11). This example is a simplified version. In reality, if fewer than t participants group together, they won't be able to directly recover the secret but can still limit it from an infinite number of natural numbers to a finite set. This method is currently not considered secure. Geometrically speaking, this kind of attack makes use of the fact that we know the order of the polynomial, so we can guess its possible path between known points. This reduces the potential values of unknown points, as they must be on the smooth curve. This issue can be addressed with finite field arithmetic: Now we have a large prime $p > n$ and $p > a_i, i = 1, \dots, n$. Our calculation of the sample point-function value pairs $(x_i, y_i = f(x_i))$ changes to $(x_i, y_i = f(x_i) \bmod p)$, where $\bmod$ refers to the modulus operation. Detailed proofs can be found in other relevant literature, which we won't elaborate here.

Shamir's secret sharing scheme can be described as follows: We can determine a line with two points, define a quadratic curve with three points, a cubic curve with four points, and so on. That is, t points can be used to define a polynomial curve of $t - 1$ degree.

Blakley's Secret Sharing Scheme: Blakley's secret sharing scheme shares a similar idea with Shamir's scheme: two non-parallel lines on the same plane intersect at a point. Three non-parallel planes in space intersect precisely at one point. More generally, any n non-parallel $(n - 1)$ -dimensional hyperplane intersects at a specific point. So, the secret can be encoded as the coordinates of any such intersection, as shown in Figure 3.6.¹

Considering security, if the secret is encoded using all of the n coordinates, even if they are random, then an insider (someone who possesses one or more $(n - 1)$ -dimensional hyperplanes) would gain information about the secret because they know that the secret must lie on their plane. If the insider can gain more knowledge about the secret than an outsider, then the system no longer has information-theoretic security. If only one of the n coordinates is used, then the insider will not know more than an outsider (in other words, for

¹Image from Wikipedia's "Secret sharing" entry.

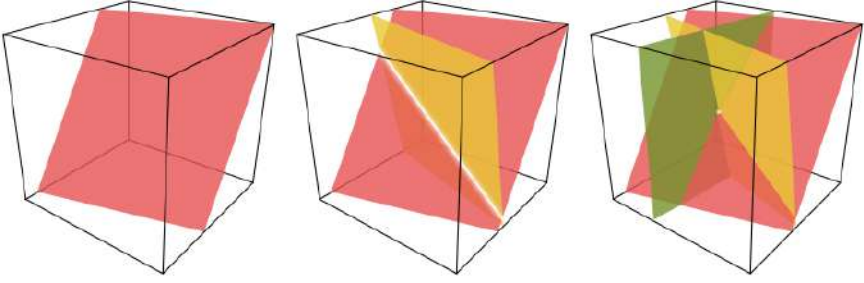


Figure 3.6. Blakley’s secret sharing scheme in three dimensions. Each share can be seen as a plane, so the current secret can be viewed as the intersection of three shares. Even though two shares already constrain the secret to be somewhere on the line of intersection between the shares, it’s still not enough to determine the secret.

a two-dimensional system, the secret must lie on the x axis). Each participant has enough information to define a hyperplane; the secret can then be recovered by calculating the intersection of the planes and taking the specified coordinate of the intersection point.

Blakley’s scheme is less space-efficient than the Shamir’s scheme; that is, the size of Shamir’s shares is the same as the original secret, while Blakley’s shares need to be t times the size of the original secret. If we impose some additional constraints on the planes in Blakley’s scheme, then Blakley’s scheme will reduce to Shamir’s scheme. In other words, Blakley’s scheme is a more general version of Shamir’s scheme.

It’s worth mentioning that Yao’s garbled circuits can also be described using secret sharing, referred to as Yao sharing in literature. However, combining Yao sharing with other secret sharing modes is not straightforward and requires a special transformation function (refer to the word “SecureML” by Mohassel *et al.* (2017)). For more detailed and up-to-date information on secret sharing, refer to the review written by Beimel (2011).

3.2.2.4 Oblivious transfer

OT is another cornerstone of MPC. Rabin proposed the earliest OT protocol in 1981, which implemented the “1-out-of-2” function, denoted as OT_1^2 . Subsequently, the 1-out-of- n protocol OT_1^n and the

m -out-of- n protocol OT_m^n were proposed. Following is the most common definition of the m -out-of- n protocol OT_m^n .

Definition 3.3 (m -out-of- n OT protocol OT_m^n). Party A has input data $d_1, d_2, \dots, d_n$, and party B has input indices $\{i_1, i_2, \dots, i_m\} \in \{1, 2, \dots, n\}$. The m -out-of- n OT OT_m^n protocol ensures that party B can always get $d_{c_1}, d_{c_2}, \dots, d_{c_m}$ but cannot get any other values from party A, and party A is unaware of party B's choices $\{i_1, i_2, \dots, i_m\}$.

In simpler terms, in an OT protocol, the sender has a database, and the receiver wants to access specific items in this database. However, the receiver does not want the sender to know which items they need to access, and similarly, the sender does not want to leak the entire database to the receiver. In MPC, OT is often used as a complementary technology to garbled circuits or secret sharing. For example, in garbled circuits, the evaluator (Bob) encodes his data (note that the random key is held by Alice) using the OT method.

3.2.2.5 Applications of secure multi-party computation

MPC is widely used in tasks, such as secure database construction and distributed data mining. In data mining, early work primarily focused on models such as decision trees, K-means, SVM, and linear regression. Recently, with the development of deep neural networks, more attention has been paid to developing efficient MPC methods suitable for deep models, like EzPC proposed by Chandran *et al.* (2017), Gazelle proposed by Juvekar *et al.* (2018), and Oblivious proposed by Liu *et al.* (2017).

Feasibility Analysis of Deep Models Based on Multi-Party Computation: Specifically, we are more interested in distributed data mining. There are many scenarios where we cannot train models locally, as data may be scattered among multiple entities, such as in a horizontal data distribution (each party owns a subset of data samples) or vertical data distribution (each party owns a subset of features), or when data and models are located in different entities. In such cases, SMC is a promising technical solution, but there are

still significant challenges, such as efficiency, scalability, and expressiveness. For example, a typical neural network includes linear matrix multiplication, max-pooling, nonlinear activation functions (ReLU, Softmax, etc.), batch normalization, and dropout. Using secret sharing techniques, such as Beaver triples, we can effectively compute matrix multiplication, but there is not yet a universal solution for nonlinear operations. Yao’s garbled circuit can perform nonlinear operations by defining a Boolean circuit, but it has very high computational and communication overhead. Due to the simplicity and compatibility of secret sharing with linear operations, it is another promising choice for performing these nonlinear operations.

A Simple Comparison with Homomorphic Encryption-Based Deep Learning Frameworks: HE, although it only requires a single round of interaction, does not support efficient non linearity. For instance, nGraph-HE and its extensions, built on the Microsoft SEAL library, provide a secure evaluation framework that greatly improves upon the work of CryptoNet, but it uses polynomials (such as squares) for the activation functions.

In comparison to HE, MPC frameworks typically use lightweight encryption technology and can provide faster implementation methods. MiniONN and DeepSecure use optimized garbled circuits, requiring few communication rounds, but they do not support training and altering the neural network structure to accelerate algorithm execution. Other frameworks, like ShareMind, SecureML, SecureNN, or the latest FALCON, rely more on additive secret sharing and allow secure model evaluation and training. They use simpler, more efficient primitives but require a large number of communication rounds; for example, for the ReLU function, SecureNN requires 11 rounds and FALCON needs $5 + \log_2(l)$ (usually 10 when $l = 32$) rounds. ABY, Chameleon, and the recent ABY³ mix garbled circuits and additive or binary secret sharing based on the idea of which operation is most efficient to consider. However, the transition between the two is costly, and these methods, except for ABY³, do not support training this transition. Lastly, works like Gazelle combine HE and SMC, taking full advantage of both, but the conversion cost is also high.

For example, AriaNN, proposed by Ryffel *et al.* (2020), designed secure comparisons based on secret sharing and applied them to the

evaluation of ReLU and MaxPooling. As a result, they successfully trained a VGG16 network with satisfactory accuracy on multiple benchmark datasets. However, many nonlinear operations are not yet supported by the current state-of-the-art frameworks, such as Softmax and Dropout. These operations are essential for neural networks to achieve good performance. Therefore, our aim is to bridge this gap; in other words, our goal is to develop effective secret sharing methods for other basic nonlinear operations. Lastly, note that we still need about 60 hours in a local network environment to train an Alexnet with only four epochs on the CIFAR-10 dataset, as reported in Ryffel's report. For reference, training one epoch with the same settings locally takes only a few minutes. Nonlinear operations are still the bottleneck, for instance, the random keys used in comparison operations are proportional to the input size, so the batch size is limited to a very small range, thereby slowing down the convergence speed. To further improve efficiency, we can develop better secret sharing protocols or exploit the parallelism of distributed learning, such as dispersing comparisons to multiple servers, so we can evaluate them in parallel.

3.2.3 Homomorphic Encryption

The rapid development of technologies such as artificial intelligence, big data, and cloud computing has provided effective means for processing, storing, and using massive amounts of user data. However, the application of these technologies inevitably raises concerns about data privacy and security. HE is a unique encryption scheme that allows a third party to process encrypted data (ciphertext) without the need for prior decryption, thus effectively addressing these concerns. This useful feature of HE has been known for 30 years, and a significant amount of research has been devoted to advancing this technology. To give researchers and practitioners a holistic understanding of HE, this section focuses on algorithm descriptions, homomorphic property analyses, etc., to introduce the research on Partially Homomorphic Encryption (PHE), Somewhat Homomorphic Encryption (SWHE), and Fully Homomorphic Encryption (FHE).

3.2.3.1 Introduction

Encryption is an essential scheme to protect sensitive information or privacy. However, traditional encryption schemes are unable to compute on ciphertexts without prior decryption. In other words, users must sacrifice their privacy to enjoy services from third parties (such as cloud services and digital tech services), and internet companies risk leaking user data privacy while using user data. This technological drawback has caused user concerns about data privacy leaks. If a scheme could perform desired operations on encrypted data, it would be perfect. HE is such a technical scheme. The word “homomorphism” is rich in meaning. In abstract algebra, a homomorphism is a mapping that preserves all algebraic structures between the domain and codomain of an algebraic set. In cryptography, homomorphism refers to a type of encryption. HE is a scheme that allows a third party to preserve the function form and characteristics of ciphertext while performing specific computations. In fact, this HE corresponds to the mapping in abstract algebra. Due to the extensive application of HE technology in fields like cloud computing and privacy-protecting ML, this chapter introduces relevant HE and representative studies in PHE, SWHE, and FHE.

Figure 3.7 illustrates an example of HE. User C first encrypts their private data (message m) (Step 1) and then sends the encrypted data to a cloud server (Step 2). When User C wishes to compute (query), i.e., execute function $f(m)$, he/she sends the function form $f(\cdot)$ to the server (Step 3). After receiving the request, the server computes function $f(m)$ homomorphically on the ciphertext (Step 4) and then

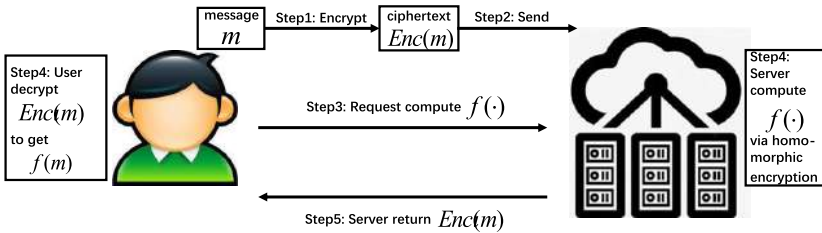


Figure 3.7. A simple diagram of homomorphic encryption in a user–server scenario.

sends back the computation result (in the form of ciphertext) to the user (Step 5). Finally, the user decrypts the result using their key, yielding $f(m)$ (in plaintext form, Step 6). Note that, in this example, the server does not need to know the user's private key when performing homomorphic computations (supporting addition and multiplication) on the ciphertext.

Before Gentry proposed the groundbreaking FHE scheme, there had already been a considerable amount of research related to HE. However, the schemes proposed in these studies either only supported one type of homomorphic operation (e.g., only supporting addition and subtraction or only supporting multiplication and division) or could only perform a limited number of operations on the ciphertext. Some schemes were even limited to specific types of problems (e.g., branching programs). Depending on the types of operations allowed on the ciphertext, existing HE technologies can be divided into three categories:

- *partially homomorphic encryption*, which supports only one type of homomorphic operation but does not limit the number of operations;
- *somewhat homomorphic encryption*, which supports multiple types of homomorphic operations but can only perform a limited number of computations;
- *fully homomorphic encryption*, which supports multiple types of homomorphic operations and does not limit the number of operations.

Given the characteristics of these three types of HE schemes, PHE can only be used in specific applications that only include addition or multiplication; SWHE supports both addition and multiplication, but the size of the ciphertext in this scheme increases with each homomorphic operation, so the maximum number of homomorphic operations it allows is limited. These issues limit the practical application of PHE and SWHE. Gentry's FHE scheme, first proposed in 2009, overcomes the limitations of these two schemes. However, its conceptual complexity and implementation difficulty, especially the high computational cost of some processes, make it less suitable for practical application. Over time, researchers have continuously proposed many improved new schemes based on this method.

3.2.3.2 Introduction to homomorphic encryption

In this section, we begin by defining HE, followed by an overview of PHE, SWHE, and FHE schemes. Finally, we delve into the security analysis of HE and the required mathematical knowledge. Due to space constraints, we merely introduce these mathematical concepts. Readers who wish to gain a deeper understanding are encouraged to refer to the relevant materials.

Definition of Homomorphic Encryption: To provide a clearer understanding of the classification of HE schemes mentioned earlier, we introduce the definition of homomorphic operations.

Definition 3.4 (Homomorphic Encryption). An encryption scheme is called a homomorphic operation over the operator “*” if the following equation holds:

$$E(m_1) * E(m_2) = E(m_1 * m_2), \quad \forall m_1, m_2 \in M \quad (3.12)$$

where m_1, m_2 are plaintexts, E is the encryption algorithm, and M is the plaintext space composed of all plaintexts.

Specifically, when designing an encryption scheme that supports homomorphic computation for any function, one only needs to consider homomorphic addition and multiplication operations, as addition and multiplication form a complete set of function operations on a finite set (i.e., any function operation on this set can be composed of addition and multiplication). Similarly, for Boolean circuits, support for XOR gate (corresponding to addition) and AND gate (corresponding to multiplication) operations is sufficient. Depending on the method of using the key during decryption, HE schemes can be divided into symmetric HE and asymmetric HE schemes. The former refers to using the same key for encryption and decryption, while the latter involves using different keys for encryption and decryption, respectively.

A HE scheme comprises four fundamental elements: key generation function, encryption function, decryption function, and evaluation function. In asymmetric HE schemes, the key generation function produces a key pair comprising a private key and a public key; whereas in symmetric HE, it generates a single key (generally, the public key). The encryption function encrypts the plaintext to produce the ciphertext; the decryption function decrypts the information in the ciphertext domain to yield the plaintext. Both are

processes found in classic encryption schemes, with only the evaluation function being unique to HE. It takes ciphertext as input and returns the corresponding computation results in the ciphertext domain. Importantly, the evaluation function can perform a function $f(\cdot)$ operation on ciphertexts (c_1, c_2) without obtaining (m_1, m_2) . A key point of HE is that the format of the ciphertext after computation must remain consistent with its pre-computation format to be correctly decrypted back to plaintext. In addition, the size of the ciphertext should remain constant to enable an infinite number of operations. Among the existing HE schemes, PHE schemes support evaluation functions involving only addition or multiplication operations. SWHE schemes only support a finite number of operations in the ciphertext domain. Only FHE schemes can execute any function (such as search, sort, and find extremes) an unlimited number of times in the ciphertext domain.

Security Analysis of Homomorphic Encryption: In HE, security is the primary concern. The security of existing HE schemes is based on complexity theory. Complexity theory can be used to judge the difficulty of solving a problem. If solving a problem requires a significant amount of resources (or an unrealistic cost), it is considered computationally hard. The security design and construction of the public-key cryptosystem in existing HE schemes often involve the following computationally hard problems:

- ❑ **Integer Factorization Problem (IFP):** Given an arbitrary positive integer n , compute its prime factorization $n = p_1^{e_1}, p_2^{e_2}, \dots, p_m^{e_m}$, where p_i and p_j are distinct prime numbers, and the positive integer $e_i \geq 1$.
- ❑ **Discrete Logarithm Problem (DLP):** Given a prime number p , a generator a of the group Z_p^* , and an element $b \in Z_p^*$, compute an integer $0 < x \leq p - 2$ that satisfies $x = \log_a^b$.
- ❑ **Quadratic Residuosity Problem (QRP):** A quadratic residue of an integer n with respect to another integer p refers to the remainder obtained when n^2 is divided by p . Let $n = pq$, where p and q are prime numbers. Let x be an integer such that the Jacobi symbol $(\frac{x}{n}) = +1$. The QRP is to determine whether x is a quadratic residue modulo n given x and n .
- ❑ **Decisional Composite Residuosity Problem (DCRP):** Given two large prime numbers p and q , let $N = pq$. Given an

arbitrary $y \in \mathbb{Z}_{N^2}^*$ such that $z = y^N \bmod N^2$, determine whether z is an N -residue or a non- N -residue.

- **Approximate Greatest Common Divisor Problem (AGCDP):** Given a set of randomly selected integers $x_1, \dots, x_n$, it is required to determine that each integer x_i is close to their approximate common factor p , where p is a large prime number.
- **Sparse Subset Sum Problem (SSSP):** Given m n -bit integers $\beta_1, \dots, \beta_m$ and an integer α . Determine whether there exists a subset $S \subset [m]$ such that $\sum_{i \in S} \beta_i = \alpha$ holds.

The above problems involve numerous concepts and knowledge of number theory and discrete mathematics. Here, we briefly introduce several commonly used concepts:

Group: A group represents an algebraic structure that is closed, adheres to the associative law, possesses a unit element, and has an inverse element with respect to binary operation.

Generator: A set containing the smallest number of group elements, which can generate all elements in the group by the product of elements in the set, is referred to as the group's generators. The number of generators is equal to the rank of the finite group.

Euler Function: The Euler function, denoted as $\varphi(n)$, signifies the count of numbers between 1 and n that are coprime (or relatively prime) to n . For a prime number p , it holds that $\varphi(p) = p - 1$.

Multiplicative Sub-group: $\mathbb{Z}_{n^2}^*$ denotes the multiplicative sub-group of integers modulo n^2 .

Ring: A binary algebraic structure $(R, +, *)$ constitutes a ring if the “+” operation obeys the commutative and associative laws, the “*” operation satisfies the commutative, associative, and distributive laws, and if there exists a zero element and an inverse element.

Lattice: A lattice is a partially ordered set where every non-empty finite subset has both a supremum (least upper bound) and an infimum (greatest lower bound).

3.2.3.3 Partial homomorphic encryption (PHE)

Several PHE schemes have been proposed, but we only discuss a few classical ones here as they form the basis for subsequent

PHE schemes. Due to the simplicity of these schemes, we provide corresponding numerical examples. However, for the upcoming SWHE and FHE schemes, due to their conceptual complexity, no specific instances are introduced.

RSA encryption scheme: The RSA encryption scheme is an early PHE scheme, named after its inventors, Rivest, Shamir, and Adleman. It was the first feasible implementation of public-key cryptography. Its homomorphic properties were demonstrated in Rivest's subsequent work in 1978. The security of the RSA encryption scheme is based on the difficulty of factoring the product of two large prime numbers. The algorithmic implementation of RSA is as follows:

Key generation: For two large prime numbers p and q (if p and q are too small, they can be easily cracked by brute force), let $n = pq$ and $L = \text{lcm}(p-1, q-1)$, where lcm denotes the least common multiple of $p-1$ and $q-1$. Select an integer $e \leq L$ such that $\text{gcd}(L, e) = 1$ (gcd generates the greatest common divisor of the two inputs). Calculate d based on $(d \times e) \bmod L = 1$. Then, the public key is $pk = (n, e)$, and the private key is $sk = d$.

Encryption: For any plaintext $0 \leq m < n$ in plaintext space M , the encrypted ciphertext is

$$c = E_{pk}(m) = m^e \bmod n \quad \forall m \in M \quad (3.13)$$

Decryption: For any ciphertext c , decryption using the key c yields the plaintext

$$m = D_{sk}(c) = c^d \bmod n \quad (3.14)$$

From the implementation process of the RSA algorithm, it is clear that its encryption and decryption operations form reversible mappings.

Homomorphic properties of the RSA scheme: For $m_1, m_2 \in M$,

$$\begin{aligned} E(m_1) * E(m_2) &= (m_1^e \bmod n) \times (m_2^e \bmod n) \\ &= (m_1 \times m_2)^e \bmod n = E(m_1 \times m_2) \end{aligned} \quad (3.15)$$

The homomorphism of the RSA encryption scheme indicates that $E(m_1) \times E(m_2)$ can be calculated directly from $E(m_1)$ and $E(m_2)$

without the need for decryption. Moreover, the RSA scheme only supports multiplicative homomorphism and not additive homomorphism.

Example: For simplicity, we choose two small prime numbers $p = 17$ and $q = 19$. Hence, $n = p \times q = 323$ and $L = lcm(16, 18) = 144$. By $gcd(e, 144) = 1$, we know that $e = 5$ is a solution. This gives us the public key $pk = (5, 323)$. According to $d \times 5 \bmod 144 = 1$, we find that $d = 29$ is a feasible solution. Therefore, the private key $sk = (29, 323)$. Suppose the plaintext is $m = 123$, then we have

$$c = m^e \bmod n = 123^5 \bmod 323 = 225$$

Given the private key $sk = (29, 323)$ and the ciphertext $c = 225$, we can correctly decrypt to get the plaintext $m = 123$:

$$m = c^d \bmod n = 225^{29} \bmod 323 = 123$$

Paillier encryption scheme: In 1999, Paillier proposed an encryption scheme based on the composite residuosity problem (CRP) in his paper. The corresponding algorithm description is as follows:

Key generation: For two large prime numbers p and q satisfying $gcd(pq, (p-1)(q-1)) = 1$ (constraining the lengths of p and q to be equal), compute $n = pq$ and $\lambda = lcm(p-1, q-1)$. Choose a random integer $g \in \mathbb{Z}_{n^2}^*$ such that $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$ exists, where the function L is defined as $L(u) = (u-1)/n$ for any u belonging to $\mathbb{Z}_{n^2}^*$. The generated public and private keys are $pk = (n, g)$ and $sk = (\lambda, \mu)$, respectively.

Encryption: For plaintext m , randomly choose $0 < r < n$ such that r is coprime with n and obtain the corresponding ciphertext

$$c = E(m) = g^m r^n \bmod n^2 \quad (3.16)$$

Decryption: For a suitable ciphertext $c < n^2$, decrypt it using the following formula:

$$D(c) = L(c^\lambda \bmod n^2) \times \mu \bmod n = m \quad (3.17)$$

From the implementation of the Paillier algorithm, it can be seen that its encryption and decryption are reversible mappings.

Homomorphic properties of the Paillier scheme: For $m_1, m_2 \in M$,

$$\begin{aligned} E(m_1) \times E(m_2) &= (g^{m_1} r_1^n \bmod n^2) \times (g^{m_2} r_2^n \bmod n^2) \\ &= g^{m_1+m_2} (r_1 \times r_2)^n \bmod n^2 = E(m_1 + m_2) \end{aligned} \quad (3.18)$$

From Equation (3.18), it can be seen that the Paillier scheme supports additive homomorphism.

Example: For simplicity, we choose two small prime numbers $p = 7$, $q = 11$. Then we have $n = p * q = 77$, $n^2 = 5929$, $\lambda = 30$. If we take $g = 5652$, then $\mu = 51^{-1} = 74 \bmod 77$. Suppose the plaintext is $m = 42$, and the random number $r = 23$, then we have

$$\begin{aligned} c &= g^m r^n \bmod n^2 = (5652)^{42} \times (23)^{77} \bmod 5929 \\ &= (4019) \times (606) \bmod 5929 = 4624 \end{aligned}$$

Given the private key $sk = (23, 59)$ and the ciphertext $c = 4624$, we can correctly decrypt to get the plaintext $m = 42$:

$$\begin{aligned} m &= L(c^\lambda \bmod n^2) \times \mu \bmod n = L(5652^{30} \bmod 5929) \times \mu \bmod 77 \\ &= L(3928) \times 74 \bmod 77 = 42 \end{aligned}$$

Other PHE schemes: In their 1982 paper, Goldwasser and Micali proposed the GM scheme, the security of which is based on the difficulty of the QRP. A drawback of this scheme is that it can only process 1 bit of information per encryption. To address this shortcoming, Benaloh (1994) proposed an improved probabilistic HE system based on the higher-order residual problem, building upon the GM scheme. Additionally, in 1998, Okamoto and Uchiyama (1998) introduced a novel PHE scheme, enhancing computational performance by changing the encryption set in previous HE schemes. In the same year, Naccache and Stern presented another PHE scheme that only altered the decryption algorithm in the Benaloh scheme, yet this modification improved the computational efficiency of the Benaloh scheme. Similarly, Damgard-Jurik (DJ) proposed an extended and improved version of Paillier in 2001. Kawachi *et al.* (2007) introduced a scheme featuring additive homomorphic properties based on the

difficulty of the underlying lattice problem, but the homomorphism in this scheme is pseudo-homomorphism. Pseudo-homomorphism is an algebraic property that still allows homomorphic operations on the ciphertext but introduces a very small decryption error during decryption.

3.2.3.4 Somewhat homomorphic encryption (SWHE)

Before the proposal of FHE in 2009, several useful SWHE schemes were put forward. Here, we mainly focus on the foundations of SWHE, rather than detailing numerous specific schemes. This is because many SWHE schemes can be transformed into FHE schemes using Gentry's method.

BGN encryption scheme: Prior to 2005, the homomorphic properties of all proposed encryption systems were limited to either addition or multiplication operations. The Boneh-Goh-Nissim (BGN) scheme, proposed by Boneh in his paper in 2005, came closest to realizing FHE. By keeping the ciphertext size constant, it supports any number of addition operations and a single multiplication operation. In 2010, Van Dijk *et al.* proposed an improved version, the GHV scheme, whose security is based on the hardness of error-correcting problems. The BGN scheme is described in detail in the following.

Key generation: Given parameter tuple q_1, q_2, G, G_1, e , where G and G_1 are groups of order $n = p_1 q_1$ and $e : G \times G \rightarrow G_1$ is a bilinear map. If g and u are generators of group G , let $h = u^{q_2}$, then h is a random generator of the q_1 -order sub-group of G . The public key is then represented as (n, G, G_1, e, g, h) and the private key is q_1 .

Encryption: Given a plaintext m and randomly selected r from the set $\{0, 1, \dots, n-1\}$, the ciphertext can then be expressed as

$$c = E(m) = g^m h^r \mod n \quad (3.19)$$

Decryption: Given the ciphertext c and private key q_1 , compute $c^1 = c^{q_1} = (g^m h^r)^{q_1} = (g^{q_1})^m$ and $g^1 = g^{q_1}$, then the decryption algorithm $D(\cdot)$ is

$$m = D(c) = \log_{g^1} c^1 \quad (3.20)$$

Homomorphic properties of the BGN scheme: For given $m_1, m_2 \in M$, let $g_1 = e(g, g)$, $h_1 = e(g, h)$, then the orders of g_1

and h_1 are n and h_1 , respectively, and there exists an integer α such that $h = g^{\alpha q^2}$. For plaintexts m_1 and m_2 , and their corresponding ciphertexts $c_1 = E(m_1)$ and $c_2 = E(m_2)$, we have

$$\begin{aligned} c &= e(c_1, c_2)h_1^r = e(g^{m_1}h_1^{r_1}, g^{m_2}h_1^{r_2})h_1^r \\ &= g_1^{m_1 m_2} h_1^{m_1 r_2 + r_2 m_1 + \alpha q_2 r_1 r_2 + r} = g_1^{m_1 m_2} h_1^{r^1} \end{aligned} \quad (3.21)$$

From Equation 3.21, it can be seen that since r^1 and r are uniformly distributed, m_1 and m_2 can be recovered from the ciphertext. However, the current ciphertext c is in group G_1 and not in group G (so it cannot maintain the ciphertext format), thus it can only support a single homomorphic multiplication operation. However, ciphertext generated in group G_1 can undergo unlimited homomorphic addition operations.

Example: The construction of an instance of this scheme involves the complex mathematical concept of bilinear mapping, which can be constructed using Tate pairings or Weil pairings on a superelliptic curve over a field. Due to the complexity of its construction, readers are advised to refer to the paper by Yue *et al.* in 2010.

Other SWHE schemes: The first SWHE scheme, known as the Polly Cracker, was proposed by Fellows *et al.* (1994). This scheme enables homomorphic addition and multiplication operations on ciphertext. However, the size of the ciphertext increases exponentially with the number of homomorphic operations, especially multiplication. To address this issue, more efficient improved versions of the Polly Cracker were proposed by Levy *et al.* (2004) and Van Ly *et al.* (2006), but studies have shown that these schemes are susceptible to attacks. In 2011, Albrecht introduced a noisy version of the Polly Cracker, in which homomorphic addition does not increase the size of the ciphertext, but multiplication causes the size of the ciphertext to increase quadratically.

3.2.3.5 Fully homomorphic encryption (FHE)

A scheme that allows for unlimited evaluations, including various operations on encrypted data, with the output remaining in the ciphertext space, is known as a FHE scheme. The concept was first proposed by Rivest *et al.* (1978) and has received a lot of attention ever since. However, this open problem was not solved until

2009, when Craig Gentry of IBM, for the first time, constructed a FHE scheme based on ideal lattices, which was detailed in his doctoral dissertation. This milestone greatly propelled the development of HE. However, from a practical application perspective, the scheme has many shortcomings, such as high computational cost and difficulty in implementing complex mathematical concepts. Since then, a series of optimized FHE schemes have been proposed, all based on the computational difficulty of lattice problems.

A lattice is a linear combination of a set of unrelated vectors (base vectors), $b_1, b_2, \dots, b_n$. Lattice L can be expressed as

$$L = \sum_{i=1}^n \vec{b}_i * v_i, \quad v_i \in \mathbb{Z} \quad (3.22)$$

where any vector $b_1, b_2, \dots, b_n$ is called a basis of the lattice L . For a given lattice, it has infinitely many bases (the base is not unique).

Before Gentry's pioneering work, there was relevant work noticing lattice-based cryptography, especially its beautiful properties in post-quantum cryptography, such as proof of security, effective implementation, and simplicity of concepts. In recent years, another popular lattice-related problem is Learning With Errors (LWE), especially after it became the basis for constructing FHE schemes. In 1998, Hoffstein *et al.* proposed one of the most important works in lattice-based encryption systems. He proposed a new public-key encryption scheme, whose security is based on the Shortest Vector Problem (SVP) in lattices. In the SVP problem, given a set of bases of the lattice, the goal is to find the shortest non-zero vector in the lattice.

After Gentry's work, lattices have become even more popular in cryptographic research. First, there are those like Smart *et al.* (2010) who only focus on Gentry's 2009 FHE scheme based on ideal lattices. There are also schemes like the integer-based FHE scheme introduced by Van Dijk *et al.*, which is based on the approximate GCD problem. The main motivation behind this scheme is its conceptual simplicity. On this basis, in 2011, researchers like Brakerski proposed a scheme based on the difficulty of Ring Learning with Errors (RLWE), which has decent efficiency. Finally, in 2012, Lopez proposed an FHE scheme similar to Number Theory Research Unit (NTRU) because it has good efficiency and standardization characteristics. NTRU encryption is an old, strong standard lattice-based encryption scheme, and its homomorphic properties were only

realized recently. Next, we detail the FHE schemes based on ideal lattices and LWE and briefly introduce other schemes.

FHE scheme using ideal lattices: The first scheme proposed by Gentry in his doctoral thesis was a GGH-style encryption scheme, where GGH was originally proposed by Goldreich *et al.* (1997). However, Gentry used a double layer rather than a single layer of noise to encrypt the information within the GGH cryptosystem. In fact, Gentry’s breakthrough work started with SWHE schemes based on ideal lattices. As previously mentioned, SWHE schemes only allow a limited number of homomorphic operations on ciphertexts. Beyond a certain number of operations, the decryption function will not be able to correctly recover the message from the ciphertext. To convert a noisy ciphertext into a suitable ciphertext, the noise in the ciphertext must be reduced. For this, Gentry utilized methods called “squashing” and “bootstrapping” to produce a new ciphertext, which allowed for multiple homomorphic operations. By repeatedly applying these methods, any SWHE scheme can be transformed into a FHE scheme.

Initially, Gentry designed HE schemes using rings and ideals without lattices, where an ideal refers to a special subset of a ring, such as the set of even numbers. As a result, every ideal used in the scheme could be represented by a ring. The description of the SWHE scheme based on ideals and rings proposed by Gentry is as follows:

Key generation: Given a ring R and a set of basis B_I for the ideal I , the algorithm $\text{IdealGen}(R, B_I)$ generates a pair of basis (B_J^{sk}, B_J^{pk}) where the $\text{IdealGen}(\cdot)$ algorithm outputs the basis of the public and private keys over the ideal ring (represented by the basis B_I), such that $I + J = R$. The $\text{Samp}(\cdot)$ algorithm is used to sample from the coset of a given ideal, where the coset can be obtained by translating the ideal. Eventually, the generated public key is $(R, B_I, B_J^{pk}, \text{Samp}(\cdot))$, and the generated private key is B_J^{sk} .

Encryption: For random vectors $\vec{r}$ and $\vec{g}$, select a basis B_{pk} of the public key from the ideal lattice L , then the plaintext $\vec{m} \in \{0, 1\}^n$ is encrypted as follows:

$$\vec{c} = E(\vec{m}) = \vec{m} + \vec{r} \cdot B_I + \vec{g} \cdot B_J^{pk} \quad (3.23)$$

where B_I is a basis of the ideal lattice L and $\vec{m} + \vec{r} \cdot B_I$ is referred to as the noise parameter.

Decryption: The ciphertext can be decrypted using the key (basis) B_J^{sk} as follows:

$$\vec{m} = \vec{c} - B_J^{sk} \cdot \left\lfloor \left(B_J^{sk} \right)^{-1} \cdot \vec{c} \right\rfloor \bmod B_I \quad (3.24)$$

where $\lfloor \cdot \rfloor$ is the “round to the nearest integer” function, returning the nearest integer to the vector coefficient.

Additive homomorphic property: For plaintext vectors $\vec{m}_1, \vec{m}_2 \in \{0, 1\}^n$, we have

$$\begin{aligned} \vec{c}_1 + \vec{c}_2 &= E(\vec{m}_1) + E(\vec{m}_2) \\ &= \vec{e}_1 + \vec{e}_2 + (\vec{e}_1 + \vec{g}_2 + \vec{e}_2 + \vec{g}_1 + \vec{g}_1 + \vec{g}_2) \cdot B_J^{pk} \end{aligned} \quad (3.25)$$

From the above, we know that $\vec{c}_1 + \vec{c}_2$ remains in the ciphertext space and the format remains unchanged. Moreover, to decrypt the ciphertext, you only need to calculate $(\vec{c}_1 + \vec{c}_2) \bmod B_J^{pk}$; when the noise amount of the ciphertext is less than $B_J^{pk}/2$, the above calculation result is equivalent to $\vec{m}_1 + \vec{m}_2 + (\vec{r}_1 + \vec{r}_2) \cdot B_I$. At this point, the decryption algorithm can correctly recover $m_1 + m_2$ from the ciphertext.

Multiplicative homomorphic properties: Similar to the additive homomorphic properties, let $\vec{e} = \vec{m} + \vec{r} \cdot B_I$, we have

$$\begin{aligned} \vec{c}_1 \times \vec{c}_2 &= E(\vec{m}_1) \times E(\vec{m}_2) \\ &= \vec{e}_1 \times \vec{e}_2 + (\vec{e}_1 \times \vec{g}_2 + \vec{e}_2 \times \vec{g}_1 + \vec{g}_1 \times \vec{g}_2) \cdot B_J^{pk} \end{aligned} \quad (3.26)$$

where $\vec{e}_1 \times \vec{e}_2 = \vec{m}_1 \times \vec{m}_2 + (\vec{m}_1 \times \vec{r}_2 + \vec{m}_2 \times \vec{r}_1 + \vec{r}_1 \times \vec{r}_2)$. It can be verified that $\vec{c}_1 \times \vec{c}_2$ still exists in the ciphertext space. If the noise $\vec{e}_1 \times \vec{e}_2$ is sufficiently small, the plaintext $\vec{m}_1 \times \vec{m}_2$ can be correctly recovered from the ciphertext $\vec{c}_1 \times \vec{c}_2$. Note that we still call it an SWHE scheme because this scheme requires a certain amount of noise and allows only a finite number of homomorphic operations. To make the algorithm satisfy the FHE property, Gentry introduced “squashing”, which can be applied to squeezable ciphertexts, such as those with noise and small circuit depth.

Squashing: A vector set is selected that satisfies the property that the result of vector addition is the multiplicative inverse of the key.

If the ciphertext is multiplied by an element of this set, the circuit depth is squashed to a level that the scheme can handle. At this point, the ciphertext is squeezable and its security is based on the hardness assumption of the SSSP.

Example: Due to its conceptual complexity (involving ideals, lattice mathematical concepts, etc.), it is difficult to give a simple example. Readers are kindly requested to read Gentry’s doctoral thesis or other related materials for more information.

A scheme is called “bootstrappable” if it can run its own decryption algorithm. First, the ciphertext is converted into bootstrappable ciphertext using the squashing technique. Then, a new ciphertext can be obtained through the bootstrapping process.

Bootstrapping: Bootstrapping refers to the process of obtaining a new ciphertext corresponding to the original text from the noisy ciphertext through a redecryption process. The working principle is as follows: First, assume there are two pairs of different public and private keys $(pk1, sk1)$, $(pk2, sk2)$, where the private key is only held by the user and the public key is shared by the user and the server. If the server knows that $c = Enc_{pk1}(m)$, the key is sent to the server in the form of ciphertext. As the above SWHE scheme can homomorphically execute its own decryption algorithm, the noisy ciphertext can be homomorphically decrypted with $Enc_{pk1}(sk1)$. Subsequently, the result is encrypted with a different public key $pk2$, $Enc_{pk2}(Dec_{sk1}) = Enc_{pk2}(m)$. Since it is assumed that the scheme is semantically secure, an attacker cannot distinguish this result from the ciphertext corresponding to 0. The final ciphertext can be encrypted using the key $sk2$ which is solely owned by the user, i.e., $Enc_{pk2}(Dec_{sk1}(c)) = Enc_{pk2}(m)$.

In essence, the process of homomorphic decryption of a noisy ciphertext removes the noise in the ciphertext and then introduces new but smaller noise in the encryption process. In this process, the ciphertext appears as if it has been re-encrypted. Further homomorphic operations can be performed on this “new” ciphertext before it reaches the threshold point again (e.g., $B_J^{pk}/2$ in the method based on ideal lattices). However, Gentry’s bootstrapping method significantly increases computational costs, hindering it from becoming a practical FHE scheme. In short, by constructing an SWHE scheme, reducing the circuit depth of the decryption algorithm through the squashing

method, and obtaining a new ciphertext through bootstrapping, we have completed the creation of an FHE scheme. Therefore, we can repeatedly apply the bootstrapping process to perform an unlimited number of operations on the ciphertext, thus successfully obtaining an FHE scheme.

Subsequently, a series of research attempts were made to improve the original scheme proposed by Gentry. In 2010, Gentry proposed a new key generation algorithm, which improved its security. Based on Gentry's scheme, Smart *et al.* (2010) proposed a new FHE scheme. This scheme uses relatively smaller ciphertexts and keys without compromising security. In addition, some works are dedicated to studying efficient key generation algorithms to improve the efficiency of FHE.

FHE scheme over the integers: A year after Gentry proposed his original scheme, in 2010, the DGHV scheme was proposed by Dijk–Gentry–Halevi–Vaikuntanathan. The security of this scheme is based on the difficulty of the AGCDP. The biggest advantage of this scheme is the simplicity of its concept; its symmetric version may be one of the simplest FHE schemes. The details are described as follows:

Key generation: For a given security parameter λ , a random odd integer of η bits length is generated.

Encryption: For given large primes p and q , select a number r much smaller than p . The plaintext $m \in \{0, 1\}$ is encrypted in the following way:

$$c = E(m) = m + 2r + pq \quad (3.27)$$

where p is the hidden private key, c is the ciphertext, and $m + 2r$ is referred to as noise.

Decryption: The ciphertext is decrypted in the following way:

$$m = D(c) = (c \bmod p) \bmod 2 \quad (3.28)$$

From the decryption process, it can be seen that only when $m + 2r < p/2$ does $c \bmod p = m + 2r$. Further using $m + 2r \bmod 2$ can recover m . Therefore, only when the noise is less than $p/2$, can the decryption algorithm work properly, which limits the number of operations that can be performed on the ciphertext. For this reason, Van dijk *et al.* used squashing and bootstrapping to make the

method a full FHE scheme. Its homomorphic properties are described as follows:

Additive homomorphic property: For plaintext vectors $\vec{m}_1, \vec{m}_2 \in \{0, 1\}^n$, we have

$$\begin{aligned} E(m_1) + E(m_2) &= m_1 + 2r_1 + pq_1 + m_2 + 2r_2 + pq_2 \\ &= (m_1 + m_2) + 2(r_1 + r_2) + (q_1 + q_2)q \end{aligned} \quad (3.29)$$

where p is the key. Clearly, the output is in the ciphertext space. Furthermore, the noise of the sum of the ciphertexts is equal to the sum of the noise, and when the noise $|m_1 + 2r_1 + m_2 + 2r_2| < p/2$, it can be decrypted correctly.

Multiplicative homomorphic property: For plaintext vectors $\vec{m}_1, \vec{m}_2 \in \{0, 1\}^n$, we have

$$\begin{aligned} E(m_1)E(m_2) &= (m_1 + 2r_1 + pq_1)(m_2 + 2r_2 + pq_2) \\ &= m_1m_2 + 2(m_1r_2 + m_2r_1 + 2r_1r_2) + kp \end{aligned} \quad (3.30)$$

It can be seen that the result retains the format of the original ciphertext and has homomorphic properties. In particular, the noise of the product of the ciphertexts is equal to the product of the original ciphertext noise. Thus, in this scheme, the noise increases exponentially with the number of multiplicative operations, limiting the number of multiplications.

Example: Let $p = 11$, $q_1 = q_2 = 5$, $m_1 = 0$, and $m_2 = 1$; Choose $r_1 = -1$, $r_2 = 1$. According to the above encryption scheme, we can get

$$\begin{aligned} c_1 &= m_1 + 2r_1 + pq_1 = 0 - 2 \times 1 + 11 \times 5 = 53; \\ c_2 &= m_2 + 2r_2 + pq_2 = 1 + 2 \times 1 + 11 \times 5 = 58 \end{aligned}$$

The decryption process is

$$c_1 \bmod p = -2, \quad -2 \bmod 2 = 0; \quad c_2 \bmod p = 3, \quad 3 \bmod 2 = 1$$

From the above process, we know that the noise of c_1 is -2 and the noise of c_2 is 3 , both within the range $(-p/2, p/2) = (-11/2, 11/2)$, so both can be decrypted correctly. Similarly, the noise of $c_1 + c_2$ is 5 and still within the decodable range. However, for $c_1 \times c_2$, we have

$c_1 \times c_2 = 53 \times 58 = 3074$; however, $3074 \bmod 11 = 5$, $5 \bmod 2 = 1 \neq m_1 \times m_2$. The error occurred because the noise of $c_1 \times c_2$ is -6 , which is not in $(-11/2, 11/2)$.

The above algorithm is a symmetric FHE scheme. Van also proposed its asymmetric version in 2010. As can be seen from the description of the scheme, its concept is very simple. However, the simplicity of the concept comes with a significant computational cost, making the scheme very inefficient. Therefore, subsequent research has been dedicated to improving the efficiency of this method. Coron conducted a series of studies from the perspective of reducing the length of the public key and achieved improvements from $(O(\lambda^{10}) \rightarrow O(\lambda^7))$, $(O(\lambda^7) \rightarrow O(\lambda^5))$, and $(O(\lambda^5) \rightarrow O(\lambda^3))$. Importantly, in 2014, Chen *et al.* (2014a), proposed a method for batch FHE, which greatly improved the efficiency of the method. This batch encryption method allows multiple ciphertexts to be packaged into one ciphertext. In addition, other related work has improved this method from various perspectives.

FHE schemes based on Fault-tolerant learning: Fault-tolerant learning is considered one of the most challenging problems to resolve in practical scenarios, even with post-quantum algorithms. It was first introduced by Regev (2009). Regev managed to translate the complexity of the worst-case lattice problems, such as the SVP, into a fault-tolerant learning problem. This implies that if an algorithm can solve the fault-tolerant learning problem in a reasonable timeframe, it could likewise solve the SVP effectively. Consequently, this approach has become one of the most notable solutions in post-quantum cryptography. In 2011, Brakerski and Vaikuntanathan proposed a SWHE scheme based on ring fault-tolerant learning, bringing the FHE scheme a significant step closer to practical application. Their method exploited the efficiency characteristics of ring fault-tolerant learning. Using the squashing and bootstrapping techniques proposed by Gentry, they transformed this scheme into an FHE scheme. The encryption scheme based on the LWE problem is also referred to as the second-generation HE framework. Before introducing the algorithm's process, let's introduce some symbolic definitions.

The inner product of vectors a and b is denoted by $\langle a, b \rangle$. The notation $d \stackrel{\$}{\leftarrow} \mathcal{D}$ signifies that d is a random element drawn from distribution D . $\mathbb{Z}[x]/(f(x))$ represents the ring of all polynomials'

modulo $f(x)$, and if the coefficients of a polynomial $f(x)$ belong to $\mathbb{Z}_q$, it can be denoted as $\mathbb{Z}_q[x]/(f(x))$. χ represents the error distribution of ring R_q . With these symbolic definitions in place, the symmetric version of the scheme can be described as follows:

Key generation: A key element from the error distribution is randomly selected as the secret key, denoted as $s \xleftarrow{\$} \chi$. This key can then be represented as $\vec{s} = (1, s, s^2, \dots, s^D)$, where D is an integer.

Encryption: For given large prime numbers p and q , select a number r that is much smaller than p . The plaintext $m \in \{0, 1\}$ is encrypted as follows:

$$c = E(m) = m + 2r + pq \quad (3.31)$$

where p is the hidden secret key and c is the ciphertext.

Decryption: Given the ciphertext $\vec{c}$, it can be decrypted as follows:

$$m = \langle \vec{c}, \vec{s} \rangle \pmod{t} \quad (3.32)$$

The decryption algorithm can only execute normally when $\langle \vec{c}, \vec{s} \rangle \pmod{t} < q/2$.

Example: This scheme is similar to the DGHV scheme in principle, but it involves complex mathematical concepts such as error distribution in rings and cosets. No illustrative examples are provided here. Interested readers are advised to read related literature, such as the paper by Juyan Li (2018).

For the symmetric scheme, it is also necessary to generate random pairs $(a, as + te)$. The homomorphic properties are consistent with the previous two schemes and are not reiterated here. Like the previous SWHE schemes, the authors used squashing and bootstrapping methods to make the scheme fulfill FHE characteristics. Later, Brakerski and Vaikuntanathan proposed another SWHE scheme based on the standard LWE problem but employed relinearization technology. This technique can transform overly long ciphertexts into normal lengths. There are also schemes that don't require compression or bootstrapping, but these are typically only capable of handling circuits of limited depth. More recently, Gentry *et al.* (2013) proposed a simple, efficient, attribute-based FHE scheme in 2013. This scheme is straightforward and extremely fast because it uses

approximate eigenvalue methods instead of relinearization technology. This scheme restricts the size of certain parameters, thus preserving the format of the ciphertext. In addition to schemes based on LWE, some research is dedicated to designing faster compression algorithms, accelerating homomorphic operations, or creating new efficient SWHE algorithms like the NTRU algorithm.

3.2.3.6 Application of homomorphic encryption

Due to the superior technical capabilities of HE in protecting data privacy and security, and increasing concerns over data privacy and security in the application of ML models, many researches have begun to apply HE to ML models to achieve technically enforced privacy protection. Considering the importance and representativeness of logistic regression in ML, we focus on an example of applying HE to logistic regression, referred to as “secure logistic regression”.

Logistic regression is a representative classification method in supervised ML. Consider the scenario in Figure 3.7 and expand it to multiple users. Suppose a scenario where multiple users N jointly train a logistic regression model with a server, where the server is responsible for receiving, storing, sharing, and processing data. In order to design secure logistic regression in this scenario, it is first necessary to represent logistic regression as a form that can be “homomorphically computed” through function approximation. This is because the original logistic regression expression contains a logarithm function, which cannot be directly computed by HE technologies that support at most addition and multiplication. On this basis, we illustrate how to use the Paillier scheme to solve this model.

Introduction to logistic regression: Given a dataset $\{x^i, y^i\}_{i=1}^N$, where $x^i = \{1, x_1^i, \dots, x_d^i\} \in \mathbb{R}^{d+1}$, $y^i = \{0, 1\}$. The objective J is defined as follows:

$$J(\theta) = \frac{\lambda}{2N} \sum_{j=1}^d \theta_j^2 + \frac{1}{N} \sum_{i=1}^N [-y^i \log(h_\theta(x^i)) - (1 - y^i) \log(1 - h_\theta(x^i))] \quad (3.33)$$

where $\theta = (\theta_0, \theta_1, \dots, \theta_d) \in \mathbb{R}^{d+1}$ is the model parameters and $h_\theta(x) : \mathbb{R}^{d+1} \rightarrow \mathbb{R}$ is the sigmoid function, which is given by

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^T x)} = \frac{1}{1 + \exp\left(-\sum_{j=0}^d \theta_j x_j\right)} \quad (3.34)$$

The purpose of model training is to optimize the objective function J and find the optimal solution satisfying $\theta^* = \operatorname{argmin}_\theta J(\theta)$.

Homomorphically computable logistic regression: Since the original logistic regression model contains logarithmic operations, it cannot be directly computed by existing HE schemes. To solve this problem, one can approximate the logarithm function to obtain a homomorphically computable logistic regression model. For example, using a k -degree polynomial, the logarithm function can be approximated by $\log\left(\frac{1}{1+\exp(u)}\right) \approx \sum_{j=0}^k a_j u^j$, then we get

$$\log(1 - h_\theta(x)) \approx \sum_{j=0}^k a_j (\theta^T x)^j \quad \log(h_\theta(x)) \approx \sum_{j=0}^k (-1)^j a_j (\theta^T x)^j \quad (3.35)$$

Substituting the approximation terms from Equation 3.35 into Equation 3.33, we obtain the approximation of the objective function J as

$$J_{\text{approx}}(\theta) = \frac{\lambda}{2N} \sum_{j=1}^d \theta_j^2 + J_{\text{approx}}^*(\theta) \quad (3.36)$$

where $J_{\text{approx}}^*(\theta) = \frac{1}{N} \sum_{i=1}^{N_{\text{data}}} \sum_{j=1}^k (y^i - y^i(-1)^j - 1) a_j (\theta^T x^i)^j - a_0$. Also, expand $(\theta^T x^i)^j$, $i = 1, \dots, n$, as

$$(\theta^T x)^j = \sum_{r_1, \dots, r_j=0}^d (\theta_{r_1} \dots \theta_{r_j}) (x_{r_1} \dots x_{r_j}) \quad (3.37)$$

Let $A_{j,r_1, \dots, r_j} = \sum_{i=1}^N (y^i - y^i(-1)^j - 1) (x_{r_1}^i \dots x_{r_j}^i)$, and set k to 2 (as $k = 2$ is a common choice due to its reasonable computational cost and good approximation effect), then we have

$$J_{\text{approx}}(\theta) = \frac{1}{N} \sum_{j=1}^2 \sum_{r_1, \dots, r_j=0}^d a_j (\theta_{r_1} \dots \theta_{r_j}) A_{j,r_1, \dots, r_j} - a_0 \quad (3.38)$$

where $A_{1,r_1} = \sum_{i=1}^N (2y^i - 1)x_{r_1}^i$ and $A_{2,r_1,r_2} = \sum_{i=1}^N (-1)(x_{r_1}^i x_{r_2}^i)$. If approximations are made using the Taylor series, we have

$$a_0 = -\log 2, \quad a_1 = -0.5, \quad a_2 = -0.125 \quad (3.39)$$

Other approximation methods may be employed, such as minimizing error under quadratic polynomials. Aono *et al.* (2016) demonstrated through their paper that the two approximation methods performed similarly and both yielded good results.

Secure computation of the model: As illustrated in Figure 3.7, each user begins by encrypting their data. The collective data from all users form the dataset. For sample i , it is required to calculate the real numbers $a_{1,r_1}^i = (2y^i - 1)x_{r_1}^i \in \mathbb{R}$ and $a_{2,r_1,r_2}^i = (-1)(x_{r_1}^i x_{r_2}^i) \in \mathbb{R}$ for all $0 \leq r_1, r_2 \leq d$. This process is performed by the data owner and needs to calculate $n_d = \frac{(d+1)(d+4)}{2}$ numbers. For the sake of brevity, we denote all n_d numbers that need to be calculated on sample i as dat^i .

We adopt an additive homomorphic scheme to encrypt dat^i , resulting in ciphertext CT:

$$CT^i = \mathbf{E}_{pk} \left(\{a_{1,r_1}^i\}_{0 \leq r_1 \leq d}, \{a_{2,r_1,r_2}^i\}_{0 \leq r_1, r_2 \leq d} \right) \quad (3.40)$$

where $\mathbf{E}_{pk}$ represents the required encryption algorithm.

After the server receives ciphertexts from all users, it computes $CT = \sum_{i=1}^N CT^i$ in a homomorphic manner. Following this calculation, the server feeds the ciphertext CT back to the users. Users decrypt the ciphertext using a key, yielding

$$\sum_{i=1}^N a_{1,r_1}^i \in \mathbb{R}, \quad \sum_{i=1}^N a_{2,r_1,r_2}^i \in \mathbb{R} \quad (3.41)$$

where $0 \leq r_1, r_2 \leq d$. Once the plaintext is decrypted, users can compute θ^* using the analytical solution.

During the inference phase, suppose that $\theta^* = (\theta_0^*, \dots, \theta_d^*)$ is publicly known, and the input data $x = (x_1, \dots, x_d)$ for prediction is encrypted. According to Equation (3.34), we know

$$h_{\theta^*}(x) = \frac{1}{1 + \exp \left(-\theta_0^* - \sum_{j=1}^d \theta_j^* x_j \right)} \quad (3.42)$$

To compute $h_{\theta^*}(x)$, it is necessary to first know $\sum_{j=1}^d \theta_j^* x_j$. If the data to be predicted and by the user is encrypted, then $\sum_{j=1}^d \theta_j^* \mathbf{E}_{pk}(x_j)$ needs to be calculated, then decrypted to obtain the required parameters for prediction.

Next, we introduce the steps to implement secure computation of this model using the Paillier scheme. Detailed implementation can be extended from the example in the Paillier scheme.

Secure logistic regression based on the Paillier scheme: Next, we introduce the implementation details of the previous section using the Paillier scheme. Assume that $\text{PaiEnc}_{pk}(\cdot)$ represents the public key encryption algorithm under the Paillier scheme. Given $n = pq$, if the size of the plaintext space is more than $\log_2 n \geq 2,048$ bits, then multiple integers $dat_1, \dots, dat_t$ with precision (a real number $0 \leq r < 1$ can be represented as an integer in the form of $\lfloor r \cdot 2^{\text{prec}} \rfloor$ in binary, similar to the decimal representation, for example, $0.564 = 564 \cdot 10^{-3}$, thus representing precision bit as 564 and precision as 10^{-3}) can be packed into Paillier ciphertext in the following way:

$$\text{PaiEnc}_{pk}(\overbrace{[\underbrace{[dat_1 \ 0_{\text{pad}}]}_{\text{prec+pad bits}} \cdots \underbrace{[dat_t \ 0_{\text{pad}}]}_{\text{prec+pad bits}}]}^{\lfloor \log_2 n \text{ bits} \rfloor}) \quad (3.43)$$

where 0_{pad} denotes zero elements padded with pad bits. This operation prevents overflow that may occur during homomorphic addition. Generally, $\text{pad} \approx \log_2 N$ because it needs N (sample size) ciphertext additions. Moreover, as the plaintext size (in bits) should be less than $\log_2 n$, $t(\text{prec} + \text{pad}) \leq \log_2 n$ must be true. From this relation, we obtain

$$t = \left\lfloor \frac{\lfloor \log_2 n \rfloor}{\text{prec} + \text{pad}} \right\rfloor \quad (3.44)$$

This formula indicates the upper bound of the precision bits that a single Paillier plaintext can contain. Hence, the packing method can be used to encrypt approximately $\lfloor \log_2 n \rfloor / (\text{prec} + \text{pad})$ real numbers with precision prec within the range $[0, 1)$, which can support about 2^{pad} ciphertext addition operations.

From the above analysis, the number of Paillier ciphertexts sent to the server by the data owner is $\frac{n_d}{t} = \frac{n_d(\text{prec} + \text{pad})}{\lfloor \log_2 n \rfloor}$. As each Paillier ciphertext size is $2 \lfloor \log_2 n \rfloor$ bits, the communication resources

required to send ciphertexts are $\text{CommCost} = 2n_d(\text{prec} + \text{pad})$ (in bits).

In Step 3, the server needs to perform homomorphic addition operations on $\frac{n_d}{t}$ columns and N rows of ciphertexts, with a computational complexity of roughly $N \cdot \frac{n_d}{t}$. Finally, the sums of n_d/t ciphertexts obtained by row addition are fed back to the data owners by the server:

$$\begin{array}{ccc}
 & \overbrace{\hspace{10em}}^{\lfloor \log_2 n \rfloor} & \\
 \text{PaiEnc}_{pk}(\underbrace{[dat_1^1 \ 0_{\text{pad}}]}_{\text{prec} + \text{pad} \text{ bits}} \cdots \underbrace{[dat_t^1 \ 0_{\text{pad}}]}_{\text{prec} + \text{pad} \text{ bits}} \text{ bits}), \dots & & \\
 & \vdots & \\
 & + & \\
 & \vdots & \\
 & \overbrace{\hspace{10em}}^{\lfloor \log_2 n \rfloor} & \\
 \text{PaiEnc}_{pk}(\underbrace{[dat_1^i \ 0_{\text{pad}}]}_{\text{prec} + \text{pad} \text{ bits}} \cdots \underbrace{[dat_t^i \ 0_{\text{pad}}]}_{\text{prec} + \text{pad} \text{ bits}} \text{ bits}), \dots & & \\
 & \vdots &
 \end{array} \quad (3.45)$$

The user decrypts the sums of the n_d/t received ciphertexts to obtain the required values $\sum_{i=1}^N dat_1^i \in \mathbb{Z}, \dots, \sum_{i=1}^N dat_t^i \in \mathbb{Z}, \dots, \sum_{i=1}^N dat_{n_d}^i \in \mathbb{Z}$. With these data, the desired ML model can be solved, and the process does not involve information leakage, realizing secure logistic regression.

Other models and implementations: Beyond logistic regression, HE has been applied to other statistical ML models, such as linear regression, K -means clustering, and PCA, among others. Additionally, it has found use in several specific ML applications. Interested readers are encouraged to seek further information from related literature.

3.2.3.7 Implementation of homomorphic encryption

As mentioned in the previous section, HE has found widespread applications in ML. However, as stated in Sections 3.2.3.3–3.2.3.5, due to conceptual complexity, the implementation of many HE schemes is not straightforward. Most implementations are designed for specific publications, and only a few HE libraries support reuse.

Moreover, most of these libraries and packages use C and C++ language interfaces, which can increase the learning curve for ML researchers. We briefly introduce some libraries and packages for implementing HE to help readers choose these libraries based on their specific needs. First, we introduce several HE libraries based on C and C++, followed by a HE package based on R language.

Homomorphic encryption libraries based on C/C++: Minar (2010) provided an open-source C language implementation of the Gentry scheme, “libfhe”. This implementation is based on a binary scheme, but it does have routines that allow for encryption of integers via binary representation, subsequently encrypting each binary digit to realize a binary adder algorithm. This scheme does not involve bootstrapping, and it has not been updated since its publication. “Scarab” is a C-language-based library that implements Smart’s HE scheme proposed in 2010 for integer ciphertext spaces. This library allows for the encryption of binary messages, and although it supports addition (XOR) and multiplication (AND) operations, computation still has many restrictions. Similarly, this library has not been updated for a long time. “HElib” is a C++-implemented HE library. The initial algorithm details of this library can be found in Halevi’s paper published in 2014, and the library is still under maintenance and updates.

Another HE library, “HElib” (2019 version), is also developed in C++ and is continually updated and improved. It is regarded as one of the most mature and feature-complete HE libraries globally, suitable for relatively professional researchers and practitioners. Its comprehensive functions are too extensive to showcase fully in this article, so we recommend that readers review its technical documentation independently. Recently, IBM (2019) has open-sourced FHE libraries for Android, iOS, MacOS, and Linux platforms, namely IBMHE. For example, the open-source library on the Linux platform has been packaged into a Docker container, making it easier to get started with and use FHE technology. Moreover, it provides many code examples that beginners can follow to learn step-by-step. For these mature and comprehensive HE libraries, we do not delve too much into details, and we suggest readers read their introductory documents based on their specific needs.

Homomorphic encryption package based on R: The HE libraries mentioned above are all based on languages such as C and C++, which may make it difficult for some ML practitioners to quickly master these libraries. Therefore, we introduce an easy-to-use HE package based on R language that was published by Aslett *et al.* (2014). This R package provides a graphical interface, making development and testing relatively straightforward. It contains a few general-purpose functions that can be used with different encryption schemes. Its underlying implementation uses high-performance languages, such as C and C++. In addition, it supports multi-threading and can significantly improve efficiency on multi-core machines.

The R package includes several commonly used functions: (1) *Scheme function “pars”*: The first argument of this function specifies which encryption scheme to use, and it allows for modifications of parameter options in the default encryption scheme through parameters. To understand each parameter’s command in detail, help can be obtained through the function `parsHelp`. (2) *Key generation function “keygen”*: It only has one parameter object returned by `pars`. The `keygen` function generates the corresponding public and private keys based on the parameters returned by `pars`. (3) *Encryption function “enc”*: Given a public key (like the one generated by `keygen`) and plaintext, the `enc` function returns the ciphertext encrypted according to the established public key and encryption scheme. This scheme can encrypt not only a single integer but can also integer vectors and matrices defined in R language. The encryption process automatically implements multithreaded acceleration, greatly enhancing the efficiency of researchers. (4) *Decryption function “dec”*: Given a key (generated by `keygen`) and the ciphertext that needs to be decrypted (which can be in scalar, vector, matrix, etc., formats), this function can return the corresponding plaintext while maintaining the original ciphertext format, such as scalar, vector, or matrix.

Clearly, compared to the HE libraries discussed in the previous section, this R package is very user-friendly to researchers. Simple parameter settings can implement the desired HE scheme. It can effectively alleviate the difficulty for ML practitioners to implement ML models based on HE due to a lack of background knowledge in related fields, such as mathematics and cryptography.

3.2.3.8 Conclusion

In this internet-centric world, data privacy plays a more critical role than ever before. For systems that are highly sensitive to data, such as e-commerce companies, digital technology companies, banks, hospitals, and government departments, protecting user data privacy is crucial. HE is a promising direction for protecting data privacy technically because it allows any third party to operate on encrypted data without decrypting it in advance. This desirable feature has attracted a lot of attention. In this chapter, we started with the basics of HE, sorted out the mainstream directions and their related methods, and introduced representative schemes in three types of HE and their improved versions. Existing FHE schemes still need significant improvements to enhance their practicality. There is considerable room for improvement in the speed and performance of existing homomorphic encryption schemes, which is an important future research direction. With the advancement of research, we look forward to more widespread applications of HE in privacy protection and FL.

Part II

Federated Learning Algorithms

This page intentionally left blank

Chapter 4

Tree-Based Models in Vertical Federated Learning

This chapter primarily introduces tree-based models in vertical federated learning tasks. We explain the design principles and safety verification of the federated learning algorithm in depth through the example of the random forest algorithm.

Algorithm highlights: Tree-based models in this chapter consist of two algorithms based on different ensemble methods, bagging and boosting, namely the vertical federated random forest algorithm and the vertical federated gradient boosting algorithm.

4.1 Introduction to Tree-Based Models

Among the vast array of machine learning models, tree-based models can be considered as one of the most widely used models. They have two notable characteristics: (1) The model can intuitively reveal causal relationships, making the understanding of the underlying decision logic less difficult, thus showcasing a significant advantage in terms of interpretability; (2) the basic structure and decision path of the model are very concise, with low demands on computational resources, making it well received in practical applications.

Let's start by briefly reviewing decision trees. A decision tree is one of the basic tree-based models, typically using a binary tree structure to record each level's decision branches and deriving the final decision through layered inference. Decision trees can solve common

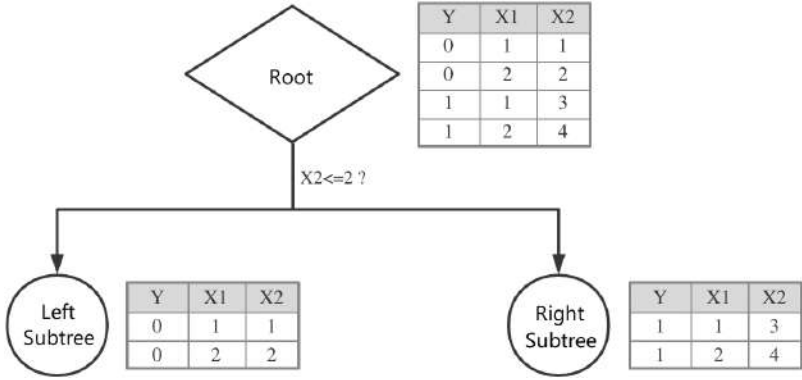


Figure 4.1. Example of decision tree (classification tree) construction.

classification or regression problems as classification trees or regression trees, respectively. Therefore, various tree-based models are primarily evolved from decision trees. The core step in decision tree modeling is to sort and enumerate features on each node waiting for a decision, as shown in Figure 4.1. Depending on the modeling problem, we can compute, minimizing cross-entropy (classification tree) or variance (regression tree) as indicators and statistically identify the most distinguishing feature and splitting point for the current sample labels as decision criteria. The samples of the current node are split according to the decision criteria, and the left and right subtrees are established to continue the iteration to the next level.

Typically, a single decision tree can only complete a machine learning task to a certain extent. On the one hand, a single decision tree often has problems with insufficient use of data. For instance, a decision tree model with a depth of 10 can only divide the data into at most 1024 different intervals, thus not generating more than 1024 different decision paths. On the other hand, single decision trees have a high probability of overfitting. The feature and splitting point with the most distinguishing ability as decision criteria are determined by the data distribution of the training set, which often results in surprisingly poor performance in the test set. Therefore, in actual machine learning tasks, we typically use the idea of ensemble learning to integrate the decision results of multiple decision trees into one model to generate the final decision. The two most commonly used ensemble methods in traditional machine learning are bagging and boosting.

In the following sections, we use the federal random forest algorithm corresponding to bagging and the federal gradient boosting algorithm corresponding to boosting as examples to introduce the design principles of tree-based models in federated learning.

4.2 Vertical Federated Random Forest Algorithm

4.2.1 Algorithm Structure

Random forest algorithm is a typical tree-based algorithm, whose principle is still based on tree structure to divide feature space. Compared to the basic form of a single decision tree, the random forest algorithm randomly selects samples and features to establish multiple independent decision trees and integrates their respective decision results through bagging to complete training and prediction. Common bagging methods are majority voting (classification tree) or averaging (regression tree). Due to the double randomness brought by samples and features, the random forest algorithm naturally has the characteristics of dimensionality reduction and sampling. It can easily handle high-dimensional or large-scale data, can maintain accuracy even in the case of feature loss, and can even balance errors for unbalanced datasets. It's a widely applicable machine learning method.

The federated random forest algorithm implementation introduced in this chapter can retain all the advantages of traditional random forest algorithm. To achieve data exchange between multiple participants, the algorithm will be divided into four sub-algorithm steps executed by different participants, as shown in Figure 4.2.



Figure 4.2. FedLearn random forest algorithm workflow.

4.2.1.1 Plaintext and ciphertext transmission

The data exchanged between multiple participants can be broadly divided into two categories: The first category includes user IDs, sample dimensions, sample numbers, etc., which do not need to be kept confidential from other participants and can be simply transmitted in plaintext; the second category includes features of samples at each participant's side and the most critical labels of the active participant's samples, which are the most privacy-sensitive contents. During the exchange, it is necessary to ensure the use of encrypted form to prevent any participant from possessing plaintext data they shouldn't have, thereby avoiding issues of privacy leakage. In this example, Paillier homomorphic encryption is used during data transmission.

4.2.2 Algorithm Description

First, let's look at the sub-process `RandomForestPhase1`, which is responsible for managing the overall algorithm workflow. The pseudocode logic is shown in Algorithm 4.1. This process needs to select the optimal splitting point to divide the feature space into two parts when the decision tree can continue to grow. If it is found that no matter how the decision tree is split, it cannot bring better classification effects, the entire process should be stopped.

The following process, `RandomForestPhase2`, will be executed on the passive side, with the filled algorithm being Algorithm 4.2. Its primary purpose is to sort the features and calculate the information gain of the decision tree's branch points. This step is a prerequisite in Algorithm 4.1 to determine whether the decision tree split can be effective.

In this context, y represents the label owned by the active side and $\langle \rangle$ denotes the homomorphic encryption operator. In this example, we use the Paillier algorithm to implement the homomorphic encryption operation, ensuring that the passive participant can only use encrypted labels. When labels are used by the active side (who originally has the authority to use labels), the step of using the homomorphic encryption operator can be skipped.

The next process, `RandomForestPhase3`, will return to the active side owning the labels for execution, with the filled algorithm being

Algorithm 4.1 Main Algorithm (Executed by the Active Side)

Input:

Sample space I ;
 Feature dimensions D ;

Output:

Decision tree

- 1: Check if the algorithm termination condition is met, such as the number of instances is less than a preset value;
 - 2: **while** Logic is true **do**
 - 3: Form the encrypted input to Algorithm 4.2 and receive return values from the passive side;
 - 4: Call Algorithm 4.3 to find the current optimal split and return corresponding values;
 - 5: **if** Termination signal is true **then**
 - 6: Break (terminate) the loop;
 - 7: **else**
 - 8: Call Algorithm 4.4 to split the instance space into I_L and I_R and create a new record $[record_{id}, server_{id}]$ (used for reference purposes);
 - 9: **end if**
 - 10: Recursively construct the left tree with I_L and the right tree with I_R
 - 11: **end while**
-

Algorithm 4.3. Its goal is to select an optimal splitting point through the information gain at each splitting point from the passive side.

In Algorithm 4.1, the process of calculating the score in line 10 is essentially the negative sum of the variances of the left and right subtrees. Since the entire information gain can be decomposed into the original variance minus the sum of the variances of the left and right subtrees, after eliminating the original variance as a common term, the optimal score can be deduced through the extremum of the sum of the left and right subtrees, which also gives us the optimal splitting point.

After finding the optimal splitting point, the process needs to execute RandomForestPhase4, which populates Algorithm 4.4 to expand the depth of the tree. Both the active and passive sides split the data

Algorithm 4.2 Sort Features and Return Aggregated Values
(Executed by the Passive Side)

Input:

Feature dimension (passive side's information) d ;
 Current node's instance space I ;
 Quantile l ;
 $\{\langle y_i \rangle\}_{i \in I}$

Output:

$\mathbf{Y} \in \mathbb{R}^{d \times l}$

- 1: **for** $k = 0 \rightarrow d$ **do**
 - 2: Propose $S_k = \{s_{k1}, s_{k2}, \dots, s_{kl}\}$ through the quantiles on feature k
 - 3: **end for**
 - 4: **for** $k = 0 \rightarrow d$ **do**
 - 5: $\mathbf{Y}_{kv} = \sum_{i \in (i | s_{k,v} \geq x_{i,k} \geq s_{k,v-1})} \langle y_i \rangle$
 - 6: **end for**
-

into two parts based on the optimal splitting point and then return to the main control flow of Algorithm 4.1.

4.2.3 Security Analysis

In federated learning, the security of data is a very important aspect to consider in algorithm design. In practice, we analyze data security from two perspectives: the security of data transmission and whether the receiving participant can infer data owned by other data sources.

4.2.3.1 Data transmission and statistics

At the beginning of training, the active side needs to homomorphically encrypt the labels of the samples and send them to the passive sides. During each splitting process, participants need to distribute the sample spaces of the left subtree and right subtree to other participants for subsequent training processes. Also, during each computation of the optimal splitting point, all passive sides need to sum up the corresponding labels according to the quantiles of the features they hold (due to homomorphic encryption, this sum remains encrypted) and send it back to the active side. After computing,

Algorithm 4.3 Find the Optimal Split (Executed by the Active Side)

Input:

Instance space of current samples I ;
 Aggregated encrypted gradient statistics from m participants
 (Output of Algorithm 4.2) $\{\mathbf{Y}^i\}_{i=1}^m$;
 Minimum information gain ϵ ;

Output:

Optimal feature k_{opt} ;
 Optimal quantile v_{opt} ;

```

1: for  $i = 0 \rightarrow m$  do
2:   // Enumerate all features
3:   for  $k = 0 \rightarrow d_i$  do
4:      $y_i \leftarrow 0$ 
5:     // Enumerate all quantiles
6:     for  $v = 0 \rightarrow l_k$  do
7:       Obtain the encrypted value  $D(\mathbf{Y}_{kv}^i)$ 
8:        $y_l \leftarrow y_l + D(\mathbf{Y}_{kv}^i)$ 
9:        $y_r \leftarrow y - y_l$ 
10:       $score \leftarrow \max(score, E[Y_l]^2 \times n_l + E[Y_r]^2 \times n_r)$  where  $n$  represents the number of samples (instances)
11:      Keep the maximum score and the corresponding  $k_{opt}, v_{opt}$ 
12:      If  $score > \epsilon + E[Y]^2 \times n$ , return  $k_{opt}, v_{opt}$ , otherwise return the termination signal
13:    end for
14:  end for
15: end for

```

the active side sends the features and quantiles of the optimal splitting point to the corresponding participants to perform the splitting operation on the node.

Table 4.1 summarizes the necessary data transmission and corresponding direction in the federated random forest algorithm, and Table 4.2 summarizes the information held by the active and passive sides during the training process.

Algorithm 4.4 Split Sample Space (Executed by the Passive Sides)

Input:

- Current node’s instance space I ;
- Selected feature k_{opt} ;
- Selected quantile v_{opt} ;

Output:

- Left subtree instance space I_L ;
 - Right subtree instance space I_R ;
 - Training records and flags;
- 1: Split I into I_L and I_R based on k_{opt} and v_{opt}
 - 2: Generate a record for the current split ($record_{id}$, $feature_{id}$, $split$ value)
-

Table 4.1. Data transmission statistics of federated random forest algorithm.

Content	Transmission direction
Encrypted labels	Active side $\rightarrow$ Passive sides
Sample space	Passive sides $\leftrightarrow$ Active side
Sum of labels based on feature quantiles	Passive sides $\rightarrow$ Active side

Table 4.2. Data held by participants in the federated random forest algorithm.

Content	Held by active side	Held by the passive side
Sample space of left and right subtrees	Yes	Yes
Number of features from each participant	Yes	No
Sum of labels based on feature quantiles	Yes	Yes
Split quantiles for individual features	Yes	Yes

4.2.3.2 Security of labels

Next, based on the data transmission and the data held by the participants in the federated random forest training process, we will analyze the security properties of the federated random forest algorithm. First, let's analyze whether the active side's labels can be detected by the passive sides.

From the perspective of labels, we can consider that the encrypted labels themselves are hard to crack due to asymmetric encryption technology. Secondly, since all leaf nodes are stored on the active side, the prediction values saved on the leaf nodes (mean for regression tree, a quantile for classification tree, such as 50% quantile for binary classification tree) will not be obtained by the passive participants. Thirdly, we need to consider whether the root node of the leaf nodes can detect the information of the leaf nodes. Assuming a parent node of two leaf nodes is stored at a passive participant, because the sample space splitting method of the child nodes is decided by this passive participant, this participant knows the sample space information of these two leaf nodes. In this case, the passive participant can guess the classification of these two child nodes, and the accuracy of its guess is determined by the purity of the child nodes. Meanwhile, since each tree in the random forest algorithm tends to overfit, the risk of leaf node classification detection on a single tree is somewhat higher compared to other algorithms. Therefore, we need certain technical means to reduce the risk of the active side's leaf nodes being detected.

4.2.3.3 Security of features

The second task to analyze is whether the distribution of features of the passive participants can be detected by the active side.

According to the information transmitted and held by the participants, we know that the active side's understanding of the passive participants' feature information is limited to the number of features, the sum of labels based on feature quantiles, and the sample space after optimal splitting. Generally, this information is insufficient to infer the passive participants' features. However, in extreme cases, there is a certain risk that the features (or their distribution) of the passive participants can be detected.

Consider a situation where one active participant and one passive participant use federated random forests for modeling. The active

side has no features, and the passive participant has only one feature. At this point, the active side can initiate a training by randomly generating a set of labels and attempt to perform a node splitting operation. After splitting, the left subtree's sample space I_L and the right subtree's sample space I_R are obtained, and then the active participant stops training. At this point, the active side has obtained the grouping relationship of a feature of the passive participant. Afterward, the active side can adjust the sample space to I_L and I_R to train two sets of random tree models again to obtain more refined sample space divisions and repeatedly carry out such operations, until the active participant believes the sample classification is sufficiently fine to stop. Thus, the order of the passive participant's sample features can be detected by the active side to a certain extent (depending on the precision of the active participant's detection), which can lead to a certain leakage of feature information. Therefore, we also need to use some technical means to protect the features of the passive participants.

4.2.3.4 Privacy-enhanced federated random forest algorithm

In a paper published by Cheng *et al.* in 2019, the authors proposed a “Completely SecureBoost” algorithm to ensure that the leaf node information of the active side would not be leaked due to the guesswork of the passive participants described earlier. The core idea is to first carry out a round of boosting tree modeling locally on the active side, and from the second round onward, all participants cooperate to carry out the subsequent modeling of the federated gradient boosting tree. The authors demonstrated that under the “honest-but-curious” setting, the degree of leakage of the active participant's label information is equivalent to the purity of the leaf nodes in the first round of modeling. This purity can reach 0.5 in normal local modeling, implying that it is impossible to guess the decision information carried by the leaf node through the sample space information of each leaf node. Based on this idea, we iterated the federated random forest algorithm and proposed a privacy-enhanced version of federated random forest algorithm. Following is a brief description of the algorithm.

Similar to the algorithm proposed by Cheng *et al.* 2019, in the first round of the algorithm, we use the strategy of gradient boosting tree to carry out a round of modeling on the active side. In the second

round of modeling, through the calculation of the boosting tree, we know that in the scenario of the boosting tree, we need to optimize the following objective function:

$$\mathcal{L}^{(2)} = \sum_{i=1}^n [l(y_i, \hat{y}_i^1) + g_i f_2(\mathbf{x}_i) + \frac{1}{2} h_i f_2^2(\mathbf{x}_i)] + \Omega(f_2) \quad (4.1)$$

Here, $\Omega(f_2) = \gamma T + \frac{1}{2} \lambda \|w\|^2$ is the penalty function to prevent overfitting. $g_i = \partial_{\hat{y}_i^1} l(y_i, \hat{y}_i^1)$ is the first derivative of the loss function of the second round of boosting tree modeling with respect to $f_1(\mathbf{x}_i)$, and $h_i = \partial_{\hat{y}_i^1}^2 l(y_i, \hat{y}_i^1)$ is the second derivative of the loss function of the second round of boosting tree modeling with respect to $f_1(\mathbf{x}_i)$. After the completion of the first round of boosting tree modeling, $l(y_i, \hat{y}_i^1)$ is a known scalar, g_i and h_i can also be obtained by calculation. Therefore, without considering the overfitting penalty function, we can rewrite the loss function as follows:

$$\begin{aligned} \mathcal{L}^{(2)} &= \sum_{i=1}^n \left[g_i f_2(\mathbf{x}_i) + \frac{1}{2} h_i f_2^2(\mathbf{x}_i) \right] \\ &= \frac{1}{2} \sum_{i=1}^n [h_i f_2^2(\mathbf{x}_i) + 2g_i f_2(\mathbf{x}_i)] \\ &= \frac{1}{2} \sum_{i=1}^n \left[\left(\sqrt{h_i} f_2(\mathbf{x}_i) + \frac{g_i}{\sqrt{h_i}} \right)^2 - \frac{g_i^2}{h_i} \right] \\ &= \frac{1}{2} \sum_{i=1}^n \left[\left(\sqrt{h_i} f_2(\mathbf{x}_i) - \frac{-g_i}{\sqrt{h_i}} \right)^2 \right] - \frac{1}{2} \sum_{i=1}^n \frac{g_i^2}{h_i} \quad (4.2) \end{aligned}$$

At this point, we have a minimization problem for the squared loss function, which can also be seen as a regression problem for each sample label $\frac{-g_i}{\sqrt{h_i}}$. We can use the regression tree to solve and predict $\sqrt{h_i} f_2(\mathbf{x}_i)$. After the predicted result, $\sqrt{h_i} f_2(\mathbf{x}_i)$ is sent back to the active side, the active participant initiator can obtain the true second-round prediction result $f_2(\mathbf{x}_i)$ by correcting the result with the second-order derivative coefficient.

In actual modeling, we homomorphically encrypt $\frac{-g_i}{\sqrt{h_i}}$ to ensure that the first-order derivative coefficient and the second-order derivative coefficient itself will not be leaked. If the passive party guesses

the approximate value of $\frac{-g_i}{\sqrt{h_i}}$ by some means, it is unable to know the real label of the sample without the information of g_i or h_i .

For the protection of the passive participants' features, in response to the attack methods proposed above, we made the following improvements in the tree-construction process: Usually, the splitting decision equations given in the tree modeling process are all in the form of $a \leq b$, that is, samples less than or equal to the threshold are split into the left subtree, and samples greater than the threshold are split into the right subtree. In our tree-construction process, we randomize the direction of each node's splitting decision equation, i.e., each node's decision equation has a 50% chance of being $a \leq b$ and a 50% chance of being $a \geq b$. In this case, the sample spaces of the left and right subtrees do not have a fixed directional correspondence, thereby ensuring that even in extreme cases, the initiator cannot detect the passive participants' feature distributions through some specific modeling methods.

4.2.3.5 Privacy-enhanced universal federated learning algorithm

It is worth mentioning that the loss function in Equation (4.2) is a fixed squared loss function. Therefore, while ensuring that g_i and h_i are not leaked, by relabeling the samples, we can use other modeling methods to fit the loss function without revealing our feature distribution to the active or passive participants. These include methods such as linear regression, kernel methods, deep learning methods, and so forth. Due to space limitations, we will not discuss these methods in detail here.

4.3 Vertical Federated Gradient Boosting Algorithm

The gradient boosting algorithm is another typical tree-based modeling algorithm, with multiple variants including GBDT and XGBoost. Compared to the basic form using a single decision tree, the gradient boosting algorithm builds a series of related decision trees in sequence. When establishing each decision tree, it optimizes the current tree based solely on the residuals of all previously trained and fixed decision trees. The overall boosting form manifests as the sum

of each tree's decisions. Since the iterative tree-construction process utilizes previous information, each new decision tree should reduce the residuals from the last round, making the direction of iteration clear. Theoretically, continuous iteration will result in improved predictive performance. It doesn't require much hyperparameter tuning, and it achieves high prediction accuracy, making it a highly efficient tree-based modeling algorithm.

4.3.1 Review of XGBoost Algorithm

XGBoost is an open-source machine learning project developed by Chen *et al.* (2016b). It's a type of supervised learning algorithm. Its principle is to weight the results of multiple weak classifiers through boosting to obtain a strong classifier, and thus it belongs to gradient boosting tree models. XGBoost strives to maximize the speed and efficiency of the GBDT algorithm, implementing the GBDT algorithm efficiently and making many algorithmic and engineering improvements, hence it's named as X(Extreme) GBoosted Tree. It has been widely applied in practice and has achieved good results.

XGBoost is based on the classification and regression tree (CART) implementation. As the name suggests, it can handle both classification and regression problems. CART generates a binary tree by constantly dividing samples: First, it sorts the sample data on the current tree node according to the feature values and then it divides the sorted feature data into left and right subtrees. The process of making splitting decisions is the process of selecting the optimal splitting point, whose corresponding feature and value serve as the basis for tree node division.

$$\begin{aligned}
 \text{Obj}^{(t)} &\simeq \sum_{i=1}^n \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\
 &= \sum_{i=1}^n \left[g_i \omega_{q(x_i)} + \frac{1}{2} h_i \omega_{q(x_i)}^2 \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^T \omega_j^2 \\
 &= \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) \omega_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) \omega_j^2 \right] + \gamma T \quad (4.3)
 \end{aligned}$$

The best splitting point of CART is determined by some evaluation index. After merging and transforming the loss function (as shown in Equation (4.3)) of the XGBoost algorithm, it is only related to the sum of the first- and second-order derivatives of all the samples on each leaf node. For a single split, its impact on the overall training objective is to change from calculating the loss function using the current tree node to calculating with the left and right child nodes after the split. Therefore, as shown in Equation (4.4), the evaluation index of the split is calculated by subtracting the score of the tree node when not split from the score after the split, which is referred to as split gain:

$$\text{Gain} = \frac{1}{2} \left[\frac{G_{\text{left}}^2}{H_{\text{left}} + \lambda} + \frac{G_{\text{right}}^2}{H_{\text{right}} + \lambda} - \frac{(G_{\text{left}} + G_{\text{right}})^2}{H_{\text{left}} + H_{\text{right}} + \lambda} \right] - \gamma$$

where $G_j = \sum_{i \in I_j} g_i$, $H_j = \sum_{i \in I_j} h_i$ (4.4)

4.3.2 Review of SecureBoost Algorithm

SecureBoost is a vertical federated learning algorithm proposed by WeBank. It is a reimplement of the XGBoost algorithm in a federated learning environment and supports a variety of scenarios in WeBank's FATE platform, completing numerous federated learning tasks.

Similar to other boosting tree models in non-federated environments, the core step of SecureBoost is also to model the following loss function through gradient computation:

$$F = \sum_{i=1}^n \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

where $g_i = \partial_{\hat{y}} l(y_i, \hat{y}^{(t-1)})$, $h_i = \partial_{\hat{y}}^2 l(y_i, \hat{y}^{(t-1)})$ (4.5)

where $l(\cdot, \cdot)$ is the base loss function (like cross-entropy for classification trees and mean square error for regression trees), $\Omega(f_t)$ is the penalty function added to prevent overfitting, g_i and h_i are the first and second derivatives of the loss function $l(\cdot, \cdot)$ with respect to the predicted value, respectively.

In a federated learning environment, we need to consider the privacy protection of the data in practical modeling. By calculating and analyzing the transmitted data, the sensitive data transmitted among various parties are g_i and h_i . By collecting these two values, we can easily write the mathematical derivation of y in the loss function using g_i and h_i , which is the label of the sample that needs the most protection in the federated learning task. Therefore, SecureBoost applies additive homomorphic encryption technology to homomorphically encrypt the data g_i and h_i that need to be transmitted, ensuring that the participants without the label cannot decrypt and get the true values of g_i and h_i and reverse engineer the sample's label, causing privacy data leakage, thus completing the modeling of the boosting tree while protecting privacy data.

It is suitable for scenarios where multiple platforms have the same data sample ID, but each has different feature sets. Its training and inference require two roles: one is the active side (Active participant) who owns the label and some feature data, and the other is the passive side (Passive participant) who only has feature data. In a SecureBoost scenario, there should only be one active participant but can allow multiple passive participants, and training and inference can only be initiated by the active side.

The difference between SecureBoost algorithm and the federated random forest algorithm is that the former uses the method of gradient boosting to transmit g_i and h_i , iteratively constructing new decision trees, while the latter builds independent decision trees based on random sampling of y . Once the calculations related to gradient boosting are completed, the overall algorithmic process of the two is largely similar, so it will not be detailed here.

4.3.3 Vertical Federated Gradient Boosting Algorithm

JD Technology's federated learning system has restructured the architecture of SecureBoost algorithm, proposing a new vertical federated gradient boosting tree model FederatedGBoost. Its training and inference includes two roles: One is the client (Client) who owns the feature or label, and the other is the server (Server) responsible for coordination and data forwarding. The clients do not communicate directly with each other; instead, they complete the training and

inference processes through the server's communication and scheduling. All clients can initiate federated learning modeling. The specific design of the coordination end and client architecture can be seen in Chapter 14.

4.3.3.1 Encrypted gradient sharing

Consistent with XGBoost algorithm, the federated boosting tree's feature pre-sorting and feature binning are performed locally at the client side. However, **only one client owns the labels**, and all clients need gradient values to calculate potential split gains, necessitating a secure gradient sharing mechanism to fulfill this purpose for clients without labels.

We refer to the client with the label data as the active participant. To ensure data security, the vertical federated boosting tree uses homomorphic encryption:

- Step 1:** The active client j generates a pair of homomorphic encryption keys (p_k, s_k) , publishes the public key p_k to the server, while the client j retains the pair of public and private keys. For each round of gradient update, client j encrypts the local gradient value with the public key and sends it to the server.
- Step 2:** The server forwards the encrypted gradient values and the public key p_k to other clients.
- Step 3:** Clients without label data, based on the results of feature binning, perform binning summation of the encrypted gradient values. This process uses homomorphic addition computation. Since the federated boosting tree does not involve ciphertext multiplication, additive partial homomorphic encryption scheme is enough. Clients send the sum of binned gradient value ciphertexts back to the server.
- Step 4:** Upon receiving the forwarded binned gradient ciphertexts from the server, the active client j uses s_K for decryption and calculates the split gain according to the XGBoost algorithm's formula (4.4). At this point, active client j obtains the split gain of all global feature and threshold candidates, selects the optimal split, and notifies the server to inform the client that owns the splitting feature.

4.3.3.2 Model storage and inference

Similar to distributed XGBoost, the tree structure and node split information of the vertical federated boosting tree are stored separately on different platforms. Each tree node's splitting feature and threshold are only stored on the client where the feature is located. The client with the labels records the entire tree structure and maintains *queryTable* that keeps track of which client the node split information is on. In Step 4, client j marks the current tree node as sequence number *recordID* and records the corresponding client in *queryTable*. The leaf nodes are stored on client j to prevent other clients from reversing the original data label value through the leaf node weights.

As a result, each client ultimately possesses a portion of the model's substructure, and the collection of these substructures forms the complete federated boosting tree model. Inference must be initiated by any client, by querying client j through the server, thereby knowing which client to query next for the sample's direction in the tree. When the inference is complete, client j calculates the sum of the weights of the leaves reached for each inference sample, obtains the inference result, and returns it to the participant that initiated the inference.

This page intentionally left blank

Chapter 5

Vertical Federated Linear Regression Algorithm

Linear regression is one of the most basic models in machine learning and an essential analytical tool for many problems, with extensive applications in various scenarios. For instance, in financial risk control scenarios, linear regression is often used to predict the credit level of loan users. In the internet advertising business, it is frequently used to predict ad click-through rates. In medical scenarios, it is commonly used for disease prediction. In federated learning, linear regression also holds a significant position, so it is necessary to discuss the federated linear regression model.

In the work of Qiang Yang *et al.* (2019b), the authors categorized federated learning models into three types based on data overlap: horizontal federated learning, vertical federated learning, and federated transfer learning. Horizontal models are for data with consistent features but inconsistent IDs, vertical models target data with consistent IDs but inconsistent features, and federated transfer models cater to data with both inconsistent features and IDs. In this chapter, we adopt this categorization. Thanks to the additivity of linear models, horizontal linear regression is relatively simple, and we do not focus on it. In this chapter, we mainly pay attention to the case of vertical federated learning and introduce a federated linear regression model proposed by Qiang Yang *et al.* (2019b).

Algorithm highlight: Based on practical situations and the vertical federated linear regression algorithm, a new vertical federated multi-view linear regression algorithm is proposed. By modeling objects

from multiple perspectives and jointly optimizing all functions, this method can often more accurately mine the correlation between data in different subspaces, thus improving the model's performance.

5.1 Vertical Federated Linear Regression

In this section, we revisit the vertical federated linear regression method proposed in the paper “Federated Machine Learning: Concept and Applications” by Qiang Yang *et al.* (2019). In this paper, the authors proposed a vertical federated linear regression algorithm based on partial homomorphic encryption and the existence of a trusted third party.

Overall, the loss function and prediction function of vertical federated linear regression are not significantly different from those of regular linear regression algorithms. They share the same loss function and both can be trained using gradient descent. However, what sets them apart is that the training data is distributed among different parties. To facilitate training through gradient descent, the computation of the gradient needs to be jointly conducted by all participants.

In federated linear regression, when calculating the gradient, each participant uses the data they own to compute the intermediate results of the gradient. The gradient aggregation is completed under the participation of a trusted third party, which generates the final gradient value for parameter updates. To prevent the gradient and original data from being leaked, all data transferred to other participants are homomorphically encrypted, so computations conducted on other parties are carried out under partially homomorphic encryption.

Specifically, given a learning rate η , regularization coefficient λ , and p participants with their datasets denoted as $\{X_i^{(1)}\}_{i \in D_1}$, $\{X_i^{(2)}\}_{i \in D_2}, \dots, \{X_i^{(p)}\}_{i \in D_p}$, each participant's feature space corresponding to the linear regression parameters are $W_1, W_2, \dots, W_p$ (See Table 5.1 for notations). The total parameter set is $W = W_1 \cup W_2 \cup \dots \cup W_p$. Assume that one participant has the label denoted as $\{y_i\}_{i \in D}$. Also, denote the local prediction function of participant j as $f^{(j)}(x)$, which is the prediction value obtained from that

Table 5.1. Symbol table.

η	Step size
λ	Regularization parameter
S	Space of data from all participants
S_i	Subspace of data from the i th participant
m	Dimension of S
m_i	Dimension of S_i
f_i	Local prediction function of participant i
U_{p_i}	Indicator matrix from S to S_{p_i}
n	Total number of samples of all participants after ID alignment
p	Total number of participants
W_i	Training parameters corresponding to sample features in S_i
W	Training parameters corresponding to sample features in S
D_i	Training dataset of the i th participant
D	Union of training data from all participants
I	Union of data IDs from all participants
O	Overlap patterns of all IDs in I
$X_j^{(i)}$	The j th sample in D_i
y_i	Label of the i th sample in D
O_i	Mixed mode when the sample ID is i
α_{O_i}	Mixing parameter when the mixed mode is O_i
$L(W, \alpha)$	Loss function
$\nabla_{W_j} L$	Gradient of the loss function L with respect to the parameter vector W_j
$\ \cdot\ _2^2$	Squared Euclidean norm of a vector
$[[\cdot]]$	Ciphertext under additive homomorphic encryption
$\mathbb{Z}_N^*$	Set of integers $1, 2, 3, \dots, N-1$

participant's data and corresponding features, i.e., $f^{(j)}(x) = W_j X_i^{(j)}$. The training loss function can then be represented as

$$L(W) = \frac{1}{n} \sum_{i=1}^n \left(\sum_{j=1}^p f^{(j)}(x) - y_i \right)^2 + \frac{\lambda}{2} \left(\sum_{j=1}^p \|W_j\|^2 \right)$$

The gradient of the training parameters with respect to the loss function is

$$\nabla_W L = \frac{2}{n} \sum_{i=1}^n \left(\sum_{j=1}^p f^{(j)}(x) - y_i \right) x + \lambda \|W_j\|$$

Denote $d_i = \sum_{j=1}^p f^{(j)}(x) - y_i$, in the calculation of d_i , each participant's $\hat{y}$ needs to be homomorphically encrypted and sent to the participant with the label. To prevent privacy leakage, the $\hat{y}_i^{(j)}$ from other participants should be technically encrypted. Denote $[[\cdot]]$ as additive homomorphic encryption, then $[[d_i]] = \sum_{j=1}^p [[f^{(j)}(x)]] - [[y_i]]$. Therefore, under homomorphic encryption, the gradient of the training parameters with respect to the loss function is

$$[[\nabla_{W_j} L]] = \frac{2}{n} \sum_{i=1}^n ([[d_i]] x_i^{(j)}) + [[W_j]]$$

5.1.1 Training Process

Based on the above discussion, the training process of vertical federated linear regression is presented in Table 5.2. In addition to those participants, a trusted third party is required during the training process. The third party is responsible for generating the public and private keys of homomorphic encryption and decrypting the gradient ciphertext. To prevent the third party from acquiring the plaintext of the gradient from the participating parties, a random mask is added to the data to be decrypted before it is sent to the third party.

Under the assumption of the semi-honest model, this approach technically prevents the third party from obtaining the privacy information of the training participants. In Table 5.2, it is assumed that only one participant has the label, denoted as Participant *B*. Since the behavior of other participants besides *B* is identical, we denote one of them as Participant *A* and only describe *A*'s procedures.

We briefly discuss the security of the protocol in Table 5.2 under the assumption of a semi-honest model. Under the semi-honest assumption, all participants strictly follow the protocol. In Steps 1 and 2, the data transmitted by each participant is encrypted homomorphically, and both *A* and *B* have no private keys, so neither *A* nor *B* can obtain any information other than their own data. In Step 3, each participant adds a random mask before transmitting their gradient to the third party for decryption, so the third party also cannot obtain any information about the training participants. Therefore, the protocol in 5.2 is secure under a semi-honest model.

Table 5.2. Training process of vertical federated linear regression.

	Other participants (One denoted as A)	Participant with label (Assumed to be B)	Third party
Step 1	Initialize local training parameters	Initialize local training parameters	Generate public and private keys for homomorphic encryption and send public keys to the training participants
Step 2	Compute local prediction $[[f^A(x)]]$	Compute $[[f^B(x)]]$, receive other participant's local predictions, calculate $[[d_i]]$ and return to other participants; calculate $[[L]]$ and send to third party	
Step 3	Generate mask R_A , calculate $[[\nabla_{w_A} L]]$, add $[[R_A]]$, get $[[\nabla_{w_A} L]] + [[R_A]]$, and send to third party	Generate mask R_B , calculate $[[\nabla_{w_B} L]]$, add $[[R_B]]$, get $[[\nabla_{w_B} L]] + [[R_B]]$, and send to third party	Decrypt $[[L]]$, $[[\nabla_{w_A} L]] + [[R_A]]$, $[[\nabla_{w_B} L]] + [[R_B]]$ and return results to respective participants
Step 4	Update local training parameters	Update local training parameters	

Table 5.3. Inference process of vertical federated linear regression.

	Participants (One denoted as A)	Third party
Step 1		Send data ID i to be predicted to each participant
Step 2	Compute $[[f^A(x)]]$ and send to third party	Calculate $\sum_{j=1}^p [[f^j(x)]]$ and return the result

5.1.2 Inference Process

The inference process of vertical federated linear regression also requires the cooperation of all participants and a trusted third party to output the final result. Table 5.3 summarizes the inference process.

Note that each participant sends data directly to the trusted third party. In cases where the third party is fully trusted, the data and results of each participant will not be exposed to other parties.

5.1.3 A Dilemma of Vertical Federated Learning

Horizontal and vertical federated learning, due to the restrictions on data samples in ID and features, must choose all IDs or features to align during training and inference, and samples that cannot be aligned cannot participate in training and inference. In real application scenarios, this setting often poses many difficulties.

Example: We illustrate this problem with a real-world business scenario. Financial companies often target advertisements toward potential customers in their lending business to increase product visibility. In a case involving internet advertising, Financial Companies A and B aim to promote their products to target customers to enhance the registration and credit approval rate. Both Companies A and B have their respective user IDs and tags but lack comprehensive user features. An external data supplier, C , possesses more complete user information. Hence, in the joint modeling process between parties ABC , A and B provide user IDs, a few user features, and tags, while C provides more complete user features. Note that conventional horizontal or vertical approaches are not effective for three-party joint modeling. In the horizontal scenario, the user overlap between A and B is too small; in the vertical scenario, the overlapping feature count among ABC is too limited. Even more importantly, even if horizontal or vertical data were used for training, during the inference phase, the samples cannot overlap effectively, leading to ineffective predictions.

5.2 Federal Multi-View Linear Regression

Therefore, we aim to find a new federated learning method to solve this problem. In this section, we introduce a federated learning method based on multi-view learning. Multi-view learning assumes that the object being modeled can be described from multiple views, each of which often has a strong correlation with the others. By modeling the object from multiple views and jointly optimizing all

functions, this method can more accurately mine the correlation among data in different subspaces, thus improving the model's effectiveness. For more detailed background on multi-view learning, you can refer to the literature proposed by Xu *et al.* (2013) and Li *et al.* (2019), respectively.

As can be seen in Section 5.1.3 (Example), on the datasets of different participants, the same ID corresponds to the same user, indicating strong data correlation. If we consider the samples corresponding to the same ID as a user's description under different views, we can obtain a stronger data representation. Furthermore, if we assume that all observed data come from the combination of different participant's views, and we model and jointly train the combination of views appearing in the dataset, we can resolve the problem of misaligned IDs mentioned earlier.

To be more rigorous, let's assume that the samples of each participant are samples within different subspaces of the same high-dimensional space, denoted as S , and the subspaces generated by the sample sets of each participant are $S_1, S_2, \dots, S_p$. Our goal is to learn a linear function from S by weighting the overlap patterns of samples in different subspaces.

Due to the potential overlap of datasets from different participants, the overlap patterns of samples corresponding to different IDs may vary. As an example, Figure 5.1 demonstrates various sample overlaps in Section 5.1.3. From the figure, it can be seen that sample i overlaps simultaneously on participants ABC, while sample j only overlaps on participants BC.



Figure 5.1. Overlap patterns: Data from different participants vary. Sample i overlaps simultaneously on participants ABC, while sample j only overlaps on participants BC.

overlaps on participants BC. Thus, before defining multi-view federated learning, we first need to define the overlap patterns of samples on the datasets of various participants.

Definition 5.1 (Overlap pattern). Assume there are p participants in the federated modeling, with the sample sets of the p participants denoted as $X_1, X_2, \dots, X_p$ and the sets of sample IDs as $I_1, I_2, \dots, I_p$. The union of all participant samples is denoted as $X = X_1 \cup X_2 \cup \dots \cup X_p$, and the union of all participant IDs as $I = I_1 \cup I_2 \cup \dots \cup I_p$. $\forall i \in I$, the overlap pattern O_i of the sample i is defined as the set of participant IDs that contain this ID.

For instance, in Figure 5.1, the overlapping pattern O corresponding to sample i is $\{A, B, C\}$, and the overlapping pattern for sample j is $\{B, C\}$.

Supposing there is a certain correlation among data from multiple participants, in federated multi-view learning, we can consider the samples of the same ID from different participants as results sampled from different subspaces within the same high-dimensional population. Denote the distribution of this high-dimensional population as $P_{X \in S}(x)$, then $X_1, X_2, \dots, X_p$ respectively follow $P_{X \in S_1}(x_1), P_{X \in S_2}(x_2), \dots, P_{X \in S_p}(x_p)$. And $P_{X \in S_1}(x_1), P_{X \in S_2}(x_2), \dots, P_{X \in S_p}(x_p)$ are marginal distributions of $P_S(X)$.

Let $H(\Theta, x_i, O_i)$ be a function that maps x_i to S , $H(\Theta, x_i, O_i) : \bigcup_{j \in O_i} S_j \mapsto S$, then federated multi-view linear regression can be defined as

$$\underset{\Theta, W}{\text{minimize}} \sum_{i \in I} (WH(\Theta, x_i, O_i) - y_i)^2 + \frac{\lambda}{2} \|W\|_2^2$$

$H(x_i) = \alpha_A U_A x_i^{(A)} + \alpha_B U_B x_i^{(B)} + \alpha_C U_C x_i^{(C)}$, Next, we examine the definition of H , still using sample i from Figure 5.1 as an example. As sample i overlaps on ABC, and we assume all samples are drawn from the same high-dimensional distribution, we can propose $H(x_i) = U_A x_i^{(A)} + U_B x_i^{(B)} + U_C x_i^{(C)}$, where U_A, U_B, U_C are linear transformation matrices from S_A, S_B, S_C to S . These linear transformations move features in S_A, S_B, S_C to corresponding positions in S and take the arithmetic average for each feature. Taking U_A as an example, the j th column is a one-hot vector of length m . If the i th position is non-zero, it indicates that the j th feature of S_A is at

the i th position in S . Since we need to take an arithmetic average, if the j th feature of S_A overlaps k times ($0 < k < 4$) in ABC, the corresponding non-zero value should be $1/k$. Furthermore, to adjust the importance of each sample in summation, we introduce importance factors $\{\alpha_A, \alpha_B, \alpha_C, |\alpha_A + \alpha_B + \alpha_C = 1\}$ for each participant's samples. Hence, $H(x_i) = \alpha_A U_A x_i^{(A)} + \alpha_B U_B x_i^{(B)} + \alpha_C U_C x_i^{(C)}$.

It's worth noting that the x_i in this case belongs to the situation where ABC all overlap. For different overlapping patterns, the values of $\alpha_A, \alpha_B, \alpha_C$ should be different. Therefore, each overlapping pattern should have different importance factors, ensuring their sum is 1. If a sample corresponding to a certain ID does not exist in a certain participant, then the α of that participant will be 0.

Therefore, the loss function and prediction function of federated multi-view linear regression can be defined as follows:

Definition 5.2 (Federated multi-view linear regression).

Assume there are p participants in federated modeling, with the samples of p participants denoted respectively as $X_1, X_2, \dots, X_p$. Let $x_i^{(j)} \in X_j$ be the sample of ID i in participant j , U_j be the transformation matrix from S_j to S as defined above, O_i be the overlap pattern of sample with ID i , and $\alpha^{(i)} = \{\alpha_1^{(i)}, \alpha_2^{(i)}, \dots, \alpha_p^{(i)} \mid \sum_{k=1}^p \alpha_k^{(i)} = 1\}$, then we have

$$f(x_i, W, \alpha^{(i)}) = W \left(\sum_{j=1}^p \alpha_j^{(i)} U_j x_i^{(j)} \right) \quad (5.1)$$

Let $\alpha = \{\alpha^{(i)} \mid i \in I\}$, the loss function of federated multi-view linear regression can then be obtained as

$$L(W, \alpha) = \frac{1}{n} \sum_{i \in I} (f(x_i, W, \alpha^{(i)}) - y_i)^2 + \frac{\lambda}{2} \|W\|_2^2 \quad (5.2)$$

Sometimes in practical applications, we may obtain prior knowledge about the feature weights of each participant. For instance, in certain cases, we could know the data collection methods, preprocessing methods, as well as relevant expert experience concerning feature importance. Thus, we could assign weights to various overlap patterns based on this experience, and in this case, the multi-view federated linear regression degenerates into general linear regression.

5.2.1 Second-Order Optimization Method Based on BFGS

In this section, we use the second-order optimization method BFGS to find the local minimum of the objective function. Unlike distributed optimization algorithms commonly used in machine learning, federated learning has more substantial time and space overheads during training. The reasons primarily lie in the following: (1) Homomorphic encryption is required for multi-party communication interactions, where a single multiplication or addition operation can take up to a hundred times longer than a normal operation, making computation under homomorphic encryption the system's bottleneck; (2) the data volume under homomorphic encryption grows exponentially, posing challenges to network transmission. Therefore, we hope to improve traditional second-order optimization methods: Strive to perform as many calculations as possible locally with unencrypted data. As different participants have different dataset overlap patterns, many sample IDs sometimes exist only in one participant, and calculations related to these data often do not need interaction with other participants. Thus, during computation, we can complete relevant calculations of these samples locally.

1. BFGS algorithm

BFGS is an efficient quasi-Newton method that determines the search direction using both the first- and second-order information of the objective function. Unlike Newton's method, it doesn't directly compute the Hessian matrix but approximates the inverse of the Hessian matrix. Let's denote the approximated inverse Hessian matrix as H . In BFGS, the iterative formula of H is given by

$$H_{k+1} = (I - \rho_k s_k d_k^T) H_k (I - \rho_k d_k s_k^T) + \rho_k s_k s_k^T \quad (5.3)$$

where

$$\begin{aligned} s_k &= w_{k+1} - w_k \\ d_k &= g^{(k+1)} - g^{(k)} \\ \rho_k &= \frac{1}{s_k^T d_k} \end{aligned}$$

2. Computation of H and g across multiple participants

As mentioned earlier, to further reduce computational overhead, we aim to separate each sample's first- and second-order derivatives into two parts: those that need joint computation and those that can be completed locally. Therefore, we first classify the samples of each participant into private samples and non-private samples.

Definition 5.3 (Private samples). For any participant, if a sample does not overlap with any other participant, we call this sample a **private sample** of this participant. Let the private samples of participant j be denoted as X_j^{priv} , where $X_j^{\text{priv}} = \{x \in X_j | O_{I_x} = \{j\}\}$. The **non-private samples** are then defined as $X_j \setminus X_j^{\text{priv}}$.

3. Computation for private samples: $x \in X_j^{\text{priv}}$

Let the dimension of the entire space S be m , the dimension of the subspace S_i of the samples of participant p_i be mi , and X be the set of all samples, whose size is n . For convenience, we abbreviate the error function $l(f(x), y) = (y - f(x))^2$ (in the case of logistic regression, the logistic function) as $l(f)$. We define W^i as the parameter corresponding to the sample space of the i -th participant. According to Definition 5.2, the loss function can be written as follows:

$$L(W) = \sum_{x \in X_j^{\text{priv}}} l(W_i x, y) + \sum_{x \in X_j \setminus X_j^{\text{priv}}} l(f(x, W), y)$$

Let $Z = W_i x$, by applying the chain rule to the above formula,

$$\nabla_W L = \sum_{x \in X_j^{\text{priv}}} U_x x \nabla_Z l(Z) + \sum_{x \in X_j \setminus X_j^{\text{priv}}} \nabla_f l(f) \nabla_W f(W)$$

Note that in the above formula, the term on the left side of the plus sign does not require any information from other participants when being computed and can be fully computed locally and then mapped to the entire space S via U_j . Further, for the Hessian matrix,

$$\nabla_W^2 L = \sum_{x \in X_j^{\text{priv}}} U_x \nabla_Z^2 l(Z) U_x^T + \sum_{x \in X_j \setminus X_j^{\text{priv}}} \nabla_f l(f) \nabla_W^2 f(W)$$

Similarly, the part related to private samples can also be fully computed locally when calculating the second-order derivative. In the

approximation of the Hessian matrix $\nabla_W^2 L$ of L with respect to W using quasi-Newton method, the approximation to $\nabla_Z^2 l(Z)$ can be directly performed locally at each participant and then the approximated result is mapped to the entire space. This not only reduces the computational complexity but also reduces the errors produced when approximating high-dimensional sparse matrices.

4. For non-private samples: $x \in X_j \setminus X_j^{\text{priv}}$

Different from the computation of private samples, the computation related to non-private samples cannot be completed by a single participant and involves data interaction. Therefore, in order to protect the security of the participants' data, the computation process needs to be protected according to the methods in Secure Multi-Party Computation (SMPC). In this section, we first give the calculation formula in plaintext. In Section 5.2.2, we introduce a computation protocol based on secret sharing and homomorphic encryption in detail.

Gradient: For $\forall x \in X_j$, let $f^{(j)}(x, W, \alpha)$ be the prediction of x on participant j , abbreviated as $f^{(j)}(x)$.

Compute: $\partial L / \partial w_i$ The derivative of the loss function to the training parameter w_i , $\frac{\partial L}{\partial w_i} = \frac{\partial l}{\partial f} \frac{\partial f}{\partial w_i}$.

The calculation of $\partial f / \partial w_i$ requires the addition of the prediction values $f^{(j)}$ from each participant to obtain the global prediction value f :

$$\frac{\partial f}{\partial w_i} = \frac{\partial}{\partial w_i} \left(\sum_{j=1}^p \sum_{x \in X_j} f^{(j)}(x) \right)$$

After calculating f , we can obtain $\partial L / \partial w_i$:

$$\frac{\partial l}{\partial f} = \sum_{j=1}^p \frac{\partial l^{(j)}}{\partial f} = \sum_{j=1}^p \frac{\partial}{\partial f} \left(\sum_{x \in X_j} l(f^{(j)}) \right) \quad (5.4)$$

Compute: $\partial L / \partial \alpha$ Since each overlapping pattern corresponds to a different α , the number of α parameters depends on the number of types of overlap patterns in the participants' datasets. Given an

overlap pattern o , let the sample set that meets this overlap pattern be X_o , then the derivative of L to α_o is

$$\frac{\partial L}{\partial \alpha_o} = \frac{\partial l}{\partial f} \frac{\partial}{\partial \alpha_o} \left(\sum_{j=1}^p \sum_{x \in X_o \cap X_j} f^{(j)}(x, \alpha) \right) \quad (5.5)$$

From the above formula, we can summarize the calculation process of the gradient as follows: each participant calculates $\frac{\partial l_{(j)}}{\partial f}, \frac{\partial f^{(j)}}{\partial w_i}, \frac{\partial f^{(j)}}{\partial \alpha_o}$ locally, then the participants aggregate to get $\frac{\partial L}{\partial f}$; then each participant calculates $\frac{\partial L}{\partial f} \frac{\partial f^{(j)}}{\partial w_i}$; finally, the first-order derivatives $\frac{\partial L}{\partial w_i}, \frac{\partial L}{\partial \alpha_o}$ are obtained by aggregating the results from each participant again.

Approximation of the Hessian Matrix: According to formula (12.13), the calculation of H requires the calculation of the changes of w, g in each iteration, namely s and d , but s, d cannot be calculated by a single participant, they need to be collected and processed to get the final result after each participant gets g, w in this round.

5.2.2 Secure Computation Protocol

In this section, we incorporate SMPC into the aforementioned computation process to ensure data security. This protocol is based on the semi-honest model, assuming that the corrupted party merely collects information from the protocol without deviating from the protocol standards. Unlike the method proposed by Yang *et al.* (2019a), which relies on a trusted third party, for the sake of security and fairness, we adopt a decentralized architecture in this protocol. In other words, all computations in this protocol are carried out by the participants, without relying on a trusted third party. It should be pointed out that in this protocol, we randomly select a Master from the participants. This Master can be any of the participants and does not have to be trusted. The Master and other participants only differ in their computational tasks: The work performed by the Master includes aggregation under homomorphic encryption and processing of its own data, while non-Master nodes are only responsible for processing their own data. In contrast, the method proposed by Yang *et al.* (2019a) requires the third party to be independent of any participant and to be trustworthy.

5.2.2.1 Basic cryptographic components and assumptions

This section introduces several basic cryptographic components and assumptions that we use.

Shamir’s secret sharing: Shamir’s secret sharing was proposed by Adi Shamir. Utilizing the property that a t -degree polynomial can be completely determined by $t + 1$ points on it, Shamir’s scheme hides a plaintext message within a random polynomial and creates shares by evaluating this polynomial at various points. By using Lagrange interpolation on these shares, the plaintext can be recovered. More specifically, suppose we want to split a secret s into n shares such that any subset of at least $t + 1$ ($n > t + 1$) shares can recover s . The scheme proceeds by defining a t -degree random polynomial $P(X)$ over a finite field F , setting $P(0) = s$, and taking n different values to form $(x_1, P(x_1)), (x_2, P(x_2)), \dots, (x_n, P(x_n))$. In this section, we default to taking $x_1 = 1, x_2 = 2, \dots, x_n = n$ as these n values, which constitute the n shares. To recover the secret, we take any $t + 1$ points and apply Lagrange interpolation:

$$P(X) = \sum_{i=1}^{t+1} l_i P(x_i)$$

where l_i is known as the Lagrange coefficient

$$l_i = \prod_{j=1, j \neq i} \frac{X - x_j}{x_i - x_j} \quad (5.6)$$

Taking $P(0)$ recovers s .

BGW secure multiparty computation protocol: The BGW protocol is one of the earliest secure multiparty computation protocols. It utilizes the homomorphic properties of each share in Shamir’s secret sharing under addition and multiplication to perform these operations in ciphertext (where plaintext is shared among different parties). It can be used to evaluate any arithmetic expression. The security of the BGW protocol under the honest majority assumption was proved by Ben-Or *et al.* (1988). Beaver (1991) proposed a method to speed up the multiplication operation in the BGW protocol, separating the protocol execution into online and offline phases and

using triples generated during offline preprocessing (Beaver triples or multiplication triples) to eliminate the need for additional network interaction during multiplication.

Homomorphic encryption: Homomorphic encryption is one of the effective methods for implementing secure multiparty computation. It allows operations like retrieval and comparison to be performed on encrypted data, yielding correct results without decrypting the data during the processing. Fully homomorphic encryption allows both addition and multiplication operations on ciphertext but is often slower, while partially homomorphic encryption only allows either addition or multiplication but is typically faster. For a detailed overview of homomorphic encryption, we recommend the review paper by Acar *et al.* (2018). As federated learning often has high time requirements, and for most linear models such as linear regression and logistic regression, ciphertext addition operations are sufficient (the logistic function can be linearly approximated), we use the efficient Paillier cryptosystem for ciphertext computation. For a detailed and rigorous discussion, please refer to the work of Paillier in 1999.

We provide a brief overview of the Paillier public-private key generation process here: Two large prime numbers p and q are randomly selected in $\mathbb{Z}_N^*$ and their product N is calculated, the public key is $g = N + 1$, and the private key is Euler's totient function of N , $\phi(N) = (p - 1)(q - 1)$.

During encryption, a random $r \in \mathbb{Z}_{N^2}^*$ is chosen and the plaintext $m \in \mathbb{Z}_N^*$ is encrypted to get the ciphertext c through the following computation:

$$c = g^m r^N \mod N^2$$

During decryption, the following is computed:

$$m = L(c^\lambda \mod N^2) \lambda^{-1} \mod N$$

where $L(x) = (x - 1)/N$.

Security assumptions: Considering the practical situation in the vast majority of business scenarios, we assume that all participating parties in federated learning satisfy the **semi-honest assumption**. Compared to the stronger security assumption of a malicious

model, the semi-honest model is significantly more time-efficient; as for security, the semi-honest assumption aligns with the reality of most business cases.

Communication network structure: Each participant can directly communicate with the others. Before executing the protocol, a participant is randomly selected as the Master to collect and aggregate the encrypted local information from all parties.

5.2.2.2 Paillier algorithm based on distributed key generation and threshold decryption

In the absence of a trusted third party, the processes of key generation and decryption are relatively independent from the entire training process. We separately introduce these two processes in this section and introduce the training and inference phases in the following section.

Due to the decentralized architecture adopted for secure training and inference, we used the Paillier algorithm with distributed key generation and threshold decryption without a trusted third party. This algorithm consists of two parts: distributed key generation and threshold decryption, both without a trusted third party. They were first proposed by Boneh (2001) and Fouque *et al.* (2000), and Nishide and Sakurai (2010) provided a security proof for the entire protocol. Veugen *et al.* (2019) described the implementation of the algorithm in their paper and provided several optimizations.

Before introducing the Paillier algorithm based on distributed key generation and threshold decryption, it is necessary to first introduce its foundation — the Paillier algorithm based on threshold decryption.

Paillier with threshold decryption: Threshold decryption is one application of secret sharing in public key cryptosystems. It uses secret sharing to divide the private key into multiple shares. During decryption, the ciphertext is first decrypted into multiple intermediate results using the shares, and then the intermediate results are aggregated and decrypted again to get the plaintext. The Paillier algorithm based on threshold decryption was first introduced by Fouque *et al.* (2000) in their paper. The authors assumed the existence of a trusted third party outside of the multiple participants.

This third party uses Shamir's secret sharing to generate different Paillier keys for each participant, and decryption requires a certain number of participants to work together.

In our application, each participant is required to generate a key, and every participant must participate in decryption. Therefore, the degree of the random polynomial in Shamir's secret sharing is equal to the number of participants minus one. The third-party key generation process can be briefly described as follows: Assuming the number of participants is p , a random $\beta \in \mathbb{Z}_N^*$ is selected, and p shares of $\lambda\beta$ are generated. Each share is the private key of each participant. The public key is $(g, \beta\lambda \bmod N)$. If the polynomial corresponding to the share of $\beta\lambda$ is $h(X)$, the shares at participants $1, 2, \dots, p$ are $h(1), h(2), \dots, h(p)$, respectively.

Assuming the ciphertext is c and participant j needs to decrypt c . Figure 5.2 describes the following decryption steps. Each participant calculates the intermediate result. At the beginning of decryption, each participant first calculates the intermediate result c_i and passes it to participant j :

$$c_i = c^{h(i)} \bmod N^2$$

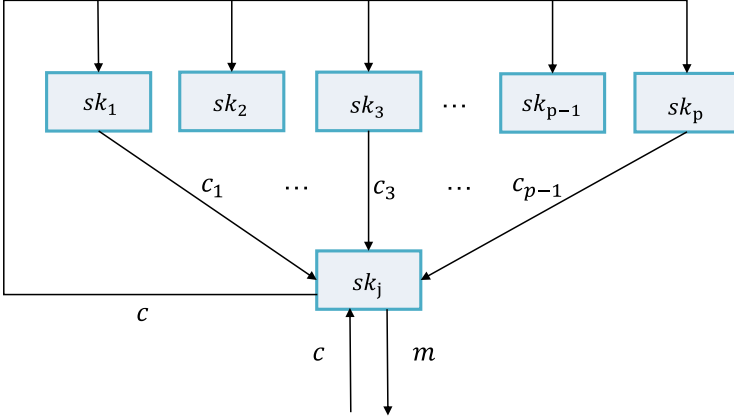


Figure 5.2. The decryption steps of the Paillier algorithm based on threshold: First, each party uses their private keys $\{sk_j | j \in \{1, 2, \dots, p\}\}$ to decrypt the ciphertext c and produces multiple intermediate results $\{c_j | j \in \{1, 2, \dots, p\}\}$. These intermediate results are then aggregated and decrypted again to obtain the plaintext m .

Participant requiring the plaintext decrypts again: After collecting c_i from each participant, participant j calculates the Lagrange coefficient l_i using formula 5.6 and decrypts again to obtain the plaintext m

$$m = L(c^{\lambda\beta} \bmod N^2)\theta^{-1} \bmod N$$

where $c^{\lambda\beta} = \prod_{j=1}^q c_i^{l_i} \bmod N^2$.

Paillier-distributed key generation without a trusted third party: Different from the Paillier algorithm based on threshold decryption, in the absence of a trusted third party, key generation cannot be completed by one party alone, so a new public key and private key generation method is needed. In his 2001 paper, Boneh first introduced the distributed key generation protocol for RSA. Nishide *et al.* extended it to Paillier key generation and provided a proof of security for the entire protocol. Veugen *et al.* described the implementation process of the algorithm in their paper and proposed several optimizations.

Next, based on the literature by Veugen *et al.* (2019), we briefly introduce the process of Paillier's distributed key generation without a trusted third party. Paillier distributed key generation can be divided into two stages: (1) distributed generation of the modulus N ; (2) multiple parties generate public keys and their private keys.

Distributed generation of modulus N : Similar to RSA, the Paillier key is also based on the product of two large primes p and q , N , during the process of distributed key generation, each participant needs to cooperate to generate their own p_i, q_i so that $p = \sum_i p_i, q = \sum_i q_i, N = (\sum_i p_i)(\sum_i q_i)$, but it is required that each participant only knows N and their own p_i, q_i and cannot know the specific values of p, q .

In the literature by Boneh *et al.* (2001), each participant first generates random numbers p_i and q_i . Through the BGW protocol, they compute $N = (\sum_i p_i)(\sum_i q_i)$. Since this N might not be the product of two prime numbers, a biprimality test is required. If it passes, they proceed to the next stage, otherwise, the generated N is discarded, and they start over. Once an appropriate N has been generated through the BGW protocol, its value is revealed to all parties.

It should be noted that the first two steps are usually repeated numerous times, but empirical evidence shows that its time consumption remains within an acceptable range. In the experiment conducted by Malkin *et al.* (1999), three participants each used a machine with six threads to generate a 2048-bit RSA key, which took about 18 minutes on average. Additionally, as the key generation process is independent of the training and inference, it can be performed as an offline preprocessing step to save time.

Multi-party generation of public and private keys: In the absence of a trusted third party, each participant can use the BGW protocol to calculate $(\beta\lambda \bmod N)$ and the share of $\beta\lambda$. The specific computation process is quite complicated. To ensure the coherence of this chapter, the readers are advised to refer to the detailed process in the work of Veugen *et al.*

5.2.2.3 Training protocol

The training process can be roughly divided into four steps: (1) Generate public and private keys for each participant, and offline generation of triples needed for BGW multiplication calculations; (2) participants compute gradient g using Paillier and decrypt it for each participant; (3) each participant calculates their local $s_k d_k$ and then uses the BGW protocol to calculate H ; (4) finally, each participant updates the value of W , decrypts the updated W , and moves to the next iteration or terminates.

The detailed process of generating public keys and distributed secret keys in Step (1) has been introduced in Section 12.6.1, and the calculation of triples has been detailed in Beaver's (1991) literature, hence it won't be further explained here. Next, we provide a detailed process for Steps (2)–(4). Since all operations are identical for all participants except the Master, we illustrate using participant j as an example.

Computing gradient g : The process of computing the gradient is divided into three steps:

(1) **All participants compute local prediction values:** First, each participant calculates the predicted value $f^{(j)}$ of their non-private samples according to formula 5.1 and homomorphically

encrypts it to get $[[f^{(j)}]]$, where

$$f^{(j)}(x_i, W, \alpha) = W \left(\alpha_j^{(i)} U_j x_i^{(j)} \right), \forall x_i \in X_j \setminus X_j^{\text{priv}}$$

The participant with the label sends the encrypted label $[[\{y\}]]$ to the Master.

(2) **Master aggregates $f^{(j)}$ from all parties:** Each participant sends $[[f^{(j)}]]$ to the Master for summation to obtain $[[\partial L / \partial w_i]] = [[f - y]]$ and returns it to each participant:

$$[[f(W, \alpha) - y]] = \sum_{x \in X \setminus X^{\text{priv}}} \left(\sum_{j=1}^p [[f^{(j)}(x)]] - [[y]] \right)$$

Note that, although the Master obtained $f^{(j)}$ from all parties in this step, it cannot gain any useful information as threshold decryption requires participation from all parties.

(3) **Each participant calculates the intermediate value of their gradient:** Each participant receives $[[f - y]]$ from the Master and calculates the intermediate value of their gradient:

$$[[g_j]] = \sum_{x \in X_j \setminus X_j^{\text{priv}}} [[f(W, \alpha) - y]] x + \left[\left[\sum_{x \in X_j^{\text{priv}}} \left(f^{(j)}(x, W, \alpha) - y \right) x \right] \right]$$

(4) **Master aggregates $[[g_j]]$ from all parties:** Master aggregates $[[g_j]], j \in \{1, 2, \dots, p\}$, received from each participant to obtain the gradient ciphertext $[[g]] = \sum_{j=1}^p g_j$ and returns it to each participant. Each participant then decrypts $[[g]]$ (refer to Section 12.6.1) to obtain the gradient g for this round of iteration. Similarly, based on the property of threshold decryption, the Master cannot obtain any useful information unless all participants collectively decrypt or the Master obtains the secret keys of all participants.

Computing H : The process of calculating H involves ciphertext multiplication. As Paillier can't perform this operation, we choose to use the BGW protocol based on Beaver triples. The content to be computed is given by formula (12.13). During the computation, the

communication complexity of the BGW protocol is $O(pm^2)$, where m is the number of sample features. Since in the majority of cases the number of sample features is relatively small, the total time consumption is within a small range.

A detailed introduction of the BGW execution process can be found in the work of Cramer *et al.* (2015). To avoid verbosity, we recommend that readers directly refer to it.

Updating the value of W : Each participant updates to get $W_j^{k+1}, j \in \{1, 2, \dots, p\}$, where k denotes the k th iteration. They then check the 2-norm of g and stop the iteration when the condition is met:

$$W_j^{k+1} = W_j^k - \eta Hg$$

Brief proof of security: We briefly explain the security of the entire computation process. During the process of obtaining g , any data sent to other parties is encrypted, and the decryption process must involve all participants. Therefore, all participants can only obtain ciphertexts and cannot understand the information contained in them. In the process of solving H , all calculations are carried out through BGW, so the security of the computation process is guaranteed by the security of the BGW protocol. Each participant only obtains the gradient and parameter W of their own features during the entire training process, so the computation process is secure.

5.2.2.4 Inference protocol

Inference processes: The inference processes are similar to the first two steps of the gradient calculation process in training:

(1) **All participants calculate local prediction values:** Each participant calculates the predicted value $f^{(j)}$ of their non-private samples and homomorphically encrypts them to obtain $[[f^{(j)}]]$, where

$$f^{(j)}(x_i, W, \alpha) = W \left(\alpha_j^{(i)} U_j x_i^{(j)} \right), \forall x_i \in X_j \setminus X_j^{\text{priv}}$$

(2) **The Master aggregates $f^{(j)}$ from all parties:** Each participant encrypts their $f^{(j)}$ to obtain $[[f^{(j)}]]$ and sends it to the Master.

The Master then sums these up to obtain the prediction $[[f]]$ and sends it back to each participant.

$$[[f(W, \alpha)]] = \sum_{x \in X \setminus X^{\text{priv}}} \left(\sum_{j=1}^p [[f^{(j)}(x)]] \right)$$

(3) **Participants decrypt to obtain final results:** Each participant receives $[[f]]$, decrypts it in two steps as described in (2), calculates the prediction for their private samples, and returns the results. Note that not all participants need to obtain the prediction results. In these cases, the decryption request can be initiated by the participant who needs the prediction. After local decryption, the decryption intermediate result is returned to the requesting party, who finally decrypts to obtain the result.

Brief security proof: Similar to the training process, all data transmitted by each participant are encrypted, and the decryption process requires the participation of all parties. Each participant can only obtain their own prediction results.

5.2.3 Conclusion

In this chapter, we focused on introducing two methods of vertical federated linear regression. In Section 5.1, we introduced a basic method of vertical federated linear regression, which provides a design paradigm for algorithms in federated learning and serves as a reference for subsequent practical and efficient methods. In Section 5.2, based on the review by Yang *et al.* (2019), we expanded the method in the first section from a practical business scenario and replaced the trusted third party with a decentralized structure, making the algorithm more suitable for actual business scenarios. It is important to note that there are far more than just these two linear regression algorithms in federated learning. In practice, linear regression problems in federated learning often need to change according to actual business needs and system conditions.

Furthermore, in related fields such as privacy-preserving machine learning and SMPC, some works are also noteworthy. They often focus on security, training efficiency, system stability, and other aspects rather than sticking to horizontal and vertical federations.

Related to linear regression, for example, Zheng *et al.* (2019) proposed a linear joint learning system based on a malicious attack model in 2019; SecureML introduced a privacy-protected training system based on 2PC (Secure Two-Party Computation); Ang *et al.*'s 2020 paper discussed joint training of linear models in noisy network environments.

This page intentionally left blank

Chapter 6

Vertically Federated Kernel Learning

In a lot of real-world data mining and machine learning applications, data are provided by multiple clients and each maintains private records of different feature sets about common entities. These types of data are generally called vertically partitioned data. It is challenging to train these vertically partitioned data effectively and efficiently while keeping data privacy for traditional data mining and machine learning algorithms. Given that there is little literature on designing for nonlinear methods, especially kernel methods, we focus on nonlinear learning with kernels in this section and propose a Vertically Federated Kernel Learning (VFKL) algorithm for vertically partitioned data. Specifically, we use Random Fourier Features (RFF) to approximate the kernel mapping function and use a specially designed doubly stochastic gradient (DSG) to update the solutions, which are all computed federatedly without the disclosure of data. Importantly, we prove that VFKL has a sublinear convergence rate and can guarantee the data security under the semi-honest assumption. Extensive experimental results on a variety of benchmark datasets show that VFKL is significantly faster than state-of-the-art federated learning methods when dealing with kernels, while retaining the similar generalization performance.

Algorithm highlights: It is a novel VFKL method, which is based on the decomposability of random features. It can provide a sublinear convergence rate. This algorithm demonstrates superior experimental results across various datasets. Furthermore, it is capable of resisting inference attacks.

6.1 Introduction

In order to adapt to the variety of data in modern society, current data mining methods are facing new challenges. For instance, data are often distributed across multiple data providers (such as private financial and e-commerce companies, hospitals, and government departments), each of which holds different features or aspects of the data. This kind of data situation is referred to as *vertically partitioned data*. Let's take a specific example: Suppose we have a digital finance company, an e-commerce company, and a bank collect different pieces of information about the same individual. The digital finance company has access to online consumption, loans, and repayments; the e-commerce company has access to online shopping behaviors; the bank holds customer information, such as average monthly deposits, account balance, and bills. If this individual wishes to secure a loan from the digital finance company, that company can collaboratively utilize the information stored in these three places to assess the credit risk of this financial loan. This process is summarized in Figure 6.1.

However, with the demands of government policies and corporate business profits, direct access to data from other participants would be unrealistic. Specifically, in response to growing concerns about personal privacy, the European Union has issued the General Data Protection Regulation (GDPR). For companies, on the one hand, users' data are valuable assets that they have a responsibility to protect. For instance, a recent data breach at an overseas social networking company has drawn much attention. On the other hand, real user data are useful for companies to train robust business learning models (such as recommendation systems). Therefore, it is essential to ensure privacy is not compromised when conducting federated learning on vertically partitioned data.

In the literature, there are many privacy-preserving federated learning algorithms for vertically partitioned data in various applications, for example, cooperative statistical analysis, linear regression, association rule-mining, logistic regression, random forest, XGBoost and support vector machine. From the optimization standpoint, Wan *et al.* (2007) proposed privacy-preservation gradient descent algorithm for vertically partitioned data in 2007. Zhang *et al.* (2018) proposed a feature-distributed stochastic variance reducing gradient algorithm for high-dimensional linear classification.

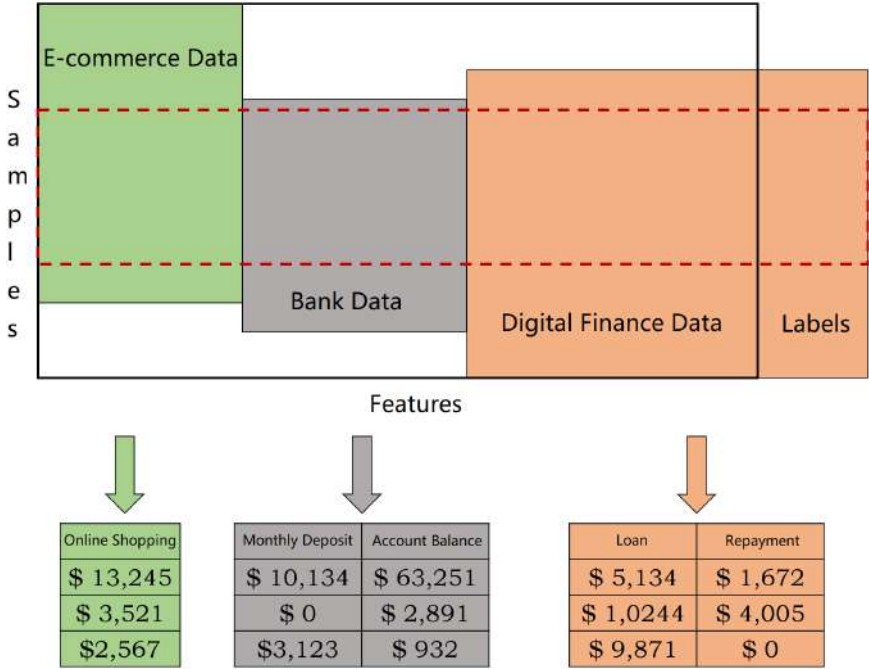


Figure 6.1. The schematic diagram of vertically partitioned data is as follows. Each sample can be viewed as an individual. Different colors of the feature dimensions for each sample represent the different information about the person held by each organization. Dashed boxes indicate the same customer information owned by multiple organizations, from which the vertically partitioned data are constructed. As our scenario is set in a digital finance company, the labels coincide with the same sample size as the digital finance company. Labels can either be binary, i.e., $(+1, -1)$ corresponding to binary classification (BC) problems, or continuous values corresponding to regression problems. In practical applications, BC problems are more common, where the given label $+1$ implies granting a loan and -1 implies not granting.

However, existing privacy-preservation federated learning algorithms assume the models are implicitly linearly separable, i.e., in the form of $f(x) = \mathcal{G} \circ h(x)$, where $\mathcal{G}$ is any differentiable function, $\{g_1, g_2, \dots, g_m\}$ is a partition of the features, and $h(x)$ is a linearly separable function of the form of $\sum_{\ell=1}^m h^\ell(w_{g_\ell}, x_{g_\ell})$. Thus, almost all the existing privacy-preservation federated learning algorithms are limited to linear models. In fact, we know that the nonlinear models often can achieve better risk prediction performance than

linear methods. Kernel methods are an important class of nonlinear approaches to handle linearly non-separable data. Kernel models usually take the form $f(x) = \sum_i^N \alpha_i k(x_i, \cdot)$ which do not satisfy the assumption of implicitly linear separability, where $k(\cdot, \cdot)$ is a kernel function. To the best of our knowledge, PP-SVMV is the only privacy-preserving method for learning vertically partitioned data with nonlinear kernels. However, PP-SVMV needs to merge the local kernel matrices in different workers to a global kernel matrix, which would cause a high communication cost. It is still an open question to train the vertically partitioned data efficiently and scalably by kernel methods while keeping data privacy.

To address this challenging problem, in this section, we propose a new Vertical Federated Kernel Learning (VFKL) algorithm for vertically partitioned data. Specifically, we use random Fourier features (RFF) to approximate the kernel mapping function and use doubly stochastic gradients (DSG) to update kernel model, which are all computed federatedly without revealing the whole data sample to each worker. We prove that VFKL can converge to the optimal solution in $\mathcal{O}(1/t)$, where t is the number of iterations. We also provide the analysis of the data security under the semi-honest assumption (i.e., Assumption 6.2). Extensive experimental results on a variety of benchmark datasets show that VFKL is significantly faster than state-of-the-art federated learning methods.

6.2 Brief Review of Doubly Stochastic Kernel Methods

In this section, we first introduce kernel methods and random Fourier feature approximation techniques. We then briefly review the DSG method and finally, discuss the relationship between the DSG method and our VFKL approach. The DSG method is a scalable and efficient kernel method that uses DSG, i.e., random samples and random features, to update the kernel function. We extend the DSG method to the case of vertically partitioned data, leading to the proposed VFKL method.

6.2.1 Problem Statement

Given a training set $\{(x_i, y_i)\}_{i=1}^N$ where $x_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$ for BC or $y_i \in \mathbb{R}$ for regression. Let $l(u, y)$ be a scalar loss function

which is convex with respect to $u \in \mathbb{R}$. (We list several common loss functions in Table 6.1 including square loss, hinge loss, and logistic loss, and here we adopt square loss in this section.) Given a positive definite kernel function $k(\cdot, \cdot)$ and the associated reproducing kernel Hilbert spaces (RKHS) $\mathcal{H}$. A kernel method tries to find a function $f \in \mathcal{H}$ for solving:

$$\arg \min_{f \in \mathcal{H}} \mathcal{L}(f) = \mathbb{E}_{(x,y)} l(f(x), y) + \frac{c}{2} \|f\|_{\mathcal{H}}^2 \quad (6.1)$$

where $c > 0$ is a regularization parameter.

Problem 6.1 is a convex optimization problem. A key feature of convex optimization problems is that all local optima are also global optima. This characteristic ensures that we won't get stuck in a local optimal solution when solving the problem. That is, if we find a local optimal solution to the problem, it must also be the global optimal solution, greatly simplifying the problem-solving process. Given this property, when seeking to maximize or minimize a convex optimization, we can directly search for points where the function's gradient is zero. However, directly solving for points where the gradient is zero can be difficult for some complex functions. In practice, we often use the Gradient Descent (GD) method to progressively approximate the optimal solution. Since GD training requires the use of all samples in the dataset to calculate the gradient of the entire convex optimization problem, this introduces extra computational costs. Hence, for some large-scale datasets, we can randomly select a mini-batch of samples to approximate the gradient of the

Table 6.1. Common loss functions used in Binary Classification (BC) and Regression (R) problems.

Name of loss	Type of task	Convex	The loss function
Square loss	BC+R	Yes	$L(f(x_i), y_i) = (f(x_i) - y_i)^2$
Logistic loss	BC	Yes	$L(f(x_i), y_i) = \log(1 + \exp(-y_i f(x_i)))$
Smooth hinge loss	BC	Yes	$L(f(x_i), y_i) = \begin{cases} \frac{1}{2} - z_i & \text{if } z_i \leq 0 \\ \frac{1}{2}(1 - z_i)^2 & \text{if } 0 < z_i < 1, \\ 0 & \text{if } z_i \geq 1 \end{cases}$ where $z_i = y_i f(x_i)$

entire convex optimization problem. This approach involves using the Stochastic Gradient Descent (SGD) method to optimize the problem.

6.2.2 A Brief Introduction to Kernel Methods

In machine learning algorithms, we typically use kernel methods to map low-dimensional, linearly inseparable data into a high-dimensional space where they become linearly separable, thus facilitating the training of improved classifiers. As the data undergoes this high-dimensional transformation, the relationship between input data and output becomes nonlinear. Classifiers using this method are typically referred to as nonlinear classifiers. Though deep neural networks are well-known nonlinear classifiers, their extensive parameterization makes them impractical for many real-world applications, especially in federated settings. Kernel-based Support Vector Machines (SVM) are among the best-performing nonlinear classifiers outside deep networks. In the following, we briefly introduce kernel functions and kernel methods.

Definition 6.1. Let $\mathcal{X}$ denote the input space (either a subset of Euclidean space or a discrete set). Let $\mathcal{H}$ represent the feature space, i.e., a Hilbert space. If there exists a mapping function, $\phi(x) : \mathcal{X} \rightarrow \mathcal{H}$, such that for all $x_i, x_j \in \mathcal{X}$, the function $k(\cdot, \cdot)$ satisfies $k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$, then k is termed a kernel function, ϕ is the mapping function, and the expression $\phi(x_i) \cdot \phi(x_j)$ represents the inner product of $\phi(x_i)$ and $\phi(x_j)$.

The uniqueness of this concept lies in our ability to define only the kernel function k during learning and prediction, without the need to explicitly define the mapping function ϕ . This is because, in general situations, calculating k directly is more feasible.

However, not all functions can serve as kernel functions. The necessary and sufficient conditions for a function to be a kernel function are given in the following:

Theorem 6.1. Let $\mathcal{X}$ be the input space, and $k(\cdot, \cdot)$ be a symmetric function defined on $\mathcal{X} \times \mathcal{X}$. Then k is a kernel function if and only if for any dataset $D = \{x_1, x_2, \dots, x_n\} \in \mathcal{X}$, the kernel matrix K is

always semi-positive definite:

$$K = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ k(x_i, x_1) & \cdots & k(x_i, x_j) & \cdots & k(x_i, x_n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_j) & \cdots & k(x_n, x_n) \end{bmatrix}. \quad (6.2)$$

Theorem 6.1 indicates that as long as the matrix corresponding to a symmetric function is semi-positive definite, it can be used as a kernel function. In practice, for a semi-positive definite kernel matrix, there always exists a corresponding mapping function ϕ . This means that any kernel function implicitly defines a RKHS. This RKHS is a complete inner product space and has the property of *reproducing kernel*: $f \cdot k(x, \cdot) = f(x), k(x, \cdot) \cdot k(y, \cdot) = k(x, y)$.

We often refer to the kernel functions corresponding to semi-positive definite kernel matrices as *positive definite kernels*. From the discussion above, we want samples to be linearly separable in the mapped feature space. Thus, the quality of the feature space directly impacts the performance of kernel methods. Consequently, the selection of kernel functions becomes crucial. As shown in Table 6.2, commonly used kernel functions include linear, polynomial, Gaussian,

Table 6.2. Common kernel functions.

Name	Expression	Parameters
Linear kernel	$k(x_i, x_j) = x_i^T x_j$	
Polynomial kernel	$k(x_i, x_j) = (x_i^T x_j)^d$	$d \geq 1$ is the degree of the polynomial. For $d = 1$, it reduces to the linear kernel
Gaussian kernel (RBF kernel)	$k(x_i, x_j) = \exp(-\frac{\ x_i - x_j\ ^2}{2\sigma^2})$	$\sigma > 0$ is the width of the Gaussian kernel
Laplacian kernel	$k(x_i, x_j) = \exp(-\frac{\ x_i - x_j\ }{\sigma})$	$\sigma > 0$
Sigmoid kernel	$k(x_i, x_j) = \tanh(\beta x_i^T x_j + \theta)$	$\tanh$ is the hyperbolic tangent function, $\beta > 0, \theta < 0$

Laplace, and sigmoid kernels. Additionally, we can obtain more kernel functions through linear combinations, direct products, etc., among existing kernel functions.

With the related definitions of kernel functions established, there is a more general result for the loss function called the representation theorem. It states that for optimization problems like (6.1), we have the following.

Theorem 6.2. *Let $\mathcal{H}$ be the RKHS corresponding to the kernel function k and $\|f\|_{\mathcal{H}}$ be the norm defined in $\mathcal{H}$ with respect to f . For any non-negative loss function $l : \mathbb{R}^m \rightarrow [0, \infty]$ and any monotonically increasing function $\Delta : [0, \infty] \rightarrow \mathbb{R}^m$:*

$$\min_{f \in \mathcal{H}} \mathcal{L}(f) = l(f) + \Delta(\|f\|_{\mathcal{H}}) \quad (6.3)$$

the solution can always be written in the form $f(x) = \sum_i \alpha_i k(x_i, \cdot)$.

For models like the linear support vector machine, the decision hyperplane can be represented as $f(x) = w^T x + b$, where w and b are model parameters. Let $\phi(x)$ denote the mapped vector of x . Consequently, the linear model becomes a nonlinear model (or linearly separable in the feature space) given by $f(x) = w^T \phi(x) + b$. This transforms the optimization objective from

$$\min_{w, b} \frac{1}{2} \|w\|^2 \quad (6.4)$$

$$\text{s.t. } y_i(w^T x_i + b) \geq 1, \quad i = 1, 2, \dots, m$$

to

$$\min_{w, b} \frac{1}{2} \|w\|^2 \quad (6.5)$$

$$\text{s.t. } y_i(w^T \phi(x_i) + b) \geq 1, \quad i = 1, 2, \dots, m$$

Accordingly, the solution changes from

$$f(x) = w^T x + b = \sum_{i=1}^m a_i y_i x_i^T x + b \quad (6.6)$$

to

$$f(x) = w^T \phi(x) + b = \sum_{i=1}^m a_i y_i \phi(x_i)^T \phi(x) + b = \sum_{i=1}^m a_i y_i k(x, x_i) + b \quad (6.7)$$

As b is a constant term, we can express the solution as a linear combination of kernel functions, as shown in Theorem 6.2.

Thus, continuing to denote ϕ as the mapping from low to high-dimensional space, the solution in Theorem 6.2 can be represented as linearly separable in the high-dimensional form, i.e., $f(x) = \omega^\top \phi(x)$, where $\omega = \sum_i \alpha_i \phi(x_i)$. In other words, the inner product of sample points x_i and x_j in the high-dimensional feature space $\phi(x_i) \cdot \phi(x_j)$ is equivalent to the kernel function calculation in their original sample space. However, traditional kernel methods are computationally slow on large-scale or high-dimensional datasets. To accelerate kernel function computations, researchers have proposed approximating it using the inner products of basis functions obtained from the Fourier transform of the kernel functions, which has achieved promising results. We delve into this method in the following section.

6.2.3 Random Fourier Features Approximation

RFF is a powerful technique to make kernel methods scalable. It uses the intriguing duality between positive definite kernels which are continuous and shift invariant (i.e., $k(x_i, x_j) = k(x_i - x_j)$) and stochastic processes, as shown in Theorem 6.3.

Theorem 6.3. *A continuous, real-valued, symmetric and shift-invariant function $k(x_i - x_j)$ on $\mathbb{R}^d$ is a positive definite kernel if and only if there is a finite non-negative measure $\mathbb{P}(\omega)$ on $\mathbb{R}^d$, such that*

$$\begin{aligned} k(x_i - x_j) &= \int_{\mathbb{R}^d} e^{i\omega^T(x_i - x_j)} d\mathbb{P}(\omega) \\ &= \int_{\mathbb{R}^d \times [0, 2\pi]} 2 \cos(\omega^T x_i + b) \cos(\omega^T x_j + b) d(\mathbb{P}(\omega) \times \mathbb{P}(b)) \end{aligned} \quad (6.8)$$

where $\mathbb{P}(b)$ is a uniform distribution on $[0, 2\pi]$ and $\phi_\omega(x) = \sqrt{2} \cos(\omega^T x + b)$.

To efficiently compute the integral form in Theorem 6.3, we approximate it using Monte Carlo sampling:

$$k(x_i - x_j) \approx \frac{1}{D} \sum_{z=1}^D \phi_{\omega_z}(x_i) \cdot \phi_{\omega_z}(x_j) = \phi_\omega(x_i) \cdot \phi_\omega(x_j) \quad (6.9)$$

where D denotes the number of RFF, and $\omega_z \in \mathbb{R}^d$ is sampled from $\mathbb{P}(\omega)$. For convenience, we stack the ω_z from the equation into a matrix ω (i.e., $\omega \in \mathbb{R}^{d \times D}$). Specifically, for the Gaussian Radial Basis Function (RBF) kernel $k(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$, $\mathbb{P}(\omega)$ follows a Gaussian distribution proportional in density to $\exp(-\frac{\sigma^2 \|\omega\|^2}{2})$. For the Laplace kernel, $\mathbb{P}(\omega)$ is a Cauchy distribution. To compute the random feature mapping ϕ , one has to compute the linear combination of the original input features (i.e., $\omega^T x + b$), which is partitionable along its axis. Such a characteristic makes the random feature approximation particularly suitable for the federated learning setting.

6.2.4 Doubly Stochastic Gradient

According to the representation theorem and properties of reproducing kernels, for the function $f = \sum_i \alpha_i k(x_i, \cdot) \in \mathcal{H}$, the gradient of $f(x)$ is

$$\nabla f(x) = \nabla \langle f, k(x, \cdot) \rangle_{\mathcal{H}} = \nabla \sum_i \alpha_i k(x_i, x) = k(x, \cdot) \quad (6.10)$$

The gradient of $\|f\|_{\mathcal{H}}^2$ is

$$\nabla \|f\|_{\mathcal{H}}^2 = \nabla \langle f, f \rangle_{\mathcal{H}} = \nabla \sum_i \sum_j \alpha_i \alpha_j k(x_i, x_j) = 2 \sum_i \alpha_i k(x_i, \cdot) = 2f \quad (6.11)$$

Thus, the stochastic gradient of $\nabla f(x)$ with respect to the random features ω (as per Equation (6.10)) can be reformulated as follows:

$$\nabla \hat{f}(x) = \hat{k}(x, \cdot) = \phi_{\omega}(x) \cdot \phi_{\omega}(\cdot) \quad (6.12)$$

Given a randomly sampled data point (x_i, y_i) and random feature ω_i , the DSG of the loss function $l(f(x_i), y_i)$ in the RKHS with respect to the random sample (x_i, y_i) and the random feature ω_i is recast as follows:

$$\xi(\cdot) = l'(f(x_i), y_i) \phi_{\omega_i}(x_i) \cdot \phi_{\omega_i}(\cdot) \quad (6.13)$$

Since $\nabla \|f\|_{\mathcal{H}}^2 = 2f$ (as per Eq.(6.11)), the stochastic gradient of $\mathcal{L}(f)$ is

$$\widehat{\xi}(\cdot) = \xi(\cdot) + cf(\cdot) = l'(f(x_i), y_i) \phi_{\omega_i}(x_i) \cdot \phi_{\omega_i}(\cdot) + cf(\cdot) \quad (6.14)$$

We have $\mathbb{E}_{(x,y)} \mathbb{E}_{\omega} \widehat{\xi}(\cdot) = \nabla \mathcal{L}(f)$. Based on the stochastic gradient (formula 6.14), we can update the solution using the step size η_t . Letting $f_1(\cdot) = \mathbf{0}$, we then get

$$\begin{aligned} f_{t+1}(\cdot) &= f_t(\cdot) - \eta_t (\xi(\cdot) + cf(\cdot)) = \sum_{i=1}^t -\eta_i \prod_{j=i+1}^t (1 - \eta_j c) \xi_i(\cdot) \\ &= \sum_{i=1}^t -\eta_i \prod_{j=i+1}^t (1 - \eta_j c) l'(f(x_i), y_i) \phi_{\omega_i}(x_i) \cdot \phi_{\omega_i}(\cdot) = \sum_{i=1}^t \alpha_i^t \phi_{\omega_i}(\cdot) \end{aligned} \quad (6.15)$$

According to formula 6.15, the function $f(\cdot)$ can be viewed as a weighted sum of $\phi_{\omega_i}(\cdot)$, where the weights are α_i^t , for $i = 1, \dots, t$. Contrary to the implicitly linearly separable assumption as mentioned in the Introduction (i.e., $f(x) = \sum_i^N \alpha_i k(x_i, x)$ in Section 6.1), the prediction function $f(x)$ does not meet this requirement. Moreover, $f(x)$ also represents the high-dimensional linearly separable form discussed in Section 6.2.2.

Discussion: The key to extending DSG to vertically partitioned data is that the computation of random features is linearly separable. As a result, we compute the random features of our portion of the vertically partitioned data on a single node. When we need to use the global kernel function to compute the global function gradient, we can reconstruct the overall random feature by accumulating the random features from individual nodes, thus approximating the entire kernel function.

6.3 Vertical Federated Kernel Learning Approach

In this section, we present a reformulation of nonlinear problems in the context of vertically partitioned data. Subsequently, we outline the entire approach's structure. Finally, a detailed description of the VFKL algorithm is provided.

6.3.1 Problem Statement

As mentioned previously, in a lot of real-world data mining and machine learning applications, the input of training sample (x, y) can be vertically partitioned into m part, i.e., $x = [x_{g_1}, x_{g_2}, \dots, x_{g_m}]$ and $x_{g_\ell} \in \mathbb{R}^{d_\ell}$ is stored on the ℓ worker and $\sum_{\ell=1}^m d_\ell = d$. According to whether the label is included in a worker, we divide the workers into two types: one is *active worker* and the other is *passive worker*, where the active worker is the data provider who holds the label of a sample and the passive worker only has the input of a sample. The active worker would be a dominating server in federated learning, while passive workers play the role of clients. We let S_ℓ denote the data stored on the ℓ worker, where the labels $\{y_i\}_{i=1}^l$ are distributed on active workers. Under this setup, the goal of the VFKL algorithm is as follows:

To collaboratively train across these active and passive nodes on the vertically partitioned data $\{S_\ell\}_{\ell=1}^m$ to solve the nonlinear learning Problem (6.1) while ensuring data privacy.

6.3.2 Algorithm Structure

Consistent with the vertical federated learning setting, each node retains its local vertically partitioned data. The term “local” here refers to individual nodes. Figure 6.2 depicts the system architecture of the VFKL method. As previously mentioned, the main idea behind VFKL is that the computation of random features can be vertically partitioned. Specifically, detailed introductions to data privacy protection, model privacy protection, and tree-structured communication are provided:

- **Data privacy:** To keep the vertically partitioned data privacy, we need to divide the computation of the value of $\phi_{\omega_i}(x_i) = \sqrt{2} \cos(\omega_i^T x_i + b)$ to avoid transferring the local $(x_i)_{g_\ell}$ to other workers. Specifically, we send a random seed to the ℓ -th worker. Once the ℓ -th worker receives the random seed, it can generate the random direction ω_i uniquely according to the random seed. Thus, we can locally compute $(\omega_i)_{g_\ell}^T (x_i)_{g_\ell} + b$ which avoids directly transferring the local data $(x_i)_{g_\ell}$ to other workers for computing $\omega_i^T x_i + b$. In the following section, we discuss that it is hard to

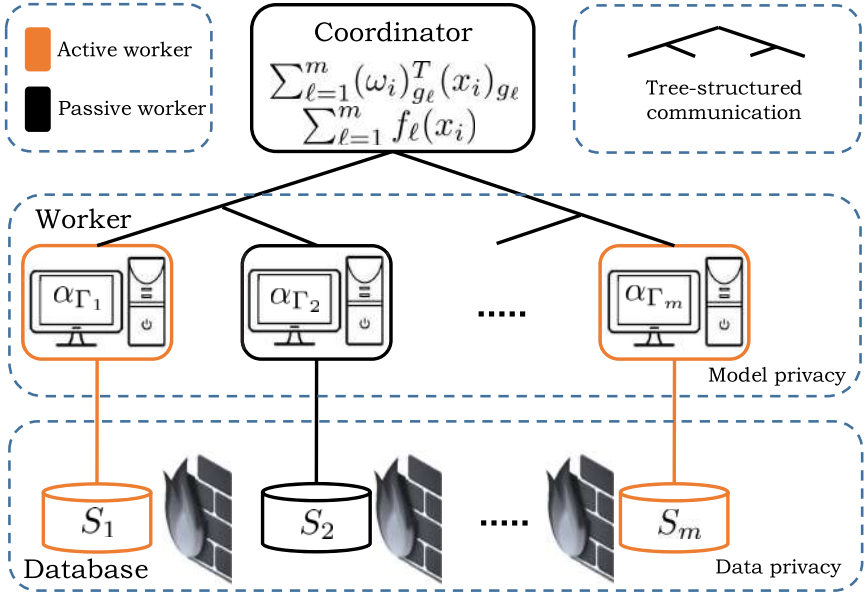


Figure 6.2. System architecture of VFKL. The aggregator in this section simply denotes the aggregation (summing) operation.

infer any $(x_i)_{g_\ell}$ according to the value of $(\omega_i)_{g_\ell}^T (x_i)_{g_\ell} + b$ from other workers.

- **Model privacy:** The model coefficients α_i are stored in different workers separately and privately. According to the location of the model coefficients α_i , we partition the model coefficients $\{\alpha_i\}_{i=1}^t$ as $\{\alpha_{\Gamma_\ell}\}_{\ell=1}^m$, where α_{Γ_ℓ} denotes the model coefficients at the ℓ -th worker and Γ_ℓ is the set of corresponding iteration indices. We do not directly transfer the local model coefficients α_{Γ_ℓ} to other workers. To compute $f(x)$, we locally compute $f_\ell(x) = \sum_{i \in \Gamma_\ell} \alpha_i \phi_{\omega_i}(x)$ and transfer it to other workers, and $f(x)$ can be reconstructed by summing over all the $f_\ell(x)$. It is difficult to infer the local model coefficients α_{Γ_ℓ} based on the value of $f_\ell(x)$ if $|\Gamma_\ell| \geq 2$. Thus, we achieve the model privacy.
- **Tree-structured communication:** Zhang *et al.* (2018) proposed an efficient tree-structured communication scheme to get the global sum which is faster than the simple strategies of star-structured communication and ring-structured communication. Take four workers as an example, we pair the workers so that

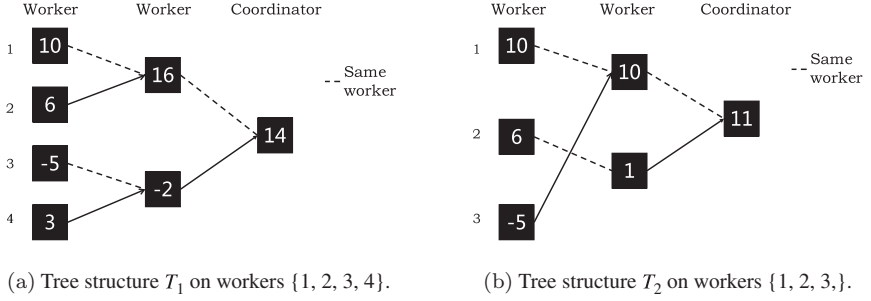


Figure 6.3. Illustration of tree-structured communication with two totally different tree structures T_1 and T_2 .

while worker 1 adds the result from worker 2, worker 3 can add the result from worker 4 simultaneously. Finally, the results from the two pairs of workers are sent to the coordinator and we obtain the global sum (see Figure 6.3(a)). If the above procedure is in a reverse order, we call it a reverse-order tree-structured communication. Note that both the tree-structured communication scheme and its reverse-order scheme are synchronous procedures.

6.3.3 Algorithm Design

To extend DSG to federated learning on vertically partitioned data while keeping data privacy, we need to carefully design the procedures of computing $\omega_i^T x_i + b$ and the solution updates:

1. **Computing $\omega_i^T x_i + b$:** We generate the random direction ω_i according to a same random seed i and a probability measure $\mathbb{P}$ for each worker. Thus, we can locally compute $(\omega_i)_{g_\ell}^T (x_i)_{g_\ell}$. To keep $(x_i)_{g_\ell}$ private, instead of directly transferring $(\omega_i)_{g_\ell}^T (x_i)_{g_\ell}$ to other workers, we randomly generate b_ℓ uniformly from $[0, 2\pi]$ and transfer $(\omega_i)_{g_\ell}^T (x_i)_{g_\ell} + b_\ell$ to another worker. After all workers have calculated $(\omega_i)_{g_\ell}^T (x_i)_{g_\ell} + b_\ell$ locally, we can get the global sum $\sum_{\ell=1}^m \left((\omega_i)_{g_\ell}^T (x_i)_{g_\ell} + b_\ell \right)$ efficiently and safely by using a tree-structured communication scheme based on the tree structure T_1 for workers $\{1, \dots, m\}$. Currently, for the ℓ -th worker, we get multiple values of b with m times. To recover the value of $\sum_{\ell=1}^m \left((\omega_i)_{g_\ell}^T (x_i)_{g_\ell} \right) + b$, we pick up one $b_{\ell'}$ from $\{1, \dots, m\} \setminus \{\ell\}$

Algorithm 6.1 VKFL algorithm on the ℓ worker

Input: $\mathbb{P}(\omega)$, local data S^ℓ , regularization parameter c , constant learning rate η , total rounds of iteration t .

- 1: **keep doing in parallel:**
- 2: Pick up an instance $(x_i)_{g_\ell}$ from the local data S^ℓ with index i .
- 3: Send i to other workers using a reverse-order tree structure T_0 .
- 4: Sample $\omega_i \sim \mathbb{P}(\omega)$ with the random seed i for all workers.
- 5: Use Algorithm 6.3 to compute $\omega_i^T x_i + b$ and locally save it.
- 6: Compute $f_{\ell'}(x_i)$ for $\ell' = 1, \dots, m$ by calling Algorithm 6.2.
- 7: Use tree-structured communication scheme based T_0 to compute $f(x_i) = \sum_{\ell=1}^m f_\ell(x_i)$.
- 8: Compute $\phi_{\omega_i}(x_i)$ according to $\omega_i^T x_i + b$.
- 9: Compute $\alpha_i = -\eta (\ell'(f(x_i), y_i) \phi_{\omega_i}(x_i))$ and locally save α_i .
- 10: Update $\alpha_j = (1 - \eta c) \alpha_j$ for all workers before i .
- 11: **end parallel loop**

Output: α_{Γ^ℓ} .

Algorithm 6.2 Compute $f_\ell(x)$ on the ℓ -th active worker

Input: $\mathbb{P}(\omega)$, α_{Γ^ℓ} , Γ_ℓ , x .

- 1: Set $f_\ell(x) = 0$.
- 2: **for** each $i \in \Gamma_\ell$ **do**
- 3: Sample $\omega_i \sim \mathbb{P}(\omega)$ with the random seed i for all workers.
- 4: Obtain $\omega_i^T x + b$ if it is locally saved, otherwise compute $\omega_i^T x + b$ by using Algorithm 6.3.
- 5: Compute $\phi_{\omega_i}(x)$ according to $\omega_i^T x + b$.
- 6: $f_\ell(x) = f_\ell(x) + \alpha_i \phi_{\omega_i}(x) \circ$,
- 7: **end for**

Output: $f_\ell(x)$

as the value of b by removing other values of b_ℓ (i.e., removing $\bar{b}_{\ell'} = \sum_{\hat{\ell} \neq \ell'} b_{\hat{\ell}}$). In order to prevent leaking any information of b_ℓ , we use a totally different tree structure T_2 (see Definition 6.2) for workers $\{1, \dots, m\} / \{\ell'\}$ to compute $\bar{b}_{\ell'} = \sum_{\hat{\ell} \neq \ell'} b_{\hat{\ell}}$. The detailed procedure of computing $\omega_i^T x_i + b$ is summarized in Algorithm 6.3.

Algorithm 6.3 Compute $\omega_i^T x_i + b$ on the ℓ -th active worker

Input: ω_i, x_i

{// This loop asks multiple workers running in parallel.}

1: **for** $\hat{\ell} = 1, \dots, m$ **do**

2: Compute $(\omega_i)_{g_{\hat{\ell}}}^T(x_i)_{g_{\hat{\ell}}}$ and randomly generate a uniform number $b_{\hat{\ell}}$ from $[0, 2\pi]$ with the seed $\sigma_{\hat{\ell}}(i)$.

3: Calculate $(\omega_i)_{g_{\hat{\ell}}}^T(x_i)_{g_{\hat{\ell}}} + b_{\hat{\ell}}$.

4: **end for**

5: Use tree-structured communication scheme based on the tree structure T_1 for workers $\{1, \dots, m\}$ to compute $\Delta = \sum_{\hat{\ell}=1}^m \left((\omega_i)_{g_{\hat{\ell}}}^T(x_i)_{g_{\hat{\ell}}} + b_{\hat{\ell}} \right) \circ$,

6: Pick up $\ell' \in \{1, \dots, m\}/\{\ell\}$ uniformly at random.

7: Use tree-structured communication scheme based on the totally different tree structure T_2 for workers $\{1, \dots, m\}/\{\ell'\}$ to compute $\bar{b}_{\ell'} = \sum_{\hat{\ell} \neq \ell'} b_{\hat{\ell}}$.

Output: $\Delta - \bar{b}_{\ell'}$

Definition 6.2 (Two totally different tree structures). For two tree structures T_1 and T_2 , they are totally different if there does not exist a subtree with more than one leaf which belongs to both T_1 and T_2 .

2. **Computing $f(x_i)$:** According to Formula (6.15), we have that $f(x_i) = \sum_{i=1}^t \alpha_i^t \phi_{\omega_i}(x_i)$. However, α_i^t and $\phi_{\omega_i}(x_i)$ are stored in different workers. Thus, we first locally compute $f_{\ell}(x_i) = \sum_{i \in \Gamma_{\ell}} \alpha_i^t \phi_{\omega_i}(x_i)$ which is summarized in Algorithm 6.2. By using a tree-structured communication scheme, we can get the global sum $\sum_{\ell=1}^m f_{\ell}(x_i)$ efficiently which is equal to $f(x_i)$ (see Line 7 in Algorithm 6.1).
3. **Updating rules:** Since α_i^t is stored in different workers, we use a communication scheme with a reverse-order tree structure to update α_i^t in each workers by the coefficient $(1 - \eta c)$ (see line 10 in Algorithm 6.1).

Based on these key procedures, we summarize our VFKL algorithm in Algorithm 6.1. Different to the diminishing learning rate used in DSG, our VFKL uses a constant learning rate η which can

be implemented more easily in the parallel computing environment. However, the convergence analysis for constant learning rate is more difficult than the one for diminishing learning rate. We give the theoretic analysis in the following section.

6.3.4 Scenario Case Study

In this section, to offer readers a more intuitive understanding of the discussed algorithm, we present a case study aligned with a real-world application. The aim of this case is to employ a machine learning model to address an issue on the genuine financial dataset UCICreditCard: predicting loan default based on various user personal details. This case assists readers in comprehending how to implement the proposed algorithm in specific tasks. Specifically, we employ the introduced VFKL algorithm on the UCICreditCard dataset to solve the optimization problem formulated as Equation (6.1).

Scenario Description: This dataset encompasses 23 attributes, which include loan amount, gender, education level, marital status, age, monthly repayment status, monthly total bill, and monthly payment amounts. The label indicates loan default where 1 signifies a default and 0 denotes non-default. The dataset holds a total of 30,000 samples. Given this dataset, consider the following scenario: Companies A, B, C, and D each possess distinct customer attributes. Company A has nine features, including loan amount, gender, age, and the monthly repayment status over 6 months. Company B holds features like loan amount, gender, age, education level, and the monthly total bill over 6 months. Company C has gender, age, marital status, and monthly payment amounts over 6 months. The clientele of each company is not identical, meaning the sample IDs differ. Notably, in this scenario, Companies A, B, and C are loan issuers keen on leveraging machine models to discern potential loan defaulters. Typically, each company would use its data to train a machine learning model. However, since each company has data for only nine attributes, lacking other potentially informative features (e.g., Company A doesn't have education level, marital status, and monthly bill data), models trained on their sole datasets might underperform. Other companies possess different sets of user data that could potentially enhance

model performance. Ideally, consolidating feature data across these entities to train a model would yield superior results. However, due to privacy regulations and business competition, direct data sharing is unfeasible. In this context, federated learning emerges as a viable solution to facilitate collaborative learning among these firms while ensuring privacy.

Modeling: Initially, we assume that data from these companies has been processed to match the vertical federated learning standard distribution, aligning with the UCICreditCard dataset. Company *A* provides nine features, specifically loan amount, gender, age, and 6 months of repayment status ($x_{g_1} = [x_1, x_2, x_6, \dots, x_{11}] \in \mathbb{R}^9$). Company *B* contributes with education level and 6 months of total bill, summing up to seven features ($x_{g_2} = [x_3, x_{12}, \dots, x_{17}] \in \mathbb{R}^7$). Company *C* supplies marital status and 6 months of total bill data, amounting to seven features ($x_{g_3} = [x_4, x_{18}, \dots, x_{23}] \in \mathbb{R}^7$). Each company maintains 30,000 common sample IDs (in our experiments, we split the dataset into training and validation sets in a 3:1 ratio). Given that all companies in this scenario are loan issuers, Companies *A*, *B*, and *C* all possess label data. Adopting an ℓ_2 regularized optimization problem for modeling, the problem in this context can be expressed as follows:

$$\min_{f \in \mathcal{H}} = \frac{1}{n} \sum_{i=1}^n l(f(x_i), y_i) + \frac{1}{c} \sum_{\ell=1}^m \|f_\ell\|_{\mathcal{H}}^2 \quad (6.16)$$

where $m = 3$, $n = 22,500$, $d = 23$, $f = \alpha\phi(x)$ (refer to Equation 6.15), $x = [x_{g_1}, x_{g_2}, x_{g_3}]$, and the feature data along with model parameters α are stored within their respective companies.

Model learning: Given the scenario, Companies *A*, *B*, and *C*, all having label data, are initiators. Hence, Companies *A*, *B*, and *C* can actively execute Algorithm 6.1. We elucidate the learning process using Company *A* as an exemplar. According to Algorithm 6.1, Company *A* randomly selects a sample with index i , ensuring all companies have a consistent random seed of i , and then samples a random direction ω_i . Subsequently, it locally invokes Algorithm 6.3 to compute $\omega_i^T x + b$ and retains it. Next, Company *A* locally invokes Algorithm 6.2 to compute its $f_\ell(x_i)$. Simultaneously, Companies *B* and *C* utilize their local model parameters and feature data to jointly

compute $\omega_i^T x + b$ and $f_\ell(x_i)$. Note that operations involving aggregation and summation in Algorithm 6.3 (lines 5 and 7) are executed on distinct tree structures, ensuring privacy during computation. Since Company A initiates the computation, it acts as a collaborator in Algorithm 6.3, computing and obtaining $\Delta - \bar{b}_{\ell'}$. In Step 9 of Algorithm 6.1, Company A computes the model parameter α based on the aggregated $\sum_{\ell=1}^m f_\ell$ and label data, stores it, and then updates its local model parameters in Step 10. Analogously, Companies B and C can execute Algorithm 6.1 and update their local models. The three entities, A, B, and C, can iteratively and concurrently repeat the mentioned process until convergence, eventually acquiring their respective models.

6.4 Theoretical Analysis

In this section, we introduce the convergence, security, and complexity analysis of VFKL. Comprehensive proofs for all claims are provided in our KDD2020 paper.¹

6.4.1 Convergence Analysis

As the basis of our analysis, our first lemma states that the output of Algorithm 6.3 is actually equal to $\omega_i^T x + b$.

Lemma 6.1. *The output of Algorithm 6.3 (i.e., $\sum_{\hat{\ell}=1}^m ((\omega_i)_{g_{\hat{\ell}}}^T(x)_{g_{\hat{\ell}}} + b_{\hat{\ell}}) - \bar{b}_{\ell'}$) is equal to $\omega_i^T x + b$, where each $b_{\hat{\ell}}$ and b are drawn from a uniform distribution on $[0, 2\pi]$, $\bar{b}_{\ell'} = \sum_{\hat{\ell} \neq \ell'} b_{\hat{\ell}}$ and $\ell' \in \{1, \dots, q\} \setminus \{\ell\}$.*

Based on Lemma 6.1, we can conclude that the federated learning algorithm VFKL can produce the same DSG as that of a DSG algorithm with constant learning rate. Thus, under Assumption 6.1, we can prove that VFKL converges to the optimal solution almost at a rate of $\mathcal{O}(1/t)$, as shown in Theorem 6.4. Note that the convergence

¹Accessible in the supplementary material of “Federated doubly stochastic kernel learning for vertically partitioned data”.

proof of the original DSG algorithm is limited to diminishing learning rate.

Assumption 6.1. Suppose the following conditions hold:

1. There exists an optimal solution, denoted as f_* , to Problem 6.1.
2. We have an upper bound for the derivative of $l(u, y)$ w.r.t. its 1st argument, i.e., $|l'(u, y)| < M$.
3. The loss function $l(u, y)$ and its first-order derivative are L -Lipschitz continuous in terms of the first argument.
4. We have an upper bound for the kernel value, i.e., $k(x, x') \leq \kappa$. We have an upper bound for random feature mapping, i.e., $|\phi_\omega(x) \cdot \phi_\omega(x')| \leq \phi$.

Theorem 6.4. Set $\epsilon > 0$, $\min\{\frac{1}{c}, \frac{\epsilon c}{4M^2(\sqrt{\kappa} + \sqrt{\phi})^2}\} > \eta > 0$, for Algorithm 6.1, with $\eta = \frac{\epsilon \vartheta}{8\kappa B}$ for $\vartheta \in (0, 1]$, under Assumption 6.1, we reach $\mathbb{E}[|f_t(x) - f_*(x)|^2] \leq \epsilon$ after

$$t \geq \frac{8\kappa B \log(8\kappa e_1/\epsilon)}{\vartheta \epsilon c} \quad (6.17)$$

iterations, where $B = \left[\sqrt{G_2^2 + G_1} + G_2\right]^2$, $G_1 = \frac{2\kappa M^2}{c}$, $G_2 = \frac{\kappa^{1/2} M(\sqrt{\kappa} + \sqrt{\phi})}{2c^{3/2}}$, and $e_1 = \mathbb{E}[\|h_1 - f_*\|_{\mathcal{H}}^2]$.

Remark 6.1. Based on Theorem 6.4, we have that for any given data x , the evaluated value of f_{t+1} at x will converge to that of a solution close to f_* in terms of the Euclidean distance. The rate of convergence is almost $\mathcal{O}(1/t)$, if eliminating the $\log(1/\epsilon)$ factor. Even though our algorithm has included more randomness by using random features, this rate is nearly the same as standard SGD. As a result, this guarantees the efficiency of the proposed algorithm.

6.4.2 Security Analysis

We discuss the data security (in other words, prevent local data on one worker leaked to or inferred by other workers) of VFKL under the *semi-honest* assumption.

Assumption 6.2 (Semi-honest security). All workers follow the protocol or algorithm to perform the correct computations. However,

they may retain records of the intermediate computation results which they may use later to infer the data of other workers.

Since each worker knows the parameter ω given a random seed, we can have a linear system of $o_j = (\omega_j)_{g_\ell}^T (x_j)_{g_\ell}$ with a sequence of trials of ω_j and o_j . It has the potential to infer $(x_j)_{g_\ell}$ from the linear system of $o_j = (\omega_j)_{g_\ell}^T (x_j)_{g_\ell}$ if the sequence of o_j is also known. We call it *inference attack*. Please see its formal definition in Definition 6.3. In this part, we prove that VFKL can prevent inference attack (i.e., Theorem 6.5).

Definition 6.3 (Inference attack). An inference attack on the ℓ th worker is to infer a certain feature group g of sample x_j which belongs to other workers without directly accessing it.

Theorem 6.5. *Under the semi-honest assumption, the VFKL algorithm can prevent inference attack.*

As discussed above, the key of preventing the inference attack is to mask the value of o_j . As described on lines 2–3 in Algorithm 6.3, we add an extra random variable $b_{\hat{\ell}}$ into $(\omega_j)_{g_\ell}^T (x_j)_{g_\ell}$. Each time, the algorithm only transfers the value of $(\omega_j)_{g_\ell}^T (x_j)_{g_\ell} + b_{\hat{\ell}}$ to another worker. Thus, it is impossible for the receiver worker to directly infer the value of o_j . Finally, the ℓ th active worker gets the global sum $\sum_{\hat{\ell}=1}^m ((\omega_j)_{g_{\hat{\ell}}}^T (x_j)_{g_{\hat{\ell}}} + b_{\hat{\ell}})$ by using a tree-structured communication scheme based on the tree structure T_1 . Thus, lines 2–4 in Algorithm 6.3 keep data privacy.

As proved in Lemma 6.1, lines 5–7 in Algorithm 6.3 are to get $\omega_j^T x + b$ by removing $\bar{b}_{\ell'} = \sum_{\hat{\ell} \neq \ell'} b_{\hat{\ell}}$ from the sum $\sum_{\hat{\ell}=1}^m ((\omega_j)_{g_{\hat{\ell}}}^T (x_j)_{g_{\hat{\ell}}} + b_{\hat{\ell}})$. To prove that VFKL can prevent the inference attack, we only need to prove that the calculation of $\bar{b}_{\ell'} = \sum_{\hat{\ell} \neq \ell'} b_{\hat{\ell}}$ on line 7 in Algorithm 6.3 does not disclose the value of $b_{\hat{\ell}}$ or the sum of $b_{\hat{\ell}}$ on a node of tree T_1 , which is indicated in Lemma 6.2.

Lemma 6.2. *In Algorithm 6.3, if T_1 and T_2 are totally different tree structures, for any worker $\hat{\ell}$, there is no risk to disclose the value of $b_{\hat{\ell}}$ or the sum of $b_{\hat{\ell}}$ to other workers.*

6.4.3 Complexity Analysis

The computational complexity for one iteration of VFKL is $\mathcal{O}(dmt)$. The total computational complexity of VFKL is $\mathcal{O}(dmt^2)$. Further, the communication cost for one iteration of VFKL is $\mathcal{O}(mt)$, and the total communication cost of VFKL is $\mathcal{O}(mt^2)$.

6.5 Experiments

In this section, we first introduce the overall setup of the experiment in detail, then give the experimental results (including the comparison of test errors over time for multiple kernel methods, the comparison of training time with the increase in training samples, and the comparison of time spent on different communication structures, comparison of box plots of multiple methods and comparison of hyperparameter selection), and finally discuss the various results.

6.5.1 Experimental Setup

To demonstrate the superiority of VFKL on federated kernel learning with vertically partitioned data, we compare VFKL with PP-SVMV, which is the state-of-the-art algorithm of the field. Additionally, we also compare with SecureBoost, which is recently proposed to generalize the gradient tree-boosting algorithm to federated scenarios. Moreover, to verify the predictive accuracy of VFKL on vertically partitioned data, we compare with the oracle model² that can access the whole data samples without the federated learning constraint. For the oracle learners, we use state-of-the-art kernel classification solvers, including LIBSVM and DSG. Finally, we include FD-SVRG, which uses a linear model, to comparatively verify the accuracy of VFKL.

Our experiments were performed on a 24-core two-socket Intel Xeon CPU E5-2650 v4 machine with 256GB RAM. We implemented our VFKL in Python, where the parallel computation was handled via MPI4py. We utilized the SecureBoost algorithm through the official unified framework.³ The code of LIBSVM is provided by the

²The Oracle model in this chapter refers to a model with a complete dataset.

³The code is available at <https://github.com/FederatedAI/FATE>.

Table 6.3. The benchmark datasets used in the experiments.

Datasets	Features	Sample size
gisette	5,000	6,000
phishing	68	11,055
a9a	123	48,842
ijcnn1	22	49,990
w8a	300	64,700
real-sim	20,958	72,309
epsilon	2,000	400,000
defaultcredit	23	30,000
givemecredit	10	150,000

LIBSVM’s homepage. We used the implementation⁴ provided by Dai *et al.* (2014) for DSG. We modified the implementation of DSG such that it uses constant learning rate.

We have summarized the datasets used in our experiments in Table 6.3, which includes seven BC datasets and two real-world financial datasets. The first seven datasets are sourced from the LIBSVM website.⁵ The “defaultcredit” (also known as “UCICreditCard”) dataset is taken from UCI.⁶ The “givemecredit” (also known as “GiveMeSomeCredit”) dataset is sourced from Kaggle.⁷

We split the dataset as 3:1 for training and testing, respectively. Note that in the experiments of the *real-sim*, *givemecredit*, and *epsilon* datasets, PP-SVMV always runs out of memory, which means this method only works when the number of instance is below around 45,000 when using the computation resources specified above. Since the training time is over 15 hours, the result of SecureBoost algorithm on *epsilon* dataset is absent.

6.5.2 Results and Discussions

In Figure 6.4, we present a comparison of the test error rates in relation to training time for VFKL and three state-of-the-art kernel

⁴The DSG code is available at https://github.com/zixu1986/Doubly_Stochastic_Gradients.

⁵LIBSVM dataset <https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>.

⁶<https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients>.

⁷<https://www.kaggle.com/c/GiveMeSomeCredit/data>.

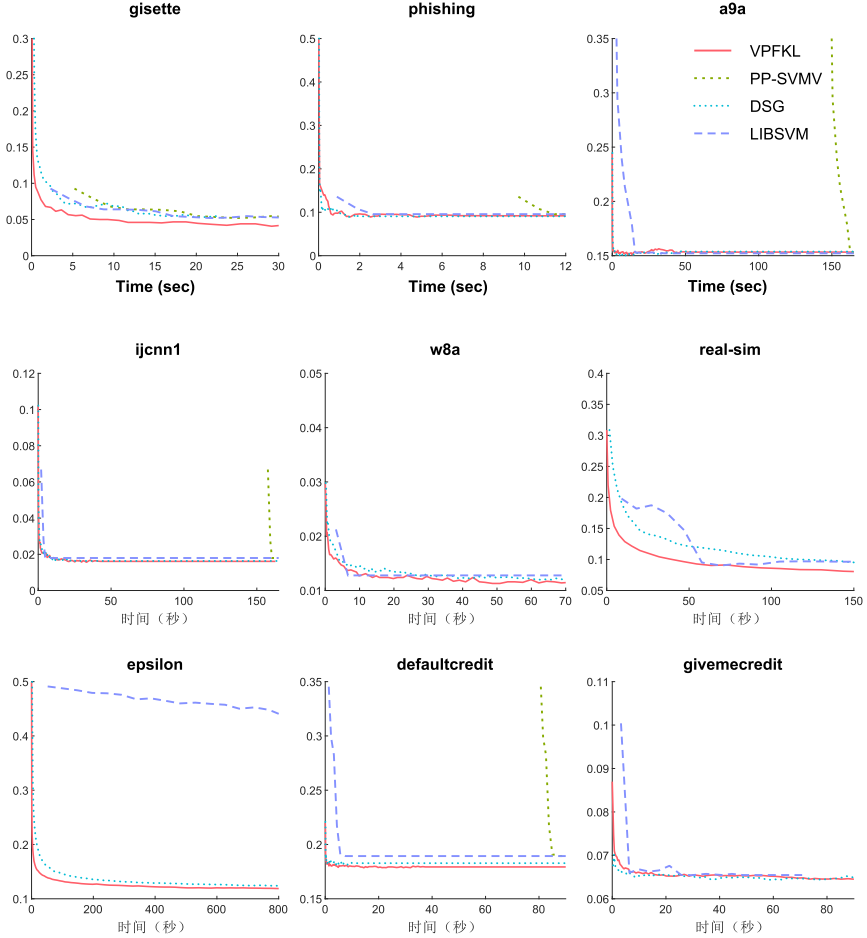


Figure 6.4. The results of binary classification. Unit: Test error/time (seconds).

methods. Clearly, our algorithm consistently achieves convergence faster than the other advanced kernel methods.

In Figure 6.5, we demonstrate the training time v.s. different training sizes of VFKL and PP-SVMV. Similarly, the absence of results for the a9a and w8a datasets for PP-SVMV is due to insufficient machine memory. It is obvious that our method has much better scalability than PP-SVMV. The reason for this edge of scalability is comprehensive, mostly because VFKL have adopted the random feature approach, which is efficient and easily parallelizable.

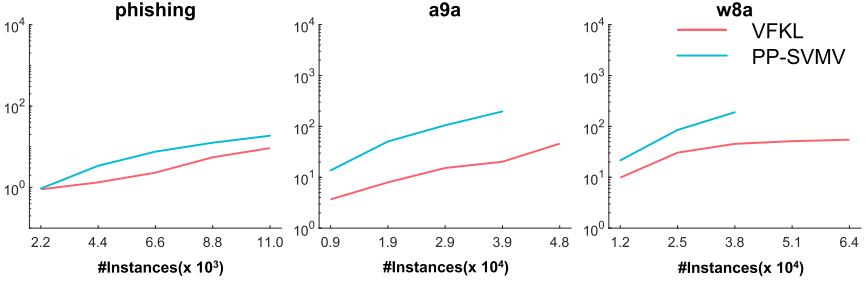


Figure 6.5. The change of training time when increasing the number of training samples. Unit: Training time (seconds)/number of samples.

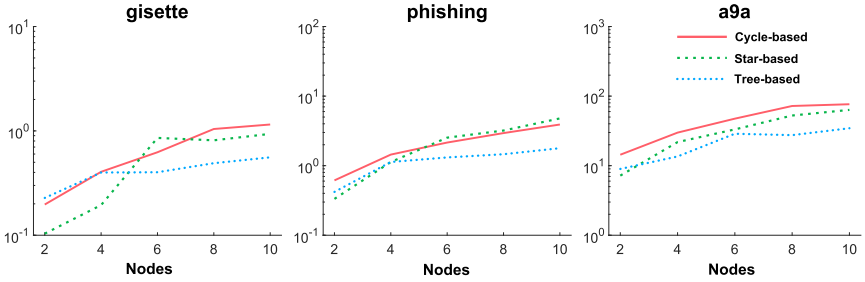


Figure 6.6. The elapsed time of different structures on three datasets. Unit: Time spent (seconds)/number of nodes.

Besides, we could also demonstrate that the communication structure used in PP-SVMV is not optimal, which means more time is spent in sending and receiving the partitioned kernel matrix.

As mentioned in previous section, VFKL used a tree-structured communication scheme to distribute and aggregate computation. To verify such a systematic design, we compare the efficiency of three commonly used communication structures, i.e., cycle-based, tree-based, and star-based communication structures. The goal of the comparison task is to compute the kernel matrix (linear kernel) of the training set of three datasets. Specifically, each node maintains a feature subset of the training set and is asked to compute the kernel matrix using the feature subset only. The computed local kernel matrices on each node are then summed by using one of the three communication structures. Our experiment compares the efficiency (elapsed communication time) of obtaining the final kernel matrix, and the results are given in Figure 6.6.

From Figure 6.6, we could make a statement that with the increase in nodes, our tree-based communication structure has the lowest communication cost. This explains the poor efficiency of PP-SVMV, which used a cycle-based communication structure, as given in Figure 6.5.

We present the test errors of three state-of-the-art kernel methods: tree-boosting method (SecureBoost), linear method (FD-SVRG), and our VFKL in Figure 6.7. All results are averaged over 10 different train-test split trials. From the results, we find that our VFKL always has the lowest test error and variance. In addition, tree-boosting method SecureBoost performs poor in high-dimensional datasets, such as *real-sim*, and the linear method normally has worse results than other kernel methods.

Since the official implementation of SecureBoost algorithm (Version 1.1) did not provide the elapsed time of each iteration, we present the total training time between our VFKL and SecureBoost in Table 6.4. From these results, we could make a statement that the SecureBoost algorithm is not suitable for the high dimension or large-scale datasets, which limits its generalization.

Finally, we provide the results of hyperparameter sensitivity analysis, as shown in Figure 6.8, for step size η and regularization parameter c . In the experiment, we choose η from $[1 : 1e^{-3}]$ and c from $[1e^1 : 1e^{-5}]$. The results show that the method performs better when the step size is smaller. Moreover, the method is insensitive to small regularization parameters.

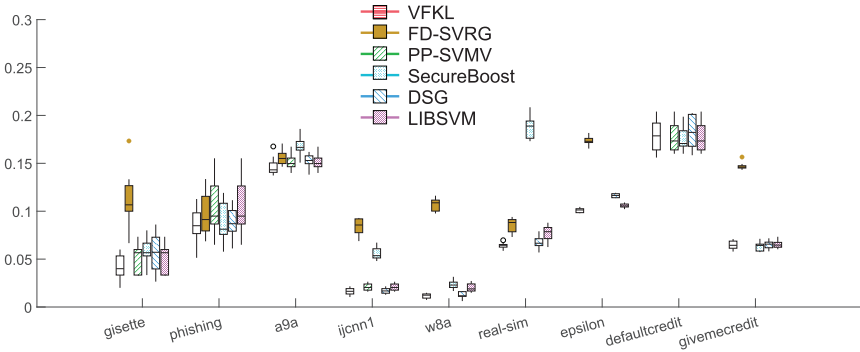
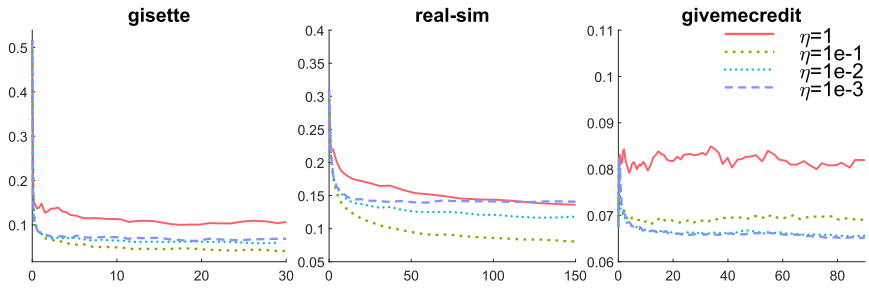
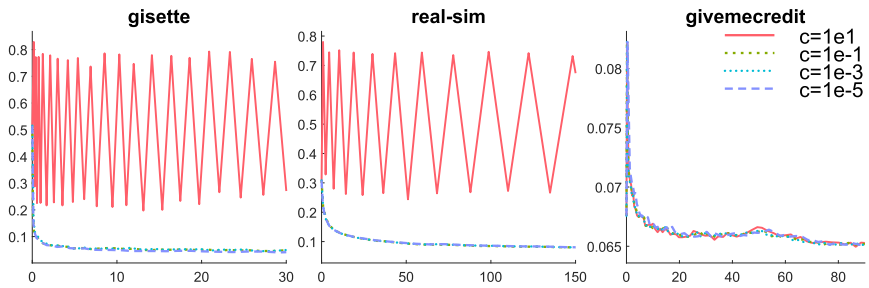


Figure 6.7. The boxplot of test errors of three state-of-the-art kernel methods (PP-SVMV, DSG and LIBSVM), tree-boosting method (SecureBoost), linear method (FD-SVRG), and our VFKL. Units: Test error/method name.

Table 6.4. The comparison of total training time between our VFKL and SecureBoost algorithm.

Datasets	VFKL (min)	SecureBoost (min)
gisette	1	46
phishing	1	10
a9a	2	15
ijcnn1	1	17
w8a	1	17
real-sim	14	65
epsilon	29	>900
defaultcredit	1	13
givemecredit	3	32

(a) Learning rate η .(b) Regularization parameter c .**Figure 6.8.** Hyperparameter sensitivity analysis of the VFKL method. Unit: Test error/time (seconds).

6.6 Conclusion

Privacy-preserving federated learning for vertically partitioned data is urgent currently in data mining and machine learning. In this chapter, we proposed a federated doubly stochastic kernel learning (i.e., VFKL) algorithm for vertically partitioned data, which breaks the limitation of implicitly linear separability used in the existing privacy-preserving federated learning algorithms. We proved that VFKL has a sublinear convergence rate and can guarantee data security under the semi-honest assumption. To the best of our knowledge, VFKL is the first efficient and scalable privacy-preserving federated kernel method. Extensive experimental results show that VFKL is more efficient than the existing state-of-the-art kernel methods for high-dimensional data while retaining the similar generalization performance.

Chapter 7

Asynchronous Vertical Federated Learning Algorithm

The privacy-preserving federated learning for vertically partitioned data has shown promising results as the solution of the emerging multi-party joint modeling application, in which the data holders (such as government branches, private finance, and e-business companies) collaborate throughout the learning process rather than relying on a trusted third party to hold data. However, existing federated learning algorithms for vertically partitioned data are limited to synchronous computation. To improve the efficiency when the unbalanced computation/communication resources are common among the parties in the federated learning system, it is essential to develop asynchronous training algorithms for vertically partitioned data while keeping the data privacy. This chapter introduces the Asynchronous Federated Stochastic Gradient Descent on Vertically Partitioned Data (AFSGD-VP) and its variance-reduced versions, namely AFSVRG-VP and AFSAGA-VP. Theoretical convergence under strong convexity conditions for these three algorithms is presented, showing no specific requirement for asynchronous latency. Additionally, this chapter discusses model privacy, data privacy, computational complexity, and communication cost from a technical perspective. Experimental results on vertically partitioned datasets not only validate the theoretical findings but also demonstrate that the asynchronous algorithms are much more efficient than their synchronous counterparts.

Algorithm highlights: This algorithm is an asynchronously updated vertical federated learning method, offering higher efficiency than synchronous approaches. It possesses theoretical convergence speed under strong convexity conditions. The algorithm encompasses the regular Stochastic Gradient Descent (SGD) method and two variants based on SVRG and SAGA (variance-reduced versions of SGD). It can defend against both exact and approximate inference attacks.

7.1 Introduction

Federated learning facilitates the collaborative model learning without the sharing of raw data and increasingly attracts attentions from both tech giants and industries where privacy protection is required. Especially, in the emerging multi-party joint modeling application, the data locate at multiple (two or more) data holders and each maintain its own records of different feature sets with common entities, which are called as vertically partitioned data. While an integrated dataset improves the performance of a trained learning model, organizations cannot share data due to legal restrictions or competition between participants. For example, a digital finance company, an e-commerce company, and a bank collect different information of the same person. The digital finance company has access to online consumption, loan and repayment information. The e-commerce company has access to the online shopping information. The bank has customer information like average monthly deposit and account balance. If the person submits a loan application to the digital finance company, it might want to evaluate the credit risk of approving this financial loan by comprehensively utilizing the information stored in all the three parties. Such scenarios have been popularly appearing in recent industrial applications and raise the need of efficient federated learning algorithms on the vertically partitioned data.

For the vertically partitioned data, the direct access to the data in other providers or sharing of the data is often prohibited due to the legal and commercial issues. For the legal reason, most countries worldwide have made laws in protection of data security and privacy. For example, the European Union made the General Data Protection Regulation (GDPR) to protect users' personal privacy

and data security. For the commercial reason, customer data is usually a valuable business asset for corporations. Thus, both of the causes require federated learning on the vertically partitioned data without the disclosure of data.

From an algorithmic design standpoint, there's a wealth of research on vertically partitioned data. For various applications, numerous privacy-preserving federated learning algorithms have been proposed, including collaborative statistical analysis, linear regression, association rule mining, logistic regression, XGBoost, random forests, support vector machines, etc. However, these algorithms for vertically partitioned data all adhere to synchronous computation, which is much less efficient than asynchronous computation due to the wastage of vast amounts of idle computational resources (as shown in Figure 7.1).

SGD algorithm and its variants have been dominant to train large-scale machine learning problems. Specifically, at each iteration, SGD independently samples a sample and uses the stochastic gradient with respect to the sampled sample to update the solution. The stochasticity makes each iteration of SGD cheap while it also causes a large variance of stochastic gradients due to random sampling. To reduce

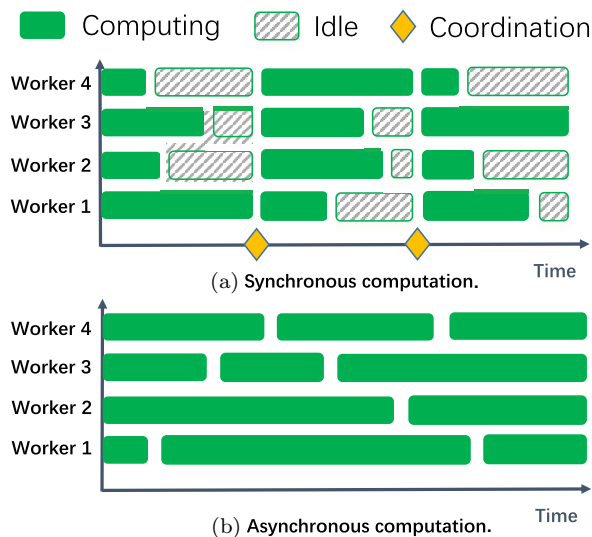


Figure 7.1. Comparison of asynchronous computation to synchronous computation.

this variance, various techniques have been employed on SGD, including SVRG, SAG, and SAGA.

However, it is still vacant for SGD and its various variance reduction variants to train vertically partitioned data in parallel and asynchronously while keeping data and model privacy. For a detailed discussion on this topic, please refer to Section 7.2. To address this challenge, this chapter introduces an Asynchronous Vertical Federated Learning algorithm (AFSGD-VP) along with its SVRG and SAGA variants. Importantly, we provide the theoretical convergence rates of the three algorithms under strong convexity conditions. We also discuss model privacy, data privacy, computational complexity, and communication costs at the technical level. Extensive experimental results on multiple vertically partitioned datasets not only validate the theoretical outcomes of AFSGD-VP and its SVRG and SAGA variants but also demonstrate that these algorithms outperform their synchronous counterparts in efficiency.

7.2 Related Work

This section briefly reviews asynchronous distributed stochastic optimization algorithms for vertically partitioned data and provides an overview of SGD-type algorithms.

7.2.1 Overview of Existing Work

Currently, many asynchronous distributed stochastic optimization algorithms are designed for large-scale learning challenges. Still, most are limited to horizontal federated learning. Specifically, the following:

For smooth convex optimization problems, Zhao *et al.* (2016) proposed an asynchronous stochastic algorithm based on SVRG, showing its linear convergence rate; Mania *et al.* (2017) introduced the perturbed iterate framework to analyze the asynchronous SVRG algorithm based on sparse gradients; Huo *et al.* (2016) extended the asynchronous SVRG algorithm to non-convex problems, demonstrating its sublinear convergence rate; Leblond *et al.* (2017) proposed an asynchronous SAGA algorithm and confirmed its linear convergence rate. For non-smooth regularization convex optimization challenges, Gu *et al.* (2018) independently proposed an asynchronous

stochastic proximal gradient algorithm based on SVRG, proving its linear convergence.

Zhang *et al.*'s (2018) FD-SVRG algorithm is the first privacy-preserving SGD method for vertically partitioned data. However, it is synchronous. As far as we know, the Feature Distributed Machine Learning (FDML) algorithm is the only asynchronous vertical federated learning algorithm. But due to the FDML's strong dependency on the upper bound of latency (assuming latency is bounded and there exists an upper limit), the theoretical value of this latency upper bound often remains unknown in practical systems. This makes the algorithm's convergence in real-world problems not always guaranteed, inevitably resulting in a loss of expressivity. Experimental results shown in Figure 7.8 also corroborate this issue by comparing classification accuracies. In contrast, the algorithm proposed in this chapter is lossless as it imposes no specific requirements or constraints on latency in both theoretical analysis and practical implementation. The primary differences between our proposed algorithm and FDML are as follows:

1. While both the FDML algorithm and the one in this chapter are asynchronous distributed algorithms, FDML adopts a Master-Slave structure. In contrast, our proposed method employs a decentralized approach. The decentralized architecture is more robust, with the Master-Slave being a static and special case of the decentralized structure. Moreover, we provide theoretical convergence rates for our algorithm.
2. FDML supports decreasing step sizes. However, our algorithm allows for a fixed step size, making it more adaptable for asynchronous distributed setups without the need to adjust the step size at every iteration.
3. FDML only supports SGD, while our proposed method supports both SGD-based as well as SVRG- and SAGA-based algorithms.

7.2.2 Review of SGD-Type Algorithms

This section provides a brief overview of SGD-type algorithms.

As mentioned earlier, SGD-like algorithms have become popular for tackling large-scale machine learning problems. We first review the update framework of SGD-like algorithms, which includes various variance reduction techniques. Specifically, given an unbiased

random gradient v (i.e., $\mathbb{E}v = \nabla f(w)$), the update rule for SGD-like algorithms can be expressed as follows:

$$w \leftarrow w - \gamma v \quad (7.1)$$

where γ represents the learning rate. Next, we introduce the specific forms of the gradients for SGD, SVRG, and SAGA.

SGD: In each iteration of the SGD algorithm, a sample (x_i, y_i) is independently chosen. The solution is then updated using the random gradient $\nabla f_i(w)$ relative to the sampled data (x_i, y_i) :

$$v = \nabla f_i(w) \quad (7.2)$$

SVRG: For the SVRG algorithm, instead of directly using the gradient $\nabla f_i(w)$, an unbiased random gradient v is used for updates as shown in the following:

$$v = \nabla f_i(w) - \nabla f_i(\tilde{w}) + \nabla f(\tilde{w}) \quad (7.3)$$

where $\tilde{w}$ denotes the value of w after a certain number of iterations.

SAGA: For the SAGA algorithm, the unbiased random gradient v is formulated as follows:

$$v = \nabla f_i(w) - \alpha_i + \frac{1}{l} \sum_{i=1}^l \alpha_i \quad (7.4)$$

where α_i is the most recent historical gradient of $\nabla f_i(w)$.

7.3 Problem Statement

In this chapter, we consider the model in a linear form of $w^T x$. Given a training set $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^l$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$ for binary classification or $y_i \in \mathbb{R}$ for regression.¹ The loss function w.r.t. the sample (x_i, y_i) and the model weights w can be

¹For classification and one-dimensional regression problems, we do not enforce that the label y belongs to $\{+1, -1\}$ or $\mathbb{R}$. In fact, the label y can be other choices, even multi-dimensional problems, such as multi-modal regression Problems (7.22) based on ℓ_2 norm regularization, as long as there is a corresponding loss function.

formulated as $L(w^T x_i, y_i)$. Thus, we consider to optimize the following regularized empirical risk minimization problem:

$$\min_{w \in \mathbb{R}^d} f(w) = \frac{1}{l} \sum_{i=1}^l \underbrace{L(w^T x_i, y_i) + g(w)}_{f_i(w)} \quad (7.5)$$

where $g(w)$ is a regularization term and $f_i : \mathbb{R}^d \rightarrow \mathbb{R}$ is considered as a smooth, possibly non-convex function. Obviously, the empirical risk minimization problem is a special case of this problem. In addition to the empirical risk minimization problem, Problem 7.5 summarizes an extensive number of important regularized learning problems, such as ℓ_2 -regularized logistic regression, ridge regression, and least squares SVM.

As mentioned previously, in a lot of real-world machine learning applications, the input of training sample (x, y) is partitioned vertically into q parts, i.e., we have a partition $\{\mathcal{G}_1, \dots, \mathcal{G}_q\}$ of d features. Thus, we have $x = [x_{\mathcal{G}_1}, x_{\mathcal{G}_2}, \dots, x_{\mathcal{G}_q}]$, where $x_{\mathcal{G}_\ell} \in \mathbb{R}^{d_\ell}$ is stored on the ℓ th worker, and $\sum_{\ell=1}^q d_\ell = d$. According to whether the label is included in a worker, we divide the workers into two types: One is active worker and the other is passive worker, where the active worker is the data provider who holds the label of a sample, and the passive worker only has the input of a sample. Let D^ℓ represent the data stored on the ℓ th party. The label y_i is only distributed on the active side. Then the algorithm goal can be summarized as follows:

Goal: Make active workers to cooperate with passive workers to solve the regularized empirical risk minimization Problem 7.5 on the vertically partitioned data $\{D^\ell\}_{\ell=1}^q$ in parallel and asynchronously with the SGD and its SVRG and SAGA variants, while keeping the vertically partitioned data private.

7.4 Asynchronous Vertical Federated Learning Algorithm

This section first introduces the framework of the asynchronous vertical federated learning algorithm. Subsequently, three algorithms of asynchronous vertical federated learning based on SGD-type

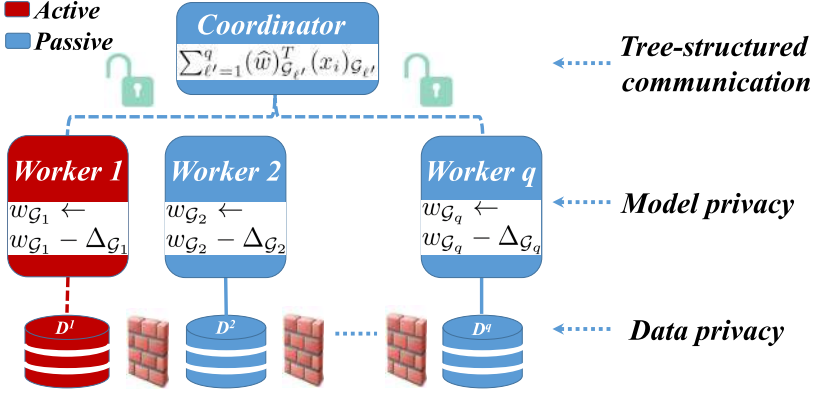


Figure 7.2. System structure of the privacy-preserving asynchronous federated learning algorithm.

algorithms are presented, namely AFSGD-VP, AFSVRG-VP, and AFSAGA-VP.

7.4.1 Algorithm Framework

As mentioned earlier, AFSGD-VP, AFSVRG-VP, and AFSAGA-VP are privacy-preserving asynchronous federated learning algorithms. Figure 7.2 outlines their system architecture. Detailed descriptions of tree-structured communication, data privacy, and model privacy are as follows:

Tree-structured communication: To obtain $w^T x_i$, we need to accumulate the local results from different workers. Zhang *et al.* (2018) proposed an efficient tree-structured communication scheme to get the global sum which is faster than the simple strategy of sending the results from all workers directly to the coordinator for sum. The efficiency of various aggregation methods depends on communication and computational efficiency. In basic aggregation methods, both communication and summation computations are borne by the main node. In contrast, with tree-structured communication, communication and summation calculations are evenly distributed among different nodes or participants. Therefore, when invoking multiple such operations simultaneously, the tree-based structure is faster than simple summation.

In this section, we use the tree-structured communication scheme to obtain $w^T x_i$. Note that, our tree-structured communication scheme works with the asynchronous pattern to obtain $w^T x_i$, which means the iteration rounds in each participant are not aligned with $w_{\mathcal{G}_\ell}$. This is considerably different from the synchronous mode used in other literature, where all $w_{\mathcal{G}_\ell}$ iterations belong to the same iteration round.

Based on the tree-structured communication scheme, we summarize the basic algorithm of computing $\sum_{\ell'=1}^q w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}}$ on the ℓ -th active worker in Algorithm 7.1.

Algorithm 7.1 Basic algorithm of computing $\sum_{\ell'=1}^q w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}}$ on ℓ th active worker

Input: w, x_i

{This loop asks multiple workers running in parallel.}

1: **for** $\ell' = 1, \dots, q$ **do**

2: Calculate $w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}}$.

3: **end for**

4: Use tree-structured communication scheme to compute $\xi = \sum_{\ell'=1}^q w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}} \circ$

Output: ξ .

Data and model privacy: To keep the vertically partitioned data and model privacy, we save the data $(x_i)_{\mathcal{G}_\ell}$ and model weights $w_{\mathcal{G}_\ell}$ in the ℓ -th worker separately and privately. We do not directly transfer the local data $(x_i)_{\mathcal{G}_\ell}$ and local model weights $w_{\mathcal{G}_\ell}$ to other workers. To obtain $w^T x_i$, we locally compute $w_{\mathcal{G}_\ell}^T(x_i)_{\mathcal{G}_\ell}$ and only transfer $w_{\mathcal{G}_\ell}^T(x_i)_{\mathcal{G}_\ell}$ to other workers for computing $w^T x$, as shown in Algorithm 7.1. It is not trivial to infer the local model coefficients $w_{\mathcal{G}_\ell}$ and $(x_i)_{\mathcal{G}_\ell}$ based on the value of $w_{\mathcal{G}_\ell}^T(x_i)_{\mathcal{G}_\ell}$ which is discussed in detail in Section 7.5.2. Therefore, the proposed asynchronous vertical federated learning algorithm achieves the data and model privacy.

Although it is not trivial to exactly infer the local model coefficients $w_{\mathcal{G}_\ell}$ and $(x_i)_{\mathcal{G}_\ell}$ based on the value of $w_{\mathcal{G}_\ell}^T(x_i)_{\mathcal{G}_\ell}$, it has the risk of approximate inference attack (see Definition 7.5). To address this issue, we propose a safer algorithm to compute $\sum_{\ell'=1}^q w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}}$ in Algorithm 7.2. Specifically, we add a random number $b^{\ell'}$ into

Algorithm 7.2 Safer algorithm of computing $\sum_{\ell'=1}^q w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}}$ on the ℓ th active worker

Input: w, x_i

{This loop asks multiple workers running in parallel.}

1: **for** $\ell' = 1, \dots, q$ **do**

2: Generate a random number $b^{\ell'}$

3: Calculate $w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}} + b^{\ell'}$

4: **end for**

5: Use tree-structured communication scheme based on the tree structure T_1 to compute $\xi = \sum_{\ell'=1}^q (w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}} + b^{\ell'})$.

6: Use tree-structured communication scheme based on the totally different tree structure T_2 from T_1 to compute $\bar{b} = \sum_{\ell'=1}^q b^{\ell'}$.

Output: $\xi - \bar{b}$

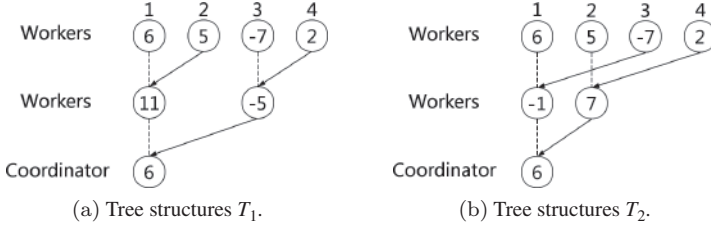


Figure 7.3. Illustration of tree-structured communication with two significantly different tree structures.

$w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}}$ and then use the tree-structured communication scheme on a tree structure T_1 to compute $\sum_{\ell'=1}^q (w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}} + b^{\ell'})$ which can improve the data and model security for the operation of transferring the value of $w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}} + b^{\ell'}$.

Finally, we need to recover the value of $\sum_{\ell'=1}^q w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}}$ from $\sum_{\ell'=1}^q (w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}} + b^{\ell'})$. In order to prevent leaking any sum of $b^{\ell'}$ of a subtree of T_1 , we use a significantly different tree structure T_2 on all workers (see Definition 7.1 and Figure 7.3) to compute $\bar{b} = \sum_{\ell'=1}^q b^{\ell'}$.

Definition 7.1 (Two significantly different tree structures).

For two tree structures T_1 and T_2 , they are significantly different if there does not exist a subtree $\widehat{T}_1$ of T_1 and a subtree $\widehat{T}_2$ of T_2 whose sizes are larger than 1 and smaller than T_1 and T_2 , respectively, such that $Leaf(\widehat{T}_1) = Leaf(\widehat{T}_2)$.

7.4.2 Algorithm Description

In this section, we introduce three asynchronous vertical federated SGD algorithms, namely AFSGD-VP, AFSVRG-VP, and AFSAGA-VP. Their specific descriptions are as follows:

AFSGD-VP Algorithm: For the AFSGD-VP algorithm, each active participant must repeatedly perform the following steps:

1. **Pick up an index:** Picks up an index i randomly from $\{1, \dots, l\}$ and obtains the local instance $(x_i)_{\mathcal{G}_\ell}$ from the local data.
2. **Compute $\widehat{w}^T x_i$:** Uses the tree-structured communication scheme with asynchronous pattern (i.e., Algorithm 7.1 or 7.2) to obtain $\widehat{w}^T x_i = \sum_{\ell'=1}^q (\widehat{w})_{\mathcal{G}_{\ell'}}^T (x_i)_{\mathcal{G}_{\ell'}}$, where $\widehat{w}$ denotes w inconsistently read from different workers and two $(\widehat{w})_{\mathcal{G}_{\ell'}}$ may be in different local iteration stages. Note that we always have that $(w)_{\mathcal{G}_\ell} = (\widehat{w})_{\mathcal{G}_\ell}$.
3. **Compute stochastic local gradient:** Based on $\widehat{w}^T x_i$, we can compute the unbiased stochastic local gradient as $\widehat{v}^\ell = \nabla_{\mathcal{G}_\ell} f_i(\widehat{w})$, where $\nabla f_i(w) = \frac{\partial L(\widehat{w}^T x_i, y_i)}{\partial (\widehat{w}^T x_i)} x_i + \nabla g(\widehat{w})$.
4. **Update:** Updates the local model weights $w_{\mathcal{G}_\ell}$ by $w_{\mathcal{G}_\ell} \leftarrow w_{\mathcal{G}_\ell} - \gamma \cdot \widehat{v}^\ell$, where γ is the learning rate.

For the detailed steps of the AFSGD-VP algorithm, see Algorithm 7.3.

AFSVRG-VP Algorithm: Like the SGD algorithm, the random gradient in AFSGD-VP has high variance due to random sampling. To address this high variance issue, AFSVRG-VP employs the SVRG technique to reduce the variance of the stochastic gradient. The specific steps of the AFSVRG-VP algorithm are shown in Algorithm 7.4. Compared to AFSGD-VP, the AFSVRG-VP algorithm has the following differences:

Algorithm 7.3 Asynchronous federated SGD algorithm (AFSGD-VP) for vertically partitioned data on the ℓ -th active worker

Input: Local data D^ℓ , Learning rate γ .

- 1: Initialize $w_{\mathcal{G}_\ell} \in \mathbb{R}^{d_\ell}$.
- 2: **Keep doing in parallel**
- 3: Pick up an index i randomly from $\{1, \dots, l\}$ and obtain the local instance $(x_i)_{\mathcal{G}_\ell}$.
- 4: Compute $(w)_{\mathcal{G}_\ell}^T(x_i)_{\mathcal{G}_\ell} \circ$
- 5: Compute $\hat{w}^T x_i = \sum_{\ell'=1}^q (\hat{w})_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}}$ based on Algorithm 7.1 or 7.2
- 6: Compute $\hat{v}^\ell = \nabla_{\mathcal{G}_\ell} f_i(\hat{w})$ based on $\hat{w}^T x_i$.
- 7: Update $w_{\mathcal{G}_\ell} \leftarrow w_{\mathcal{G}_\ell} - \gamma \cdot \hat{v}^\ell \circ$
- 8: **End parallel loop**

Output: $w_{\mathcal{G}_\ell}$

Algorithm 7.4 Asynchronous federated SVRG algorithm (AFSVRG-VP) for vertically partitioned data on the ℓ -th active worker

Input: Local data D^ℓ , Learning rate γ .

- 1: Initialize $w_{\mathcal{G}_\ell}^0 \in \mathbb{R}^{d_\ell \circ}$
- 2: **for** $s = 0, 1, 2, \dots, S - 1$ **do**
- 3: Compute the full local gradient $\nabla_{\mathcal{G}_\ell} f(w^s) = \frac{1}{l} \sum_{i=1}^l \nabla_{\mathcal{G}_\ell} f_i(w^s)$ by using tree-structured communication scheme.
- 4: $w_{\mathcal{G}_\ell}^s = w_{\mathcal{G}_\ell}^0 \circ$
- 5: **Keep doing in parallel**
- 6: Pick up an index i randomly from $\{1, \dots, l\}$ and obtain the local instance $(x_i)_{\mathcal{G}_\ell}$.
- 7: Compute $(w)_{\mathcal{G}_\ell}^T(x_i)_{\mathcal{G}_\ell}$ and $(w^s)_{\mathcal{G}_\ell}^T(x_i)_{\mathcal{G}_\ell}$.
- 8: Compute $\hat{w}^T x_i = \sum_{\ell'=1}^q (\hat{w})_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}}$ and $(w^s)^T x_i = \sum_{\ell'=1}^q (w^s)_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}}$ based on Algorithm 7.1 or 7.2.
- 9: Compute $\hat{v}^\ell = \nabla_{\mathcal{G}_\ell} f_i(\hat{w}) - \nabla_{\mathcal{G}_\ell} f_i(w^s) + \nabla_{\mathcal{G}_\ell} f(w^s)$.
- 10: Update $w_{\mathcal{G}_\ell} \leftarrow w_{\mathcal{G}_\ell}^s - \gamma \cdot \hat{v}^\ell$.
- 11: **End parallel loop**
- 12: $w_{\mathcal{G}_\ell}^{s+1} = w_{\mathcal{G}_\ell}$.
- 13: **end for**

Output: $w_{\mathcal{G}_\ell}$

1. The first one is that AFSVRG-VP is to compute the full local gradient $\nabla_{\mathcal{G}_\ell} f(w^s) = \frac{1}{l} \sum_{i=1}^l \nabla_{\mathcal{G}_\ell} f_i(w^s)$ in the outer loop which will be used as the snapshot of full gradient, where the superscript s denotes the s -th outer loop.
2. The second one is that we compute not only $\hat{w}^T x_i$ but also $(w^s)^T x_i$ for each iteration.
3. The third one is that AFSVRG-VP computes the unbiased stochastic local gradient as $\hat{v}^\ell = \nabla_{\mathcal{G}_\ell} f_i(\hat{w}) - \nabla_{\mathcal{G}_\ell} f_i(w^s) + \nabla_{\mathcal{G}_\ell} f(w^s)$.

AFSAGA-VP Algorithm: As mentioned above, the stochastic gradients in SGD have a large variance due to the random sampling. To handle the large variance, AFSAGA-VP uses the SAGA technique to reduce the variance of the stochastic gradients. The algorithmic steps are illustrated in Algorithm 7.5. Specifically, this algorithm needs to maintain a matrix of the latest historical local gradient α_i^ℓ which is achieved by the updating rule of $\hat{\alpha}_i^\ell \leftarrow \nabla_{\mathcal{G}_\ell} f_i(w)$ for each iteration. Based on the matrix of latest historical local gradients $\hat{\alpha}_i^\ell$, AFSAGA-VP computes the local gradient as $\hat{v}^\ell = \nabla_{\mathcal{G}_\ell} f_i(\hat{w}) - \hat{\alpha}_i^\ell + \frac{1}{l} \sum_{i=1}^l \hat{\alpha}_i^\ell$.

Algorithm 7.5 Asynchronous federated SAGA algorithm (AFSAGA-VP) for vertically partitioned data on the ℓ -th active worker

Input: Local data D^ℓ , Learning rate γ .

- 1: Initialize $w_{\mathcal{G}_\ell} \in \mathbb{R}^{d_\ell}$.
- 2: Compute the local gradients $\alpha_i^\ell = \nabla_{\mathcal{G}_\ell} f_i(w)$, $\forall i \in \{1, \dots, n\}$ by using tree-structured communication scheme, and locally save them.
- 3: **Keep doing in parallel**
- 4: Pick up an index i randomly from $\{1, \dots, l\}$ and obtain the local instance $(x_i)_{\mathcal{G}_\ell}$.
- 5: Compute $(w)_{\mathcal{G}_\ell}^T (x_i)_{\mathcal{G}_\ell}$.
- 6: Compute $\hat{w}^T x_i = \sum_{\ell=1}^q (\hat{w})_{\mathcal{G}_\ell}^T (x_i)_{\mathcal{G}_\ell}$ based on Algorithm 7.1 or 7.2.
- 7: Compute $\hat{v}^\ell = \nabla_{\mathcal{G}_\ell} f_i(\hat{w}) - \hat{\alpha}_i^\ell + \frac{1}{l} \sum_{i=1}^l \hat{\alpha}_i^\ell$.
- 8: Update $w_{\mathcal{G}_\ell} \leftarrow w_{\mathcal{G}_\ell} - \gamma \cdot \hat{v}^\ell$.
- 9: Update $\hat{\alpha}_i^\ell \leftarrow \nabla_{\mathcal{G}_\ell} f_i(\hat{w})$.
- 10: **End parallel loop**

Output: $w_{\mathcal{G}_\ell}$.

7.4.3 Scenario Case Study

In this section, to provide readers with a more intuitive understanding of the described algorithm, we present a case that aligns with a real-world application scenario. The objective of this case is to predict, using a machine learning model, whether a user will default on their loan based on various personal details, using the UCICreditCard dataset. This case can aid readers in understanding how to apply the proposed algorithm in specific tasks. Specifically, we employ the proposed algorithm on the UCICreditCard dataset to solve a logistic regression model of the form Equation (7.5).

Scenario Description: This dataset comprises 23 features, including loan amount, gender, educational level, marital status, age, monthly repayment status, monthly bill total, and monthly payment amount. The label indicates whether the loan will default (1 for default and 0 for non-default) and has a total of 30,000 samples. Based on this dataset, we imagine the following scenario: Four companies, A, B, C, and D, each hold different customer features. Company A has nine features like loan amount, gender, age, and 6 months of repayment status. Company B has nine features, including loan amount, gender, age, educational level, and 6 months of monthly bill total. Company C has nine features such as gender, age, marital status, and 6 months of monthly payment amount. Each company's customer base is unique. Notably, companies A, B, and C are loan issuers who wish to use machine learning to predict potential loan defaults. Typically, each company would use its data to train a model. However, because each has only nine features and lacks other crucial information, the performance might be subpar. Since other companies possess other pertinent user information beneficial for the learning model, an ideal approach would be to aggregate feature data from several companies. But direct data sharing is prohibited due to privacy laws and commercial competition. Thus, a feasible solution is to use federated learning for collaborative learning, ensuring data privacy among the four companies.

Modeling: First, we assume that each company's data have been processed into the standard distribution format for vertical federated learning, corresponding to the UCICreditCard dataset. Company A provides nine features including loan amount, gender, age, and

6 months of repayment status (i.e., $\mathbf{x}_{\mathcal{G}_1} = [x_1, x_2, x_6, \dots, x_{11}] \in \mathbb{R}^9$), B provides seven features including education level and 6 months of total bill (i.e., $\mathbf{x}_{\mathcal{G}_2} = [x_3, x_{12}, \dots, x_{17}] \in \mathbb{R}^7$), and C provides seven features including marital status and 6 months of total bill data ($\mathbf{x}_{\mathcal{G}_3} = [x_4, x_{18}, \dots, x_{23}] \in \mathbb{R}^7$). All companies share 30,000 common sample IDs, with 24,000 samples for training and 6,000 for testing. Since all companies are loan providers, Companies A, B, and C all have label data. Using a ℓ_2 regularized logistic regression model, the problem for this scenario can be represented as follows:

$$\min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w}) := \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\theta_i, y_i) + \lambda \sum_{m=1}^q \|\mathbf{w}_{\mathcal{G}_m}\|^2 \quad (7.6)$$

where $q = 3$, $n = 24,000$, $d = 23$, $\theta = \mathbf{w}^T \mathbf{x}_i = \sum_{m=1}^3 \mathbf{w}_{\mathcal{G}_m}^T(\mathbf{x}_i)_{\mathcal{G}_m}$, $\mathbf{x} = [\mathbf{x}_{\mathcal{G}_1}, \mathbf{x}_{\mathcal{G}_2}, \mathbf{x}_{\mathcal{G}_3}]$, $\mathbf{w} = [\mathbf{w}_{\mathcal{G}_1}, \mathbf{w}_{\mathcal{G}_2}, \mathbf{w}_{\mathcal{G}_3}]$, and the feature data and model parameters are respectively stored in the corresponding companies.

Model Learning: From the scenario setup, it's understood that Companies A, B, and C all have labeled data and all are the active parties. Hence, Companies A, B, and C can actively execute Algorithm 7.3. We now explain the learning process using Company A as an example. According to Algorithm 7.3, Company A randomly selects sample i and then computes locally $\theta_1 = \widehat{\mathbf{w}}_{\mathcal{G}_1}^T(\mathbf{x}_i)_{\mathcal{G}_1}$. It then invokes either Algorithm 7.1 or 7.2 to compute $\theta = \widehat{\mathbf{w}}^T \mathbf{x}_i$. Specifically, taking Algorithm 7.2 as an example, Companies B and C also utilize their current local model parameters (which, due to asynchronous updates, might not align with real-time model parameters) and their local feature data to compute $\theta_2 = \widehat{\mathbf{w}}_{\mathcal{G}_2}^T(\mathbf{x}_i)_{\mathcal{G}_2}$ and $\theta_3 = \widehat{\mathbf{w}}_{\mathcal{G}_3}^T(\mathbf{x}_i)_{\mathcal{G}_3}$, respectively. Meanwhile, Companies A, B, and C each generate random numbers δ^1 , δ^2 , and δ^3 . They then aggregate ξ and $\bar{b}$ through different tree structures (as in Steps 5 and 6 of Algorithm 7.3), i.e., $\xi = \sum_{m=1}^3 (\theta_m = \widehat{\mathbf{w}}_m^T(\mathbf{x}_i)_{\mathcal{G}_m} + \delta^m)$ and $\bar{b} = \sum_{m=1}^3 \delta^m$. Since Company A is the initiating party for this computation, it acts as a collaborator and calculates $\xi - \bar{b}$ to derive $\theta = \widehat{\mathbf{w}}^T \mathbf{x}_i$. In Step 6 of Algorithm 7.3, Company A uses the aggregated $\widehat{\mathbf{w}}^T \mathbf{x}_i$ and labeled data to compute $\vartheta = \frac{\partial \mathcal{L}(\mathbf{T}\mathbf{x}_i, y_i)}{\partial \widehat{\mathbf{w}}^T \mathbf{x}_i} = \frac{-y_i \exp(-y_i \theta)}{1 + \exp(-y_i \theta)}$. It then calculates its stochastic gradient as $\vartheta(\mathbf{x}_i)_{\mathcal{G}_1} + 2\lambda \widehat{\mathbf{w}}_1$ and updates its local model parameters in Step 7. Companies B and C can similarly execute Algorithm 7.3 and update their local models accordingly.

Companies A, B, and C repeatedly carry out the above process asynchronously and in parallel until convergence, at which point the corresponding models can be learned. For Algorithms 7.4 and 7.5, one simply needs to replace the respective algorithmic procedures.

7.5 Theoretical Analysis

In this section, we analyze the convergence, security, and complexity of AFSGD-VP, AFSVRG-VP, and AFSAGA-VP.

7.5.1 Convergence Analyses

We first make several basic assumptions and then provide the results of convergence of AFSGD-VP, AFSVRG-VP, and AFSAGA-VP.

Preliminaries: In this part, we give the assumptions of strong convexity (Assumption 7.1), different Lipschitz smoothnesses (Assumption 7.2), and block-coordinate bounded gradients (Assumption 7.3), which are standard for convex analysis.

Assumption 7.1 (Strong convexity). The differentiable function f_i ($\forall i \in \{1, \dots, l\}$) in Problem (7.5) is strongly convex with parameter $\mu > 0$, which means that $\forall w$ and $\forall w'$, we have

$$f_i(w) \geq f_i(w') + \langle \nabla f_i(w'), w - w' \rangle + \frac{\mu}{2} \|w - w'\|^2 \quad (7.7)$$

Assumption 7.2 (Lipschitz smoothness). The function f_i ($\forall i \in \{1, \dots, l\}$) in Problem (7.5) is Lipschitz smooth with constant L , which means that, $\forall w$ and $\forall w'$, we have

$$\|\nabla f_i(w) - \nabla f_i(w')\| \leq L\|w - w'\| \quad (7.8)$$

The function f_i ($\forall i \in \{1, \dots, l\}$) in Problem (7.5) is block-coordinate Lipschitz smooth w.r.t. the ℓ th block $\mathcal{G}_\ell$ with constant L_ℓ , such that, $\forall w$, and $\forall \ell \in \{1, \dots, q\}$, we have:

$$\|\nabla_{\mathcal{G}_\ell} f_i(w + \mathbf{U}_\ell \Delta_\ell) - \nabla_{\mathcal{G}_\ell} f_i(w)\| \leq L_\ell \|\Delta_\ell\| \quad (7.9)$$

where $\Delta_\ell \in \mathbb{R}^{d_\ell}$, $\mathbf{U}_\ell \in \mathbb{R}^{d \times d_\ell}$ and $[\mathbf{U}_1, \mathbf{U}_2, \dots, \mathbf{U}_q] = \mathbf{I}_d$.

According to the hypothetical definition above, define $L_{\max} = \max_{\ell=1, \dots, q} L_\ell$.

Assumption 7.3 (Block-coordinate bounded gradients). For smooth function $f_i(x)$ ($\forall i \in \{1, \dots, l\}$) in Problem (7.5), the block-coordinate gradient $\nabla_{\mathcal{G}_\ell} f_i(w)$ is called bounded if there exists a parameter G such that $\|\nabla_{\mathcal{G}_\ell} f_i(w)\|^2 \leq G$, $\forall i \in \{1, \dots, l\}$ and $\forall \ell \in \{1, \dots, q\}$.

Next, we discuss the difficult issues in the following asynchronous analysis.

Globally labeling the iterates: As shown in Algorithms 7.3–7.5, we do not globally label the iterates from different workers. Although it is fine for the implementation, it is very important to use a global iteration counter in theoretical analysis.

More specifically, the global time counter plays a fundamental role in the convergence rate analyses of AFSGD-VP, AFSVRG-VP, and AFSAGA-VP. To address this issue, we propose the strategy of “after communication” labeling, in which we update our iterate counter as one worker finishes computing $\hat{w}^T x_i$. This means that $\hat{w}_t$ (or $\hat{w}_t^s$) is the $(t + 1)$ th fully completed the computation of $\hat{w}^T x_i$. The strategy of “after communication” labeling guarantees both that it are uniformly distributed and that i_t and $\hat{w}_t$ are independent.

We define a minimum set of successive iterations of fully visiting all coordinates from the time counter t as $K(t)$ in Definition 7.2.

Definition 7.2 (Set $K(t)$). Let $\overline{K}(t) = \{\{t, t + 1, \dots, t + \sigma\} : \psi(\{t, t + 1, \dots, t + \sigma\}) = \{1, \dots, q\}\}$. The minimum set of successive iterations of fully visiting all coordinates from the time counter t is defined as $K(t) = \min_{K'(t) \in \overline{K}(t)} |K'(t)|$.

Let $\psi^{-1}(\ell, K)$ denote all the elements in K such that $\psi(\psi^{-1}(\ell, K)) = \ell$. We assume that there exists an upper bound η_1 to the size of $\psi^{-1}(\ell, K(t))$ (i.e., Assumption 7.4).

Assumption 7.4 (Bounded size of $\psi^{-1}(\ell, K(t))$). $\forall t$ and $\forall \ell \in \{1, \dots, q\}$, the sizes of all $\psi^{-1}(\ell, K(t))$ are upper bounded by η_1 , i.e., $|\psi^{-1}(\ell, K(t))| \leq \eta_1$.

Based on the definition of $K(t)$, we define the epoch number of fully visiting all coordinates for the global t th iteration as $v(t)$ and the start time counter in one epoch as $\varphi(t)$ in Definition 7.3. Our convergence rate analyses are build on the epoch number $v(t)$.

Definition 7.3 ((Epoch number $v(t)$ and start time counter $\varphi(t)$)). Let $P(t)$ be a partition of $\{0, 1, \dots, t - \sigma'\}$, where $\sigma' \geq 0$. For any $\kappa \in P(t)$ we have that, there exists $t' \leq t$ such that $K(t') = \kappa$, and there exists $\kappa_1 \in P(t)$ such that $K(0) = \kappa_1$. The epoch number $v(t)$ is defined as the maximum cardinality of $P(t)$. Given a global time counter $u \leq t$, if there exists $\kappa \in P(t)$ such that $u \in \kappa$, we define the start time counter $\varphi(t)$ as the minimum element of κ , otherwise $\varphi(t) = t - \sigma' + 1$.

Global updating rule: The updating rules (such as $w_{\mathcal{G}_\ell} \leftarrow w_{\mathcal{G}_\ell} - \gamma \cdot \hat{v}^\ell$) in Algorithms 7.3, 7.4, and 7.5 are locally working on a certain worker. To provide the convergence rate analyses of AFSGD-VP, AFSVRG-VP, and AFSAGA-VP, we need provide the global updating rules of corresponding algorithms. Due to the commutativity of the add operations used in $w_{\mathcal{G}_\ell} \leftarrow w_{\mathcal{G}_\ell} - \gamma \cdot \hat{v}^\ell$, the order in which these updates are finished in the corresponding worker is irrelevant. Hence, we provide the global updating rules of AFSGD-VP, AFSVRG-VP, and AFSAGA-VP as follows:

$$w_{t+1} = w_t - \gamma \mathbf{U}_{\psi(t)} \hat{v}_t^{\psi(t)} \quad (7.10)$$

Note that the global updating rule (7.10) which defines the relation of two adjacent iterates does not conflict with the rule of globally labeling the iterates due to the commutativity of the add operations.

Relationship between w_t and $\hat{w}_t$: As mentioned before, AFSGD-VP, AFSVRG-VP, and AFSAGA-VP use the tree-structured communication scheme with asynchronous pattern to obtain $\hat{w}^T x_i = \sum_{\ell'=1}^q (\hat{w})_{\mathcal{G}_{\ell'}}^T (x_i)_{\mathcal{G}_{\ell'}}$, where $\hat{w}$ denotes w inconsistently read from different workers. Thus, the vector $(\hat{w}_t)_{\mathcal{G}_{\ell'}}$ for $\ell' \neq \ell$ may be inconsistent to the vector $(w_t)_{\mathcal{G}_{\ell'}}$, which means that some blocks of $\hat{w}_t$ are same with the ones in w_t (e.g., $(w)_{\mathcal{G}_\ell} = (\hat{w})_{\mathcal{G}_\ell}$), but others are different to the ones in w_t . To address the challenge, we assume an upper bound to the delay of updating. Specifically, we define a set $D(t)$ of iterations, such that

$$\hat{w}_t - w_t = \gamma \sum_{u \in D(t)} \mathbf{U}_{\psi(u)} \hat{v}_u^{\psi(u)} \quad (7.11)$$

where $\forall u \in D(t)$, we have $u < t$. It is reasonable to assume that there exists an upper bound τ such that $\tau \geq t - \min\{t' | t' \in D(t)\}$ (i.e., Assumption 7.5).

Assumption 7.5 (Bounded overlap). There exists an upper bound τ such that $\tau \geq t - \min\{u | u \in D(t)\}$ for all iterations t in AFSGD-VP, AFSVRG-VP, and AFSAGA-VP.

In addition, we assume that there exist an upper bound η_2 to the size of $\psi^{-1}(\ell, D(t))$ (Assumption 7.6).

Assumption 7.6 (Bounded size of $\psi^{-1}(\ell, D(t))$). $\forall t$ and $\forall \ell \in \{1, \dots, q\}$, the sizes of all $\psi(\psi^{-1}(\ell, D(t)))$ are upper bounded by η_2 , i.e., $|\psi(\psi^{-1}(\ell, K))| \leq \eta_2$.

AFSGD-VP algorithm: We provide the convergence result of AFSGD-VP in Theorem 7.1.

Theorem 7.1. Under Assumptions 7.1–7.6, to achieve the accuracy ϵ of Problem (7.5) for AFSGD-VP, i.e., $\mathbb{E}f(w_t) - f(w^*) \leq \epsilon$, we set

$$\gamma = \frac{-L_{\max} + \sqrt{L_{\max}^2 + \frac{2\mu\epsilon(L^2q\eta_1^2 + \eta_2L^2\tau)}{G\eta_1q}}}{2L^2(q\eta_1^2 + \eta_2\tau)} \quad (7.12)$$

and the epoch number $v(t)$ should satisfy the following condition:

$$v(t) \geq \frac{2}{\mu} \frac{2L^2(q\eta_1^2 + \eta_2\tau)}{-L_{\max} + \sqrt{L_{\max}^2 + \frac{2\mu\epsilon(L^2q\eta_1^2 + \eta_2L^2\tau)}{G\eta_1q}}} \log \left(\frac{2(f(w_0) - f(w^*))}{\epsilon} \right) \quad (7.13)$$

Remark 7.1. Theorem 7.1 shows that, the convergence rate of AFSGD-VP is $\mathcal{O}(\frac{1}{\sqrt{\epsilon}} \log(\frac{1}{\epsilon}))$. The theorem shows that if we try to obtain a more accurate solution with a smaller stepsize, the convergence rate slows down.

AFSVRG-VP algorithm: We provide the convergence result of AFSVRG-VP in Theorem 7.2.

Theorem 7.2. Under Assumptions 7.1–7.6 to achieve the accuracy ϵ of Problem (7.5) for AFSVRG-VP, i.e., $\mathbb{E}f(w_t) - f(w^*) \leq \epsilon$, let

$C = (\eta_1 \gamma L^2 q \eta_1 + L_{\max}) \frac{\gamma^2}{2}$ and $\rho = \frac{\gamma \mu}{2} - \frac{16L^2 \eta_1 q C}{\mu}$, we choose γ such that

$$\rho < 0 \quad (7.14)$$

$$\frac{8L^2 \eta_1 q C}{\rho \mu} \leq 0.5 \quad (7.15)$$

$$\gamma^3 \left(\left(\frac{1}{2} + \frac{2C}{\gamma} \right) \eta_2 \tau + 4 \frac{C}{\gamma} \eta_1^2 q \right) \frac{\eta_1 q L^2 G}{\rho} \leq \frac{\epsilon}{8} \quad (7.16)$$

the inner epoch number should satisfy $v(t) \geq \frac{\log 0.25}{\log(1-\rho)}$, and the outer loop number should satisfy $S \geq \frac{\log \frac{2(f(w_0) - f(w^*))}{\log \frac{\epsilon}{4}}}{\log \frac{1}{3}}$.

Remark 7.2. Theorem 7.2 shows that the convergence rate of AFSVRG-VP is $\mathcal{O}(\log(\frac{1}{\epsilon}))$.

AFSAGA-VP algorithm: We provide the convergence result of AFSAGA-VP in Theorem 7.3.

Theorem 7.3. Under Assumptions 7.1–7.6, to achieve the accuracy ϵ of Problem (7.5) for AFSAGA-VP, i.e., $\mathbb{E}f(w_t) - f(w^*) \leq \epsilon$, let $c_0 = ((\frac{\eta_2}{2} + 3(\gamma q \eta_1^2 + L_{\max})(\eta_1 + 2\eta_2))\tau + (\gamma L^2 q \eta_1^2 + 8L_{\max})\eta_1 q \eta_1)\gamma^4 L^2 \eta_1 q G$, $c_1 = (\gamma L^2 q \eta_1^2 + L_{\max})\gamma^2 \eta_1 q 2L^2$, and $\rho \in (1 - \frac{1}{l}, 1)$, we choose γ such that

$$\frac{4c_0}{\gamma \mu (1 - \rho) \left(\frac{\gamma \mu^2}{4} - 2c_1 - c_2 \right)} \leq \frac{\epsilon}{2} \quad (7.17)$$

$$0 < 1 - \frac{\gamma \mu}{4} < 1 \quad (7.18)$$

$$-\frac{\gamma \mu^2}{4} + 2c_1 + c_2 \left(1 + \frac{1}{1 - \frac{1-\frac{1}{l}}{\rho}} \right) \leq 0 \quad (7.19)$$

$$-\frac{\gamma \mu^2}{4} + c_2 + c_1 \left(2 + \frac{1}{1 - \frac{1-\frac{1}{l}}{\rho}} \right) \leq 0 \quad (7.20)$$

the epoch number should satisfy $v(t) \geq \frac{\log \frac{2(2\rho - 1 + \frac{\gamma \mu}{4})(f(w_0) - f(w^*))}{\epsilon(\rho - 1 + \frac{\gamma \mu}{4}) \left(\frac{\gamma \mu^2}{4} - 2c_1 - c_2 \right)}}{\log \frac{1}{\rho}} \circ$

Remark 7.3. Theorem 7.2 shows that, the convergence rate of AFSAGA-VP is $\mathcal{O}(\log(\frac{1}{\epsilon}))$.

7.5.2 Security Analysis

We discuss the data and model security (in other words, prevent local data and model on one worker leaked to or inferred by other workers) of AFSGD-VP, AFSVRG-VP, and AFSAGA-VP under the semi-honest assumption. Note that the semi-honest assumption (i.e., Assumption 7.7) is commonly used in related works.

Assumption 7.7 (Semi-honest security). All workers will follow the algorithm to perform the correct computations. However, they may retain records of the intermediate computation results which they may use later to infer the other work's data and model.

Before discussing the data and model privacy in detail, we first introduce the concepts of exact and approximate inference attacks in Definitions 7.4 and 7.5.

Definition 7.4 (Exact inference attack). An exact inference attack on the ℓ -th worker is to exactly infer one sample x or model w which belongs to other workers without directly accessing it.

Definition 7.5 (ϵ -approximate inference attack). An ϵ -approximate inference attack on the ℓ th worker is to infer one sample x or model w as $\hat{x}_{\mathcal{G}}$ or $\hat{w}_{\mathcal{G}}$ with the accuracy of ϵ (i.e., $\|\hat{x}_{\mathcal{G}} - x_{\mathcal{G}}\|_{\infty} \leq \epsilon$ or $\|\hat{w}_{\mathcal{G}} - w_{\mathcal{G}}\|_{\infty} \leq \epsilon$) which belongs to other workers without directly accessing it.

Security analysis based on Algorithm 7.1: First, we show that AFSGD-VP, AFSVRGVP, and AFSAGA-VP based on Algorithm 7.1 can prevent the exact inference attack but has the risk of approximate inference attack.

Specifically, in order to infer the information of $(w_t)_{\mathcal{G}_{\ell}}$, we have a sequence of linear system of $o_t = (w_t)_{\mathcal{G}_{\ell}}^T (x_i)_{\mathcal{G}_{\ell}}$ while only o_t are known. Thus, it is impossible to infer the exact information of $(w_t)_{\mathcal{G}_{\ell}}$ from the linear system of $o_t = (w_t)_{\mathcal{G}_{\ell}}^T (x_i)_{\mathcal{G}_{\ell}}$ even the size of feature group is one. Similarly, we cannot infer the exact information of $(x_i)_{\mathcal{G}_{\ell}}$.

However, it has the potential to approximately infer $(w_t)_{\mathcal{G}_\ell}$ from the linear system of $o_j = w_{\mathcal{G}_\ell}^T(x_i)_{\mathcal{G}_\ell}$ if the size of feature group is one. Specifically, if we know the region of $(x_i)_{\mathcal{G}_\ell}$ as $\mathcal{I}$, we can have that $o_j/w_{\mathcal{G}_\ell} \in \mathcal{I}$ which can infer $w_{\mathcal{G}_\ell}$ approximately. Further, we can infer $(x_i)_{\mathcal{G}_\ell}$ approximately. We say that Algorithm 7.1 has the risk of approximate inference attack.

Security analysis based on Algorithm 7.2: Next, we show that AFSGD-VP, AFSVRGVP, and AFSAGA-VP based on Algorithm 7.2 can prevent the approximate inference attack.

As discussed above, the key to preventing the approximate inference attack is to mask the value of o_j . As described in lines 2–3 of Algorithm 7.2, we add an extra random variable $b^{\ell'}$ into $w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}}$ and transfer the value of $w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}} + b^{\ell'}$ to another worker. This operation makes the received part not directly get the value of o_j . Finally, the ℓ th active worker gets the global sum $\sum_{\ell'=1}^q \left(w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}} + b^{\ell'} \right)$ by using a tree-structured communication scheme based on the tree structure T_1 . Thus, the lines 2–5 of Algorithm 7.2 keep data privacy.

Line 6 of Algorithm 7.2 is trying to get $w^T x$ by removing $\bar{b} = \sum_{\ell'=1}^q b^{\ell'}$ from the sum $\sum_{\ell'=1}^q \left(w_{\mathcal{G}_{\ell'}}^T(x_i)_{\mathcal{G}_{\ell'}} + b^{\ell'} \right)$. To prove that Algorithm 7.2 can reduce the risk of approximate inference attack, we only need to prove that the calculation of $\bar{b} = \sum_{\ell'=1}^q b^{\ell'}$ does not disclose the value of $b^{\ell'}$ or the sum of $b^{\ell'}$. Please refer to Lemma 7.1 for details.

Lemma 7.1. *Using a tree structure T_2 on all workers which is significantly different to the tree T_1 to compute $\bar{b} = \sum_{\ell'=1}^q b^{\ell'}$, there is no risk to disclose the value of $b^{\ell'}$, or the sum of $b^{\ell'}$.*

7.5.3 Complexity Analysis

We give the computational complexities and communication costs of AFSGD-VP, AFSVRG-VP, and AFSAGA-VP as follows:

- The computational complexity for one iteration of AFSGD-VP is $\mathcal{O}(d+q)$. Thus, the total computational complexity of AFSGD-VP is $\mathcal{O}((d+q)t)$, where t denotes the iteration number. Further, the communication cost for one iteration of AFSGD-VP is $\mathcal{O}(q)$, and the total communication cost is $\mathcal{O}(qt)$.

- For AFSVRG-VP, the computational complexity and communication cost of line 3 are $\mathcal{O}((d+q)l)$ and $\mathcal{O}(ql)$, respectively. Assume that the inner loop number of AFSVRG-VP is t . Thus, the total computational complexity of AFSVRG-VP is $\mathcal{O}((d+q)(l+t)S)$, and the communication cost is $\mathcal{O}(q(l+t)S)$.
- For AFSAGA-VP, the computational complexity and communication cost of line 2 are $\mathcal{O}((d+q)l)$ and $\mathcal{O}(ql)$, respectively. Assume that the loop number of AFSAGA-VP is t . Thus, the total computational complexity of AFSAGA-VP is $\mathcal{O}((d+q)(l+t))$, and the communication cost is $\mathcal{O}(q(l+t))$.

7.6 Experimental Results

In the experiments, we not only validated the theoretical results of AFSGD-VP, AFSVRG-VP, and AFSVRG-VP but also demonstrated that our algorithms are more efficient than their synchronous counterparts, namely FSGD-VP, FSVRG-VP, and FSAGA-VP.

7.6.1 Experimental Setup

All our experiments were conducted on 16 instance nodes of Amazon Cloud EC2, with 8 instances located in Virginia and 8 in Ohio. Each node is equipped with 8 Intel Xeon vcpus, 32GB of RAM, and a 25 GB network bandwidth. For communication, we utilized MPI and implemented efficient matrix operations with Armadillo v9.700.3. In all the experiments, the regularization coefficient was set to $\lambda = 1e^{-4}$, and an optimal learning rate γ was selected from the set $\{5e^{-1}, 1e^{-1}, 5e^{-2}, 1e^{-2}, \dots\}$. We assumed the presence of one straggler node that operates at a speed 200% slower, or equivalently 1/3 the speed of a typical working node, to simulate real-world scenarios. In practice, it's common in federated learning systems for different participants to possess varying computation and communication capabilities and resources.

Datasets: To fully demonstrate the scalability of our asynchronous vertical federated learning algorithms, we conduct experiments on eight datasets as summarized in Table 7.1 for classification and regression tasks. Two real and relatively small financial datasets, UCICreditCard (Credit1) and GiveMeSomeCredit (Credit2), are

Table 7.1. The datasets used in the experiments.

	Classification tasks						Regression tasks	
	Financial		Large-scale		Multi-class			
	Credit1	Credit2	rcv1.binary	url	news20	rcv1.multiclass	E2006-tfidf	Year
#Categories	2	2	2	2	20	53	—	—
#Train	24,000	96,257	677,399	1,916,904	15,935	518,571	16,087	463,715
#Test	6,000	24,012	20,242	479,226	3,993	15,564	3,308	51,630
#Feature	90	92	47,236	3,231,961	62,061	47,236	150,360	90

from the Kaggle² website. The other six datasets are from the LIB-SVM³ website. We split news20 and url datasets into training data and testing data randomly with a ratio of 4:1. We also use rcv1's testing data for training and training data for testing as there are more instances in the testing data.

Problem statement: We have made corresponding comparisons on classification and regression tasks, where the algorithm FSVRG-VP is almost the same as the algorithm FD-SVRG. For the classification tasks, we consider the ℓ_2 -norm regularized logistic regression model as follows:

$$\min_{\mathbf{w}} f(\mathbf{w}) = \frac{1}{l} \sum_{i=1}^l \log(1 + e^{-y_i \mathbf{w}^T x_i}) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \quad (7.21)$$

Among them, the data $x_i \in \mathbb{R}^d$ and the corresponding label $y_i = \pm 1$. For multi-classification problems, consider the following ℓ_2 -norm regularized multi-nomial logistic regression modulo problem:

$$\min_{\mathbf{w} \in \mathcal{R}^{d \times c}} f(\mathbf{w}) = \frac{1}{l} \sum_{i=1}^l \left[\log \left(\sum_{j=1}^c e^{\mathbf{w}_{\cdot j}^T x_i} \right) - \sum_{j=1}^c Y_{ij} \mathbf{w}_{\cdot j}^T x_i \right] + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \quad (7.22)$$

where c is the number of categories, $\mathbf{w}_{\cdot j}$ is the j th column of $\mathbf{w}$, and label $y_i \in \{1, 2, \dots, c\}$, for $Y \in \mathcal{R}^{l \times c}$ if $j = y_i$, $Y_{ij} = 1$, otherwise $Y_{ij} = 0$.

For the regression tasks, we use the ridge linear regression method with ℓ_2 -norm regularization as follows:

$$\min_{\mathbf{w}, b} f(\mathbf{w}, b) = \frac{1}{l} \sum_{i=1}^l (\mathbf{w}^T x_i + b - y_i)^2 + \frac{\lambda}{2} (\|\mathbf{w}\|_2^2 + b^2) \quad (7.23)$$

Implementation details: Our asynchronous algorithms are implemented under the decentralized framework, where a worker owns its

²<https://www.kaggle.com/datasets>.

³<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>.

own part of data and model parameters. There is no master node for aggregating data/features/gradients which may lead to undesired user information disclosure. Instead, we utilize a coordinator as in Figure 7.2 to collect the product computed from local data and parameters from other workers. Each worker node can independently call the coordinator to enable the asynchronous model update. The aggregation of local product is performed in a demand-based manner, which means that only when a worker node needs to update its local parameter will it request the coordinator to pull the local product from other worker nodes. Different from horizontal federated learning, it will be much harder for an attacker to restore the information of the user data in a worker node using the local product than the gradient.

We note that if the dimensionality of the data in Problem (7.5) is d , the magnitude of the gradient is also d . But no matter how large the dimensionality of the data is, the dimension size of the local product is 1, which means that the gradient contains more data information than the local product. Therefore, it is easier to infer the characteristics of the original data by using the gradient than by using the local product. Data reconstruction attacks can use generative adversarial networks to solve optimization problems, thereby reconstructing data using gradient information.

Specifically, in our asynchronous algorithms, each worker node performs computation rather independently. The main thread of a worker process performs the major workload of gradient computation and model update operation. Another listener thread keeps listening for the request and sends back the local product to the requesting source. The computation diagram can be summarized as follows for a worker:

1. Randomly select an index of the data.
2. Call the coordinator to broadcast the index to the listeners of other workers.
3. Reduce the sum of the local product back from the listeners.
4. Perform gradient computation and model parameters update.

Note that the local product is computed based on a worker's current parameters. Overall speaking, however, some workers may

have updated their parameters more times than other workers. The broadcast and reduce operation are also realized in a tree-structured scheme to reduce communication costs.

7.6.2 Results and Discussions

Classification tasks: We first compared our asynchronous federated learning algorithm with synchronous version on financial datasets to demonstrate the ability to address real application. In asynchronous algorithms, each worker saves its local parameters every fixed interval for testing. In the synchronous setting, each worker saves the parameters every fixed number of iterations as all the workers run at the same pace. We follow this scheme for the other experiments.

The original total numbers of features of UCICreditCard and GiveMeSomeCredit dataset are 23 and 10, respectively. We apply one-hot encoding for categorical features and standardize other features column-wisely. The number of features becomes 90 and 92 respectively after the simple data preprocessing.

Four worker nodes are used in this part of the experiment. As shown by the left two figures in Figure 7.4, our asynchronous vertical algorithms consistently surpass their synchronous counterparts. The y-axis function sub-optimality represents the error of objective function to the global optimal. The shape of the convergence curve is first determined by the optimization method we choose, i.e., SGD, SVRG, and SAGA. The error precision of SGD is usually higher than SVRG, while that of SAGA is similar to SVRG. Then the convergence speed is mostly influenced by the computation and communication complexity. In asynchronous settings, there is no inefficient idle time to wait for other workers, so the update frequency is much higher, which results in faster convergence speed of our asynchronous algorithm with regard to wall clock time.

Previous experiments show that our asynchronous federated learning algorithms could address real financial problems more efficiently. In this part, we use large-scale benchmark datasets, i.e., large number of data instances and high-dimensional features, for further validations. In our experiments, eight worker nodes are used for experiments on rcv1.binary, news20, and rcv1.multiclass datasets;

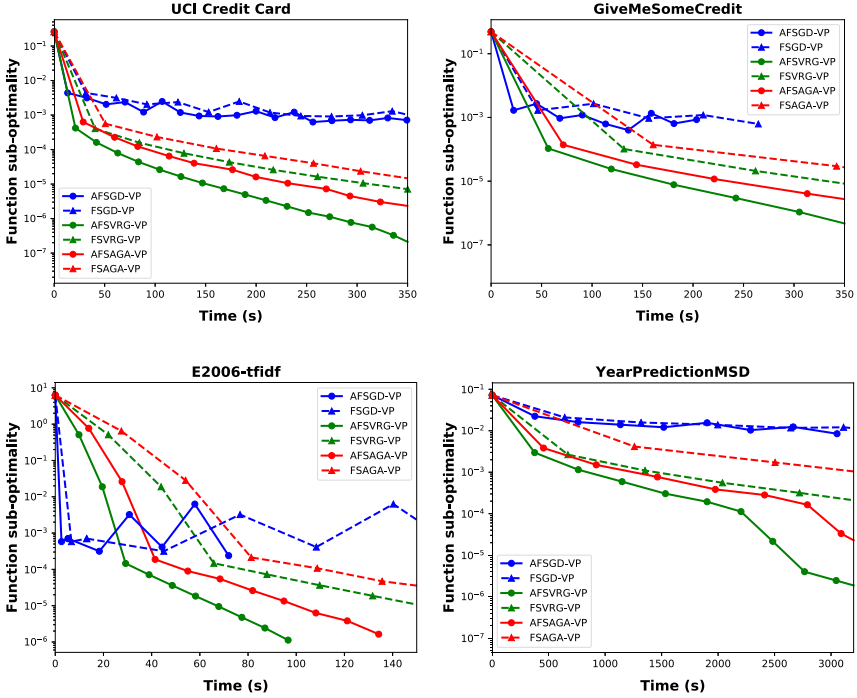


Figure 7.4. Convergence of different algorithms for classification task (left two) and regression task (right two).

16 worker nodes are used for experiments on url dataset. The results are visualized in Figure 7.5. Our asynchronous SGD, SVRG, and SAGA still surpass their synchronous counterparts in the experiments on all the four datasets.

Regression tasks: To further illustrate the advantages asynchronous algorithms can scale to various tasks, we also conduct experiments on regression problems, as shown by the right two figures in Figure 7.4. Both the E20060-tfidf with a smaller number of data instances but a larger number of features and the YearPredictionMSD with larger number of instances but a smaller number of features are tested. Similar conclusions as previous can be reached.

Asynchronous efficiency: The speedup results of asynchronous algorithms compared with synchronous ones are summarized in Table 7.2. The speedup is computed based on the time when the

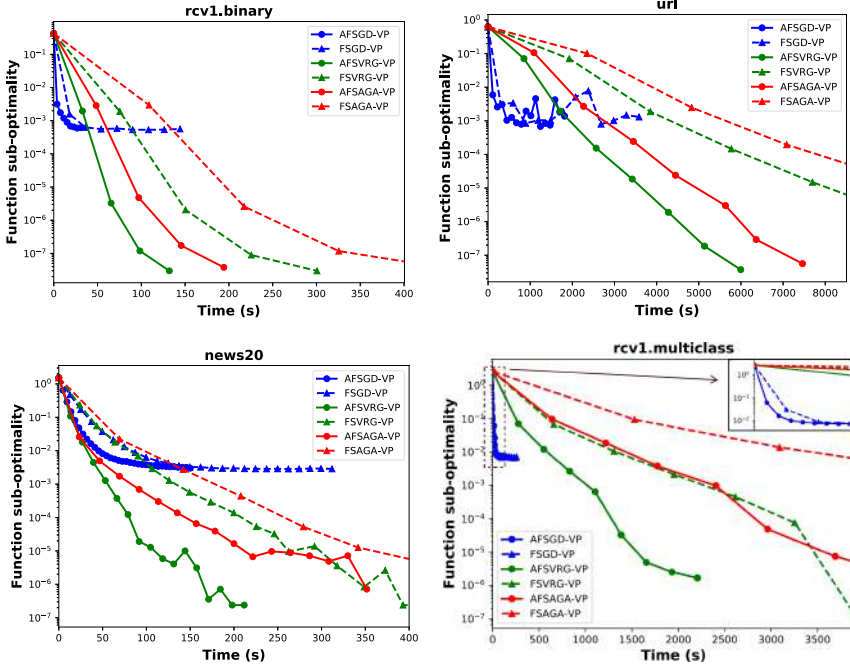


Figure 7.5. Convergence of different algorithms for binary classification task (left two) and multi-class classification (right two) task on more largescale datasets.

algorithm reaches a certain precision of optimality ($1 \times e^{-4}$ for SVRG and SAGA; $1 \times e^{-2}$ or $1 \times e^{-3}$ for SGD based on different datasets).

To further analyze the efficiency of our asynchronous algorithms, we quantify the composition of the time consumption of asynchronous and synchronous algorithms as in Figure 7.7. The execution time and update frequency are scaled by those of the straggler in the synchronous algorithm. The computation time of stragglers is much higher than non-stragglers, which leads to a large amount of synchronization time for non-stragglers in synchronous algorithms. While in our asynchronous algorithms, non-stragglers pull the update-to-date product information from stragglers without waiting the straggler to finish its current iteration. As a result, the synchronization time is eliminated and we can achieve a large gain in terms of the update frequency.

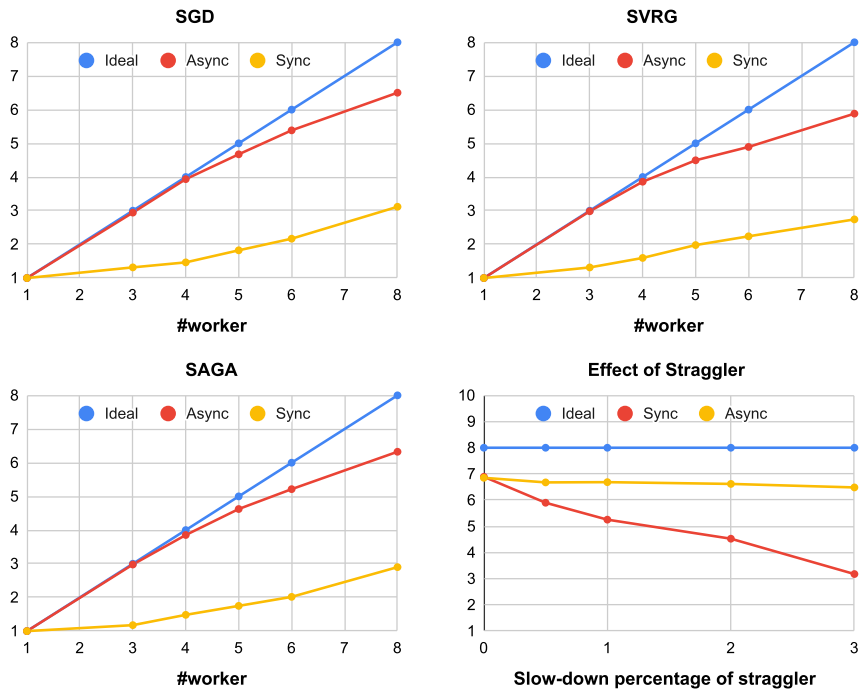


Figure 7.6. Scalability (url dataset for classification task). The y-axis demonstrates run-time speedup.

Table 7.2. Asynchronous speedup.

Dataset	Speedup		
	SGD	SVRG	SAGA
UCICreditCard	2.14	2.49	1.79
GiveMeSomeCredit	2.40	2.26	2.38
rcv1.binary	2.02	2.30	2.22
url	2.11	2.64	2.05
news20	2.85	1.83	1.95
rcv1.multiclass	2.32	2.81	2.61
E2006-tfidf	2.24	2.12	2.02
YearPredictionMSD	2.26	2.15	2.38

Scalability: The scalability in terms of number of workers is shown in Figure 7.6. The right-hand side shows that asynchronous methods scale to hysteresis effects. Synchronous algorithms cannot address

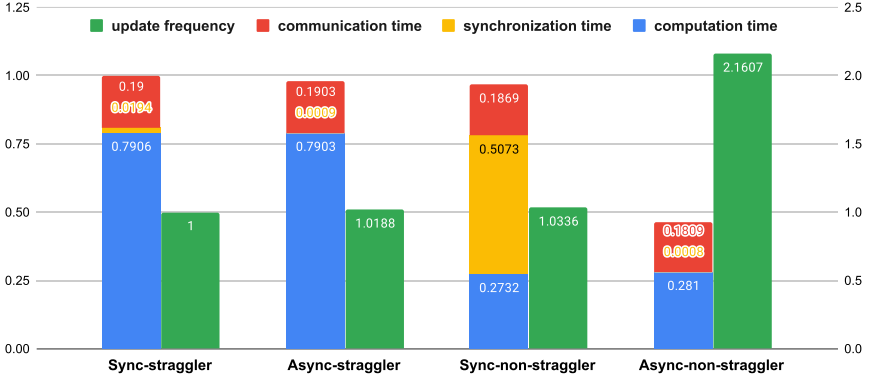


Figure 7.7. Time-consumption breakdown of an asynchronous algorithm to demonstrate its efficiency (eight workers on url dataset for binary classification task).

the problem of straggler and scale poorly as non-stragglers are inefficiently waiting for the straggler. Asynchronous algorithms behave like ideal in the beginning as they can address the straggler problem well, and deviate from ideal when the number of workers continues to grow because the communication overheads will limit the speedup.

Comparison with FDML Algorithm: It's noteworthy that the FDML algorithm constructs a local sub-model for each participant based on Equation (7.22), and these local predictions are then combined at the server node to jointly optimize the entire model. Importantly, the convergence of FDML heavily relies on the theoretical value of the delay upper bound. However, this value remains unknown in real systems. Therefore, any inconsistency between the theoretical and actual values of the delay upper bound inevitably leads to a loss in the expressiveness of FDML in real-world applications, as depicted in Figure 7.8. We did not compare the functional loss values of the two because FDML does not fully optimize the problem posed in Equation (7.22), treating it instead as a sub-model. FDML only introduced an algorithm based on the SGD version, so we compared it using asynchronous SGD as the training method. In the experiments, the performance with two participants declined compared to that with a single participant. This was also reflected in Hu *et al.*'s logistic regression experiments. However, as we incrementally increased the node count, FDML's performance deteriorated rapidly, whereas ours remained consistent. As the number of nodes increased

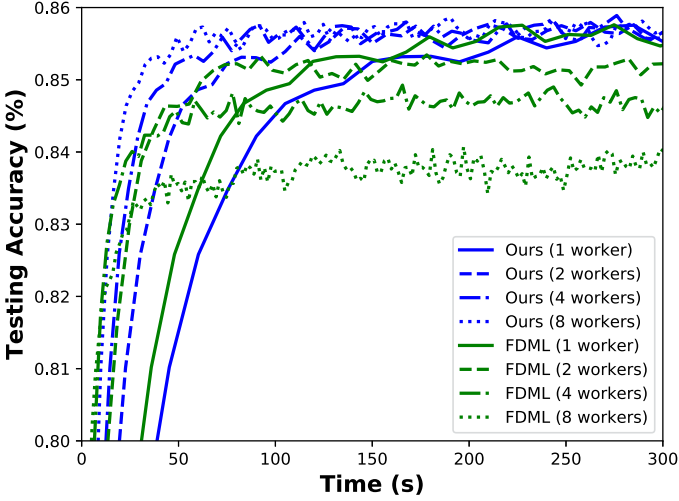


Figure 7.8. Comparison with FDML on multi-class classification tasks.

and the number of features obtained by each node in each sample decreased, the loss of expressiveness became more pronounced. In contrast, the number of our participants is scalable because it genuinely optimizes the problem given in Equation (7.22).

7.7 Conclusion

In this chapter, we introduced an asynchronous vertical federated learning algorithm (AFSGD-VP) and its SVRG and SAGA variants. Currently, these algorithms stand as the first asynchronous vertical federated learning algorithms with theoretical guarantees. Crucially, we provided the convergence rates of AFSGD-VP and its SVRG and SAGA variants under strongly convex objective functions without any bounds on delay. We also established the model’s technical privacy from both the model and numerical perspectives. Extensive experiments on multiple vertically partitioned datasets not only validated the theoretical results of AFSGD-VP and its SVRG and SAGA variants but also demonstrated that these asynchronous algorithms are more efficient than their synchronous counterparts.

Chapter 8

Secure Bilevel Asynchronous Vertical Federated Learning with Backward Updating

Vertical federated learning (VFL) attracts increasing attention due to the emerging demands of multi-party collaborative modeling and concerns of privacy leakage. In the real VFL applications, usually only one or partial parties hold labels, which makes it challenging for all parties to collaboratively learn the model without privacy leakage. Meanwhile, most existing VFL algorithms are trapped in the synchronous computations so that training time is dragged down by straggler, which leads to inefficiency in their real-world applications. To address these challenging problems, we propose a novel VFL framework integrated with new backward updating mechanism (BUM) and bilevel asynchronous parallel architecture (BAPA) (VFB²), under which three new algorithms, VFB²-SGD, -SVRG and -SAGA, are proposed. We derive the theoretical results of the convergence rates of these three algorithms under both strongly convex and non-convex conditions. We also prove the security of VFB² under semi-honest threat models. Extensive experiments on benchmark datasets demonstrate that our algorithms are efficient, scalable and lossless.

8.1 Introduction

Federated learning has emerged as a paradigm for collaborative modeling with privacy preservation. A line of recent works (McMahan *et al.*, 2016; Smith *et al.*, 2017) focus on the horizontal federated learning, where each party has a subset of samples with complete features. There are also some works (Gascon *et al.*, 2016; Yang *et al.*, 2019b; Dang *et al.*, 2020) studying the vertical federated learning (VFL), where each party holds a disjoint subset of features for all samples. In this chapter, we focus on VFL that has attracted much attention from the academia and industry due to its wide applications to emerging multi-party collaborative modeling with privacy preservation.

Currently, there are two mainstream methods for VFL: homomorphic encryption (HE) based methods and exchanging the raw computational results (ERCR) based methods. The HE-based methods (Hardy *et al.*, 2017; Cheng *et al.*, 2019) leverage HE techniques to encrypt the raw data and then use the encrypted data (ciphertext) for training model with privacy preservation. However, there are two major drawbacks of HE-based methods. First, the complexity of homomorphic mathematical operation on ciphertext field is very high, thus HE is extremely time-consuming for modeling (Liu *et al.*, 2015, 2019). Second, approximation is required for HE to support operations of nonlinear functions, such as sigmoid and logarithmic functions, which inevitably causes loss of accuracy for various machine learning (ML) models using nonlinear functions (Kim *et al.*, 2018; Yang *et al.*, 2019a). Thus, the inefficiency and inaccuracy of HE-based methods dramatically limit their wide applications to realistic VFL tasks.

ERCR-based methods (Zhang *et al.*, 2018; Hu *et al.*, 2019; Gu *et al.*, 2020b) leverage labels and the raw intermediate computational results transmitted from the other parties to compute stochastic gradients and thus use distributed stochastic gradient descent (SGD) methods to train VFL models efficiently. Although ERCR-based methods circumvent aforementioned drawbacks of HE-based methods, existing ERCR-based methods are designed with the only consideration that all parties have labels, which is not usually the case in real-world VFL tasks. In realistic VFL applications, usually only one or partial parties (denoted as active parties) have the labels, and the

other parties (denoted as passive parties) can only provide extra feature data but do not have labels. When these ERCR-based methods are applied to the real situation with both active and passive parties, the algorithms even cannot guarantee the convergence because only active parties can update the gradient of loss function based on labels, but the passive parties cannot, i.e., partial model parameters are not optimized during the training process. Thus, it comes to the crux of designing the proper algorithm for solving real-world VFL tasks with only one or partial parties holding labels.

Moreover, algorithms using synchronous computation (Gong *et al.*, 2016; Zhang *et al.*, 2018) are inefficient when applied to real-world VFL tasks, especially, when computational resources in the VFL system are unbalanced. Therefore, it is desired to design the efficient asynchronous algorithms for real-world VFL tasks. Although there have been several works studying asynchronous VFL algorithms (Hu *et al.*, 2019; Gu *et al.*, 2020b), it is still an open problem to design asynchronous algorithms for solving real-world VFL tasks with only one or partial parties holding labels.

In this chapter, we address these challenging problems by proposing a novel framework (VFB²) integrated with the novel backward updating mechanism (BUM) and bilevel asynchronous parallel architecture (BAPA). Specifically, the BUM enables all parties, rather than only active parties, to collaboratively update the model securely and also makes the final model lossless; the BAPA is designed for efficient asynchronous backward updating. Considering the advantages of SGD-type algorithms in optimizing ML models, we thus propose three new SGD-type algorithms, i.e., VFB²-SGD, -SVRG and -SAGA, under this framework.

Notations: $\hat{w}$ denotes the inconsistent read of w . $\bar{w}$ denotes w to compute local stochastic gradient of loss function for collaborators, which maybe stale due to communication delay. $\psi(t)$ is the corresponding party performing the t th global iteration. Given a finite set S , $|S|$ denotes its cardinality.

8.2 Problem Formulation

Given a training set $\{x_i, y_i\}_{i=1}^n$, for binary classification task $y_i \in \{-1, +1\}$ or $y_i \in \mathbb{R}$ for regression problem and $x_i \in \mathbb{R}^d$, we consider

the model in a linear form of $w^\top x$, where $w \in \mathbb{R}^d$ corresponds to the model parameters. For VFL, x_i is vertically distributed among $q \geq 2$ parties, i.e., $x_i = [(x_i)_{\mathcal{G}_1}; \dots; (x_i)_{\mathcal{G}_q}]$, where $(x_i)_{\mathcal{G}_\ell} \in \mathbb{R}^{d_\ell}$ is stored on the ℓ party and $\sum_{\ell=1}^q d_\ell = d$. Similarly, there is $w = [w_{\mathcal{G}_1}; \dots; w_{\mathcal{G}_q}]$. Particularly, we focus on the following regularized empirical risk minimization problem:

$$\min_{w \in \mathbb{R}^d} f(w) := \underbrace{\frac{1}{n} \sum_{i=1}^n \mathcal{L}(w^\top x_i, y_i)}_{f_i(w)} + \lambda \sum_{\ell=1}^q g(w_{\mathcal{G}_\ell}) \quad (8.1)$$

where $w^\top x_i = \sum_{\ell=1}^q w_{\mathcal{G}_\ell}^\top (x_i)_{\mathcal{G}_\ell}$, $\mathcal{L}$ denotes the loss function, $\sum_{\ell=1}^q g(w_{\mathcal{G}_\ell})$ is the regularization term, and $f_i : \mathbb{R}^d \rightarrow \mathbb{R}$ is smooth and possibly non-convex. Example of Problem 8.1 includes models for binary classification tasks and models for regression tasks.

In this chapter, we introduce two types of parties, **active party** and **passive party**, where the former denotes data provider holding labels while the latter does not. Particularly, in our problem setting, there are m ($1 \leq m \leq n$) active parties. Each active party can play the role of dominator in model updating by actively launching updates. All parties, including both active and passive parties, passively launching updates play the role of collaborator. To guarantee the model security, only active parties know the form of the loss function.

The problem studied in this paper is stated as follows:

Given: Vertically partitioned data $\{x_{\mathcal{G}_\ell}\}_{\ell=1}^q$ stored in q parties and the labels only held by active parties.

Learn: A ML model $\mathbf{M}$ collaboratively learned by both active and passive parties without leaking privacy.

Lossless constraint: The accuracy of $\mathbf{M}$ must be comparable to that of model $\mathbf{M}'$ learned under non-federated learning.

8.3 Understanding Algorithms

In this section, we propose the novel VFB² framework and corresponding backward updating mechanism (BUM) and BAPA,

followed by three new SGD-type asynchronous algorithms, i.e., VFB²-SGD, -SVRG and -SAGA, under this framework.

8.3.1 VFB² Framework

VFB² is composed of three components and its systemic structure is illustrated in Figure 8.1(a).

Backward updating mechanism: The key idea of BUM is to make passive parties indirectly use labels to compute stochastic gradient without directly accessing the raw label data. Specifically, the BUM embeds label y_i into an intermediate value $\vartheta := \frac{\partial \mathcal{L}(w^\top x_i, y_i)}{\partial (w^\top x_i)}$. Then ϑ and i are distributed backward to the other parties. Consequently, the passive parties can also compute the stochastic gradient and update the model by using the received ϑ and i (see Algorithms 8.2 and 8.3 for details). Figure 8.1(b) depicts the case where ϑ is distributed from party 1 to the rest of the parties. In this case, all parties, rather than only active parties, can collaboratively learn the model without privacy leakage.

For VFL algorithms with BUM, dominated updates in different active parties are performed in distributed-memory parallel, while collaborative updates within a party are performed in shared-memory parallel. The difference of parallelism fashion leads to the challenge of developing a new parallel architecture instead of just

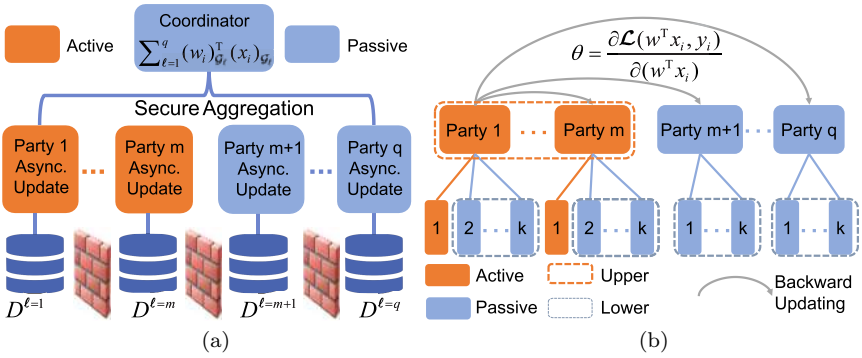


Figure 8.1. (a) System structure of VFB² framework; (b) Illustration of the BUM and BAPA, where k is defined as number of threads in each party.

directly adopting the existing asynchronous parallel architecture for VFL. To tackle this challenge, we elaborately design a novel BAPA.

Bilevel asynchronous parallel architecture: The BAPA includes two levels of parallel architectures, where the upper level denotes the inner-party parallel and the lower one is the intra-party parallel. More specifically, the inner-party parallel denotes distributed-memory parallel between active parties, which enables all active parties to asynchronously launch dominated updates; while the intra-party one denotes the shared-memory parallel of collaborative updates within each party, which enables multiple threads within a specific party to asynchronously perform the collaborative updates. Figure 8.1(b) illustrates the BAPA with m active parties.

To utilize feature data provided by other parties, a party needs to obtain $w^T x_i = \sum_{\ell=1}^q w_{\mathcal{G}_\ell}^\top(x_i)_{\mathcal{G}_\ell}$. Many recent works achieved this by aggregating the local intermediate computational results securely (Hu *et al.*, 2019; Gu *et al.*, 2020a). In this paper, we use the efficient tree-structured communication scheme (Zhang *et al.*, 2018) for secure aggregation, whose security was proved by Gu *et al.*, (2020a).

Secure aggregation strategy: The details are summarized in Algorithm 8.1. Specifically, at Step 2, $w_{\mathcal{G}_\ell}^\top(x_i)_{\mathcal{G}_\ell}$ is computed locally on the ℓ th party to prevent the direct leakage of $w_{\mathcal{G}_\ell}$ and $(x_i)_{\mathcal{G}_\ell}$. Especially, a random number δ_ℓ is added to $w_{\mathcal{G}_\ell}^\top(x_i)_{\mathcal{G}_\ell}$ to mask the value of $w_{\mathcal{G}_\ell}^\top(x_i)_{\mathcal{G}_\ell}$, which can enhance the security during aggregation process. At Steps 4 and 5, ξ_1 and ξ_2 are aggregated through tree structures T_1 and T_2 , respectively. Note that T_2 is totally

Algorithm 8.1 Safe algorithm of obtaining $w^T x_i$

Input: $\{w_{\mathcal{G}_{\ell'}}\}_{\ell'=1}^q$ and $\{(x_i)_{\mathcal{G}_{\ell'}}\}_{\ell'=1}^q$ allocating at each party, index i

Do this in parallel

- 1: **for** $\ell' = 1, \dots, q$ **do**
- 2: Generate a random number $\delta_{\ell'}$ and calculate $w_{\mathcal{G}_{\ell'}}^\top(x_i)_{\mathcal{G}_{\ell'}} + \delta_{\ell'}$
- 3: **end for**
- 4: Obtain $\xi_1 = \sum_{\ell'=1}^q (w_{\mathcal{G}_{\ell'}}^\top(x_i)_{\mathcal{G}_{\ell'}} + \delta_{\ell'})$ through tree structure T_1
- 5: Obtain $\xi_2 = \sum_{\ell'=1}^q \delta_{\ell'}$ through totally different tree structure $T_1 \neq T_2$

Output: $w^T x_i = \xi_1 - \xi_2$

different from T_1 which can prevent the random value being removed under threat model 1 (defined in Section 8.4.3). Finally, the value of $w^\top x_i = \sum_{\ell=1}^q (w_{\mathcal{G}_\ell}^\top (x_i)_{\mathcal{G}_\ell})$ is recovered by removing term $\sum_{\ell=1}^q \delta_\ell$ from $\sum_{\ell=1}^q (w_{\mathcal{G}_\ell}^\top (x_i)_{\mathcal{G}_\ell} + \delta_\ell)$ at the output step. Using such aggregation strategy, $(x_i)_{\mathcal{G}_\ell}$ and $w_{\mathcal{G}_\ell}$ are prevented from leaking during the aggregation.

8.3.2 Details of Algorithms

SGD (Bottou, 2010) is a popular method for learning ML models. However, it has a poor convergence rate due to the intrinsic variance of stochastic gradient. Thus, many popular variance reduction techniques have been proposed, including the SVRG, SAGA, SPIDER (Johnson and Zhang, 2013; Defazio *et al.*, 2014; Wang *et al.*, 2019) and their applications to other problems (Huang *et al.*, 2019; Huang *et al.*, 2020; Dang *et al.*, 2020; Yang *et al.*, 2020a,b; Li *et al.*, 2020; Wei *et al.*, 2019a). In this section, we raise three SGD-type algorithms, i.e., the SGD, SVRG and SAGA, which are the most popular ones among SGD-type methods for the appealing performance in practice. We summarize the detailed steps of VFB^2 -SGD in Algorithms 8.2 and 8.3. For VFB^2 -SVRG and -SAGA, one just needs to replace the update rule with corresponding one.

Algorithm 8.2 VFB^2 -SGD for active party ℓ launch dominated updates

Input: Local data $\{(x_i)_{\mathcal{G}_\ell}, y_i\}_{i=1}^n$ stored on the ℓ th party, learning rate γ

1: Initialize the necessary parameters.

Keep doing in parallel (distributed-memory parallel for multiple active parties)

2: Pick up an index i randomly from $\{1, \dots, n\}$

3: Compute $\hat{w}^\top x_i = \sum_{\ell'=1}^q \hat{w}_{\mathcal{G}_{\ell'}}^\top (x_i)_{\mathcal{G}_{\ell'}}$ based on Algorithm 8.1

4: Compute $\vartheta = \frac{\partial \mathcal{L}(\hat{w}^\top x_i, y_i)}{\partial (\hat{w}^\top x_i)} \circ,$

5: Send ϑ and index i to collaborators

6: Compute $\tilde{v}^\ell = \nabla_{\mathcal{G}_\ell} f_i(\hat{w})$

7: Update $w_{\mathcal{G}_\ell} \leftarrow w_{\mathcal{G}_\ell} - \gamma \tilde{v}^\ell$

End parallel

Algorithm 8.3 VFB²-SGD for the ℓ th party to passively launch collaborative updates

Input: Local data $\{(x_i)_{\mathcal{G}_\ell}, y_i\}_{i=1}^n$ stored on the ℓ th party, learning rate γ

- 1: Initialize the necessary parameters (for passive parties)
Keep doing in parallel (shared-memory parallel for multiple threads)
 - 2: Receive ϑ and index i from the dominator
 - 3: Compute $\tilde{v}^\ell = \nabla_{\mathcal{G}_\ell} \mathcal{L}(\bar{w}) + \lambda \nabla_{\mathcal{G}_\ell} g(\hat{w}) = \vartheta \cdot (x_i)_{\mathcal{G}_\ell} + \lambda \nabla g(\hat{w}_{\mathcal{G}_\ell})$
 - 4: Update $w_{\mathcal{G}_\ell} \leftarrow w_{\mathcal{G}_\ell} - \gamma \tilde{v}^\ell$
 - 5: **End parallel**
-

As shown in Algorithm 8.2, at each dominated update, the dominator (an active party) calculates ϑ and then distributes ϑ together with i to the collaborators (the rest of the $q - 1$ parties). As shown in Algorithm 8.3, for party ℓ , once it has received ϑ and i , it will launch a new collaborative update asynchronously. As for the dominator, it computes the local stochastic gradient as $\nabla_{\mathcal{G}_\ell} f_i(\hat{w}) = \nabla_{\mathcal{G}_\ell} \mathcal{L}(\hat{w}) + \lambda \nabla g(\hat{w}_{\mathcal{G}_\ell})$. While, for the collaborator, it uses the received ϑ to compute $\nabla_{\mathcal{G}_\ell} \mathcal{L}$ and local $\hat{w}$ to compute $\nabla_{\mathcal{G}_\ell} g$, as shown in Step 3 of Algorithm 8.3. Note that active parties also need perform Algorithm 8.3 to collaborate with other dominators to ensure that the model parameters of all parties are updated.

8.3.3 Scenario Case Study

In this section, we show an application over real-world scenarios to give a more intuitive understanding of the described algorithm together with the proposals to apply the proposed algorithm to a specific task. The goal is to predict whether a user will default on a loan based on multiple personal information on the real financial dataset UCICreditCard. Specifically, we propose to apply VFB² algorithms to solve a logistic regression model with the form of a Problem (8.1):

Scenario description: The dataset contains 23 personal information as features, including loan amount, gender, education level, marital status, age, monthly repayment status, monthly bill total, and monthly payment amount; labels are defined based on whether there will be a loan default (marked as 1) or not (marked as 0); in total,

there are 30,000 samples. From this dataset, we envision a scenario as described in the following. Suppose there are four companies A, B, C, and D with different characteristics of customers. Among them, company A has nine characteristics including loan amount, gender, age, and six-month monthly repayment; B has gender, education level, marital status, age and other four characteristics; C has eight characteristics, including gender, age, and six-month monthly bill total; D has eight characteristics, including gender, age, six-month monthly payment amount; and each of the company's customers are not all the same (i.e., the sample IDs are not all the same). In particular, in this scenario, company A is the loan originator and wants to use the machine model to determine whether the user will default on the loan. Typically, Company A would use I train a ML model with my own data. However, since company A has only nine characteristics of data, there is no other information that is helpful for judging whether the user is in breach of contract, such as education level, marital status, and monthly bills, and so use this data to train models that may experience poor performance. Companies B, C, and D have other information about relevant users, and this information will be helpful to learn the model required by company A. Therefore, an ideal way is to use the feature data of other companies to train the model together in order to obtain a better performing model. However, due to privacy protection-related laws and regulations and commercial competition, these user data cannot be directly shared between companies. In this case, a feasible solution is to use federated learning to realize the collaborative learning of four companies under the protection of privacy at the technical level.

From this dataset, we envision a scenario as described in the following. Suppose there are four companies A, B, C, and D with different features of customers. Among them, company A has nine features, including loan amount, gender, age, and six-month monthly repayment; B has four features of gender, education level, marital status and age; C has eight features, including gender, age, and six-month monthly bill total; finally, D has eight features including gender, age, and six-month monthly payment amount; These companies own information of different groups of customers (that is to say, the sample IDs are not all the same). Particularly, in this case, company A is the loan originator who wants to know whether the user will default on the loan.

Typically, company A would train a ML model with its own data. However, since company A only owns nine features without information such as education level, marital status, and monthly bills, the model will perform poorly due to lack of data to judge whether the user is in breach of contract.

On the other hand, companies B, C, and D have other information about relevant users, which will be helpful to support company A's model with extra useful features. Therefore, an ideal way is to use the feature data of other companies to train the model together in order to obtain a better-performing model.

However, if we take privacy protection, laws, and commercial competition into consideration, companies B, C, and D won't choose to share these user data anyway. In this case, a feasible solution is to use federated learning to realize the collaborative learning of four companies under the protection of privacy at the technical level.

Modeling: We first assume that each company has processed its own data and converted it into the standard distribution form in VFL to match UCICreditCard dataset. Among them, company A provides nine features, such as loan amount, gender, age, and monthly repayment for six months ($\mathbf{x}_{\mathcal{G}_1} = [x_1, x_2, x_6, \dots, x_{11}] \in \mathbb{R}^9$); company B provides two features of education level and marital status ($\mathbf{x}_{\mathcal{G}_2} = [x_3, x_4] \in \mathbb{R}^2$); company C provides a total of six features of the total monthly bill for six months ($\mathbf{x}_{\mathcal{G}_3} = [x_{12}, \dots, x_{17}] \in \mathbb{R}^6$); company D provides a total of six features for the monthly payment amount for six months ($\mathbf{x}_{\mathcal{G}_4} = [x_{18}, \dots, x_{23}] \in \mathbb{R}^6$). Each company has 30,000 sample IDs shared in common (24,000 samples for training set and 6,000 samples for test set; only company A has labeled data). Using the ℓ_2 regularized logistic regression model for modeling, the problem in this scenario can be expressed as

$$\min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w}) := \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\theta_i, y_i) + \lambda \sum_{m=1}^q \|\mathbf{w}_m\|^2 \quad (8.2)$$

where $q = 4$, $n = 24,000$, $d = 23$, $\theta = \mathbf{w}^T \mathbf{x}_i = \sum_{m=1}^4 \mathbf{w}_m^T(\mathbf{x}_i)_{\mathcal{G}_m} = \sum_{m=1}^4 \mathbf{w}_m^T(\mathbf{x}_i)_{\mathcal{G}_m}$, $\mathbf{x} = [\mathbf{x}_{\mathcal{G}_1}, \mathbf{x}_{\mathcal{G}_2}, \mathbf{x}_{\mathcal{G}_3}, \mathbf{x}_{\mathcal{G}_4}]$, $\mathbf{w} = [\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3, \mathbf{w}_4]$, and all features and model parameters are stored in corresponding owner companies, respectively.

Model learning: Only company A contains labels, so it is the active party in this scenario, and all other companies B, C, and D are passive parties. According to Algorithm 8.2, company A randomly selects a sample i , and then calls Algorithm 8.1 to calculate $\theta = \widehat{\mathbf{w}}^T \mathbf{x}_i$. Specifically, when calling Algorithm 8.1, all companies use their current local model parameters and the local features (due to asynchronous update, the local model parameters used in the calculation may be inconsistent with the real-time model parameters) to compute $\theta_1 = \widehat{\mathbf{w}}_1^T(\mathbf{x}_i)_{\mathcal{G}_1}$, $\theta_2 = \widehat{\mathbf{w}}_2^T(\mathbf{x}_i)_{\mathcal{G}_2}$, $\theta_3 = \widehat{\mathbf{w}}_3^T(\mathbf{x}_i)_{\mathcal{G}_3}$, $\theta_4 = \widehat{\mathbf{w}}_4^T(\mathbf{x}_i)_{\mathcal{G}_4}$ and generate random numbers $\delta_1, \delta_2, \delta_3, \delta_4$, respectively. Then ξ_1 and ξ_2 are aggregated through different tree structures, i.e., $\xi_1 = \sum_{m=1}^4 (\theta_m = \widehat{\mathbf{w}}_m^T(\mathbf{x}_i)_{\mathcal{G}_m} + \delta_m)$ and $\xi_2 = \sum_{m=1}^4 \delta_m$. Since company A is the active party who initiated the calculation, it acts as a collaborator and calculates $\xi_1 - \xi_2$ to get $\theta = \widehat{\mathbf{w}}^T \mathbf{x}_i$. In Step 4, company A calculates $\vartheta = \frac{\partial \mathcal{L}(\mathbf{T}\mathbf{x}_i, y_i)}{\partial \widehat{\mathbf{w}}^T \mathbf{x}_i} = \frac{-y_i \exp(-y_i \theta)}{1 + \exp(-y_i \theta)}$ according to θ from previous aggregation and its unique labels. Then company A sent ϑ to B, C, D. For company A, its stochastic gradient can be calculated as $\vartheta(\mathbf{x}_i)_{\mathcal{G}_1} + 2\lambda \widehat{\mathbf{w}}_1$, and its local model parameters will be updated in Step 7. For a logistic regression model, the partial gradient of the loss function to the local parameters is $\frac{(\mathbf{x}_i)_{\mathcal{G}_m} \partial \mathcal{L}(\mathbf{T}\mathbf{x}_i, y_i)}{\partial \widehat{\mathbf{w}}^T \mathbf{x}_i} = \frac{-y_i \exp(-y_i \theta)}{1 + \exp(-y_i \theta)}$. Therefore, for passive parties such as companies B, C, D, they can still calculate the partial gradient as $\frac{(\mathbf{x}_i)_{\mathcal{G}_m} \partial \mathcal{L}(\mathbf{w}^T \mathbf{x}_i, y_i)}{\partial \widehat{\mathbf{w}}^T \mathbf{x}_i} = \frac{-y_i \exp(-y_i \theta)}{1 + \exp(-y_i \theta)} = (\mathbf{x}_i)_{\mathcal{G}_m} \vartheta$ even if they don't own any labels and use their local parameters to calculate the partial gradient of the regular term $2\lambda \widehat{\mathbf{w}}_m$. Finally, the model will update according to the gradient. The above processes are repeated asynchronously and in parallel until convergence, and the corresponding model can be learned.

8.4 Theoretical Analysis

In this section, we provide the convergence analyses. More details can be found via the paper Zhang *et al.* (2021).

We first present preliminaries for strongly convex and non-convex problems.

Assumption 8.1. For $f_i(w)$ in Problem 8.1, we assume the following conditions hold:

1. **Lipschitz gradient:** Each function f_i , $i = 1, \dots, n$, there exists $L > 0$ such that for $\forall w, w' \in \mathbb{R}^d$, we have

$$\|\nabla f_i(w) - \nabla f_i(w')\| \leq L\|w - w'\| \quad (8.3)$$

2. **Block-coordinate Lipschitz gradient:** For $i = 1, \dots, n$, there exists an $L_\ell > 0$ for ℓ block $\mathcal{G}_\ell$, where $\ell = 1, \dots, q$ such that

$$\|\nabla_{\mathcal{G}_\ell} f_i(w + U_\ell \Delta_\ell) - \nabla_{\mathcal{G}_\ell} f_i(w)\| \leq L_\ell \|\Delta_\ell\| \quad (8.4)$$

where $\Delta_\ell \in \mathbb{R}^{d_\ell}$, $U_\ell \in \mathbb{R}^{d \times d_\ell}$, $[U_1, \dots, U_q] = I_d$.

3. **Bounded block-coordinate gradient:** There exists a constant G such that for f_i , $i = 1, \dots, n$ and block $\mathcal{G}_\ell$, $\ell = 1, \dots, q$, it holds that $\|\nabla_{\mathcal{G}_\ell} f_i(w)\|^2 \leq G$.

Assumption 8.2. The regularization term g is L_g -smooth, which means that there exists an $L_g > 0$ for $\ell = 1, \dots, q$ such that $\forall w_{\mathcal{G}_\ell}, w'_{\mathcal{G}_\ell} \in \mathbb{R}^{d_\ell}$, we have

$$\|\nabla g(w_{\mathcal{G}_\ell}) - \nabla g(w'_{\mathcal{G}_\ell})\| \leq L_g \|w_{\mathcal{G}_\ell} - w'_{\mathcal{G}_\ell}\| \quad (8.5)$$

Assumption 8.2 imposes the smoothness on g , which is necessary for the convergence analyses. Since, as for a specific collaborator, it uses the received $\hat{w}$ (denoted as $\bar{w}$) to compute $\nabla_{\mathcal{G}_\ell} \mathcal{L}$ and local $\hat{w}$ to compute $\nabla_{\mathcal{G}_\ell} g = \nabla g(w_{\mathcal{G}_\ell})$, which makes it necessary to track the behavior of g individually. Similar to previous research works, we introduce the bounded delay as follows.

Assumption 8.3. Bounded delay: Time delays of inconsistent reading and communication between dominator and its collaborators are upper bounded by τ_1 and τ_2 , respectively.

Given $\hat{w}$ as the inconsistent read of w , which is used to compute the stochastic gradient in dominated updates, following the analysis in (Gu *et al.*, 2020b), we have

$$\hat{w}_t - w_t = \gamma \sum_{u \in D(t)} U_{\psi(u)} \tilde{v}_u^{\psi(u)} \quad (8.6)$$

where $D(t) = \{t-1, \dots, t-\tau_0\}$ is a subset of non-overlapped previous iterations with $\tau_0 \leq \tau_1$. Given $\bar{w}$ as the parameter used to compute

the $\nabla_{\mathcal{G}_\ell} \mathcal{L}$ in collaborative updates, which is the steal state of $\hat{w}$ due to the communication delay between the specific dominator and its corresponding collaborators. Then, following the analyses by Huo and Huang (2017), we have

$$\bar{w}_t = \hat{w}_{t-\tau_0} = \hat{w}_t + \gamma \sum_{t' \in D'(t)} U_{\psi(t')} \tilde{v}_{t'}^{\psi(t')} \quad (8.7)$$

where $D'(t) = \{t-1, \dots, t-\tau_0\}$ is a subset of previous iterations performed during the communication and $\tau_0 \leq \tau_2$.

8.4.1 Convergence Analysis for Strongly Convex Problem

Assumption 8.4. Each function f_i , $i = 1, \dots, n$, is μ -strongly convex, i.e., $\forall w, w' \in \mathbb{R}^d$, there exists a $\mu > 0$ such that

$$f_i(w) \geq f_i(w') + \langle \nabla f_i(w'), w - w' \rangle + \frac{\mu}{2} \|w - w'\|^2 \quad (8.8)$$

For the strongly convex problem, we introduce notation $K(t)$ that denotes a minimum set of successive iterations fully visiting all coordinates from global iteration number t . Note that this is necessary for the asynchronous convergence analyses of the global model. Moreover, we assume that the size of $K(t)$ is upper bounded by η_1 , i.e., $|K(t)| \leq \eta_1$. Based on $K(t)$, we introduce the epoch number $v(t)$ as follows.

Definition 8.1. Let $P(t)$ be a partition of $\{0, 1, \dots, t - \sigma'\}$, where $\sigma' \geq 0$. For any $\kappa \subseteq P(t)$, we note that there exists $t' \leq t$ such that $K(t') = \kappa$, and $\kappa_1 \subseteq P(t)$ such that $K(0) = \kappa_1$. The epoch number for the t global iteration, i.e., $v(t)$ is defined as the maximum cardinality of $P(t)$.

Given the definition of epoch number $v(t)$, we have the following theoretical results for μ -strongly convex problem.

Theorem 8.1. Under Assumptions 8.1–8.4, to achieve the accuracy ϵ for Problem 8.1 for VFB²-SGD, i.e., $\mathbb{E}(f(w_t) - f(w^*)) \leq \epsilon$, let $\gamma \leq \frac{\epsilon \mu^{1/3}}{(G96L_*^2)^{1/3}}$, if $\tau \leq \min\{\epsilon^{-4/3}, \frac{(GL_*^2)^{2/3}}{\epsilon^2 \mu^{2/3}}\}$, the epoch number $v(t)$ should satisfy

$$v(t) \geq \frac{44(GL_*^2)^{1/3}}{\mu^{4/3}\epsilon} \log \left(\frac{2(f(w_0) - f(w^*))}{\epsilon} \right) \quad (8.9)$$

where $L_* = \max\{L, \{L_\ell\}_{\ell=1}^q, L_g\}$, $\tau = \max\{\tau_1^2, \tau_2^2, \eta_1^2\}$, w^0 and w^* denote the initial point and optimal point, respectively.

Theorem 8.2. Under Assumptions 8.1–8.4, to achieve the accuracy ϵ of Problem 8.1 for VFB²-SVRG, let $C = (L_*^2\gamma + L_*)\frac{\gamma^2}{2}$ and $\rho = \frac{\gamma\mu}{2} - \frac{16L_*^2\eta_1 C}{\mu}$, we can carefully choose γ such that

$$\begin{aligned} (1) \quad & 1 - 2L_*^2\gamma^2\tau > 0; \quad (2) \quad \rho > 0; \quad (3) \quad \frac{8L_*^2\tau^{1/2}C}{\rho\mu} \leq 0.05 \\ (4) \quad & L_*^2\gamma^2\tau^{3/2}(28C + 5\gamma)\frac{36G}{\rho(1 - 2L_*^2\gamma^2\tau)} \leq \frac{\epsilon}{8} \end{aligned} \quad (8.10)$$

where $\lambda_\gamma = \frac{18}{1 - 2L_*^2\gamma^2\tau}$, the inner epoch number $v(t)$ should satisfy $v(t) \geq \frac{\log 0.25}{\log(1-\rho)}$ and the outer loop number S should satisfy $S \geq \frac{1}{\log \frac{4}{3}} \log \frac{2f(w_0) - f(w^*)}{\epsilon}$.

Theorem 8.3. Under Assumptions 8.1–8.4, to achieve the accuracy ϵ of Problem 8.1 for VFB²-SAGA, let

$$\begin{aligned} c_0 &= (2\gamma^3\tau^{3/2} + (L_*^2\gamma^3\tau + L_*\gamma^2)180\gamma^2\tau^{3/2} + 8\gamma^2\tau)\frac{18GL_*^2}{1 - 72L_*^2\gamma^2\tau} \\ c_1 &= 2L_*^2\tau(L_*^2\gamma^3\tau + L_*\gamma^2) \\ c_2 &= 4(L_*^2\gamma^3\tau + L_*\gamma^2)\frac{L_*^2\tau}{n} \end{aligned} \quad (8.11)$$

and $\rho \in (1 - \frac{1}{n}, 1)$, we can choose γ such that

$$\begin{aligned} (1) \quad & 1 - 72L_*^2\gamma^2\tau > 0; \quad (2) \quad 0 < 1 - \frac{\gamma\mu}{4} < 1 \\ (3) \quad & \frac{4c_0}{\gamma\mu(1 - \rho)\left(\frac{\gamma\mu^2}{4} - 2c_1 - c_2\right)} \leq \frac{\epsilon}{2} \\ (4) \quad & -\frac{\gamma\mu^2}{4} + 2c_1 + c_2 \left(1 + \left(1 - \frac{1 - \frac{1}{n}}{\rho}\right)^{-1}\right) \leq 0 \\ (5) \quad & -\frac{\gamma\mu^2}{4} + c_2 + c_1 \left(2 + \left(1 - \frac{1 - \frac{1}{n}}{\rho}\right)^{-1}\right) \leq 0 \end{aligned} \quad (8.12)$$

the epoch number $v(t)$ should satisfy $v(t) \geq \frac{1}{\log \frac{1}{\rho}} \log \frac{2(2\rho-1+\frac{\gamma\mu}{4})(f(w_0)-f(w^*))}{\epsilon(\rho-1+\frac{\gamma\mu}{4})(\frac{\gamma\mu^2}{4}-2c_1-c_2)}.$

Remark 8.1. For strongly convex problems, given the assumptions and parameters in corresponding theorems, the convergence rate of VFB²-SGD is $\mathcal{O}(\frac{1}{\epsilon} \log(\frac{1}{\epsilon}))$ and those of VFB²-SVRG and VFB²-SAGA are $\mathcal{O}(\log(\frac{1}{\epsilon}))$.

8.4.2 Convergence Analysis for Non-convex Problem

Assumption 8.5. Non-convex function $f(w)$ is bounded as follows:

$$f^* := \inf_{w \in \mathbb{R}^d} f(w) > -\infty \quad (8.13)$$

Assumption 8.5 shows the feasibility of non-convex Problem 8.1. For the non-convex problem, we introduce the notation $K'(t)$ that denotes a set of q iterations fully visiting all coordinates, i.e., $K'(t) = \{\{t, t+\bar{t}_1, \dots, t+\bar{t}_{q-1}\} : \psi(\{t, t+\bar{t}_1, \dots, t+\bar{t}_{q-1}\}) = \{1, \dots, q\}\}$, where the t th global iteration denotes a dominated update. Moreover, these iterations are performed respectively on a dominator and $q-1$ different collaborators receiving ϑ calculated at the t th global iteration. Besides, we assume that $K'(t)$ can be completed in η_2 global iterations, i.e., for $\forall t' \in \mathcal{A}(t)$, there is $\eta_2 \geq \max\{u | u \in K'(t')\} - t'$. Note that, contains to $K(t)$, we have $|K'(t)| = q$ and the definition of $K'(t)$ does not emphasize on “successive iterations” due to the difference of analysis techniques between strongly convex and non-convex problems. Based on $K'(t)$, we introduce the epoch number $v'(t)$ as follows.

Definition 8.2. $\mathcal{A}(t)$ denotes a set of global iterations, where for $\forall t' \in \mathcal{A}(t)$, there is the t' th global iteration denoting a dominated update and $\cup_{t' \in \mathcal{A}(t)} K'(t') = \{0, 1, \dots, t\}$. The epoch number $v'(t)$ is defined as $|\mathcal{A}(t)|$.

Give the definition of epoch number $v'(t)$, we have the following theoretical results for the non-convex problem.

Theorem 8.4. Under Assumptions 8.1–8.3, to achieve the ϵ -first-order stationary point of Problem 8.1, i.e., $\mathbb{E}\|\nabla f(w)\| \leq \epsilon$ for

stochastic variable w has, for VFB²-SGD, let $\gamma = \frac{\epsilon}{L_*qG}$, if $\tau \leq \frac{512qG}{\epsilon^2}$, the total epoch number T should satisfy

$$T \geq \frac{\mathbb{E}[f(w^0) - f^*] L_*qG}{\epsilon^2} \quad (8.14)$$

where $L_* = \max\{L, \{L_\ell\}_{\ell=1}^q, L_g\}$, $\tau = \max\{\tau_1^2, \tau_2^2, \eta_2^2\}$, $f(w^0)$ is the initial function value and f^* is defined in Equation 8.13.

Theorem 8.5. Under Assumptions 8.1–8.3, to solve Problem 8.1 with VFB²-SVRG, let $\gamma = \frac{m_0}{L_*n^\alpha}$, where $0 < m_0 < \frac{1}{8}$, $0 < \alpha \leq 1$, if epoch number N in an outer loop satisfies $N \leq \lfloor \frac{n^\alpha}{2m_0} \rfloor$, and $\tau < \min\{\frac{n^{2\alpha}}{20m_0^2}, \frac{1-8m_0}{40m_0^2}\}$, we have

$$\frac{1}{T} \sum_{s=1}^S \sum_{t=0}^{N-1} \mathbb{E} \|\nabla f(w_{t_0}^s)\|^2 \leq \frac{L_*n^\alpha \mathbb{E}[f(w_0) - f(w^*)]}{T\sigma} \quad (8.15)$$

where T is the total number of epochs, t_0 is the start iteration of epoch t , σ is a small value independent of n .

Theorem 8.6. Under Assumptions 8.1–8.3, to solve Problem 8.1 with VFB²-SAGA, let $\gamma = \frac{m_0}{L_*n^\alpha}$, where $0 < m_0 < \frac{1}{20}$, $0 < \alpha \leq 1$, if total epoch number T satisfies $T \leq \lfloor \frac{n^\alpha}{4m_0} \rfloor$ and $\tau < \min\{\frac{n^{2\alpha}}{180m_0^2}, \frac{1-20m_0}{40m_0^2}\}$, we have

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E} \|\nabla f(w_{t_0})\|^2 \leq \frac{L_*n^\alpha \mathbb{E}[f(w_0) - f(w^*)]}{T\sigma} \quad (8.16)$$

Remark 8.2. For non-convex Problems 8.1, given conditions in the theorems, the convergence rate of VFB²-SGD is $\mathcal{O}(1/\sqrt{T})$, and those of VFB²-SVRG and VFB²-SAGA are $\mathcal{O}(1/T)$ (Figure 8.2).

8.4.3 Security Analysis

We discuss the data security and model security of VFB² under two semi-honest threat models commonly used in security analysis (Cheng *et al.*, 2019; Xu *et al.*, 2019; Gu *et al.*, 2020a). Specially, these two threat models have different threat abilities, where threat model 2 allows collusion between parties while threat model 1 does not.

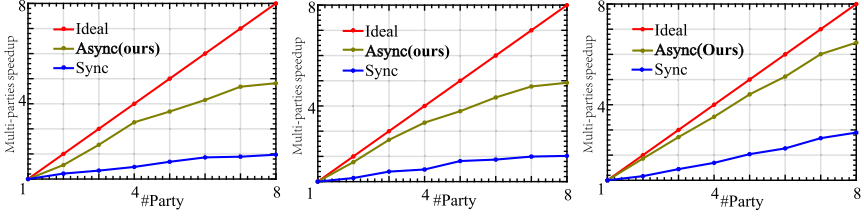


Figure 8.2. q -parties speedup scalability with $m = 2$ on Dataset D_4 . Corresponding algorithms are SGD, SVRG and SAGA from left to right, respectively.

- **Honest-but-curious (threat model 1):** All workers will follow the algorithm to perform the correct computations. However, they may use their own retained records of the intermediate computation result to infer other worker’s data and model.
- **Honest-but-colluding (threat model 2):** All workers will follow the algorithm to perform the correct computations. However, some workers may collude to infer other worker’s data and model by sharing their retained records of the intermediate computation result.

We prove the security of VFB² by analyzing and proving its ability to prevent inference attack defined as follows.

Definition 8.3 (Inference attack). An inference attack on the ℓ th party is to infer $(x_i)_{\mathcal{G}_\ell}$ (or $w_{\mathcal{G}_\ell}$) belonging to other parties or y_i hold by active parties without directly accessing them.

Lemma 8.1. *Given an equation $o_i = w_{\mathcal{G}_\ell}^\top(x_i)_{\mathcal{G}_\ell}$ or $o_i = \frac{\partial \mathcal{L}(\hat{w}^\top x_i, y_i)}{\partial (\hat{w}^\top x_i)}$ with only o_i being known, there are infinite different solutions to this equation.*

The proof of Lemma 8.1 is shown in the arXive version. Based on Lemma 8.1, we obtain the following theorem.

Theorem 8.7. *Under two semi-honest threat models, VFB² can prevent the inference attack.*

Feature and model security: During the aggregation, the value of $o_i = w_{\mathcal{G}_\ell}^\top(x_i)_{\mathcal{G}_\ell}$ is masked by δ_ℓ and just the value of $w_{\mathcal{G}_\ell}^\top(x_i)_{\mathcal{G}_\ell} + \delta_\ell$ is transmitted. Under threat model 1, one even cannot access the true value of o_i , let alone using relation $o_i = w_{\mathcal{G}_\ell}^\top(x_i)_{\mathcal{G}_\ell}$ to refer $w_{\mathcal{G}_\ell}^\top$.

and $(x_i)_{\mathcal{G}_\ell}$. Under threat model 2, the random value δ_ℓ has risk of being removed from term $w_{\mathcal{G}_\ell}^\top(x_i)_{\mathcal{G}_\ell} + \delta_\ell$ by colluding with other parties. Applying Lemma 1 to this circumstance, and we have that even if the random value is removed, it is still impossible to exactly refer $w_{\mathcal{G}_\ell}^\top$ and $(x_i)_{\mathcal{G}_\ell}$. Thus, the aggregation process can prevent inference attack under two semi-honest threat models.

Label security: When analyzing the security of label, we do not consider the collusion between active parties and passive parties, which will make preventing labels from leaking meaningless. In the backward updating process, if a passive party ℓ wants to infer y_i through the received ϑ , it must solve the equation $\vartheta = \frac{\partial \mathcal{L}(\hat{w}^\top x_i, y_i)}{\partial (\hat{w}^\top x_i)}$. However, only ϑ is known to party ℓ . Thus, following from Lemma 1, we note that it is impossible to exactly infer the labels. Moreover, the collusion between passive parties has no threats to the security of labels. Therefore, the backward updating can prevent inference attack under two semi-honest threat models.

8.5 Experiments

In this section, extensive experiments are conducted to demonstrate the efficiency, scalability and losslessness of our algorithms. More experiments are presented in the arXiv version.

8.5.1 Experiment Settings

All experiments are implemented on a machine with four sockets, and each socket has 12 cores. To simulate the environment with multiple machines (or parties), we arrange an extra thread for each party to schedule its k threads and support communication with (threads of) the other parties. We use MPI to implement the communication scheme. The data are partitioned vertically and randomly into q non-overlapped parts with nearly equal number of features. The number of threads within each parties, i.e., k , is set as m . We use the training dataset or randomly select 80% samples as the training data, and the testing dataset or the rest as the testing data. An optimal learning rate γ is chosen from $\{5e^{-1}, 1e^{-1}, 5e^{-2}, 1e^{-2}, \dots\}$ with regularization coefficient $\lambda = 1e^{-4}$ for all experiments.

Table 8.1. Dataset descriptions.

	Financial		Large scale	
	D_1	D_2	D_3	D_4
#Samples	24,000	96,257	17,996	175,000
#Features	90	92	1,355,191	16,609,143

Datasets: We use four classification datasets summarized in Table 8.1 for evaluation. Especially, D_1 (UCICreditCard) and D_2 (GiveMeSomeCredit) are the real financial datasets from the Kaggle website,¹ which can be used to demonstrate the ability to address real-world tasks; D_3 (news20) and D_4 (webspam) are the large-scale ones from the LIBSVM (Chang and Lin, 2011) website.² Note that we apply one-hot encoding to categorical features of D_1 and D_2 , thus the numbers of features become 90 and 92, respectively.

Problems: We consider ℓ_2 -norm regularized logistic regression problem for μ -strong convex case

$$\min_{w \in \mathbb{R}^d} f(w) := \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-y_i w^\top x_i}) + \frac{\lambda}{2} \|w\|^2 \quad (8.17)$$

and the non-convex logistic regression problem

$$\min_{w \in \mathbb{R}^d} f(w) := \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-y_i w^\top x_i}) + \frac{\lambda}{2} \sum_{i=1}^d \frac{w_i^2}{1 + w_i^2}$$

8.5.2 Evaluations of Asynchronous Efficiency and Scalability

To demonstrate the asynchronous efficiency, we introduce the synchronous counterparts of our algorithms (i.e., synchronous VFL algorithms with BUM, denoted as **VFB**) for comparison. When implementing the synchronous algorithms, there is a synthetic straggler party which may be 30–50% slower than the faster party

¹<https://www.kaggle.com/datasets>.

²<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>.

to simulate the real application scenario with unbalanced computational resource.

Asynchronous efficiency: In these experiments, we set $q = 8$, $m = 3$ and fix γ for algorithms with the same SGD type but in different parallel fashions. As shown in Figures 8.3 and 8.4, the loss v.s. run time curves demonstrate that our algorithms consistently outperform their synchronous counterparts regarding the efficiency.

Moreover, from the perspective of loss v.s. epoch number, we note that algorithms based on SVRG and SAGA have a better convergence rate than that of SGD-based algorithms, which is consistent with the theoretical results.

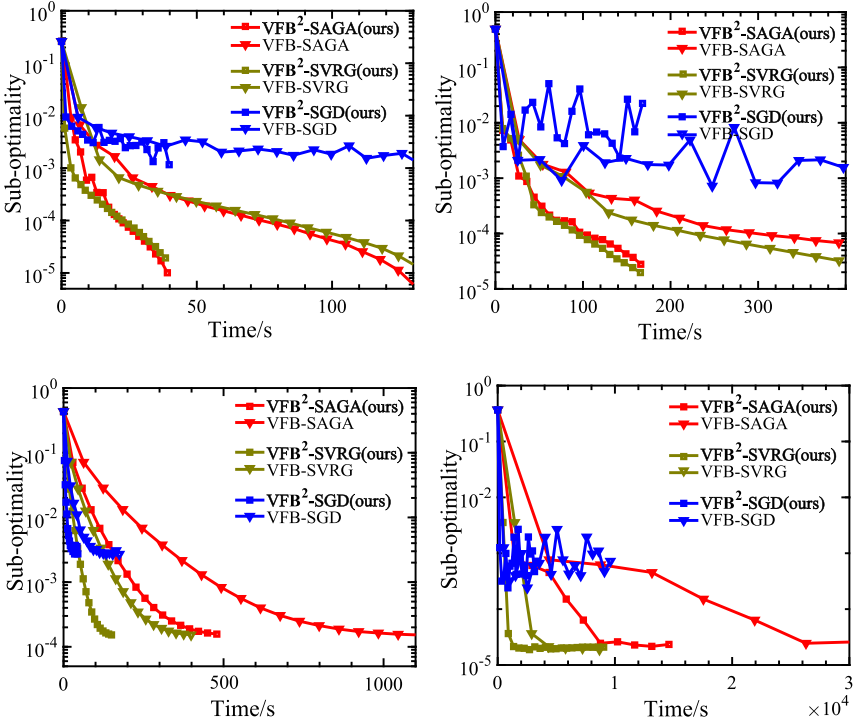


Figure 8.3. Results for solving μ -strongly convex VFL models (Problem 8.17), where the number of epochs (points) denotes how many passes over the dataset the algorithm makes. Dataset being used is D_1 , D_2 , D_3 , D_4 from left to right, respectively.

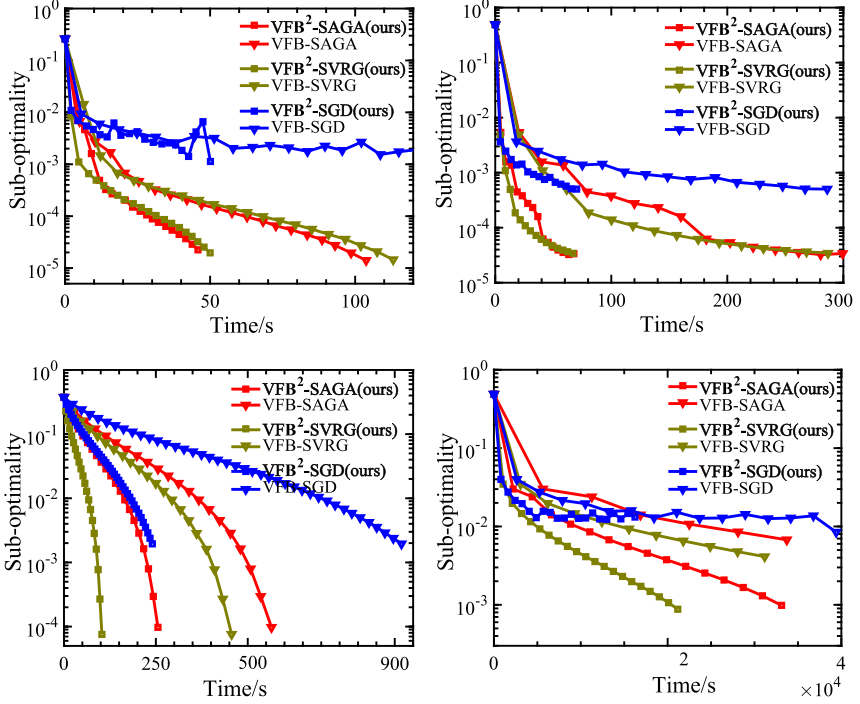


Figure 8.4. Results for solving non-convex VFL models (Problem 8.18), where the number of epochs (points) denotes how many passes over the dataset makes. Dataset being used is D_1 , D_2 , D_3 , D_4 from left to right, respectively.

Asynchronous scalability: We also consider the asynchronous speedup scalability in terms of the number of total parties q . Given a fixed m , q -parties speedup is defined as

$$q\text{-parties speedup} = \frac{\text{run time of using 1 party}}{\text{run time of using } q \text{ parties}} \quad (8.18)$$

where run time is defined as time spent on reaching a certain precision of sub-optimality, i.e., $1e^{-3}$ for D_4 . We implement experiment for Problem (8.18), results of which are shown in Figure 8.2. As depicted in Fig. 8.2, our asynchronous algorithms has much better q -parties speedup scalability than synchronous ones and can achieve near-linear speedup.

Evaluation of losslessness: To demonstrate the losslessness of our algorithms, we compare VFB²-SVRG with its non-federated

Table 8.2. Accuracy of different algorithms to evaluate the losslessness of our algorithms (10 trials).

	Algorithm	D_1 (%)	D_2 (%)	D_3 (%)	D_4 (%)
Problem (8.17)	Non-F	81.96 $\pm$ 0.25	93.56 $\pm$ 0.19	98.29 $\pm$ 0.21	92.17 $\pm$ 0.12
	AFSVRG-VP	79.35 $\pm$ 0.19	93.35 $\pm$ 0.18	97.24 $\pm$ 0.11	89.17 $\pm$ 0.10
	Ours	81.96 $\pm$ 0.22	93.56 $\pm$ 0.20	98.29 $\pm$ 0.20	92.17 $\pm$ 0.13
Problem (8.18)	Non-F	82.03 $\pm$ 0.32	93.56 $\pm$ 0.25	98.45 $\pm$ 0.29	92.71 $\pm$ 0.24
	AFSVRG-VP	79.36 $\pm$ 0.24	93.35 $\pm$ 0.22	97.59 $\pm$ 0.13	89.98 $\pm$ 0.14
	Ours	82.03 $\pm$ 0.34	93.56 $\pm$ 0.24	98.45 $\pm$ 0.33	92.71 $\pm$ 0.27

(Non-F) counterpart (all data are integrated together for modeling) and ERCR-based algorithm but without BUM, i.e., AFSVRG-VP proposed in (Gu *et al.* 2020b). Especially, AFSVRG-VP also uses distributed SGD method but cannot optimize the parameters corresponding to passive parties due to lacking labels. When implementing AFSVRG-VP, we assume that only half parties have labels, i.e., parameters corresponding to the features held by the other parties are not optimized. Each comparison is repeated 10 times with $m = 3$, $q = 8$, and the same stop criterion, e.g., $1e^{-5}$ for D_1 . As shown in Table 8.2, the accuracy of our algorithms are the same with those of Non-F algorithms and are much better than those of AFSVRG-VP, which are consistent with our claims.

8.6 Conclusion

In this chapter, we proposed a novel BUM for the real VFL system where only one or partial parties have labels for training models. Our new algorithms enable all parties, rather than only active parties, to collaboratively update the model and also guarantee the algorithm convergence, which was not held in other recently proposed ERCR-based VFL methods under the real-world setting. Moreover, we proposed a BAPA to make ERCR-based algorithms with backward updating more efficient in real-world tasks. Three practical SGD types of algorithms were also proposed with theoretical guarantee.

Chapter 9

Vertical Federated Deep Learning Algorithms

In recent years, machine learning has garnered increasing attention from both academia and industry. It has been broadly employed for various tasks, including multimedia concept retrieval, image classification, video recommendation, social network analysis, and text mining. Among a plethora of machine learning algorithms, “deep learning” (also referred to as representational learning) stands out as the most widely applied model. Furthermore, the explosive growth and accessibility of data, coupled with significant advancements in hardware technology, have spotlighted technologies like distributed learning and deep learning. Numerous state-of-the-art deep learning techniques have emerged, demonstrating promising prospects in diverse tasks, such as natural language processing, visual data processing, and audio processing. With society’s growing concern for privacy protection, privacy issues in these applications are under intense scrutiny. Previous chapters presented novel and effective vertical federated learning algorithms from the perspective of traditional machine learning methods but did not delve into deep learning models. This chapter seeks to address this by introducing vertical federated deep learning algorithms.

Algorithm highlights: This algorithm caters to deep learning models, in contrast to the prior chapters which primarily targeted linear

models. This algorithm supports asynchronous updates, rendering it more efficient than synchronous methods.

9.1 Introduction

Federated learning has drawn vast interest from both the scholarly community and industry as it technically fulfills collaborative modeling needs while upholding privacy protections. Depending on the data partitioning approach, existing federated learning frameworks can be broadly classified into two categories: horizontal and vertical federated learning. In horizontal federated learning, all participants possess identical sample features but different sample IDs. In contrast, vertical federated learning scenarios see different participants holding identical sample IDs but distinct feature subsets. Such scenarios are frequently encountered in emerging cross-organization collaborative modeling applications, including but not limited to medical research, risk assessment, and targeted marketing. For instance, e-commerce companies with online shopping data can collaborate with banks and digital finance firms, who possess other information about the same individual (e.g., monthly average deposits and online expenditures), to jointly train models, aiming for precise customer profiling and analysis.

Existing vertical federated learning algorithms are mainly tailored for traditional machine learning models, such as logistic regression, linear regression, and support vector machines, as delineated in Chapters 6–8. Thus, introducing vertical federated deep learning algorithms emerges as a vital complement to these methods.

9.2 Vertical Federated Deep Learning Algorithms

Specifically, consider a generic vertical federated learning system comprising q participants and a server (or for feature-distributed data-driven machine learning) where each participant possesses a portion of vertically partitioned feature data, while the server (a participant or a trusted third party) retains label data. In this system, all participants and the server aim to address the following composite

form of the finite-sum problem:

$$f(w_0, \mathbf{w}) := \underbrace{\frac{1}{n} \sum_{i=1}^n F_0(w_0, c_{i,1}, \dots, c_{i,q}; y_i)}_{f_i(w_0, \mathbf{w})} + \lambda \sum_{m=1}^q g(w_m)$$

where $c_{i,m} = F_m(w_m; x_{i,m}) \quad \forall m \in [q]$ (9.1)

Here, $f_i(w_0, \mathbf{w}): \mathbb{R}^d \rightarrow \mathbb{R}$ represents the loss function on the i th sample, $\mathbf{w} = \{w_1, \dots, w_q\}$, for $m \in [q]$, $w_m \in \mathbb{R}^{d_m}$ (given a positive integer q , the symbol $[q]$ represents the set $\{1, \dots, q\}$) defines a local model F_m on the participant m , which will map the input $x_{i,m}$ to output $c_{i,m}$, where $w_0 \in \mathbb{R}^{d_0}$ defines a global model F_0 maintained and learned on the server side, $d = \sum_{m=0}^q d_m$, g is the regularization term, and $\mathbf{w}$ consists of individual models $w_m \in \mathbb{R}^{d_m}$ for each participant m .

Notably, problem (9.1) embodies a generic form encapsulating numerous machine learning models, such as the typical deep neural network model elucidated in the following.

Deep neural network model: If F_m is a sophisticated nonlinear model like a neural network, then $c_{i,m}$ can be represented as follows:

$$u_0 = x_{i,m} \tag{9.2a}$$

$$u_l = \sigma_l(h_l u_{l-1} + b_l), \quad l = 1, \dots, K \tag{9.2b}$$

$$c_{i,m} = u_K \tag{9.2c}$$

Here, σ_l is a linear or nonlinear activation function, with h_l and b_l corresponding to the local model parameters w_m so that $w_m = [h_1, \dots, h_K, b_1, \dots, b_K]$. K indicates the network depth. In this case, F_0 can be a straightforward neural network layer, like a fully connected layer, or any other intricate deep neural network.

9.2.1 Algorithm Framework

For the deep neural network model in the form of problem (9.1), we introduce a new framework for Vertical Federated Deep

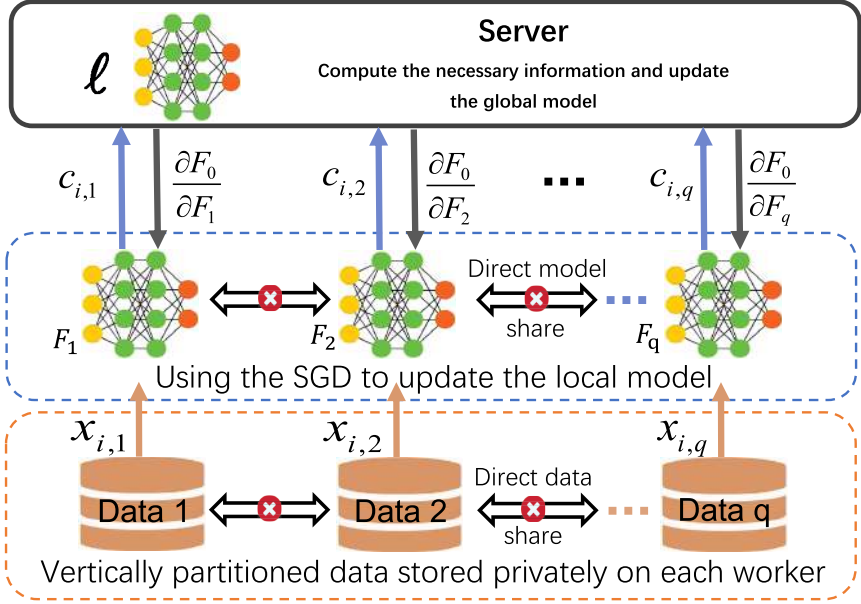


Figure 9.1. Structural diagram of VF DL.

Learning (VF DL). A schematic of this framework is shown in Figure 9.1. As depicted in Figure 9.1, the data are vertically distributed among various participants and are stored locally in a privacy-preserving manner. Both the local and global models are stored on the participants and server, respectively, with the learning occurring collaboratively. Moreover, each local model is cascaded with the global model, allowing all local models to be interconnected through this global one.

In this framework, each local model maps the input $x_{i,m}$ to its corresponding local function value. All these function values are asynchronously transmitted to the server and serve as inputs to the global model. The server then computes the partial gradient (corresponding to partial derivatives) of the global model with respect to the outputs of the local models and sends it back to the activated participants (the owners of that local model) (see line 2 of Algorithm 9.1). Upon receiving the partial gradient, the activated participants can compute the local gradient for model updating. Direct sharing of information like model parameters and data among

participants is prohibited, technically preventing direct privacy leakage. In the following, we demonstrate how to compute the corresponding gradients and introduce the vertical federated deep learning algorithm.

9.2.2 Algorithm Details

In this section, we propose an Asynchronous Vertical Federated Deep Learning (AVFDL) algorithm within the framework.

First, given a problem of the form in Equation 9.1, by the chain rule, we have

$$\frac{\partial F_0}{\partial w_m} = \frac{\partial F_0}{\partial F_m} \frac{\partial F_m}{\partial w_m} \quad (9.3)$$

where $m = [q]$. The partial derivative of the global model with respect to the output of the local model F_m is

$$\frac{\partial F_0}{\partial F_m} = \frac{\partial F_0(w_0, \mathbf{c}_i, y_i)}{\partial c_{i,m}} \quad (9.4)$$

where $\mathbf{c}_i = \{c_{i,m}\}_{m=1}^q$. It's worth noting that our proposed method is asynchronous, hence a matrix maintaining all the values of $\{\mathbf{c}_i\}_{i=1}^n$ (or vectors) is essential. If it were a synchronous version of VF DL, this matrix would not be needed. In this context, essential information, such as $c_{i,m}$ for a given sample i and all $m \in [q]$, should be efficiently transmitted during each communication. The second term in Equation (9.3) can be efficiently computed on both the client and server sides.

The specific steps of the proposed AVFDL-SGD can be found in Algorithm 9.1.

In Step 4 of the algorithm, the activated participant m computes $c_{i,m}$ using its private data and model. In Step 5, $c_{i,m}$ is sent to the server. Once the server receives $c_{i,m}$ from participant m , it calculates F_0 and its gradient using these data along with the local model function values from other participants (stored in a matrix). Note that these function values from the other $q - 1$ participants are outdated (i.e., not computed based on the latest parameters) because of asynchronous updates (this isn't an issue in the synchronous version). For participant m , it must initiate a request to the server to fetch $\frac{\partial F_0}{\partial F_m}$ and then use its locally computed $\frac{\partial F_m}{\partial w_m}$ to determine $\frac{\partial F_0}{\partial w_m}$.

Algorithm 9.1 AVFDL-SGD Algorithm

```

0: For all participants  $m \in [q]$ , initialize variables such as local
   model parameters.
1: while not converged do
2:   When User  $m$  is activated, executes:
3:     Pick a sample  $i \stackrel{\text{Unif}}{\sim} [n]$ 
4:     Calculate  $c_{i,m}$  using local data and model
5:     Upload  $c_{i,m}$  to the server
6:     Receive  $\frac{\partial F_0}{\partial F_m}$  from the server (by listening or requesting) and
       calculate  $\frac{\partial F_m}{\partial w_m}$ 
7:     Calculate the stochastic gradient estimator  $\tilde{v}_m = \frac{\partial F_0}{\partial F_m} \frac{\partial F_m}{\partial w_m}$ 
8:     Update  $w_m \leftarrow w_m - \eta_m \tilde{v}_m$ 
9:   When the server receives  $c_{i,m}$  execute:
10:    Calculate  $\frac{\partial F_0}{\partial F_m}$ 
11:    Calculate  $\tilde{v}_0 = \hat{\nabla}_{w_0} f_i(w_0, \mathbf{w})$ 
12:    Update global model parameters  $w_0 \leftarrow w_0 - \eta_0 \tilde{v}_0$ 
13:    Send  $\frac{\partial F_0}{\partial F_m}$  to user  $m$ 
14: end while

```

In this algorithm, we adopt the Stochastic Gradient Descent (SGD) as the random gradient estimation operator. Due to the inherent high variance of this estimator, the performance of the algorithm may not be optimal. To obtain a variance-reduced version, one can simply replace the update rule (or gradient estimation operator) with the corresponding rules from SVRG, SAGA, or even SPIDER algorithms. For more details on how to extend an SGD-based algorithm to SVRG or SAGA-based ones, one can refer to previous chapters, such as Chapter 7.

9.2.3 Scenario Case Study

In this section, to provide readers with an intuitive understanding of the discussed algorithm, we present a case study resembling a real-world application scenario. The objective of this case study is to use a machine learning model to address an issue on the real financial dataset UCICreditCard: predicting loan defaults based on various personal details of users. This example aims to help

readers understand how to apply the proposed algorithm in specific tasks. Specifically, we deploy the VFB² class algorithm on the UCICreditCard dataset to solve the neural network problem in the form of problem (9.1).

Scenario description: The dataset comprises 23 features, such as loan amount, gender, educational level, marital status, age, monthly repayment status, monthly bill amount, and monthly payment amount, among others. The label indicates whether there will be a loan default (where 1 denotes default and 0 signifies no default). The dataset encompasses 30,000 samples. Based on this dataset, we imagine the following scenario: Companies A, B, C, and D each possess different features of customers. Specifically, Company A has nine features, including loan amount, gender, age, and 6 months of monthly repayment status; Company B has four features: gender, education level, marital status, and age; Company C has eight features including gender, age, and 6 months of monthly bill totals; Company D has eight features: gender, age, and 6 months of monthly payment amounts. Each company's customers are not identical, meaning the sample IDs differ. Notably, Company A, the loan issuer, aims to use a machine learning model to assess potential loan defaults. Typically, Company A would employ its data to train a model. However, with only nine features and lacking crucial information like education level, marital status, and monthly bills, the model might underperform. Companies B, C, and D possess other relevant user information that can enhance Company A's model. Ideally, they would jointly train a model using features from all companies for better performance. However, due to privacy laws and business competition, direct data sharing between companies is not feasible. In this scenario, a viable approach is the use of federated learning, allowing the four companies to collaboratively learn while complying with jurisdictional regulations and ensuring data privacy.

Modeling: We initially assume that data from each company has been processed into the standard distribution format for vertical federated learning, aligning with the UCICreditCard dataset. Company A provides nine features including loan amount, gender, age, and monthly repayment for 6 months ($\mathbf{x}_{\mathcal{G}_1} = [x_1, x_2, x_6, \dots, x_{11}] \in \mathbb{R}^9$); Company B offers two features ($\mathbf{x}_{\mathcal{G}_2} = [x_3, x_4] \in \mathbb{R}^2$) of marital status and education; Company C presents six features of monthly

bill totals ($\mathbf{x}_{\mathcal{G}_3} = [x_{12}, \dots, x_{17}] \in \mathbb{R}^6$); and Company D delivers six features of monthly payment amounts ($\mathbf{x}_{\mathcal{G}_4} = [x_{18}, \dots, x_{23}] \in \mathbb{R}^6$). Each company contains 30,000 common sample IDs, with 24,000 samples used for training and 6,000 for testing. Only Company A possesses label data. The problem under this scenario can be represented as follows:

$$f(w_0, \mathbf{w}) := \underbrace{\frac{1}{n} \sum_{i=1}^n F_0(w_0, c_{i,1}, \dots, c_{i,q}; y_i)}_{f_i(w_0, \mathbf{w})} + \lambda \sum_{m=1}^q \|w_m\|^2$$

$$\text{where } c_{i,m} = F_m(w_m; x_{i,m}) \quad \forall m \in [q] \quad (9.1)$$

Here, $f_i(w_0, \mathbf{w}): \mathbb{R}^d \rightarrow \mathbb{R}$ is the loss function on the i th sample, with $q = 4$, $n = 24,000$, $d = 23$. The data $x = [x_1, x_2, x_3 \square x_4]$ are located in Companies A, B, C, and D, respectively. The parameters $\mathbf{w} = \{w_0, w_1, w_2, w_3, w_3 \square w_4\}$, and for $m \in [4]$ each w_m defines a local model F_m in Companies A, B, C, and D. This model maps input $x_{i,m}$ to output $c_{i,m}$. Notably, $w_0 \in \mathbb{R}^{d_0}$ specifies a global model F_0 maintained and learned server side.

Model learning: From the scenario, only Company A has label data, making it Participant 1 and also serving as the server or Participant 0. Companies B, C, and D represent Participants 2, 3, and 4, respectively. Supposing Company B is the client and Company A the server, according to Algorithm 9.1, Company B randomly selects sample i , computes $c_{i,2}$ using its local data and model w_2 , and uploads $c_{i,2}$ to Company A. Upon receiving $c_{i,2}$, Company A first calculates $\frac{\partial F_0}{\partial F_2}$, the partial derivative of the global model F_0 with respect to F_2 (i.e., $c_{i,2}$). Then, as shown in Step 11, it computes the derivative of f_i with respect to the global model and updates the global model in Step 12. Finally, Company A sends $\frac{\partial F_0}{\partial F_m}$ to Company B. Upon receipt, Company B uses the chain rule to determine the derivative of f_i with respect to parameter w_2 which is $\frac{\partial F_0}{\partial F_2} \frac{\partial F_m}{\partial w_m}$ and updates its local model w_2 in Step 8. Step 13 in Algorithm 9.1 strictly needs to follow Step 10. As Companies A, B, C, and D all have user roles, they must execute Steps 3–8. Additionally, since Company A also serves as the server, it also runs Steps 10–13. While Company A plays both user and server roles, the algorithmic steps for each role

run independently in parallel. Furthermore, a secure third party can also act as the server instead of Company A. Ultimately, all companies asynchronously iterate the process until convergence, achieving the corresponding models.

9.3 Theoretical Analysis

This section provides a technical analysis of the privacy, security, and complexity of the algorithm.

9.3.1 Complexity Analysis

In Step 4, the calculation of $c_{i,m}$ requires $\mathcal{O}(d_m)$ multiplication operations, resulting in a computational complexity of $\mathcal{O}(d_m)$. Similarly, the computational complexities of Steps 6, 7, and 8 are also $\mathcal{O}(d_m)$. The combined computational complexity of Steps 10, 11, and 12 is $\mathcal{O}(d_0)$. Therefore, the total computational complexity of Algorithm 9.1 is $\mathcal{O}(d_m + d_0)$. Within Algorithm 9.1, only the partial derivative of the global model with respect to the local model's output is transmitted between user m and the server. Thus, the overall communication complexity of Algorithm 9.1 is $\mathcal{O}(1)$. This is based on the assumption that the local model's output is scalar; if it's a d_m -dimensional vector, the communication complexity would be $\mathcal{O}(d_m)$. Hence, the algorithm has a relatively low communication complexity, especially when the local model's output is scalar.

9.3.2 Security Analysis

To technically ensure the privacy of the data and model within the algorithm, all data $(x_i)_m$ and corresponding model parameters w_m are securely stored only at the respective participant or server side. No participant in the scheme can directly transfer the data $(x_i)_m$ and local model parameters w_m to another participant. As demonstrated in Algorithm 9.1, only the local model's output and some partial derivatives are transmitted. Given the complexity of the local model itself, it's challenging to infer both the model parameters and feature data from its known output. Detailed security analysis on this aspect can be referred to in the previous chapters. The partial derivative

vector also offers minimal insight into valuable information, at least no current studies indicate that it might leak model or data privacy. Consequently, the proposed asynchronous federated deep learning algorithm can technically safeguard data and model privacy.

9.4 Experiments

In this section, we validate the effectiveness of the proposed algorithm on two commonly used deep learning datasets.

9.4.1 Experimental Setup

For the experiments, we use Gloo as the communication mechanism and implement the deep network using PyTorch. Similar to other sections, we partition the data by features (i.e., vertical partitioning) into q non-overlapping parts, each with almost equal feature count. For all participants (clients), their step size γ is chosen from the set $\{5e^{-1}, 1e^{-1}, 5e^{-2}, 1e^{-2}, \dots\}$, while the server-side step size is set to γ/q (since the global updates at the server side occur at q times the frequency of the local models). The datasets used are the commonly employed MNIST and Fashion MNIST. We set the number of participants to 4, i.e., $q = 4$, and assume a secure third-party on the server side.

The experiments use a model based on a Fully Connected Network (FCN). The local model consists of a two-layer FCN ($784 * 128 * 1$) with ReLU as the activation function. The global model also has two layers, with the first being a $q * 10$ fully connected layer, followed by a Softmax layer.

9.4.2 Results and Discussion

To highlight the effectiveness of the proposed asynchronous algorithm, we designed a synchronous version (i.e., the VF DL algorithm) for comparison. Figure 9.2 showcases the experimental results, where the y -axis represents the difference between function loss values and optimal values, while the x -axis indicates the runtime. The results demonstrate that our method can efficiently solve deep models. Moreover, the proposed asynchronous algorithm significantly improves efficiency compared to its synchronous counterpart.

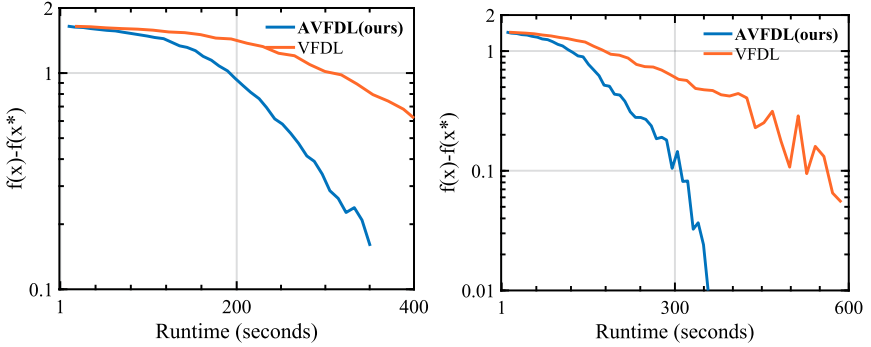


Figure 9.2. Experimental results of the algorithm on two common datasets. The left corresponds to the MNIST dataset, and the right corresponds to the Fashion MNIST dataset.

9.5 Conclusion

In this chapter, under the setting of vertical federated learning, we introduced an asynchronous framework for deep learning, termed the AVF DL framework. Within this framework, we presented the AVF DL-SGD algorithm based on stochastic gradient descent. The asynchronous nature of this algorithm allows for better utilization of computational resources across participants. Experimental results further demonstrated that this asynchronous approach is more efficient than its synchronous counterpart. Importantly, as the majority of preceding chapters mainly focus on traditional machine learning, this chapter serves as a supplementary content, ensuring comprehensiveness.

This page intentionally left blank

Chapter 10

Faster Secure Data Mining Framework via Homomorphic Encryption

Due to the rising privacy demand in data mining, Homomorphic Encryption (HE) is receiving more and more attention recently for its capability to do computations over the encrypted field. By using the HE technique, it is possible to securely outsource model learning to the not fully trustful but powerful public cloud computing environments. However, HE-based training scales badly because of the high computation complexity. It is still an open problem whether it is possible to apply HE to large-scale problems. In this chapter, we propose a novel general distributed HE-based data mining framework toward one step of solving the scaling problem. The main idea of our approach is to use the slightly more communication overhead in exchange of shallower computational circuit in HE, so as to reduce the overall complexity. We verify the efficiency and effectiveness of our new framework by testing over various data mining algorithms and benchmark datasets. For example, we successfully train a logistic regression model to recognize the digits 3 and 8 within around 5 minutes, while a centralized counterpart needs almost 2 hours.

Framework highlights: This framework is a distributed data mining framework based on homomorphic encryption; this framework uses communication costs in exchange for shallower computing circuits in homomorphic encryption, thereby greatly reducing the overall computational complexity.

10.1 Introduction

The past decade has witnessed a surge of revolutionary breakthrough in high technology, accompanied with an avalanche of the large-scale data in many fields. In industry, the data are no longer hosted on a single server or cluster and more computational resources are required to analyze these data. The cloud computing and distributed machine learning techniques have emerged to be the ideal solutions for big and cost-effective data storage and analysis. However, at the same time, it has also raised security and privacy concerns for using sensitive user data in public cloud computing, especially in the financial and medical fields. As a result, states across the world enact laws to protect the privacy of user data. For example, the famous General Data Protection Regulation (GDPR) passed by European Union in 2018 sets up strict rules for companies to use personal data. Traditional encryption techniques mainly address the security and privacy issues in the data storage and transfer tasks but not the data analysis task. This poses great challenges to the machine learning and data mining researchers, i.e., how can we securely learn the machine learning and data mining models in the public cloud computing and distributed machine learning environments, which may not be fully trustful.

Existing approaches proposed can be roughly divided into four main categories, i.e., Differential Privacy (DP) by Dwork *et al.* (2006), secure Multi-Party Computation (MPC) by Yao (1986), Homomorphic Encryption (HE) by Gentry *et al.* (2009), and secure enclave by Champagne and Lee (2010). Every approach has its advantages and drawbacks:

Differential privacy (DP): DP has strong theoretical guarantees to preserve privacy but leads to performance degradation which is not acceptable in many cases, what's more, privacy parameters' choice in DP such as ϵ is still a mystery.

Secure multi-party computation (MPC): MPC enables multiple parties to securely evaluate a function without revealing their own input, but it heavily relies on communication and is usually not trivial to formulate a model under MPC framework.

Homomorphic encryption (HE): HE is well known for its ability to do computations in the encrypted field, but it has notoriously high computation overhead.

Secure enclave (SE): SE outsources the model training to a trusted third party. Obviously, its security guarantee is totally relied on the design of this third party system. There have been many attacks targeted at such kinds of systems like the famous side channel attack.

To sum up, it is still challenging to deploy a large-scale secure data mining system based on the current status of these techniques. In this chapter, we want to take one step further to the building of a practical secure data mining system by focusing on decreasing the time complexity of the HE-based data mining systems.

More precisely, we propose a novel distributed secure data mining framework which achieves fast model learning over homomorphic encryption. The design of our framework is based on identifying a key feature of the homomorphic encryption: HE actually performs approximate computation and the noise accumulated in the ciphertext after each operation. Naturally, plaintext cannot be correctly recovered if the noise grows too much. As a result, a technique called bootstrap has been invented by the cryptographic researchers to control the noise. However, bootstrap is a major source of time complexity. It is both extremely expensive (at least $10\times$ slower compared to other basic operations) and leads to higher complexity of other operations. Naturally, we can save a lot of computation if the bootstrap is not performed. Next, we show that this is possible for data mining models, especially under the distributed learning scenario. Actually, data mining models are relatively insensitive to the noise.

Research by Zhu *et al.* (2016) shows that it is possible to compress the model parameters as low as 1 bit without losing much performance. More importantly, the depth of the computational circuit in distributed learning is bounded as the worker will refresh its parameter every few iterations. Our framework is based on this simple observation and we show later in Section 10.4 that our framework achieves at least $10\times$ times of training speed boost while keeping the same model performance. As a simple example, our framework trains a logistic regression model over MNIST dataset to the accuracy of

96.5% within around 5 minutes in contrast with nearly 2 hours training time of a traditional HE-based learning framework. Finally, note that our system protects the data privacy in that the worker has access only to the encrypted data and have no idea about the original data.

10.2 Related Work

Before proposing the framework, we first introduce some preliminaries. For the part of homomorphic encryption, please refer to Chapter 3.

HE supports arithmetic computation over the encrypted field. “Homomorphic” here means that the encryption function and decryption function are homomorphism between plaintext and ciphertext. More precisely, suppose z_i denotes the plaintext, s_i denotes the ciphertext, and π and π^{-1} denote the encryption and decryption function, respectively. If we have

$$\begin{aligned} s_1 &= \pi(z_1) \\ s_2 &= \pi(z_2) \end{aligned} \tag{10.1}$$

and

$$\begin{aligned} z_0 &= z_1 \otimes z_2 \\ s_0 &= s_1 \tilde{\otimes} s_2 \end{aligned} \tag{10.2}$$

where $\otimes$ denotes any arithmetic computation, e.g., addition and multiplication, while $\tilde{\otimes}$ denotes the encrypted version of the corresponding computation, then we have $s_0 = (z_0)$.

HE, which is viewed as the Holy Grail of cryptography, dates back to the 1970s when RSA technique was proposed. Different types of HE schemas are proposed since then. Paillier *et al.* (1999) proposed a framework that only supports homomorphic addition. Frameworks like Paillier’s which only support certain HE operations are referred as partially HE later, while more powerful frameworks like the one proposed by Brakerski *et al.* (2014) which support both addition and multiplication are called somewhat (leveled) HE. They

are not fully HE frameworks as they constrain the maximum depth of the computational circuit. The first real fully HE (FHE) framework which supports the evaluation of arbitrary computational circuit was proposed by Gentry in 2009. A key innovation of this framework is the use of bootstrap technique to control the noise of the ciphertext. Actually, the core idea of bootstrap is just to re-encrypt the ciphertext but in a homomorphic way, i.e., define a computational circuit in the encryption field that is homomorphic to the operation of decryption and encryption and then evaluate this circuit. The initial design proposed by Gentry is extremely slow (several orders slower than other basic homomorphic operations, such as addition and multiplication). An important research topic afterward, including papers by Brakerski and Vaikundanathan (2011, 2014), is to improve the practicability of the fully HE proposed by Gentry. Specially, our framework is built upon a recently proposed FHE framework named Homomorphic Encryption for Arithmetic of Approximate Numbers (HEAAN). This framework has two main features. First, it supports homomorphic rounding operation. As a result, the magnitude of the ciphertext modulus is linear with respect to the depth of the computational circuit in contrast to the exponential relation in other FHE frameworks. Moreover, the rounding operation is particularly suitable for machine learning applications, where the precision of the model parameters is of less concern. The second feature of this framework is that it supports the packing operation, where one ciphertext can encode a set of plaintext (parameters). As a result, operation on ciphertext is equivalent to doing parallel computation for the plaintext, which improves the training speed.

We now briefly overview the recent development of HE-based data mining, which can be looked from multiple angles: From the perspective of HE frameworks, partially HE frameworks are widely used in early days for their simplicity, however, their applications are greatly restricted due to limited capabilities, e.g., Hardy in 2017 showed that applications based on the paillier framework need to avoid encrypted multiplication by careful algorithmic design. In the era of FHE, it is possible to train arbitrary model completely over the encrypted data in theory, as shown in the research by Carpv and Crawford in 2019 and 2018, respectively. However, the complexity becomes the major bottleneck and prevents the full exploitation of

FHE's ability. From the perspective of data mining models, the most previous research focuses on shallow models, e.g., logistic regression by Crawford *et al.* (2018), decision tree by Cheng *et al.* (2019), and clustering by Cheon *et al.* (2018b). As for the recent popularized deep models, there is effort by Brutzkus *et al.* in 2016, 2018, respectively, on making inference over the private data but not much on training. The major difficulty comes from the high complexity. On the one hand, the computation circuit of the gradient evaluation is much deeper compared to that of the shallow models, and on the other hand, deep models typically need more iterations to converge. The two factors combined together make it impractical to train a deep model over the current HE system. Finally, there is also effort trying to make adaptations between the HE framework and the data mining models to make them more suitable for the other. In the HE side, Jiang *et al.* (2018) and Mishra *et al.* (2018) proposed fast matrix-vector multiplication kernels for the acceleration of HE-based data mining, and Crawford *et al.* (2018) made customized optimization for frequently used homomorphic operations by data mining like comparisons and inversion. In the model side, there are two main directions. One is to reduce the depth of computational circuit, e.g., Cheon *et al.* (2018b) proposed to use ensemble methods so that each model can converge within less iterations and as a result, to reduce the circuit depth. The other, like the work done by Han and Bonte and Vercauteren, is to linearize the machine learning models using methods like Taylor expansion or regression so as to alleviate the lack of efficient nonlinear operations in most HE frameworks.

10.3 Faster Secure Data Mining Framework via Homomorphic Encryption

In this section, we introduce the data mining framework and describe the proposed algorithm in detail.

10.3.1 Algorithm Framework

The various parts of the framework are described in the following.

First, we introduce the fully HE framework HEAAN, which is composed of three main components, i.e., encode/decode, encrypt/

decrypt, and basic homomorphic operations (addition, multiplication, etc.). We briefly review them as follows (refer to the work by Cheon *et al.* (2017) for more detailed description):

Encode/Decode: Process transforms a vector of plaintext to/from a polynomial in the N -th order cyclotomic polynomial ring (equivalently, the coefficient vector of the polynomial) by the canonical embedding technique.

Encrypt/Decrypt: Process transforms the plaintext to/from ciphertext which are both cyclotomic polynomials. The public key and private key are needed in this process and are also cyclotomic polynomials. The security of this process is guaranteed by the hardness of Ring Learning-with-Errors (R-LWE) problem (see Chapter 3).

Homomorphic operations: Supported by the framework are addition, multiplication, bootstrap, and scaling operation. Bootstrap is for refreshing the noisy ciphertext and scaling performs homomorphic rounding operation so as to keep the modulus of the ciphertext linear with respect to the depth of the computational circuit.

Besides, there are two important complexity parameters of the framework. One is the order N of the cyclotomic polynomial ring, which decides the number of plaintext we can pack into one ciphertext. The other is the magnitude M of the ciphertext modulus which controls the complexity of basic homomorphic operations. Roughly speaking, we want to maximize N and minimize M to accelerate the computation, but M and N are positively correlated, so there is a trade-off between the two.

In 2018, Han *et al.* (2018a) trained a logistic regression model based on HEAAN and their model is referred as the centralized HE learner and is used as the baseline for comparison. We briefly mention two useful tricks proposed by Han *et al.* (2018a) First, the authors design a customized data packing schema for logistic regression. More specifically, suppose we have an $N \times D$ data matrix X , where N is the number of samples and D denotes the sample dimension. Then to encrypt X , we divide it into $l * h$ small blocks and encrypt each block with one ciphertext. The parameter vector is grouped and encrypted using a similar way. As mentioned in the work by Han *et al.* (2018a), the proposed packing method outperforms other more straightforward ways, e.g., packing by rows or columns, in terms of

the training speed. Next, the authors propose to use regression to linearize the nonlinear functions such as the sigmoid function which contrasts with the usual Taylor expansion approach. The authors show that the regression outperforms the Taylor expansion way. The regression way works as follows: Suppose we want to fit a nonlinear function $f(x)$ with M th-order polynomial $p(x) = \sum_{i=1}^M \alpha_i x^i$, then we solve the following high-order least square problem:

$$\min_{\alpha \in \mathbb{R}^M} \sum_{j=1}^N \frac{1}{2} (f(x_j) - p(x_j))^2 \quad (10.3)$$

where $\{x_j\}_{j \in [N]}$ are randomly sampled. In this chapter, we follow this idea to approximate other nonlinear activation functions, such as the hinge loss function used in the SVM algorithm.

10.3.2 Algorithm Description

We utilize the parameter server model (detailed in Chapter 11) for distributed model training. Within this framework, the server is assumed to be trustworthy, possessing private keys for both encryption and decryption. Meanwhile, worker nodes train the model in a privacy-preserving manner, accessing only encrypted data and encrypted model parameters. As illustrated in Figure 10.1, this framework encompasses two stages: the initialization stage and the training stage. In the initialization stage, the server first generates

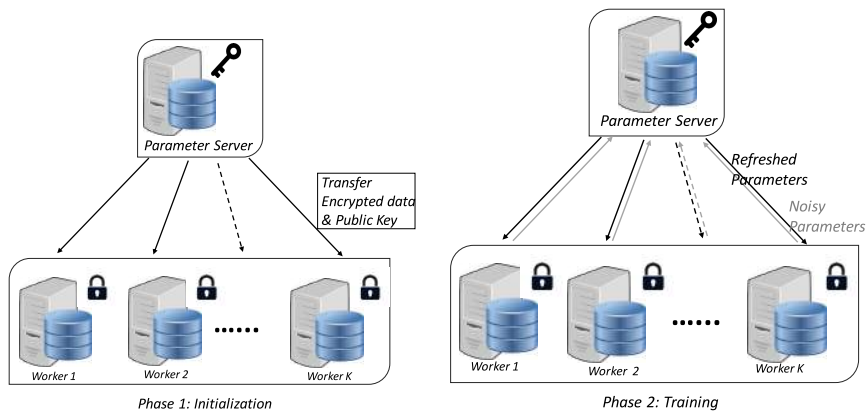


Figure 10.1. Two phases of our training framework: Left figure shows initialization phase, and right figure shows the training phase.

Algorithm 10.1 Training Algorithm for Worker

- 1: **Input:** Learning rate η , public key pk , number of iterations l to update the local weights, number of training iterations K ;
 - 2: **Output:** Converged encrypted parameter $\pi(\omega_K)$;
 - 3: **Initialization Phase:**
 - 4: Receive data from the parameter server, i.e. encrypted data-set $\{\pi(x_i), \pi(y_i)\}_{i \in [N]}$, public key pk and initial encrypted parameter $\pi(\omega_0)$;
 - 5: **Training Phase:**
 - 6: **for** $k = 1$ to K **do**
 - 7: Update $\pi(\omega)$ by evaluating Eq. 10.8 homomorphically;
 - 8: **if** $k \% l = 0$ **then**
 - 9: Send $\pi(\omega)$ to the parameter server and wait until the parameter server sends back the updated weights;
 - 10: **end if**
 - 11: **end for**
-

Algorithm 10.2 Training Algorithm for Parameter Server

- 1: **Input:** Training set $\{x_i, y_i\}_{i \in [N]}$, initial parameter ω_0 ;
 - 2: **Output:** Converged parameter ω_K ;
 - 3: **Initialization Phase:**
 - 4: Generate public and secret keys, i.e. pk and sk , encrypt training set $\{x_i, y_i\}_{i \in [N]}$ and initial parameter ω_0 ;
 - 5: Send the encrypted data-set $\{\pi(x_i), \pi(y_i)\}_{i \in [N]}$, public key pk and initial encrypted parameter $\pi(\omega_0)$ to each worker;
 - 6: **Training Phase:**
 - 7: **while** the workers are running **do**
 - 8: **if** receive request from workers **then**
 - 9: Decrypt the encrypted parameter $\pi(\omega)_{param}$ received from the worker to get ω_{param} ;
 - 10: Update its own ω with ω_{param} ;
 - 11: Encrypt ω and send back to the worker;
 - 12: **end if**
 - 13: **end while**
-

a public-private key pair and encrypts the data using the generated keys. Subsequently, the data are partitioned into chunks and allocated to the respective worker nodes. In the training stage, each worker node trains on the encrypted data until the noise in the encrypted parameters reaches a predetermined threshold. The worker nodes then send the encrypted parameters back to the server, requesting an update. On the server side, it awaits requests from the worker nodes. Upon receiving an update request, the server decrypts the parameters, updates its parameter replica, and then returns the newly encrypted parameters to the worker nodes.

It is noteworthy that this framework is versatile and isn't dependent on any specific homomorphic encryption scheme, data mining model, or distributed learning framework. Anytime the bootstrap operation in any model or framework becomes the bottleneck for application, this framework can significantly accelerate training. For the sake of simplicity, we consider synchronous training of nonlinear models based on the HEAAN framework. Specifically, the following linear models are investigated:

$$\min_{\omega \in \Omega \subset R^d} f(\omega) = \min_{\omega \in \Omega \subset R^d} \sum_{i=1}^n L(y_i \times f(x_i)) + R(\omega) \quad (10.4)$$

where $\{x_i, y_i\}_{i \in [n]}$ denotes the training dataset, $f(x_i) = \omega x_i$, and ω is the parameter. L denotes the loss function, typical choices are summarize in Table 10.1. R is some regularization function, such as L_p norm regularization. Now we describe a method to train the above models based on our framework.

Suppose we use the gradient-based method to optimize Equation (10.4), the update rule for ω is as follows:

$$\omega_{k+1} = \omega_k - \eta \nabla f(\omega_k) \quad (10.5)$$

Table 10.1. Loss functions for different linear models; details can be found in paper by Friedman *et al.* (2001)

Loss function	$L(y, f(x))$
Binomial deviance	$\log(1 + \exp(-yf(x)))$
SVM hinge loss	$[1 - yf(x)]_+$
Huber loss	$\begin{cases} -4yf(x), & \text{if } yf(x) < -1 \\ [1 - yf(x)]_+^2, & \text{if } yf(x) \geq -1 \end{cases}$

where η is the learning rate and $\nabla f(\omega_k)$ denotes the gradient of f with respect to ω_k . We can derive the explicit form of $\nabla f(\omega_k)$ for linear models:

$$\nabla f(\omega) = \sum_{i=1}^n y_i \nabla L \times x_i + \nabla R \quad (10.6)$$

where ∇L and ∇R , denote the gradients of L and R , respectively. Since the core step of learning a linear model is Equation (10.5), we need to linearize it for training under homomorphic encryption. By simple observation, the nonlinearity mainly comes from ∇L ; we follow the method proposed by Han *et al.* (2018a) to linearize it. We derive the polynomial approximation of the losses of Table 10.1 in Table 10.2. As an example, we also plot the hinge loss and its polynomial approximation based on regression in Figure 10.2.

With the linearization, the update step in Equation (10.5) can be reformulated as follows:

$$\omega_{k+1} = \omega_k - \eta \hat{\nabla} f(\omega_k) \quad (10.7)$$

where

$$\hat{\nabla} f(\omega) = \sum_i^n y_i \hat{\nabla} L \times x_i + \nabla R \quad (10.8)$$

and $\hat{\nabla} L$ is denoted in Table 10.2; what's more, we assume the regularization function is smooth (polynomial) for simplicity here, but we can apply the same procedure to linearize ∇R as ∇L if necessary.

Now the update step only consists of addition and multiplication after linearization. We can define a computational circuit which can

Table 10.2. Polynomial approximation for the gradient of the losses in Table 10.1. 4-order polynomial approximation is used so as to get a 3-order approximation of the gradient of the losses.

Loss function	Approximation of $\hat{\nabla} L(x)$
Binomial deviance	$0.5 - 0.0843x + 0.0002x^3$
SVM hinge loss	$0.5875 - 0.1005x + 0.0008x^2 - 0.00039x^3$
Huber loss	$2 - 0.1311x + 0.00005x^3$

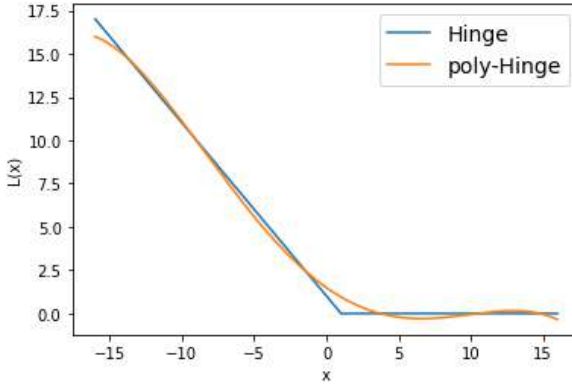


Figure 10.2. Hinge loss and its polynomial approximation based on linear regression. The blue curve is the hinge loss and the orange curve is its approximation.

be evaluated by the HEAAN framework. The base learning algorithm used by every worker is shown in Algorithm 10.1. A key innovation here is that we avoid the use of bootstrap operation, and instead we send the encrypted parameter $\pi(\omega)$ back to the parameter server for refreshing whenever the noise encoded in it goes beyond a threshold. Empirically, we simply refresh $\pi(\omega)$ every l iterations, as shown in Line 8–10 of Algorithm 10.1. Note this is critical in improving the performance of the framework and we show later in Section 10.4 that this leads to more than $10\times$ training speed boost compared to the central version in the work by Han *et al.* (2018a). What’s more, we pack multiple training samples into one ciphertext using the pack technique to make use of the parallel capability of the HEAAN framework. As for the parameter server, it performs the function of combining the results from each worker and refreshing corrupted parameters on the request of the worker, and its procedure is summarized in Algorithm 10.2.

10.4 Experimental Results

In this section, we present the experimental results. To show the effectiveness of our framework, we compare with the centralized HE learning framework and make ablation studies of the key factors that influence the performance of our framework. In the figure, we use

“Plain” to represent results without encryption and “Centralized” to show the experimental results based on the centralized learning framework.

More specifically, in Section 10.4.1, we test under the distributed learning setting, where we train different linear models on a simplified MNIST dataset and a financial dataset, and also do various ablation studies to study the influence of different factors (hyper parameters) to the performance of our framework. In Section 10.4.2, we test on a federated learning dataset called FEMNIST by Caldas *et al.* (2018b) to show that our framework is also effective under the federated learning scenario.

The experiments are conducted over an Intel Xeon E5-2683 v4@2.10GHz server with 64 cores. We use OpenMPI library by Graham *et al.* (2005) to simulate the communication between workers and the parameter server. OpenMPI is an open-source Message Passing Interface (MPI) library widely used in high-performance computing. What’s more, we use a C++ implementation of HEAAN¹ to perform HE-related computations, such as homomorphic addition and multiplication.

10.4.1 Distributed Learning

In this section, we test our framework under distributed learning settings. More specifically, we learn different linear models over a simplified MNIST dataset and a financial dataset. The dataset construction process is as follows: We extract the digits 3 and 8 from the MNIST dataset to construct a dataset with 11,982 training samples and 1,984 validation samples. Then we randomly split the training set to construct the training set and the test set (the test set is kept by the parameter server for evaluation). We also sub-sample the digit images to 14×14 . As for the financial dataset, we make use of the public subset of the original dataset mentioned in the work of Han *et al.* (2018a) which includes the credit information of 30,000 individuals and 24 features for each individual. We split the dataset to a training set of 27,000 samples and a validation set of 3,000 samples and then randomly split the training set further to construct dataset for each worker.

¹<https://github.com/snucrypto/HEAAN>.

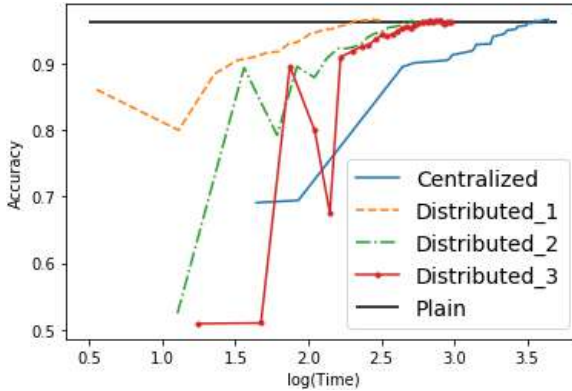


Figure 10.3. Validation accuracy with respect to the training time (log scale) of a logistic regression model trained on the simplified MNIST dataset. “Distributed- k ” means workers refresh the parameters every k iterations for our framework.

Now we present some experimental results. In Figure 10.3, we plot results of the validation accuracy with respect to the training time (log scale) when we train a logistic regression model over the MNIST dataset. As shown in the figure, both our distributed learning algorithm and the centralized learning algorithm reach the accuracy of around 96.5% which is the same as the plain accuracy, however, when comparing the training time, our framework converges much faster than the centralized version. The centralized framework consumes around 4,000–5,000s to converge, while our framework typically needs around 300–400s. We have carefully tuned the centralized framework to reach the best time performance; more specifically, we perform bootstrap every three iterations for the centralized framework to reach the best compromise between the magnitude of the ciphertext and the frequency of bootstrap. As for our framework, we vary the frequency that workers refresh their parameters. Apparently, the more frequent the worker refreshes its parameter, the higher communication overhead it leads to, however, the variant that refreshes the parameter every iteration surpasses all the other variants, although with the largest communication overhead. This actually shows that communication cost is negligible even compared to the cost of basic homomorphic operations, let alone the more expensive bootstrap operation.

We now analyze our framework in more details. The acceleration actually comes from two parts. First, our framework saves a great deal of the computational time by not performing the super expensive bootstrap operation. Furthermore, an indirect effect of eliminating the bootstrap operation is the smaller magnitude of the ciphertext modulus. This is because the magnitude of the ciphertext modulus is positively correlated with the depth of the computational circuit, while bootstrap operation takes up a significant part of the circuit depth. Naturally, we can use smaller ciphertext modulus when the computational circuit becomes shallower. Actually, the computational time for other basic homomorphic operations also decreases when we are able to use a smaller ciphertext modulus. These all benefit the training speed of our framework. To illustrate more about the effect of the ciphertext modulus, we train a variant where we only eliminate bootstrap operation but keep the magnitude of the ciphertext modulus the same. The results are shown in Figure 10.4. As we can see, simply eliminating the bootstrap operation can save around half of the computational time, which is still at the order of 10^3 s, however, if we also make use of the saved ciphertext modulus, the training time can be further decreased to the order of 10^2 s. To sum up, our framework avoids the expensive bootstrap operation and also decreases the complexity of other basic operations by reducing the

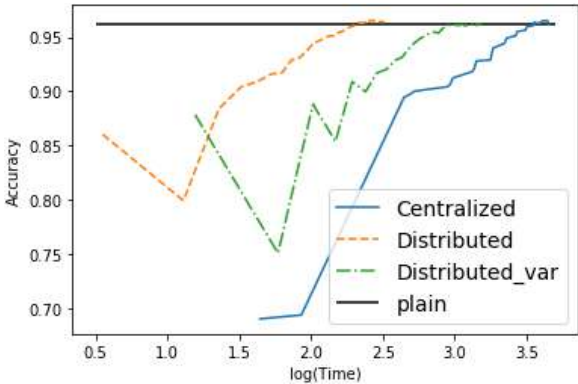


Figure 10.4. Validation accuracy with respect to training time (log scale) of a logistic regression model trained on the simplified MNIST dataset. “Distributed” is the result of our framework, while “Distributed-var” means the variant of our framework where the magnitude of ciphertext modulus is kept the same.

ciphertext modulus. Though with the cost of communication, this is negligible as shown by the experimental results.

To illustrate the robustness of our framework, we train different linear models on the MNIST dataset with the polynomial approximation computed in Table 10.2. The best validation accuracy and the corresponding training time for each model are shown in Table 10.3. The accuracy and training time are roughly the same for each loss, which shows the robustness of our framework. Next, we vary the hyper parameters to test their effect to the learning process. The results are shown in Figures 10.5 and 10.6. The hyper parameters of the experiments related to the centralized framework are kept the same as previous experiments. For the experiments in Figure 10.5,

Table 10.3. Validation accuracy and training time for different linear models trained on MNIST dataset.

Loss function	Accuracy	Time (s)
Binomial deviance	96.52	323.44
SVM hinge loss	96.42	333.74
Huber	96.57	333.65

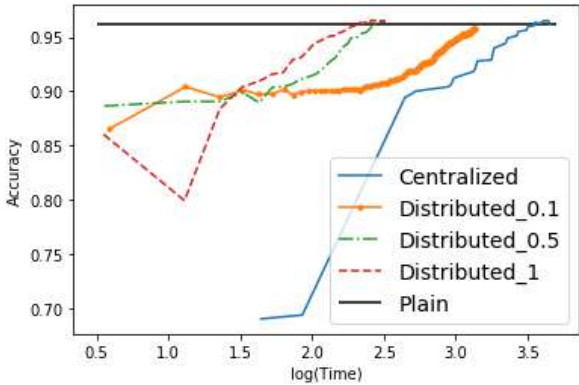


Figure 10.5. Validation accuracy with respect to the training time (log scale) of a logistic regression model trained on MNIST dataset. “Distributed- k ” means the learning rate used by our framework is k .

we fix the batch size at 64 and vary the learning rate. While for the experiments in Figure 10.6, we fix the learning rate at 1 and change the batch size. As shown by the experimental results, learning rate 1 and batch size 128 are reasonable choices with both fast and stable convergence. What’s more, we also compare the influence of the number of workers, the results are shown in Figure 10.7. We can see that

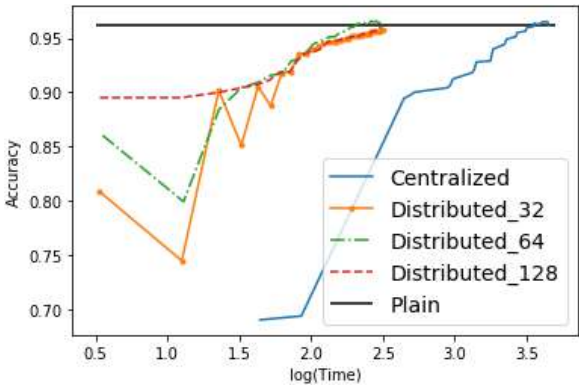


Figure 10.6. Validation accuracy with respect to the training time (log scale) of a logistic regression model trained on MNIST dataset. “Distributed- k ” means that the batch size used by our framework is k .

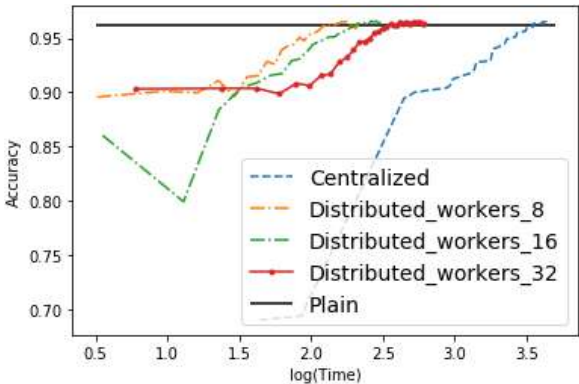


Figure 10.7. Validation accuracy with respect to training time (log scale) of a logistic regression model trained on MNIST dataset. “Distributed-worker- k ” represents the number of workers used is k .

Table 10.4. Validation accuracy and training time of logistic regression model over financial dataset.

Framework	Accuracy	Time (s)
Distributed	79.62	303.83
Centralized	79.42	4,059.68
Plain	79.42	—

more workers lead to higher communication overhead and slightly longer training time.

We also test our framework on a bigger and more realistic dataset, i.e., the financial dataset as mentioned before. We compare the validation accuracy and training time of our framework with that of the centralized framework and show the results in Table 10.4. Note that for this bigger and more complicated dataset, our model can achieve roughly the same accuracy (even slightly better) with the centralized model while still remain running $10\times$ times faster.

10.4.2 Federated Learning

In this section, we test our framework under the federated learning setting. The concept of federated learning could be referred to the work by Konečný (2016b) and Yang *et al.* (2019b). It is proposed for learning over datasets that are distributed across multiple devices and the distribution of which may be heterogeneous. Security and privacy of the user data are great challenges in federated learning, which is a great match of our fast HE-based data mining framework. We simulate the federated learning scenario with the FEMNIST dataset by Caldas *et al.* (2018). FEMNIST is generated from the MNIST dataset by partitioning the dataset according to the writer of each character. Since each writer has a special writing style, the underlying distribution of the characters written by them is different and therefore, FEMNIST is a good choice to simulate the federated learning situation. More specifically, we construct the

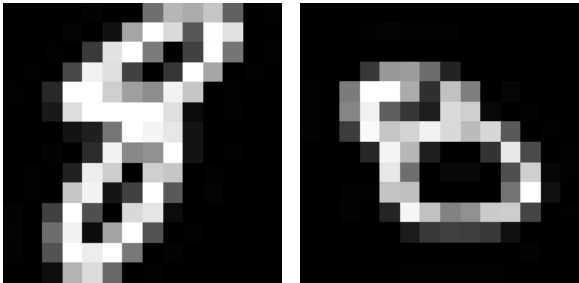


Figure 10.8. Two samples of digits from FEMNIST dataset.

Table 10.5. Validation accuracy and training time for different linear models trained on FEMNIST dataset.

Loss function	Accuracy	Time (s)
Binomial deviance	94.64	327.09
SVM hinge loss	93.66	337.63
Huber loss	94.91	327.01

dataset as follows: We randomly select 50 writers for each worker and construct a dataset with the characters they write. In Figure 10.8, we show two samples of the digits written by two different writers. The digit 8 in the left figure is slimmer and orients to the right, while the digit in the right is fatter and orients to the left. As for the dataset used by the centralized learning framework, we combine all of the users’ characters together to form a dataset.

As in the distributed learning case, we learn different models over FEMNIST dataset and the accuracy and training time are shown in Table 10.5. Their performance is roughly the same and the SVM model gets a slightly worse accuracy. What’s more, we vary the learning rate and batch size to show the robustness of our framework. The results are shown in Figures 10.9 and 10.10. The results show that our framework achieves the same training speed boost, i.e., $10\times$ times under federated learning settings.

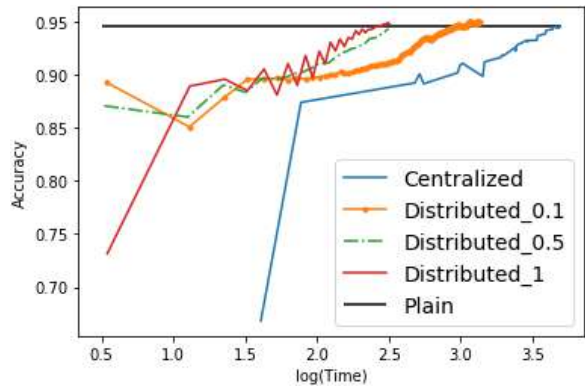


Figure 10.9. Validation accuracy with respect to training time (log scale) of a logistic regression model trained on FEMNIST dataset. “Distributed- k ” means the workers use learning rate k . We use batch size 64 in this experiment.

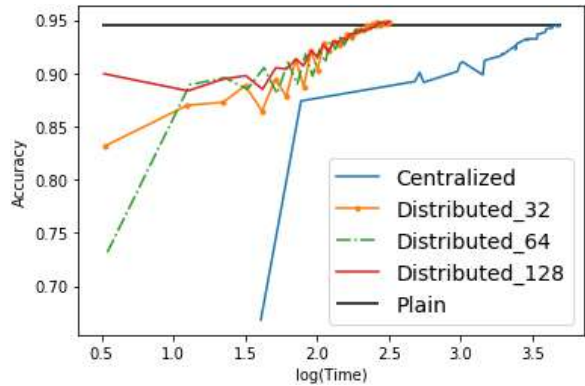


Figure 10.10. Validation accuracy with respect to training time (log scale) of a logistic regression model trained on FEMNIST dataset. “Distributed- k ” means the workers use batch size k ; we use learning rate 1 in this experiment.

10.5 Conclusion

In this chapter, we propose a distributed learning framework to drastically improve the training speed of HE-based data mining algorithms. For example, we train a logistic regression model over a

dataset with 10 thousand records within 5 minutes, while the centralized framework needs almost 2 hours. This performance boost is achieved by eliminating the bootstrap operation with the cost of communication between the workers and the parameter server. What's more, our framework is agnostic with different HE frameworks (as long as bootstrap operation is adopted), distributed learning frameworks, or specific models. We leave it as a future work to test our framework in larger scale, however, we note that the computational time of the basic arithmetic operation (especially the ciphertext multiplication) is still a bottleneck which is several orders slower than the plain operation. With the development of HE, the performance of our framework can be improved further. Finally, the model maintains privacy since it performs evaluation in the encryption field, however, the model may still contain some private information if we decrypt it; one possible remedy is to combine our framework with other privacy preserving techniques like differential privacy.

This page intentionally left blank

Chapter 11

Horizontal Federated Learning Algorithms

This chapter delves into Horizontal Federated Learning (HFL). Section 11.1 introduces the definition of HFL, its basic training processes, and the introduction of related studies. Section 11.2 discusses common distributed optimization algorithms, providing foundational insights into understanding HFL from a distributed optimization perspective. Building on this, Sections 11.3–11.5 respectively cover synchronous HFL algorithms, asynchronous HFL algorithms, and HFL algorithms with fast communication, offering readers a comprehensive understanding of HFL.

11.1 Introduction to Horizontal Federated Learning

Horizontal federated learning is also referred to as sample-based federated learning or example-partitioned federated learning. Unlike Chapters 4–9, which partition data vertically, HFL represents a “horizontal” partitioning of training data. In this scheme, each participant (e.g., a smartphone or a company in the federated learning system — often called “clients” in HFL, hence we use “participant” for consistency) possesses distinct samples (i.e., different sample IDs), but all the samples share the same feature space. For instance, two digital tech companies might have diverse customer bases, leading to unique sample IDs in their datasets. However, due to similar business models, their feature spaces might closely align. In such scenarios,

collaborative modeling between the companies can be framed under an HFL model. Formally, given datasets $\{\mathcal{D}_i\}_{i=1}^K$ distributed non-overlappingly across K participants, where the dataset $\mathcal{D}_k$ for the k th participant is expressed as a set of feature and label data pairs $\{x_i, y_i\}_{i=1}^{n_k}$ with $x_i \in \mathbb{R}^d$ representing feature data, and n_k indicating the number of samples owned by the k th participant, it follows for any two datasets D_i and D_j (where $i \neq j$ and $i, j = 1, \dots, K$):

$$\mathcal{X}_i = \mathcal{X}_j, \quad \mathcal{Y}_i = \mathcal{Y}_j, \quad \mathcal{I}_i \neq \mathcal{I}_j \quad (11.1)$$

Here, $\mathcal{X}_i$ represents the feature space of D_i , $\mathcal{Y}_i$ signifies the label space of D_i , and $\mathcal{I}_i$ denotes the sample ID space of D_i . A simple representation of an HFL system with three participants is illustrated in Figure 11.1, where the aligned training data (colored in red) has a uniform feature space across users but distinct sample spaces.

Various algorithms in HFL correspond to different frameworks. To give readers a tangible understanding of HFL, we introduce a typical HFL process. In this process, a central server orchestrates the training process by repeatedly executing the following steps until training concludes:

- **Participant selection:** The server selects participants from the pool of available ones. For example, to minimize user disturbance,

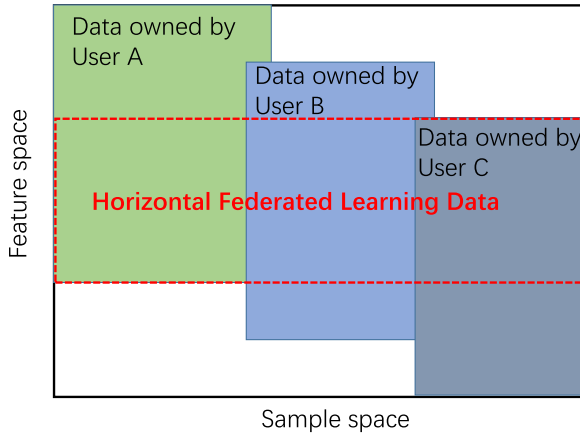


Figure 11.1. Illustration of Horizontal Federated Learning. This federated learning system consists of three users. Each user has the same feature space, but possesses different samples.

Android phones might only connect to the server (e.g., Android Cloud) when plugged in, connected to free Wi-Fi, and idle.

- **Broadcast:** The chosen participants download the current model parameters and training programs (like TensorFlow computation graphs and machine learning model weights) from the server.
- **Participant computation:** Each selected participant locally computes model updates by executing the training program, e.g., using Stochastic Gradient Descent (SGD) on their local data.
- **Aggregation:** The server collects and aggregates updates from participants. This is a hot research area, with techniques including secure aggregation with added privacy, lossy compression aggregation for improved communication efficiency, and differential privacy aggregation by adding noise.
- **Model update:** The server updates the global model locally using aggregated updates from the current round of participants.

Several studies have individually explored various parts of the aforementioned template process. Rizk *et al.* (2020) introduced a non-uniform sampling method for participant selection. Cho *et al.* (2020) investigated the influence of participant selection strategies on algorithm convergence. Wang *et al.* (2020) proposed local averaging to reduce the frequency of global averaging in their 2020 paper, thus minimizing communication costs. Ye *et al.* (2020) employed quantization methods to enhance communication efficiency. Ozfatura *et al.* (2021) introduced sparsification to boost communication efficiency in 2021, while Chai *et al.* (2020a) in their 2020 paper combined synchronous and asynchronous computations to optimize both communication and computation.

In 2016, Konečný introduced a classic model using structured updates to address issues of unreliable user network connections and slow communication speeds. McMahan in the same year presented the federated averaging algorithm, which allows each participant to train a local model and then send it to the server. The server then averages all local models to derive the global model. This method demonstrates robustness in imbalanced non-Identically and Independently Distributed (non-IID) data, producing quality models with relatively few communication rounds.

Regarding the aggregation process, Bonawitz *et al.* (2017) proposed a secure aggregation encryption method based on secret

sharing. Under this scheme, the server side (e.g., Android Cloud) can only see the gradient after the aggregation is completed and doesn't know the real gradient values privately held by each user. This method is suitable for scenarios where a large number of wireless terminals (e.g., smartphones) collaboratively model through a single server. Grama in 2020 conducted a review of the aggregation methods in existing federated learning and implemented and evaluated these aggregation methods on medical and health data.

To protect data privacy and security during training, Shokri *et al.* (2017) employed differential privacy techniques, adding noise to model parameters. Aono *et al.* (2016) utilized homomorphic encryption to secure user data. Truex *et al.* (2019) combined differential privacy and Secure Multi-Party Computation (SMC) to balance noise magnitude with privacy and security.

Furthermore, Smith *et al.* (2017) introduced a multi-task federated learning system, effectively blending federated learning with Multi-Task Learning (MTL). This system enables multiple participants to collaboratively tackle diverse learning tasks under a privacy-preserving framework, while also efficiently addressing issues like high communication overheads and network latencies.

Considering the close relationship between HFL and distributed learning, this chapter introduces HFL from a distributed optimization perspective. In the following sections, we discuss common distributed optimization algorithms as foundational knowledge.

11.2 Common Distributed Optimization Algorithms

To enhance model performance and make machine learning-based solutions suitable for more complex applications, it is often necessary to train on large volumes of data. However, processing such extensive data demands substantial computational capabilities, which may be beyond the capacity of a single computer (or work node). Thus, the field of distributed learning — which distributes the task of handling vast amounts of data among multiple machines — emerged to meet these needs. Distributed learning utilizes multiple work nodes, such as individual computers or other devices, to simultaneously process data and train models, aiming to extend machine learning to larger training datasets and models.

It is evident that horizontal federated learning and distributed learning bear significant resemblances in their form and application scenarios. A notable commonality is that both distribute data among multiple users. Several studies, like those by Kobevcny, *et al.* published in 2016, 2018, and 2019, respectively, view horizontal federated learning as a special form of distributed learning, more specifically, as privacy-preserving distributed learning that technically prevents privacy breaches. To offer readers a clearer understanding of the foundational algorithms of horizontal federated learning, this chapter introduces distributed learning. Given that distributed learning itself spans a broad spectrum of research, this section mainly focuses on some common distributed optimization algorithms rather than a comprehensive review. For those interested in a deeper dive into distributed learning, they can refer to papers by Verbraeken *et al.* published in 2020, 2016, and 2018, respectively.

Before delving into distributed optimization algorithms, it's essential to distinguish between shared-memory-based and distributed-memory-based algorithms. The former is often deployed on multicore computers, running different processes on distinct cores, achieving process-level parallel acceleration. In these algorithms, any alteration by one process to the shared memory instantly impacts other processes with access to that memory. Processes can directly access the shared memory without data transfer, thus enhancing the program's efficiency. The latter is typically deployed across multiple machines, leveraging computational resources distributed among them to accelerate data processing and model training. The primary difference is that the latter ensures atomicity in reading and writing the entire vector, while the former guarantees effective reading and writing to the bases of the vector. Unless specified otherwise in this chapter, distributed (parallel) optimization algorithms generally refer to those based on distributed memory.

Stochastic Gradient Descent (SGD) is a potent method for addressing large-scale machine learning challenges. Current distributed optimization algorithms are predominantly based on SGD and its variations. In distributed optimization, one framework is the centralized structure, like the mainstream Parameter Server (PS) model, also known as the Master-Worker model, proposed by Agarwa and others. This model comprises two main components: the parameter server (or master node) and multiple worker nodes. The master

node is responsible for storing and updating parameters, while the worker nodes process training data in parallel and communicate with the server. Each worker node operates in the following manner:

- ❑ First, the worker node fetches the latest model parameters from the parameter server required for processing the current mini-batch data.
- ❑ It then computes the stochastic gradient concerning these parameters.
- ❑ Finally, these stochastic gradients are sent back (pulled) to the parameter server.

Upon receiving the stochastic gradients from worker nodes, the server uses them to update parameters, completing one iteration. Another approach is the decentralized or peer-to-peer structure, as proposed by Bianchi *et al.* (2013). In this setup, there's no central node or parameter server. All worker nodes are interconnected, forming a connection graph. Each worker node has a copy of the model parameters. During each iteration, all worker nodes compute stochastic gradients locally and then average it with model parameters from their neighboring (directly connected) nodes.

Based on the collaboration method between nodes during training, common distributed optimization algorithms can be categorized as synchronous (parallel) algorithms (as mentioned in papers by Abadi *et al.* published in 2016, 2017, and 2012, respectively) and asynchronous (parallel) algorithms (as mentioned in papers by Zhou *et al.* published in 2018, 2015, and 2015, respectively). The following sections introduce these optimization algorithms based on this classification.

11.2.1 Synchronous Parallel Algorithms

The simplest synchronous parallel algorithm is the Bulk Synchronous Parallel (BSP) algorithm, as discussed in the classic paper by Dekel *et al.* (2012). As shown in Figure 11.2, this algorithm clearly separates the computation phase from the communication phase. As a result, running a program under this parallel algorithm is equivalent to running it serially. However, during the synchronization phase, nodes that have finished their computations must wait for other

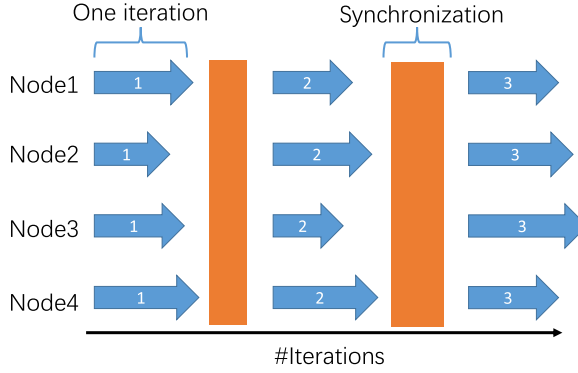


Figure 11.2. Bulk Synchronous Parallel (BSP) diagram. In this model, all worker nodes wait for synchronization at the end of each iteration and then exchange information during the synchronization phase (highlighted in orange).

nodes to complete theirs, leading to potential wastage of computational resources. Figure 11.2 provides a schematic of this algorithm.

Building on the BSP, some studies have introduced the Stale Synchronous Parallel (SSP) algorithm, as described in a 2013 paper by Ho *et al.* (2013), and the Approximate Synchronous Parallel (ASP) algorithm proposed by Hsieh *et al.* (2017). The SSP variant reduces resource wastage by allowing faster nodes to perform a few extra iterations before synchronization. If these extra iterations exceed a predefined threshold, the faster node is forced to halt its iterations. Since nodes might have undergone several updates before synchronization, other nodes might be working with outdated parameters. This leads to a drawback: The convergence rate can deteriorate rapidly when delays are excessive. A schematic of this algorithm is shown in Figure 11.3.

Unlike the SSP, which controls the degree of parameter staleness (through the delay threshold), the ASP restricts the degree of parameter inaccuracy. An advantage of this approach is that the central node can indefinitely delay synchronization when aggregated updates are not significant (e.g., only one worker node updates). However, a clear downside is defining what constitutes a “significant” update.

Focusing on synchronous parallel algorithms, we delve into the Distributed Mini-Batch (DMB) stochastic method. In the corresponding distributed system, there are k nodes, each with an independent processor (like a computer), and a network for inter-node

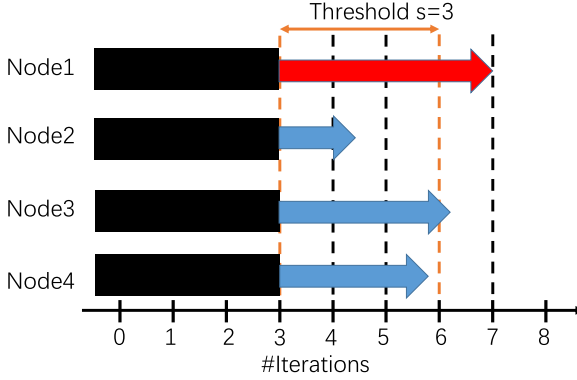


Figure 11.3. Stale Synchronous Parallel (SSP) diagram. Unlike BSP, in the SSP model, some worker nodes may advance by up to s iterations (where s is called the staleness threshold) ahead of others. Nodes that progress too quickly (as node 1 does in the figure, exceeding the threshold of 3) are forced to halt until other nodes reach the same iteration count. Like asynchronous parallel models, information about model parameters is asynchronously exchanged among nodes. The advantage of this model is that most of the time, the algorithm runs similar to the asynchronous model, but it can halt nodes based on the staleness threshold to ensure accurate outputs.

communication. Specifically, this network can be envisioned as an undirected graph $\mathcal{G}$, where edges signify bidirectional communication links between nodes. If no edge exists between two nodes, they can only communicate by relaying through other nodes. Factors influencing the nodes' inter-relationships, such as the feasibility of direct communication, can be measured by the distance between edges.

Algorithm 11.1 outlines the specific steps of this method. Here, m represents the sample size, b indicates batch size (the number of samples in a mini-batch), r signifies the number of batches, and k stands for the node count. The convergence rate of this algorithm is $\mathcal{O}(1/\sqrt{bT} + 1/T)$, where T designates iteration rounds. Given that sample computations in this algorithm equal bT (as each iteration computes b samples), the convergence rate improvement is only $\sqrt{b}$. Hence, as b increases, the convergence speed actually deteriorates. Addressing this, Li *et al.* (2014) proposed solving a regularized sub-problem in each iteration. The resultant convergence rate is $\mathcal{O}(1/\sqrt{bT})$. For larger b values, this rate surpasses the DMB algorithm. For other related methodologies, one can refer to the review article by Li *et al.* (2014).

Algorithm 11.1 Synchronous Mini-batch Stochastic Optimization on Any Worker Node

- 1: **Input:** Dataset and associated parameters
 - 2: **Output:** $\frac{1}{r} \sum_{i=1}^r w_i$
 - 3: **Parallel Training Phase:**
 - 4: **for** $i = 1, \dots, r$ **do**
 - 5: Initialize $\hat{g}_i = 0$
 - 6: **for** $s = 1, \dots, b/k$ **do**
 - 7: Sample independently and identically from the given data to obtain sample z_s
 - 8: Compute $g_s = \nabla_w f(w_i, z_s)$
 - 9: Update $\hat{g}_i \leftarrow \hat{g}_i + g_s$
 - 10: **end for**
 - 11: Compute the stochastic gradient $\hat{g}_i$ across all nodes and then calculate the average gradient $\bar{g}_i$ in a distributed manner.
 - 12: **end for**
-

11.2.2 Asynchronous Parallel Algorithms

Asynchronous parallel algorithms allow worker nodes to communicate in parallel without waiting for other nodes, as illustrated in Figure 11.4. The advantage of such algorithms is that they can typically achieve the highest training speedup compared to synchronous algorithms. However, when compared to fully synchronous parallel algorithms, the downside is that the convergence speed of the model may slow down or even fail to converge as the delay increases.

We delve into the classic algorithm proposed by Agarwal *et al.* (2012) for this category of algorithms. This algorithm is based on a cyclic delayed update framework, as depicted in Figure 11.5. The authors analyzed the convergence of the algorithm in scenarios where the stochastic gradient information is delayed. Specifically, in a structure containing n nodes, even if the computed stochastic gradient is delayed, the convergence rate of the algorithm can reach $\mathcal{O}(1/\sqrt{nT})$, where T represents the number of iterations. This convergence rate was considered optimal even when compared to algorithms with no delay. The algorithm explicitly considers two delay-robust frameworks: (1) local averaging delayed framework and (2) cyclic delayed framework.

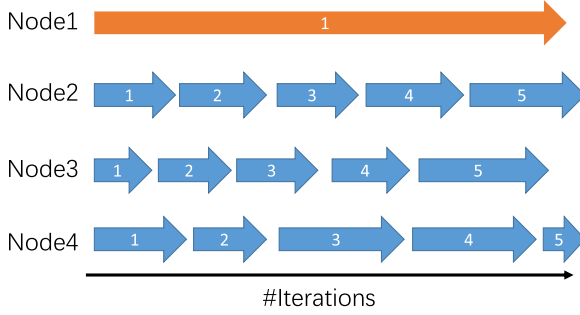


Figure 11.4. Illustration of asynchronous parallelism. Worker nodes executing the program do not need to wait for each other and exchange model parameters and other information asynchronously. Since nodes don't wait for each other, some nodes may lag significantly behind others in terms of iterations (e.g., worker node 1), leading to irrecoverable errors.

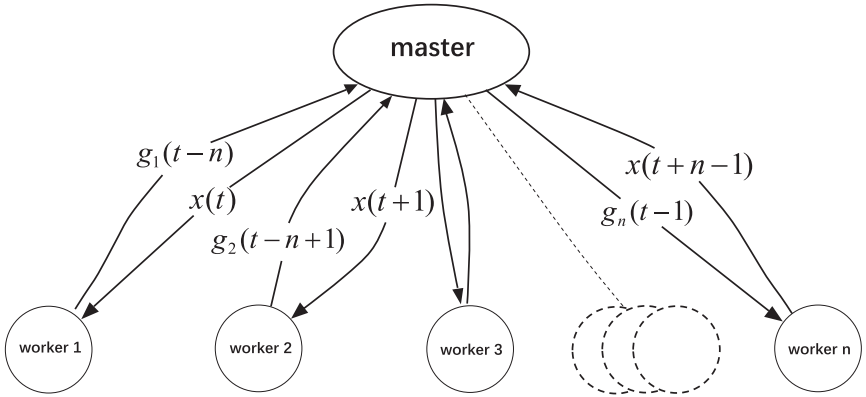


Figure 11.5. This figure shows the transfer of parameters and gradients among the nodes during the iterations from t to $t + n - 1$. Worker nodes cyclically send outdated information to the master node, which in turn sends the current parameters to the worker nodes (i.e., while the master node receives $x(t)$, the gradient information sent to it is $g_1(t - n)$).

❑ **Cyclic delayed framework:** At time t , the parameters for worker node i are $x(t)$, and it computes $g_i = F(x(t); \xi_i(t))$, where $\xi_i(t)$ is a random sample drawn by node i from the training data distribution P at time t . The master node maintains a parameter vector, and the algorithm iteratively updates in a loop. At time t , the master node receives gradient information delayed by τ rounds

from a worker node, updates the parameters, and then sends the updated parameters $x(t+1)$ back to that worker node. Meanwhile, the other nodes are unaware of this update and compute the gradient with the previous parameters.

- **Local averaging delayed framework:** This framework combines the cyclic delayed framework and the local averaging framework. Suppose there's a network $\mathcal{G} = (V, e)$, where V is a set of n nodes and e is the set of edges between the nodes. A central node is chosen to maintain the vector $x(t)$. The algorithm operates by broadcasting and aggregating across a spanning tree rooted at the central node. In the first phase, broadcasting occurs from the root node to the leaves. At time t , the central node sends its current parameters $x(t)$ to its immediate neighbors, and other nodes in the tree broadcast their current parameter vectors (for a node of depth d , the parameter vector is $x(t-d)$) to their child nodes. As shown in Figure 11.6 on the left, each node receives the latest parameters and computes local gradients based on them. In a given iteration, the second part of communication moves from the leaves back to the root. Leaf nodes convey their gradients to their parent nodes, which then average the gradients received in the previous iteration (round $t-1$) from the leaves and their gradients before passing

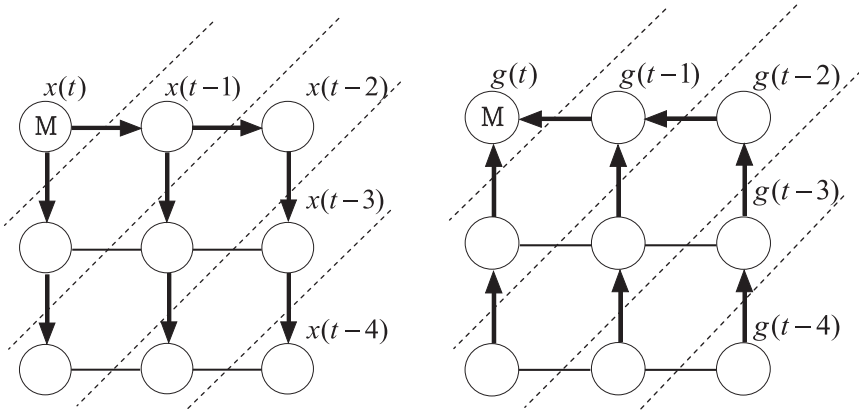


Figure 11.6. Master-Worker averaging network. (Left) Different nodes store parameters from different moments in time. If the master node stores parameters at time t , a worker node located at distance d from the master stores parameters from time $t-d$. (Right) Gradient computations for different nodes. The gradient calculated by a worker node at distance d from the master node is $g(t-d)$.

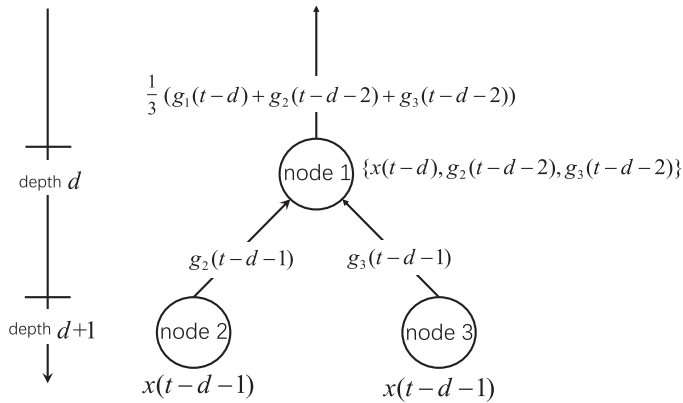


Figure 11.7. Gradient information at time t aggregates toward the master node from node 1 at distance d . The x nearby each node represents the corresponding parameter information for that node.

them back up the tree. Similarly, each node averages the gradient vectors from the previous few rounds with the current gradient vector and forwards the result up the spanning tree. The entire process is illustrated in Figure 11.7.

Subsequent work has explored and researched this algorithm for different settings in-depth, such as whether the delay is fixed or random, if the delay has an upper bound, the nature of the problem (strongly convex, convex, or non-convex), and types of stochastic gradient descent methods (e.g., variance-reducing stochastic gradient methods), etc. Due to space constraints, interested readers are referred to relevant literature.

11.3 Synchronous Horizontal Federated Learning Algorithms

In this section, we focus on introducing synchronous horizontal federated learning algorithms (henceforth simply referred to as federated learning algorithms). Before delving into synchronous federated learning algorithms, we introduce their general mathematical formulation. Suppose there are K distributed devices (e.g., computers and mobile phones). Let $\mathcal{D}_k$ denote the data on the k -th device and $n_k = |\mathcal{D}_k|$ be the number of samples on device k . The total number

of samples across all K devices is $N = \sum_{k=1}^K n_k$. We assume that for any $k \neq k'$, $\mathcal{D}_k \cap \mathcal{D}_{k'} = \emptyset$. The local empirical loss on device k is defined as follows:

$$f_k(w_k) = \frac{1}{n_k} \sum_{i \in \mathcal{D}_k} \ell_i(x_i, y_i; w_k) \quad (11.2)$$

where $\ell_i(x_i, y_i; w_k)$ is the loss function for the sample $\{x_i, y_i\}$ and w_k is the local model parameter. Based on this, we can derive the following global objective function:

$$F(w) = \sum_{k=1}^K \frac{n_k}{N} f_k(w) \quad (11.3)$$

where w is the aggregated global model. The objective of federated learning is to find the model parameter w_* such that

$$w_* = \arg \min F(w) \quad (11.4)$$

In the aforementioned generic model, we first introduce the most classical synchronous federated learning algorithm — the Federated Averaging (FedAvg) algorithm — proposed by McMahan in 2016. Before delving into the details of the algorithm, let's briefly touch upon the motivation behind its introduction by the author.

In the distributed learning scenario discussed in the previous section, computational costs generally outweigh communication costs. A common remedy for this is to utilize Graphics Processing Units (GPUs) to expedite calculations. However, in federated learning, users might be spread across different countries or cities, relying on the internet or even local area networks for communication, making each communication cycle relatively costly. Moreover, the dataset on individual users in federated learning might be smaller relative to $D_{k=1}^K$, leading to low computation costs. Consequently, communication overhead typically dominates in federated learning. Given this context, the author considered offsetting higher communication rounds (Communication Rounds) with extra computational effort. As McMahan *et al.* (2017a) mentioned in their paper, there are two primary corresponding strategies:

- **Enhance parallelism:** In every global update, include more users, allowing them to independently update their local model parameters between communications.

- **Increase the computation in each user update:** Instead of merely conducting simple single-batch stochastic gradient descent in each update, consider implementing more sophisticated update strategies. For instance, perform multiple local model updates (i.e., multiple stochastic gradient descent updates) during each iteration to amplify computational effort.

The federated averaging algorithm employs both of these strategies. The specific steps can be seen in Algorithm 11.2.

The federated averaging algorithm enhances computation based on the above strategies. Specifically, the increase in computation is

Algorithm 11.2 Federated averaging algorithm on device k

- 1: **Input:** Number of devices K , current device index k , local batch size B , local iteration count E , learning rate η .
 - 2: **Output:** Send w_k^{t+1} to the server.
 - 3: **Server-side Execution:**
 - 4: **for** global update rounds $t = 1, \dots, T$ **do**
 - 5: Server randomly selects a subset of devices S_t from K , where $|S_t| = \max(C \cdot K, 1)$.
 - 6: **for** every user $k \in S_t$ in parallel **do**
 - 7: User k executes the local update procedure ClientUpdate(k, w^t) to get w_k^{t+1} .
 - 8: User k sends the updated model parameters w_k^{t+1} to the server.
 - 9: **end for**
 - 10: Server averages all user parameters: $w^{t+1} \leftarrow \sum_{k=1}^K \frac{n_k}{N} w_k^{t+1}$.
 - 11: **end for**
 - 12: The client-side update function ClientUpdate(k, w) (executed on user k) is:
 - 13: Retrieve the latest parameters w from server
 - 14: **for** local rounds $t = 1, \dots, T$ **do**
 - 15: Randomly split dataset $\mathcal{D}_k$ into batches of size B
 - 16: **for** Iterate over E split batches, i.e., $E \cdot \frac{n_k}{B}$ rounds **do**
 - 17: Compute the stochastic gradient $\nabla \ell(w; b)$
 - 18: Update local model parameters: $w \leftarrow w - \eta \nabla \ell(w; b)$
 - 19: **end for**
 - 20: **end for**
-

governed by three key parameters: (1) Parameter C , which controls the proportion of users executing local updates in each global round, (2) Parameter E , denoting the number of rounds each user performs local updates, and (3) Parameter B , the batch size during local updates. If $B = \infty$ and $E = 1$, then the algorithm is equivalent to stochastic gradient descent with a varying batch size. Since a proportion of C users is chosen for updates in each round, when $C = 1$, it implies that all samples are involved in training, equivalent to a full-gradient update.

As shown in Algorithm 11.2, in each global update, a fraction C of users participates in the update. Each user optimizes its local objective function f_k using stochastic gradient descent. If the local objective function f_k is heterogeneous (meaning the local objective functions for users might differ), then careful tuning is required for convergence. However, excessive local rounds might lead each user to optimize toward the local objective's optimum rather than the global optimum. Note that in line 10 of Algorithm 11.2, the server averages the model parameters from all users, hence this method is termed “model averaging”. In McMahan *et al.*'s (2017a) paper, an equivalent federated averaging algorithm, called “gradient averaging”, was also introduced. As the name suggests, this algorithm doesn't average the model parameters post-update of each user but averages the gradients computed by each user. Specifically, in the t -th global update round, the local stochastic gradient computed by user k is $g_k^t = \nabla \ell(w^t; b)$, where b is a mini-batch of data randomly drawn from the local dataset. The server then aggregates the stochastic gradients as follows:

$$\bar{g}_k^t = \sum_{k=1}^K \frac{n_k}{N} g_k \quad (11.5)$$

After aggregating gradient information, the server can either directly utilize the average gradient to update the model as $w^{t+1} \leftarrow w^t - \sum_{k=1}^K \frac{n_k}{N} g_k$ and then distribute the updated model parameters to users, or the server can send the average gradient to users for them to update their local models using that gradient.

In federated learning, multiple assumptions can be made when optimizing each user's objective function. Specifically, in federated learning, the data distribution across users may vary. Thus, two

prevalent opposing assumptions are that the data are Identically and Independently Distributed (IID) and non-IID. Formally, if user data is IID, it means that the mini-batch data used for local updates statistically corresponds to uniform sampling with replacement from the entire training set (a union of K local datasets), i.e., $\mathbb{E}_{\mathcal{D}_k} [f_k(w)] = F(w)$. The federated averaging algorithm introduced above is based on this assumption. However, as each user independently collects their training data, which varies in size and distribution and is not shared with other users or servers (for privacy reasons), the IID assumption is often unrealistic. As a result, some researchers study synchronous federated learning algorithms for the non-IID case, as proposed by Khaled (2019a) and Li in (2018), respectively.

Under the non-IID setting, any user k among the K users has a local data distribution $\mathcal{P}_k$ and a local objective function:

$$f_k(w_k) = \mathbb{E}_{z \sim \mathcal{P}_i} [\ell_i(w_k; z)] \quad (11.6)$$

where $\ell_i(w_k; z)$ denotes the loss function of the model w_k on the sample z . The global optimization objective function is

$$F(w) = \sum_{k=1}^K \frac{1}{K} f_k(w_k) \quad (11.7)$$

When all $\mathcal{P}_i$ are identical, the above equation reduces to the IID setting. In this definition, F^* represents the minimum value of the function F , and the corresponding model parameters are w^* . Similarly, for the function f_k , we have f_k^* .

Similar to the federated averaging algorithm's setup, it's assumed that CK users participate in each global update, and each local update uses B samples to compute the stochastic gradient. What differs is that the batch samples $z_{k,1}, \dots, z_{k,B}$ extracted by user k follow their local distribution $\mathcal{P}_k$. To study the convergence properties of algorithms under non-IID settings, it's essential to consider an assumption about data dissimilarity. Lian *et al.* (2017) highlighted constraints between the local gradients obtained by users and the global gradient. For instance, Li *et al.* (2019) assumed there exists a positive integer κ such that

$$\frac{1}{n} \sum_{k=1}^n \|\nabla f_k(w) - \nabla F(w)\|^2 \leq \kappa^2, \quad \forall w \in \mathbb{R}^d \quad (11.8)$$

This assumption implies that when data are non-IID, locally estimated gradients may no longer be unbiased estimates of the global gradient. Other studies by Khaled (2019a) and Li (2018) assume discrepancies between local model optima and the global optimum, such as using $\Gamma = F^* - \frac{1}{K} \sum_{k=1}^K f_k^*$ to quantify the degree of dissimilarity. If the data are IID, then as the sample size increases, Γ tends toward 0.

In addition, researchers have investigated other settings and related issues of synchronous federated learning. For example, Stich *et al.* (2018) studied the convergence rate of synchronous federated learning algorithms under strong convexity conditions assuming bounded gradient norms. Yu *et al.* (2018) studied the convergence rate under non-convex conditions, while Wang and Joshi (2018) considered removing the bounded gradient assumption and achieved improved convergence rates. Due to space and expertise limitations, we do not provide specific convergence rates here. Analyzing these rates requires outlining the conditions corresponding to each rate, such as assumptions and problem formulations, and understanding these conditions requires a certain optimization knowledge. Interested readers can refer to the review paper by Kairouz *et al.* (2019).

11.4 Asynchronous Horizontal Federated Learning Algorithms

Most synchronous federated learning algorithms share a similar update framework as the federated averaging algorithm. However, this kind of update framework has an evident disadvantage — when one or more users experience significant network delays or have a large volume of data to train that takes much longer than other users, all other users must wait for these lagging participants. Chai *et al.* (2020a) pointed out that because the server only aggregates after all users have completed computation and communication, synchronous updates or optimization will lead to idle times and waste computational resources. As a result, several studies have proposed asynchronous federated learning algorithms to address this issue, such as those by Sprague *et al.* (2018) in their 2018, 2019, and 2019 papers, respectively. In asynchronous federated learning algorithms, as soon as the server receives an update from a user, it immediately updates the global model without waiting for all users to complete their

computation and communication. The server maintains the current global model, and all users maintain their local models. Note that, due to asynchrony, different users' models might differ at the same global timestamp.

Chen (2019), proposed the Asynchronous Online FEDerated learning (ASO-Fed) algorithm. The update process is illustrated in Figure 11.8. Once the server receives an update from a user, it immediately begins the aggregation process and uses the aggregated model parameters to learn feature representations, extracting cross-user feature representations. Subsequently, the server initiates the next

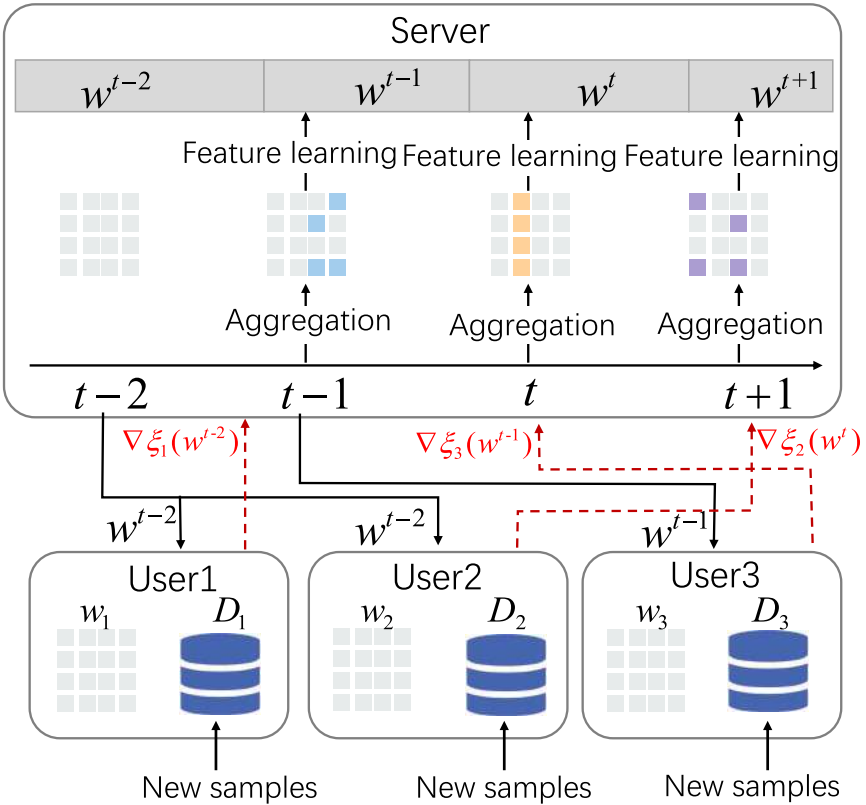


Figure 11.8. Schematic diagram of the update process of the asynchronous online federated learning algorithm. The server performs the aggregation operation immediately after receiving the update from the user. New data samples represent new samples obtained from local data during training.

global round and distributes the latest global model to ready users. Since this algorithm is an online algorithm (for more about online algorithms, readers can refer to the paper by Bent *et al.* (2005)), new data samples might join during the training process. As illustrated in Figure 11.8, when the server receives a gradient from a delayed user (e.g., User 2), the server might have already updated the global model twice. Notably, in this asynchronous update framework, there's an inconsistency when users obtain the global model from the server. This inconsistency arises because, due to communication delays, the server might have already updated the global model once more.

The pseudocode for the asynchronous online federated learning algorithm is presented in Algorithm 11.3. For the server, every global iteration aggregates model parameters to get w , with the equation $w^{t+1} = w^t - \frac{n_k}{N}(w_k^t - w_k^{t+1})$, where the term in parentheses represents the gradient. Given the algorithm's online nature, n_k and N might change over time. Due to the asynchrony, the local model w_k of a user and the global model w from the server are likely different. To alleviate the disparity between the two models, the author introduces

Algorithm 11.3 Asynchronous online federated learning algorithm

- 1: **Input:** Regularization parameter α , multiplier r_k , Learning rate η , Decay β .
 - 2: **Output:** $h_k^{pre} = h_k = 0$, $v_k = 0$
 - 3: **Server-side Execution:**
 - 4: **for** Globally update iteration rounds $t = 1, \dots, T$ **do**
 - 5: Compute $w^t \leftarrow w^t - \eta_k^t \frac{n_k}{N} \nabla f_k(w^t)$
 - 6: Update w^t based on feature learning formula (refer to Equations 5 and 6 in Chen *et al.*, 2019a)
 - 7: **end for**
 - 8: **Execution for User k during global iteration round t :**
 - 9: Obtain the latest parameters w^t from the server
 - 10: Compute ∇s_k , where $s_k(w_k) = f_k(w_k) + \frac{\lambda}{2} \|w_k - w\|^2$
 - 11: Set $h_k^{pre} = h_k$
 - 12: Set $\nabla \xi_k \leftarrow \nabla s_k - \nabla s_k^{pre} + h_k^{pre}$
 - 13: Update parameters, $w_k^{t+1} \leftarrow w_k^t - r_k^t \eta \nabla \xi_k$
 - 14: Compute and update $h_k = \beta h_k + (1 - \beta) v_k$
 - 15: Update $v_k = \nabla s_k(w^t; w_k^t)$
 - 16: Upload the local model parameters w_k^{t+1} to the server
-

the objective function s_k as a replacement for f_k :

$$s_k(w_k) = f_k(w_k) + \frac{\lambda}{2} \|w_k - w\|^2 \quad (11.9)$$

Furthermore, users don't directly employ stochastic gradient descent for updates. Instead, they utilize a kind of momentum-accelerated update method. For details on momentum-accelerated stochastic gradient descent, refer to paper by Hu *et al.* (2019). Specifically, for user k , its parameter is updated as follows:

$$w_k^{t+1} = w_k^t - \eta_k^t \left(\nabla f_k(w_k^t) - \nabla s_k^{(\text{pre})} + h_k^{(\text{pre})} + \lambda (w_k^t - w^t) \right) \quad (11.10)$$

Experiments prove that the algorithm is very robust to delays and dropped users.

Xie *et al.* (2019) also researched asynchronous federated learning algorithms. In the ASO-Fed algorithm, β is set as a constant. However, their paper suggests that varying levels of delays lead to differing levels of inconsistencies. Thus, the authors introduced the function $s(t - \tau)$, where τ indicates the delay, which plays a role similar to β in the asynchronous federated learning, to mitigate the adverse effects of delays. Notably, s should meet specific criteria: When $t = \tau$, $s(t - \tau)$ should be 1, and the value of s should decrease as $t - \tau$ increases. Chen *et al.* (2019b) proposed a two-stage analysis method in their 2019 paper, breaking down the errors during the training process. Specifically, if $w_k^* = \arg \min f_k(w)$, the training error can be decomposed as follows:

$$\underbrace{\|w^t - w^*\|}_{\text{Global error}} \leq \underbrace{\sum_{k=1}^K \frac{n_k}{N} \|w_i^t - w_k^*\|}_{\text{Initialization and local error}} + \underbrace{\sum_{i=1}^M \frac{n_k}{N} \|w_i^* - w^*\|}_{\text{local-global error}} \quad (11.11)$$

where $\|\cdot\|$ denotes the L2 norm. For a user k , the two terms on the right represent the following:

- **Initialization and local errors:** The disparity between the local model and the optimal global model at the global round t .
- **Local-global error:** The difference between the optimal local model and the global model. For a given federated learning task, this error is constant.

Therefore, to minimize global error, the first term is a potential point of focus. Based on this analysis, researchers introduced the stale-robust asynchronous federated learning algorithm. The literature theoretically proves good convergence properties for non-convex problems. Experimental results indicate that the algorithm's convergence rate is twice as fast as the baseline algorithm, resisting the negative impacts of delayed users without compromising accuracy.

In asynchronous federated algorithms, the server does not wait for lagging users before aggregating. This contrasts with synchronous federated learning algorithms, like the federated averaging algorithm introduced in the previous section, where only a subset of users are selected to communicate with the server during each global iteration. In the asynchronous federated learning setup, the server communicates asynchronously with all users. Hence, when thousands or even tens of thousands of users update their models simultaneously, the server can easily become a communication bottleneck.

To address this, Chai *et al.* (2020a) proposed a multi-tier asynchronous federated learning algorithm called “Federated Learning Method with Asynchronous Tiers” (FedAT). This algorithm combines the advantages of both synchronous and asynchronous approaches by implementing a tiered mechanism. In the multi-tier asynchronous federated learning algorithm, all users are divided into different logical tiers based on their response latency. All these tiers participate in global training simultaneously, but each tier follows its own training rhythm. Users within a single tier update synchronously with respect to the model associated with that specific tier, while each tier updates the global model asynchronously. The fastest tier, with the shortest response latency, ensures quicker convergence of the global model training. Meanwhile, slower tiers participate in global training by sending model updates asynchronously to the server, further enhancing the global model's performance.

Directly and uniformly aggregating the asynchronously updated models from each tier can result in biased training. This is because models with lower latency get updated more frequently, causing the global model to gravitate more toward these low-latency tiers. To address this issue, a weighted aggregation strategy was introduced, assigning higher weights to slower tiers.

As shown in Figure 11.9, the tiering module divides all users into M tiers based on response latency: $tier_1, \dots, tier_M$, where $tier_1$ is

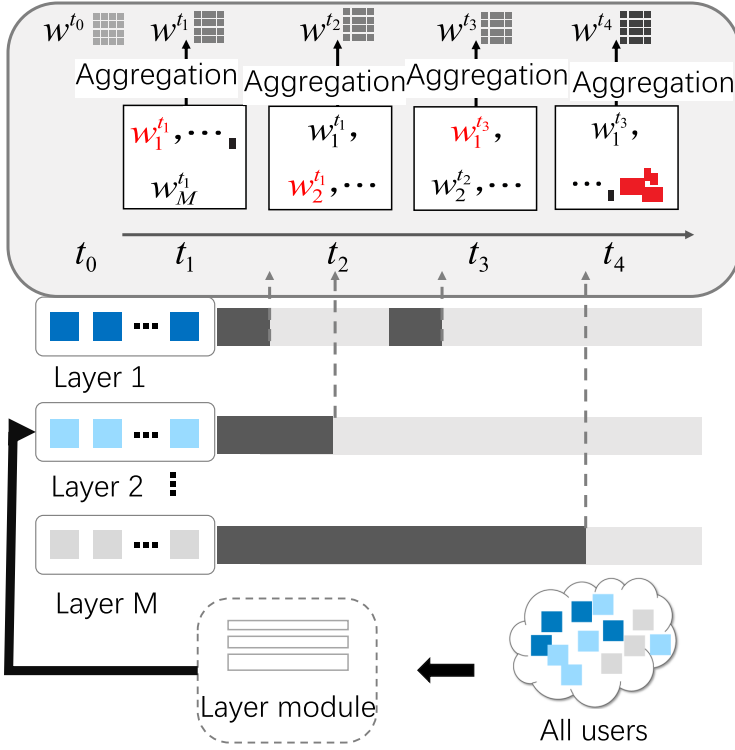


Figure 11.9. Schematic of the asynchronous tiered federated learning algorithm. Here, $tier_1, \dots, tier_M$ represent the M tiers, and $w_1^t, \dots, w_M^t$ are the corresponding model parameters.

the fastest and $tier_M$ is the slowest. The server maintains a list of models $\{w_1^t, \dots, w_M^t\}$ to track the latest model for each tier during a specific global round t . The server also holds a global model w updated asynchronously from all M tiers.

For users in a specific tier, they execute a synchronous update process together. A subset of users is randomly chosen (like in the federated averaging algorithm, selecting a subset containing $|S|$ users), and loss is calculated based on their local data. Then, weight updates (in a compressed format in this algorithm) are synchronously sent to the server, updating the model corresponding to that tier on the server. Figure 11.9 includes a simple example wherein users within the same tier train synchronously, while there's asynchronous training between different tiers. At time t_1 , users from the first tier quickly

complete their local training and send their models to the server. The server then performs the following steps: (1) synchronously updates the received model from the first tier to get $w_1^{t_1}$ (highlighted in red in the figure) and (2) aggregates the latest updates from all tiers using weighted aggregation to generate a new global model w^{t_1} .

Besides the aforementioned work, Sprague *et al.* (2018) have dedicated their efforts to applying asynchronous federated learning to geolocation.

While horizontal federated learning has seen rapid development recently, existing research predominantly focuses on designing synchronous horizontal federated learning algorithms. Therefore, asynchronous horizontal federated learning is a direction that deserves attention from professionals in the field.

11.5 Rapid Communication in Horizontal Federated Learning Algorithms

In the training process of federated learning, each round of global update requires sending and receiving the complete model from users for updating. In large-scale federated learning scenarios, we have the following: (1) A single user's machine learning model parameters can be very large (for instance, large deep neural networks might have millions of parameters); (2) during each round of global update, a large number of users need to update with the server. Consequently, communication can be a major bottleneck in federated learning. This is because wireless connections and other end-user internet communications are typically slower than intra or inter-data center communications, and the communication costs can be substantial and unreliable. Addressing this issue, many scholars have explored and proposed effective methods for rapid communication. Two common methods are: (1) gradient compression, reducing the size of gradients (or models) transferred from users to the server for updating the global model; (2) broadcast model compression, decreasing the size of models broadcasted from the server to users. Method (1) was explored by Alistarh *et al.* in papers from 2016, 2017, 2018, and 2019, respectively; while Method (2) was studied by Caldas and Khaled in papers from 2018 and 2019. A few methods are briefly introduced in the following.

Konečný *et al.* (2016b) combined federated averaging algorithms with sparsification and/or quantization (converting the model into smaller bit data) and demonstrated that communication time per round can be significantly reduced with only a slight compromise on training accuracy. Specifically, their paper introduced two effective update strategies:

- **Structured updates:** Limit the model during the training process to yield a model form with fewer parameters. For example, using low-rank or sparsity constraints (a common choice in machine learning is using the trace norm as a regularization term) results in low-rank or sparse models. The model parameters are updated in a reduced parameter space, and after updates, corresponding information is sent to the server. With fewer model parameters, communication is faster under given conditions.
- **Sketched updates:** After a regular model update, each user locally compresses the model parameters and sends the compressed parameters to the server, which then updates the global model. For instance, down-sampling and quantization coding can be used.

In these two strategies, structured updates adopt a model with fewer parameters for the training process, whereas sketched updates compress the model after the original model is updated.

In a typical horizontal federated model (excluding personalized federated learning), parameters are shared. Therefore, the aforementioned papers utilize methods to reduce parameter size to accelerate communication. Given that some methods do not communicate model parameters but rather the latest gradients (federated averaging includes both model averaging and gradient averaging), gradients can be compressed to achieve similar effects. Lin *et al.* (2017) specifically proposed the Deep Gradient Compression (DGC) method. As shown in Figure 11.10, DGC effectively reduces (single-instance) communication time to achieve rapid communication. This method can reduce the parameter size of ResNET-50 from 97 MB to 0.35 MB and DeepSpeech from 488 MB to 0.74 MB. Since it only sends important gradients (sparsified updates), it significantly reduces communication bandwidth. In determining importance, the method uses a straightforward strategy — judging by the magnitude of the gradient, and only gradients with magnitudes exceeding a preset threshold are

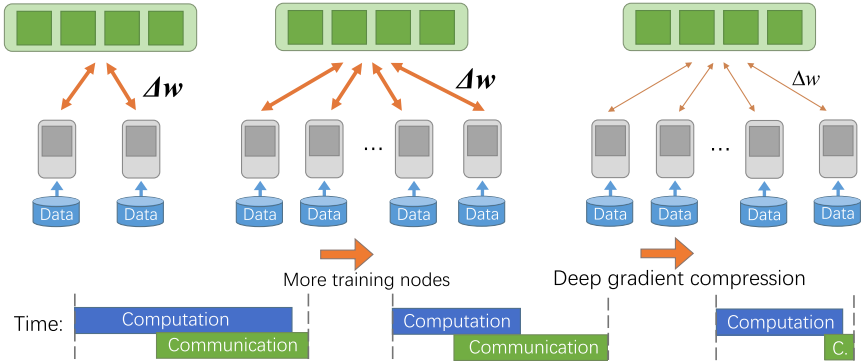


Figure 11.10. Illustration of deep compression effects. The blue in the figure represents computation time, and green represents communication time. As shown, when the number of computation nodes (users) is small, computation time is longer than communication time; when the number of nodes increases, communication time dominates. After using the deep gradient compression method, communication time is significantly reduced, realizing rapid communication, making communication time no longer the training time bottleneck.

transmitted and used to update the corresponding parameters. To avoid losing information from small magnitude gradients, the method stores gradients with magnitudes below the threshold locally and accumulates them until they exceed the threshold and are eventually transmitted. To maintain model performance, techniques like momentum correction, local gradient clipping, momentum factor masking, and warm-up training are introduced to mitigate the decline in model performance.

In addition to the methods mentioned above, Alistarh *et al.* (2016) specifically studied gradient quantization and encoding techniques to reduce communication times. Additionally, there are some methods tailored for distributed learning scenarios, such as the Sign Stochastic Gradient Descent (Sign SGD) proposed in Bernstein’s 2018 paper. Some other strategies aim to reduce the number of users selected for updates each time, as in federated averaging algorithms where a fixed number of users are chosen for communication with the server in each global update. Reducing the number of selected users can indeed accelerate communication.

However, it remains unclear if further speed improvements in communication can be achieved in federated learning and whether these methods (or combinations thereof) can strike a better balance

between fast communication and model accuracy. Some distributed learning-related studies provide theoretical insights on how to characterize the trade-off between communication (communication constraints, such as bandwidth) and model performance, such as the papers by Braverman *et al.* (2016). However, it's challenging to derive tangible insights from these theoretical works on reducing communication bandwidth, mainly because they often overlook the influence of optimization algorithms. Nevertheless, utilizing these theoretical methods to guide practical federated learning remains an area worth exploring.

11.6 Conclusion

This chapter began by defining horizontal federated learning and then introducing the templated process of its training. This led to the introduction of several relevant research directions. Given the close relationship between horizontal federated learning and distributed learning, subsequent sections provided a brief overview of related research and methods in distributed learning. Then, by drawing parallels with distributed learning classifications, synchronous and asynchronous methods for horizontal federated learning were categorized and the current state of related research was discussed. Lastly, the real-world applications of horizontal federated learning were introduced and explored, drawing from the fifth section.

In recent years, federated learning has seen rapid development and has been implemented in numerous practical applications. However, compared to distributed learning, it's still relatively nascent, leaving many challenges and issues to be addressed. Specifically, every process in the template of horizontal federated learning (introduced in the first section) encapsulates valuable and challenging research problems. Given the similarities in form and scenario setups between horizontal federated learning and distributed learning, integrating mature distributed learning techniques into federated learning presents a promising research direction. As the third and fourth sections reveal, compared to synchronous horizontal federated learning, asynchronous methods seem to be less prevalent, indicating a wealth of unresolved challenges in the asynchronous domain. As mentioned in the fifth section, communication costs in federated learning

are often high, making the pursuit of fast communication in horizontal federated learning a research direction with significant practical implications.

Through this chapter, it is hoped that readers gain a deeper understanding of horizontal federated learning from the perspective of distributed optimization. It is also the hope that readers are be inspired to contribute theoretically or practically to the advancement and application of horizontal federated learning.

This page intentionally left blank

Chapter 12

Mixed Federated Learning Algorithms

The mixed algorithm integrates the characteristics of both horizontal and vertical federated learning, enabling model training and prediction in a unified and concise manner. This chapter introduces two novel mixed federated learning algorithms: the mixed federated boosting trees and the mixed linear regression model.

Algorithm highlights: This algorithm pertains to mixed federated learning, distinguishing it from the conventional horizontal or vertical federated learning methods. It boasts advantages, such as broad data applicability, straightforward implementation steps, and reliable training outcomes.

12.1 Scenario Requirements for Mixed Federated Algorithms

Existing federated learning algorithms require one to decide between horizontal and vertical learning methods based on the nature of user data. For horizontal learning, the data must be homogenous (as shown in Figure 12.1 left), while for vertical learning, the data should be heterogeneous (as shown in Figure 12.1 right). In collaborative federated learning involving multiple parties, it's challenging to ensure that data are purely heterogeneous or homogenous (as depicted in Figure 12.2). Thus, during training, some datasets that are either homogenous or heterogeneous get discarded. When the overlap in data from the training parties is minimal or the number

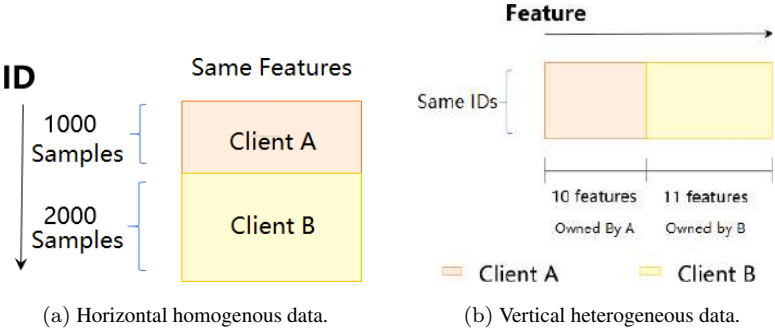


Figure 12.1. Data requirements for horizontal and vertical scenarios.

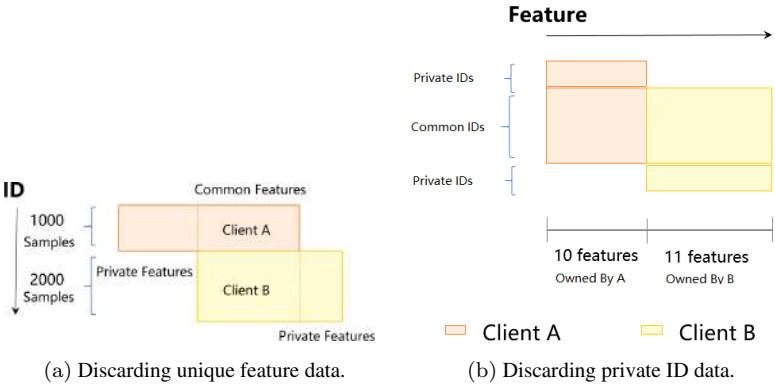


Figure 12.2. Data discarded during preprocessing.

of training parties increases, much data are discarded. This leads to subpar federated learning results on one hand and, on the other, a reduced economic benefit due to low data utilization rates.

For instance, in Figure 12.2(a), during horizontal federated learning, only data with matching features from both parties are retained, while unique features are discarded. In Figure 12.2(b), for vertical federated learning, data with private IDs are discarded.

This distinction between horizontal and vertical approaches not only reduces data utilization rates but also raises a significant practical issue: a higher threshold for model application. Before training a model, users need to address two main challenges: first, selecting between horizontal or vertical learning methods based on data characteristics and second, data preprocessing. In reality, these two steps often overlap and are more complex than they appear. Users first

need to align samples and features to determine which method better fits the data distribution. Many times, when data overlap doesn't favor a particular method, users must rely on experience or randomly choose a training approach. At times, when there's only a need for a basic usable model trained on a limited amount of data, which is refined later, it's tough to select an appropriate algorithm. Often, to optimize results, both horizontal and vertical methods are tested, necessitating two rounds of data preprocessing and training. Furthermore, as data in practical scenarios evolve and the data overlap ratio among participants changes, the initially chosen algorithm might no longer be suitable.

Due to the need to discard data, based on the varying data storage methods, security controls, and other factors of the participants, data or interfaces previously processed by database administrators might not be directly usable. This requires users to be familiar with each individual processing system, undergoing a cumbersome data filtering process to prepare the data suitable for the desired algorithm. Clearly, this process requires users to possess advanced algorithmic knowledge and technical skills. It also demands high-level collaboration among all participants, consuming their time and energy.

The mixed federated learning algorithm addresses these issues. Compared to the purely horizontal or vertical federated learning, the mixed algorithm has the following advantages:

1. **Broad data applicability:** Mixed federated learning is suitable for data that exhibit various degrees of overlap between horizontal and vertical orientations. As shown in Figure 12.3, the three participants share 5,500 data entries with 40 features, which can be directly trained and inferred using the mixed boosting tree algorithm. In horizontal federated learning, only 8 of these features can be used; in vertical federated learning, only 2,000 of the samples are usable. Even in extreme cases where overlap is almost zero, the mixed algorithm ensures the joint modeling process continues, producing a complete, usable model. Similarly, when the data distribution fully meets horizontal or vertical conditions, mixed federated learning is equivalent to the corresponding horizontal or vertical federated learning.
2. **Simplified steps:** Users don't need to decide between horizontal and vertical learning based on experience, nor is there a need for secondary data filtering. This means users can focus less on

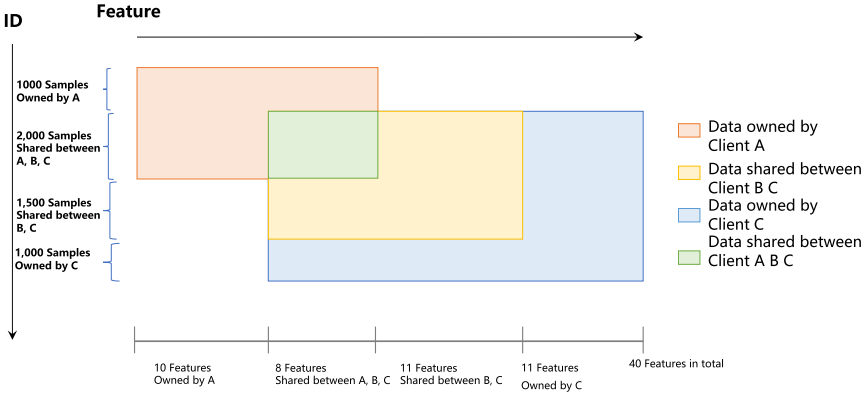


Figure 12.3. Data distribution example in mixed federated learning.

detailed data preprocessing, reducing the learning curve and making the model more accessible. In scenarios like distributed storage, this unified and simplified approach also reduces the time cost of data processing. In practical applications, the data overlap ratio between participants might change continuously, but with mixed federated learning, there's no frequent need to switch algorithms.

3. **Guaranteed training results:** Mixed federated learning fully utilizes previously discarded data to achieve a higher data utilization rate, leading to improved model training outcomes. When the sample overlap is 100%, indicating a purely vertical data distribution, the accuracy of the mixed boosting tree reaches the level of the pure vertical model. When the sample overlap is lower, the mixed boosting tree achieves higher inference accuracy, offering better generalization performance than the pure vertical model.

12.2 **Algorithm Details**

This section introduces a federated algorithm based on XGBoost¹ — the mixed federated boosting tree.

¹XGBoost is an open-source machine learning project developed by Tianqi Chen and others. It efficiently implements the GBDT algorithm and has made many algorithmic and engineering improvements, achieving commendable success in practical applications.

12.2.1 Algorithm Design

Section 4.3 already discussed the principles of the Gradient Boosted Decision Tree (GBDT) algorithm. When calculating the optimal feature for node splitting, we integrate both horizontal and vertical learning methods, evaluating splitting candidates in both orientations before making a final decision on the split.

During the training of the mixed gradient boosting tree, each tree's training process includes the following steps:

Phase 1: Update the gradient based on the initial prediction values or the previous round's prediction values.

Phase 2: Grow the mixed tree. Each node's splitting undergoes the following:

1. horizontal split candidate computation,
2. vertical split candidate computation,
3. split decision.

Phase 3: After a tree has fully grown, update the sample prediction values, compute the current training metric results, and continue with the training of the next tree.

12.2.1.1 Gradient update

During the sample alignment phase, each client registers its sample ID collection with the server. The server matches these sample IDs and simultaneously generates a new pseudo ID for each sample, which will be used in subsequent training.

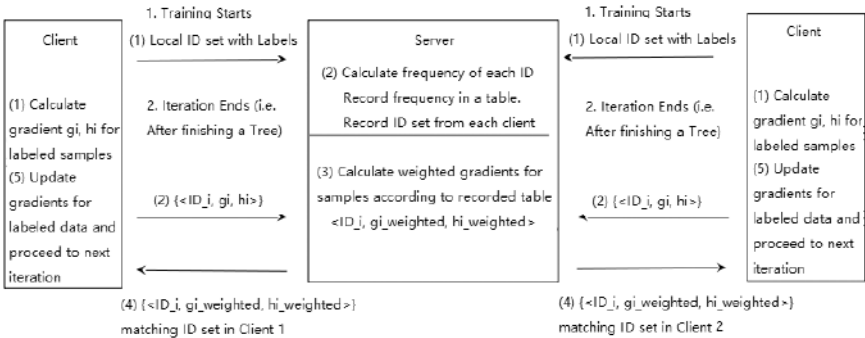


Figure 12.4. Gradient computation under mixed label distribution.

Compared to the vertical decision tree, the MixGBoost algorithm accommodates label data distributed among multiple parties. However, it's crucial to stress that this distribution only applies to non-overlapping samples between clients, i.e., on horizontal data. To safeguard label data, we require the label data of the shared samples to be provided by only one client, identical to FederatedGB, which we refer to as the “active client”.

For data security, the mixed boosting tree algorithm employs homomorphic encryption to protect gradient update values, preventing label data leakage. Each client uses the distributed key generation method described in Section 12.6.1 to agree on a homomorphic encryption key $(p_k, \{s_k\})$ in advance, publishing the public key p_k to the server while retaining their respective public-private key pairs. Before the training of each tree begins, every client recalculates the gradient based on current prediction values, with gradient values of horizontally aligned data stored locally. As described in 1, clients possessing shared sample label data must share encrypted gradient values of these samples for subsequent node split gain value computation. Based on the threshold decryption properties of the Paillier algorithm from 12.6.1, other clients cannot decrypt ciphertext using only their private key nor can they infer the original label data through gradient values.

Detailed steps of gradient update process:

- Step 1:** Each client calculates the gradient for its locally labeled sample i , including the first-order gradient $g_{i,j}$ and the second-order gradient $h_{i,j}$. The active client uses a public key to encrypt the gradient values of the shared samples, resulting in $Enc[g_{i,j}]$ and $Enc[h_{i,j}]$, where $i \in CommonSet$.
- Step 2:** The active client j sends the encrypted gradient datasets of the shared samples, $G_j = \{Enc[g_{i,j}]\}$ and $H_j = \{Enc[h_{i,j}]\}$, where $i \in CommonSet$, to the server.
- Step 3:** Upon receipt, the server distributes these encrypted gradient values to other clients.
- Step 4:** After receiving the encrypted gradient values of shared samples, clients store them locally and proceed with the tree growth process.

12.2.1.2 Mixed node splitting

We first reduce the complex scenarios of mixed data distribution to the two overlapping platform data scenarios shown in Figure 12.5. From the data perspective, the design of the mixed model needs to consider the following two issues:

1. how to utilize the private data of each participating party, i.e., the sum of data blocks outside the cross-shaped area in Figure 12.5, (1.1) + (5.2), to enhance the model's performance based on its characteristics,
2. how to mitigate the negative impact on the model due to the missing corners after data overlaps, i.e., the missing data blocks in the bottom-left and top-right corners of Figure 12.5.

Two splitting candidates: In tree models, specific feature values don't directly contribute to the model's predictions. Instead, they determine whether samples go left or right in tree nodes. If we view data from each client as a whole, then a private feature $f_{i,1}$ on client i can be considered a globally shared feature with many missing values. The mixed boosting tree model adopts the XGBoost method

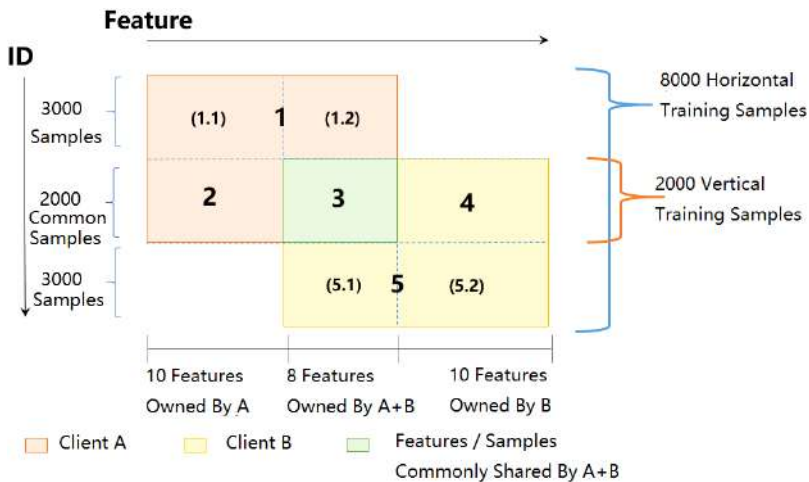


Figure 12.5. Overall data selection scheme of mixed federated learning.

for sparse matrix handling. During node splitting, it doesn't consider filling in the missing feature values. Instead, it computes losses for data with missing values being sent to the left and right sub trees, choosing the more optimal one. Abstractly, the data become a complete block, solving the problem of missing corners. Training on mixed distribution data thus transforms into a horizontal data training problem. With the participation of all samples, an optimal splitting feature is determined as the first splitting candidate.

However, when the private features of each client constitute the majority, i.e., when data blocks 2 and 4 in Figure 12.5 are relatively narrow, the aforementioned method not only compensates for the drawbacks of purely vertical federated learning by adding private ID data from each client, thereby supplementing the training data volume of shared features, but also offsets the drawbacks of purely horizontal federated learning by capturing potentially important features within these private features.

Yet, when most features are private features, the introduction of missing values affects almost all features, exacerbating the error in split computations and pushing the model away from the true distribution. In such cases, it's essential to exclude missing values and recalculate candidates for combined node splits of private features. Since contributions to split gains from private ID data of each client can't be measured on the same scale, only shared IDs, i.e., vertical data distribution, are used for calculations at this step. This approach transforms the aforementioned second issue into a vertical training problem for private features, identifying the second split candidate.

Splitting candidate computation: When the mixed federated boosting tree grows to node N , and the sample data at this node manifest as depicted in Figure 12.5 (i.e., all parties have common features as well as private features, and both common and private IDs), the sample data of node N are divided into two sets:

- The first training set, used for calculating the horizontal split candidates considering all features. This represents data from all clients, as shown in regions (1)+(2)+(3)+(4)+(5) of Figure 12.5.
- The second training set, used for calculating vertical split candidates using private features. This pertains to the shared sample data's private feature information from multiple clients, corresponding to regions (2)+(4) in Figure 12.5.

Unlike distributed machine learning algorithms, in a federated learning setup, when the data values of a feature are distributed across multiple clients, ordering these values based on the feature is not feasible. Drawing inspiration from the extremely randomized trees method proposed by Geurts *et al.* (2006), the server initially selects a subset of features, determining a random split threshold for each feature. It then calculates the gain from this random split, considering it as the split candidate for that feature. As discussed in the paper, while this approach may sacrifice the accuracy of computing the best split gain for each feature, it still yields satisfactory training results.

Specifically, the calculation for the **first feature candidate** is as follows:

Step 1: First, the server randomly selects a subset of features $F_N = \{f_1, f_2, \dots\}$ from the entire feature set and sends this subset to all clients.

Step 2: Client j cycles through the received feature subset. For each feature f_k that exists locally, it randomly selects a value from the data range of f_k in the samples of node N to serve as the classification threshold candidate for f_k . The client then sends the threshold candidate set $V_j = \{v_{1,j}, v_{2,j}, \dots\}$ to the server.

Step 3: After collecting threshold candidate information from all clients, the server constructs a list and then randomly picks thresholds from this collection, ensuring they fall within a reasonable range. For feature f_k , a value v_k is randomly chosen from the list $\{v_{k,1}, v_{k,2}, \dots\}$ to serve as the global optimal split threshold for f_k . The pair (f_k, v_k) is then broadcasted to all clients.

Step 4: Based on the threshold v_k , client j splits the node using feature f_k :

If the client possesses data for feature f_k and f_k is a common feature, the initiating client calculates the sum of gradients for the left and right subtree samples, denoted as GL_j , GR_j , HL_j , and HR_j . Non-initiating clients compute the gradient sums for all samples except the shared ones.

If f_k is a private feature in the local data, irrespective of whether client j is the initiator, it computes the gradient sum for all left and right subtree samples. During this process, non-initiating clients use homomorphic addition to compute the portions involving shared sample gradients.

If the client lacks data for feature f_k , it calculates the gradient sum for all samples of the left and right subtrees except the shared ones, denoted as $G_{j,\text{missing}}$ and $H_{j,\text{missing}}$.

Client j then notifies the server with the encrypted ciphertext of GL_j , GR_j , HL_j , HR_j , $G_{j,\text{missing}}$, and $H_{j,\text{missing}}$, using a public key.

Step 5: Upon receiving from various clients the sum of gradients needed for the feature f_k split, the server calculates the overall first-order gradient for missing values, $Enc[G_{\text{missing}}]$, and the second-order gradient, $Enc[H_{\text{missing}}]$, as well as the total first-order gradient for all samples at node N , $Enc[G_{\text{all}}]$, and the corresponding second-order gradient, $Enc[H_{\text{all}}]$:

$$Enc[G_{\text{missing}}] = \sum_j Enc[G_{j,\text{missing}}] \quad (12.1)$$

$$Enc[H_{\text{missing}}] = \sum_j Enc[H_{j,\text{missing}}] \quad (12.2)$$

$$Enc[G_{\text{all}}] = \sum_j Enc[GL_j] + \sum_j Enc[GR_j] + Enc[G_{\text{missing}}] \quad (12.3)$$

$$Enc[H_{\text{all}}] = \sum_j Enc[HL_j] + \sum_j Enc[HR_j] + Enc[H_{\text{missing}}] \quad (12.4)$$

At this point, considering the scenario where missing values are placed on both subtree sides, the total gradient sums for the left subtree under two situations are calculated as follows:

$$Enc[GL_{\text{if missing go right}}] = \sum_j Enc[GL_j] \quad (12.5)$$

$$Enc[HL_{\text{if missing go right}}] = \sum_j Enc[HL_j] \quad (12.6)$$

$$\begin{aligned}
& Enc[GL_{\text{if missing go left}}] \\
& = Enc[G_{\text{if missing go right}}] + Enc[G_{\text{missing}}] \quad (12.7)
\end{aligned}$$

$$\begin{aligned}
& Enc[HL_{\text{if missing go left}}] \\
& = Enc[H_{\text{if missing go right}}] + Enc[H_{\text{missing}}] \quad (12.8)
\end{aligned}$$

Step 6: Based on the multi-party decryption method detailed in Section 12.6.1, the server coordinates with clients to decrypt the above encrypted values. The decrypted results are held by individual clients. The initiator client calculates the split benefits of f_k for both directions of missing values according to Equation (12.9). The greater of the two situations is recorded as the split benefit for the feature f_k :

$$\text{Gain} = \frac{1}{2} \cdot \left[\frac{GL^2}{HL + \lambda} + \frac{(G_{\text{all}} - GL)^2}{H_{\text{all}} - HL + \lambda} - \frac{G_{\text{all}}^2}{H_{\text{all}} + \lambda} \right] - \gamma \quad (12.9)$$

where γ represents the minimum split threshold parameter of the boosting tree model and is effective when $\text{Gain} > 0$.

Step 7: Repeat Steps 3–6 for each feature in the selected set. Finally, determine the feature, threshold, and benefit $\{f, v, \text{Gain}\}$ with the maximum split benefit as the first feature candidate for the current node N .

For calculating the second split candidate, the second training set is used, which comprises common samples from all clients and is limited to private features available locally to each client. Thus, there are no difficulties related to cross-platform feature ranking. Consistent with the XGBoost feature selection scheme, we designed the following computation method:

Step 1: The initiator client j , based on the common samples and gradient updates at the current node N , computes the total first-order gradient G_{common} and second-order gradient H_{common} :

$$G_{\text{common}} = \sum_{\text{common } id} g_i \quad (12.10)$$

$$H_{\text{common}} = \sum_{\text{common } id} h_i \quad (12.11)$$

The decrypted results G_{common} and H_{common} are sent back to the server.

Step 2: Client j , using the common samples at the current node, randomly gets a feature set. It sorts all values for each feature f_k , maps samples to buckets, calculates the gradient $Enc[Gkv]$ and $Enc[Hkv]$ in each bucket after bucket mapping, and sends them to the server.

Step 3: The server forwards $Enc[Gkv]$ and $Enc[Hkv]$ to the initiator client.

Step 4: The initiator client, using the encryption and decryption scheme from Section 12.6.1, decrypts $Enc[Gkv]_k$ and $Enc[Hkv]_k$. It then computes the total gradient of common samples in the left sub tree for feature f_k at value v and calculates the split gain as follows:

$$\begin{aligned} \text{Gain} = \frac{1}{2} \cdot \left[\frac{GL^2}{HL + \lambda} + \frac{(G_{\text{common}} - GL)^2}{H_{\text{common}} - HL + \lambda} \right. \\ \left. - \frac{G_{\text{common}}^2}{H_{\text{common}} + \lambda} \right] - \gamma \end{aligned} \quad (12.12)$$

Ultimately, the feature, threshold, and benefit $\{f, v, \text{Gain}\}$ with the maximum split benefit among all clients are selected as the second feature candidate for the current node N .

After calculating the two split candidates, the one with the greater split gain is chosen as the final splitting method for node N . Based on the missing value direction corresponding to this split candidate, the direction of the left and right subtrees for missing feature IDs is determined, and then the next tree node is generated.

12.2.2 Model Storage and Mixed Inference

Upon completion of training, the saving and use of the mixed federal gradient boosted trees differ from vertical models. In the vertical federal gradient boosted tree model, the tree structure is entirely saved on one of the training clients. Before the inference begins, the client sends the tree structure back to the server, which then oversees the inference process. In the mixed algorithm, we adopt a different saving approach. The tree model is distributed and stored across multiple clients. Each client stores a portion of the boosted tree's node

information and substructures. Before inference, the tree substructures from each client are consolidated on the server, enabling the reconstruction of the complete tree structure, with the server also supervising the inference process.

1. **Tree substructures:** Part of the tree's node IDs and the parent-child relationships among these nodes.
2. **Node information:** Split information of intermediate nodes and the weights of leaf nodes.

12.2.2.1 Storing mixed federal gradient boosted trees

Each node's information is saved based on the client where its splitting feature resides. For private features, they are stored on the only client where they exist during the training process. For common features, the server counts the effective features (i.e., features finally adopted by the tree model) of each client during training and stores the shared features on the client that has the most effective features. Apart from that, other tree model details, including initial predictions, maximum depth, regularization terms, and other parameters, are saved on the client with the most effective features.

The way the mixed boosted trees store leaf node weights is also unique. Since a client knows the final leaf node that all its samples arrive at, there could be situations where if a client has only the feature data of a particular sample without its label, and it has knowledge of the weights of the leaf node that this sample arrives at in every tree, it can approximate and derive a value close to the actual label. In classification tasks or some special regression tasks, the label values held by other clients might be leaked to some extent. To protect the weights of the leaf nodes and further prevent data leakage of client labels, the mixed boosted trees require that no single client should store a complete tree path. If all the split information of intermediate nodes on a path is stored on a particular client, the weight of the final leaf node of that path will be randomly saved on another client that has labels.

12.2.2.2 Optimized inference process

Each node data contains depth-first traversal ID information. Before inference starts, each client sends its local tree model substructure

to the server. The server then reconstructs the tree model based on the node's ID and parent-child relationships, overseeing the inference process.

In typical federated learning tree models, the server oversees the inference process, traversing through the tree nodes. Since the feature values of the inference samples are stored on various clients, every time the server passes through a node, it needs to inquire the client where the classification feature of that node is located. The client then responds based on the feature value, directing the sample to either the left or right subtree. Normally, this approach has two optimization opportunities:

1. Optimize the traversal process of a tree, allowing parallel inquiries of nodes on the same level.
2. Optimize the inference for forest models. Since the inference and training processes differ and there's no sequential relationship in computations between trees, tree levels can be parallelized, enabling simultaneous inference across multiple trees.

This is where the advantages of the split model saving approach are evident: Further optimization of the internal tree traversal is possible. Since each client stores a portion of the tree structure, if several parent-child node features are all on the same client, then that client can combine multiple inquiries directly, recursively computing the sample's direction until there are no saved nodes locally. This can significantly reduce the number of inference communications, thereby speeding up the inference process.

Specifically, for a sample path inquiry from the server, the client's processing algorithm is as follows:

Step 1: Construct the node processing Queue. Add the current node and the corresponding sample inquiry set to the queue.

Step 2: Dequeue a node N and sample inquiry set $\{id\}$ from the queue. Using its splitting feature f and splitting threshold v , divide $\{id\}$ into the left sample set $\{id_{\text{left}}\}$ and the right sample set $\{id_{\text{right}}\}$. For the root nodes N_{left} and N_{right} of node N 's left and right subtrees:

- (a) If it's a leaf node, store the scoring based on leaf weight locally for the corresponding sample set.

- (b) If the splitting feature for the node exists locally, add the node and its corresponding sample set to the queue.
- (c) If the splitting feature for the node is not available locally, mark the direction for the corresponding sample, either for node N 's left or right subtree.

Step 3: Repeat Step 2 until the queue is empty.

Step 4: Report the situation of the samples reaching the nodes back to the server and await the next inquiry from the server.

Once the server receives a result from an inquiry, it will continue to make inquiries to the client where the next node for the sample is located, based on the sample's path in the tree and the tree structure. In extreme cases, if two clients perform hybrid inference, where one only has label data and the other has all feature data, the aforementioned method will only involve a single inquiry process. This minimizes the impact of network communication speed. In practical tests, the inference speed in such scenarios is very fast.

The inference result of a sample is the sum of the leaf node weights it reaches in each tree. At the end of the inference, each client holds some leaf nodes reached by the sample. The prediction for the sample is obtained by summing up the inference results from all clients. When the number of participants reaches three or more, we employ the distributed threshold decryption of the Paillier algorithm (refer to Section 12.6.1) to facilitate this addition process. Neither the server nor the clients can know the prediction values of any other platform, but they can obtain the sum of predictions from multiple parties, further ensuring model security.

Next, we introduce the mixed federated linear regression algorithm.

12.3 Background of the Mixed Federated Linear Regression Algorithm

In recent years, the importance of data security and privacy protection has grown significantly, and the industry has become increasingly concerned with privacy issues in joint machine learning modeling. Federated learning is considered a crucial means to achieve multi-party modeling with security guarantees (considering security

from a technical perspective). Various federated learning models and systems have been proposed, but several challenges and problems remain.

The first issue arises from the waste of data due to choosing either horizontal or vertical models.² In joint modeling scenarios, data are constrained based on ID and features; one must align either all IDs or all features during training and inference. This confines all training and prediction data to those with aligned IDs, leading to significant data wastage in many contexts. Consider a two-party modeling scenario between Party A and Party B, where Party A has data IDs $1, 2, \dots, 10,000$ and features $f_1, \dots, f_{10}$, while Party B has data IDs from $5,000, \dots, 15,000$ and features $f_5, \dots, f_{15}$. Using horizontal federated learning, only six features are considered during training and inference, whereas with vertical federated learning, only 5,001 samples can be trained upon.

Moreover, during the inference stage, data that cannot align by ID (or features) cannot be used for inference. This means that distinguishing between horizontal and vertical federated learning methods might not always be suitable for certain contexts. Taking the previous data example, suppose Party A is the primary user of the model and both A and B derive model M using the vertical federated learning method. In the inference phase, every query from Party A needs to align with Party B's existing data; otherwise, results cannot be obtained. In many scenarios, the data that can align often constitutes only a small portion of the total, preventing effective inferences for much of the data.

The second issue concerns the efficiency of federated learning training. Due to the distributed nature of federated learning and the various encryption measures applied, the computation time and network transmission in the training process are significantly higher than traditional distributed learning. Typically, training a federated learning model using homomorphic encryption requires computational resources at least a hundred times greater than conventional methods. Additionally, frequent encrypted data transfers make network communication time a significant factor limiting the efficiency

²Currently, a prevalent classification method for federated learning algorithms divides them into horizontal and vertical categories. Horizontal models address situations where IDs differ, but all features align, while vertical models handle cases where features differ, but IDs align.

of federated model training. Some systems can take dozens of hours for training, making them impractical for scenarios where time efficiency is a priority.

The third challenge revolves around the ease of use for models and systems. User experience is a vital determinant of the success of an algorithmic product. Furthermore, the ease with which a model can be deployed also impacts user experience. Federated learning algorithms encompass distributed computing, multi-party secure computing, and machine learning, usually making the underlying algorithms complex. Inexperienced users might struggle to grasp various parameters and algorithmic details of federated learning. Furthermore, distinguishing data horizontally or vertically poses an entry barrier for users. Designing user-friendly and reliable algorithms and systems remains a significant challenge for federated learning platforms.

To address the first issue, this section proposes a mixed federated learning method based on transfer learning, as discussed in Section 12.4. This method doesn't require differentiating data horizontally or vertically. During the training phase, it allows all data to be used, achieving better training accuracy. During inference, the data don't need ID alignment, enabling individual parties to perform inference. For the second issue, we introduce an algorithmic improvement based on BFGS tailored for the federated learning context. This approach utilizes the property of the objective function in federated learning being a separable function. It differentiates various types of data and features to enhance the training speed of the original algorithm, as discussed in Section 12.5. Additionally, we incorporate multi-party secure computing, employing secret sharing and efficient homomorphic encryption, to ensure the safety of model training and inference, as explored in Section 12.6. Finally, for the third challenge, the proposed method doesn't impose horizontal or vertical restrictions on input data, making it accessible even for users without any algorithmic background knowledge. Coupled with a one-click deployment system framework, it enhances product usability.

12.4 Objective Function of Mixed Linear Regression

Drawing inspiration from transfer learning and considering the computational efficiency of homomorphic encryption, we propose a

simple and effective mixed linear model. In this section, “mixed” implies that input data don’t distinguish between horizontal or vertical data sources. If data contain both horizontal and vertical data, they are combined and used as the model’s input data. This design aims to (1) improve model accuracy and (2) enhance algorithm usability.

Federated learning models often operate under the presumption that there’s some level of correlation across the data distributions from multiple parties. Here, we assume that the data from each participating party (or entity) is a sample taken from different subspaces within a high-dimensional space, which we denote as S . Intuitively and without loss of generality, it’s feasible to assume that within this space S , there exists a W such that $Y = f(W, X)$. Here, for linear regression, f is a linear function, and for logistic regression, f is the softmax function. When the data are scattered across different subspaces, they may complement or intersect with each other across various dimensions. For instance, if parties A and B are collaboratively modeling and both have the same data IDs, with party A holding features f_1, f_2, f_3 and party B having f_3, f_4, f_5 , we see an opportunity. If we can weight the shared feature f_3 in some manner and then concatenate features from both parties, we effectively form a comprehensive dataset across the entire space. Similarly, for datasets with unaligned IDs, we treat missing features as null and only weight the non-null portions. Thus, in general, we assume that different subspaces carry distinct weights. The underlying concept of the mixed linear algorithm is to fit the original function $Y = f(W, X)$ through these weighted linear combinations.

More specifically, suppose there are p parties participating in a training session, and all data across these parties are represented by X . Each party possesses a subset of features and data. For party A, if it holds the i th data from X , we denote it as x_i^a . Note that since data features might vary among parties, the i th data piece from different parties will generally differ, representing various features for the same data ID. For example, x_i^a and x_i^b could correspond to the same data but reflect the features that parties A and B respectively have. Sometimes, a piece of data might exist only in a subset of the parties, rendering it null in others.

Next, we proceed to weight the features of each party. The initial step is to discern the overlap in data features. For hybrid datasets,

like when parties are A, B, and C, some data features might overlap between A and B, some between B and C, and some might be common across all three or only exist in one. For each data piece in every party, we need to ascertain its overlap with other parties. When there are p parties, there can be up to $2^p - 1$ overlap configurations. The second step involves attributing distinct weight variables for each overlap type. Specifically, for the i th sample, assuming its overlapping parties' set is O_i , the estimation is $\hat{y}_i = \sum_{j \in O_i} \alpha_j \hat{y}_i^n$. Here, α_j denotes the weight for the j th overlapping party.

From the discussions above and by integrating the L2 norm regularization on the sum of weights, the loss function for the mixed linear regression (using MSE as the Loss) becomes

$$L(W, \alpha) = \sum_{i=1}^n l \left(\sum_{j \in O_i} \alpha_j \hat{y}_i^n \right) + \left(\sum_{i=1}^n \alpha_{p_i} - p \right)^2$$

It's noteworthy that in practical applications, we sometimes acquire prior knowledge about the feature weights of the parties (e.g., through data collection methods or preprocessing techniques). Drawing from this experience, we can manually set weights based on the overlap count (for instance, set α proportionally to the feature overlaps; for a pure vertical modeling of parties A, B, and C, set all α values to 1). In such cases, the mixed linear method reduces to standard linear regression. Conversely, without prior knowledge, the weights of the parties are treated as part of the variables and are learned during training.

12.5 Improvements in First and Second Derivative Calculations

To accelerate training, for cases with prior knowledge (such as ordinary linear regression where the loss function is convex), we employ the BFGS second-order optimization method. For cases without prior knowledge, where the loss function is non-convex, we use the steepest descent method with a step size search. Unlike commonly used distributed optimization algorithms in machine learning, federated learning often encounters issues of excessive time and space overhead in real-world applications. This is primarily because: (1) Multi-party

communication requires homomorphic encryption, where a single multiplication or addition operation can take more than a hundred times the time of regular operations. Thus, operations under homomorphic encryption become the system's time bottleneck; (2) The volume of data under homomorphic encryption surges dramatically, posing challenges to network transmission.

To address these issues, traditional first and second-order optimization methods need to be improved. The aim is to maximize the amount of computation that can be performed using local data without encryption. Afterward, only a small amount of encrypted results should be transferred to the master, from which global results can be obtained through simple homomorphic encryption operations. The method introduced in this section primarily considers: segmenting the data into private and non-private parts. For the separable function property, the overall loss function is decomposed, and the private data part is entirely computed locally through linear mapping. Then, for the non-separable vertical part, the objective function is locally approximated through the second-order separable function, transformed into a separable function, and solved.

For federated learning objective functions that mix horizontally and vertically, a separable function can be defined as follows:

$$L(W) = \sum_{x \in X_{p_i}} l(f_{p_i}(x, U_{p_i}^T W), t) + \sum_{x \in \cap_j X_{p_j}} l(f(x, W), t)$$

where f_{p_i} is the local prediction function defined on the subspace constituted by data from party p_i , i.e., a function from $R^{m_{p_i}}$ to R ; f is the global prediction function defined over the entire data dimension space, meaning f is a function from R^m to R ; U_{p_i} is a linear mapping from R_m to $R^{m_{p_i}}$. In practical implementation, as we directly supplement missing data with zeros, U_{p_i} is equivalent to a 1-hot vector. $l(W)$ is the loss function on R^m ; $l_{p_i}(Z)$ is the loss function on $R^{m_{p_i}}$ (subspace); t is the target value to be fitted.

Let $Z = U_{p_i}^T W$. To differentiate the above equation with respect to gradient and Hessian matrix, apply the chain rule:

$$\begin{aligned} \nabla_W L &= \frac{1}{n} \sum_{x \in X_{p_i}} U_{p_i} \nabla_{f_{p_i}} l(f_{p_i}) \nabla_z f_{p_i}(z) + \sum_{x \in \cap_j X_{p_j}} \nabla_f l(f) \nabla_W f(W) \\ &= \sum_{x \in X_{p_i}} U_{p_i} \nabla_Z l_{p_i}(Z, t) + \sum_{x \in \cap_j X_{p_j}} \nabla_f l(f) \nabla_W f(W) \end{aligned}$$

$$\begin{aligned}
\nabla_W^2 L &= \frac{1}{n} \sum_{x \in X_{p_i}} U_{p_i} \nabla_W (\nabla_Z l_{p_i}(Z, t)) + \sum_{x \in \cap_j X_{p_j}} \nabla_f l(f) \nabla_W^2 f(W) \\
&= \frac{1}{n} \sum_{x \in X_{p_i}} U_{p_i} \nabla_Z (\nabla_Z l_{p_i}(Z, t)) \nabla_Z W + \sum_{x \in \cap_j X_{p_j}} \nabla_f l(f) \nabla_W^2 f(W) \\
&= \frac{1}{n} \sum_{x \in X_{p_i}} U_{p_i} \nabla_Z^2 l_{p_i}(Z, t) U_{p_i}^T + \sum_{x \in \cap_j X_{p_j}} \nabla_f l(f) \nabla_W^2 f(W)
\end{aligned}$$

Noting that $l_{p_i}(Z, t)$ is a local function, the equation above can be computed across multiple clients.

12.5.1 For Private Data

From the above, it's evident that the first and second derivatives of private data (data with non-aligned IDs) can be completely calculated locally. The local results are mapped to the full space R^m through $U_{p_i}^T$ and finally uploaded to the master through one reduce-sum operation. Similarly, when approximating the Hessian matrix $\nabla_W^2 L$ of L with respect to W using the quasi-Newton method, you can directly approximate $\nabla_{W_{p_i}}^2 L_{p_i}$ locally at each party and then map the approximation results to the full space. This approach not only reduces computational complexity but also minimizes the errors generated when approximating sparse high-dimensional matrices.

From the above discussion, the pseudocode for the first-order derivative and the approximate inverse matrix of the Hessian for the private data part is as follows:

12.5.2 For ID-Overlapping Data

12.5.2.1 Gradient

Since an accurate gradient value is desired, different parties need to transmit data among each other when computing $\nabla_f l(f)$ and $\nabla_W f(W)$. This approach is similar to the gradient computation method used in previous mixed linear methods.

Algorithm 12.1 Calculate the First and Second Derivatives for Private Data (Using party p , iteration k as an example)

Input: Optimization parameters from the $k-1$ round W_{k-1}^p and the gradient g_p and the inverse matrix of the Hessian H_p from the loss function derived from the party's own data, the training data Z_p of the p th party, and the feature transformation matrix U_p .

Output: The local gradient g_p of party p and the inverse matrix of the Hessian H_p .

```

1: Function getGradInvhessPrivWp
2:  $g_{plast} = g_p$ 
3:  $g_p = \nabla_w L(Z_p, W_p)$ 
4: // Update W based on the BFGS method
5:  $W_p = \text{UPDATEW}(W_p, H_p)$ 
6:  $d = g_p - g_{plast}$ 
7:  $s = W_p - W_{k-1}^p$ 
8:  $\rho = 1/(d^T s)$ 
9: // Update H using the BFGS method
10:  $H_p = \text{UPDATEH}(d, s, \rho)$ 
11: return  $g_p, H_p$ 
12: End Function

```

12.5.2.2 Hessian matrix

Due to the large computational demand of the Hessian matrix, it is not computed directly. Instead, an approximation of the inverse Hessian matrix is made using the BFGS method. Let's denote the approximated inverse Hessian matrix as H . In BFGS, the iterative formula for H is

$$H_{k+1} = (I - \rho_k s_k d_k^T) H_k (I - \rho_k d_k s_k^T) + \rho_k s_k s_k^T \quad (12.13)$$

where

$$\begin{aligned}
s_k &= w_{k+1} - w_k \\
d_k &= \nabla f_W(W)^{(k+1)} - \nabla f_W(W)^{(k)} \\
\rho_k &= \frac{1}{s_k d_k^T}
\end{aligned}$$

Table 12.1. Precise computation process for H and g .

	Master	Client
Phase 1	Compute $\frac{\partial L^{(p)}}{\partial f}, \frac{\partial f^{(p)}}{\partial w_i}$	Compute $\frac{\partial l^{(p)}}{\partial f}, \frac{\partial f^{(p)}}{\partial w_i}$ and send to Master
Phase 2	Master aggregates all $\frac{\partial L^{(p)}}{\partial f}$ to get $\frac{\partial L}{\partial f}$ and sends to all clients	
Phase 3	Compute $\frac{\partial L}{\partial f} \frac{\partial f^{(p)}}{\partial w_i}$	Compute $\frac{\partial L}{\partial f} \frac{\partial f^{(p)}}{\partial w_i}$ and send to Master
Phase 4	Master aggregates results from all parties to obtain the first-order derivative $\frac{\partial L}{\partial w_i}$	
Phase 5	Master computes H , similarly to Algorithm 12.1	

Observing the above, to compute H accurately, clients need to transmit their respective s . The master then needs to aggregate and compute $\nabla_W f(W)$, s, d, ρ to obtain the final value.

Further, by

$$\begin{aligned}
 \frac{\partial L}{\partial w_i} &= \frac{\partial \left(\sum_p L^{(p)} \right)}{\partial f} \frac{\partial \left(\sum_p f^{(p)} \right)}{\partial w_i} \\
 &= \left(\sum_p \frac{\partial L^{(p)}}{\partial f} \right) \left(\sum_p \frac{\partial f^{(p)}}{\partial w_i} \right) = \sum_p \left[\frac{\partial f^{(p)}}{\partial w_i} \sum_p \frac{\partial L^{(p)}}{\partial f} \right]
 \end{aligned}$$

the accurate computation process for H and g is summarized in Table 12.1 as follows.

12.6 Secure Computation Protocol

In this section, we integrate secure Multi-Party Computation (MPC) into the aforementioned algorithmic process, ensuring data privacy during computation from a technical standpoint. We operate on the semi-honest model, assuming that the corrupted party merely gathers information from the protocol without deviating from its specifications. For reasons of security and fairness, we employ a decentralized architecture. This means that key generation and decryption require the collaboration of all parties involved.

Each party's key must be unique, and decryption requires the full participation of all parties to be successful. During training and inference, we use homomorphic encryption to compute in the ciphertext state, ensuring technical privacy and improved computational efficiency. To meet the demands of various real-world scenarios, our approach guarantees that raw data and first-order derivatives during computation remain confidential.

Next, we introduce two basic components: we utilize (1) secret sharing and (2) homomorphic encryption.

12.6.1 Key Generation Based on Secret Sharing

Shamir's secret sharing, a method based on polynomial interpolation, was introduced by Shamir (1979). It divides plaintext into several parts, and combining these parts is essential for successful decryption. Given that this paper employs the Paillier encryption algorithm, key generation relies on two large prime numbers. When generating a multi-party key, the prime numbers p and q are generated with the involvement of each party, and their product N is calculated. Each party retains part of the prime numbers, unaware of their actual values. Following the method introduced by Boneh *et al.* (2005), we implement this using BGW protocol and secret sharing. The key generation process can be summarized in three phases: (1) Each party produces random p_i and q_i . Through the BGW protocol, p_i and q_i are gathered, and N is computed; (2) prime testing — the parties verify if the generated N is indeed the product of two primes. If so, they proceed; otherwise, the generated N is discarded, and the process starts anew; (3) after obtaining N , parties generate the key through secret sharing and distribute its parts (shares) to each participant. A public key is also distributed to all. Note that the first two phases are often repeated multiple times, but as Malkin *et al.* (1999) demonstrated, the time overhead remains acceptable.

12.6.2 Homomorphic Encryption with Threshold Decryption

Homomorphic encryption is a powerful tool for secure multi-party computation. It allows specific algebraic operations on ciphertexts,

producing encrypted results. When decrypted, these match the outcomes of similar operations on plaintext. This capability enables operations such as searches and comparisons on encrypted data, yielding accurate results without needing to decrypt the data in the process. While full homomorphic encryption allows for both addition and multiplication on ciphertexts (albeit more slowly), partial homomorphic encryption supports only one of these operations but is generally faster. Given the stringent time requirements in many federated learning scenarios, we opted for the efficient Paillier scheme for ciphertext computation. Since multiplication on ciphertexts isn't feasible, nonlinear functions like the logistic function are approximated using piecewise linear functions. As outlined in Section 12.6.1, we employ secret sharing to generate Paillier keys, with each party retaining part of the key, facilitating homomorphic encryption computations. During decryption, all participating parties collaborate to obtain the final value.

12.6.3 Secure Federated Training Method

Based on the multi-party key generation (Section 12.6.1) and threshold homomorphic encryption (Section 12.6.2) methods described above, we propose a secure federated learning training approach. In most practical business scenarios, we assume that the parties involved in federated learning adhere to the semi-honest model. That is, they will faithfully execute cryptographic protocols but may attempt to extract private data from intermediate results. Compared to a stronger security assumption — the malicious model (where participants can entirely disregard cryptographic protocols and use any means to attack and extract private information) — the semi-honest model is significantly more time-efficient. Importantly, this approach can be further enhanced by incorporating encrypted authentication and zero-knowledge proofs to construct a method based on the malicious model.

Under the semi-honest assumption, our secure training aims to ensure that, apart from the input data (data and training parameters from all parties) and the final outcomes (model parameters for all parties), no other information can be accessed by any party. Notably, for performance reasons, the master node is aware of the approximate inverse Hessian matrix H . However, since the master

only ever accesses H during the entire training process and all other information remains encrypted, it cannot obtain specific first-order derivative values, let alone original data or training parameters. This meets the security requirements of most business scenarios.

The entire secure training process can be broadly divided into three steps: (1) public and private keys are generated for each participant, as detailed in Section 12.6.1; (2) participants encrypt their data using Paillier and compute the gradient g in its encrypted state. As per Equation (12.5), the master node is responsible for computing the ciphertext of g . Once this is obtained, the master sends corresponding feature values to each participant. Participants then decrypt (see Section 12.6.2) to obtain the updated gradient g . To further compute s and t in BFGS (Equation (12.13)), participants first calculate their individual s_i and t_i values (noting that for party i , if a particular feature is absent, it defaults to 0). They then use the BGW method to generate shares, which are sent to all parties to calculate shares of matrix H . The master subsequently decrypts these to obtain the approximate inverse Hessian matrix H . The master then inserts H into Equation (12.13) to determine the ciphertext of w , which is sent to each client; (3) each participant decrypts the ciphertext of w to obtain the updated w .

Chapter 13

Federated Reinforcement Learning

Reinforcement learning (RL) is the field within machine learning that deals with sequential decision-making. In this chapter, we describe how the RL problem is formalized as an agent that must make decisions within an environment to optimize a given cumulative reward. Evidently, this formalization applies to a wide variety of tasks and captures many fundamental features of artificial intelligence, such as a sense of causality and uncertainty. This chapter also introduces different methods for learning sequential decision tasks and the utility of deep learning. A key aspect of RL is that the agent learns good behavior. This means it progressively modifies or acquires new actions and skills. Another significant aspect of RL is that it employs trial-and-error experiences, in contrast to dynamic programming that presupposes environmental prior knowledge. As such, an RL agent doesn't need a full understanding or control of the environment; it just needs to interact with it and gather information. In an *Offline Setting*, experiences are acquired *a priori* and then used as batch data for learning (hence, the offline setting is also known as Batch RL). In contrast, in an *Online Setting*, data are available sequentially and used for stepwise updates to the agent's actions. In both cases, the core learning algorithm is essentially the same. The primary distinction in the online setting is that the agent can influence the way it gathers experiences, which is beneficial for learning. This is an added challenge, primarily because the agent must address the exploration/exploitation dilemma while learning. Yet, learning in an online setting is advantageous, as the agent can specifically gather information from the most intriguing parts of the environment.

Thus, even for well-understood environments, RL methods might still offer the most computationally efficient approach, compared to certain dynamic programming methods that lack specificity.

13.1 Overview of Reinforcement Learning

A general RL problem can be formalized as a discrete-time stochastic control process where an agent interacts with its environment as follows: The agent begins in a given *state* $s_0 \in \mathcal{S}$ within its environment, informed by an initial *observation* $\omega_0 \in \Omega$. At each time step t , the agent must take an *action* $a_t \in \mathcal{A}$. As illustrated in Figure 13.1, there are three outcomes: (1) the agent receives a *reward* $r_t \in \mathcal{R}$, (2) the state transitions to $s_{t+1} \in \mathcal{S}$, and (3) the agent receives a new observation $\omega_{t+1} \in \Omega$.

13.1.1 Markov Property

We first provide the definition of Markovian stochastic control processes.

Definition 13.1. If a discrete-time stochastic control process possesses the properties,

$$\square \mathbb{P}(\omega_{t+1}|\omega_t, a_t) = \mathbb{P}(\omega_{t+1}|\omega_t, a_t, \dots, \omega_0, a_0) \text{ and}$$

$$\square \mathbb{P}(r_t|\omega_t, a_t) = \mathbb{P}(r_t|\omega_t, a_t, \dots, \omega_0, a_0)$$

then we say the process is Markovian (i.e., it has the Markov property).

The Markov property implies that the future of the process depends solely on the current observation, and the agent is not

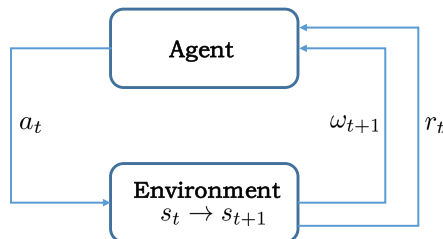


Figure 13.1. Interaction of the agent–environment in RL.

interested in viewing the complete history. A *Markov Decision Process* (MDP) is a discrete-time stochastic control process, defined as follows.

Definition 13.2. An MDP is a quintuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where

- $\mathcal{S}$ is the state space,
- $\mathcal{A}$ is the action space,
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function (also a collection of conditional state transition probabilities),
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$ is the reward function, where $\mathcal{R}$ is the continuous set of possible rewards in the range $R_{\max} \in \mathbb{R}^+$ (i.e., $[0, R_{\max}]$),
- $\gamma \in [0, 1]$ is the discount factor.

In the MDP, the system is fully observable, implying that observations are the same as the environment's states: $\omega_t = s_t$. At each time step t , the probability of moving to s_{t+1} is given by the state transition function $T(s_t, a_t, s_{t+1})$, and the reward is given by a bounded reward function $R(s_t, a_t, s_{t+1}) \in \mathcal{R}$. This is illustrated in Figure 13.2.

13.1.2 Various Types of Policies

A *Policy* dictates how an agent chooses its actions. Policies can be classified based on stationary or non-stationary criteria. Non-stationary policies depend on time steps and are useful for environments with a finite horizon, where the cumulative reward the agent seeks to optimize is limited to a finite number of future

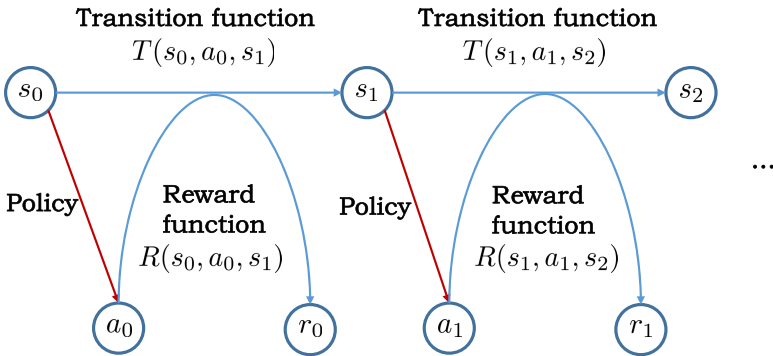


Figure 13.2. Illustration of an MDP. At each step, the agent takes an action, changes its state in the environment, and receives a reward.

time steps. In the context of introducing deep learning, with an infinite horizon, the policy is stationary.

Policies can also be categorized based on a second criterion, i.e., determinism or stochasticity:

- In the deterministic case, a policy can be represented as $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$.
- In the stochastic case, a policy can be represented as $\pi(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, where $\pi(s, a)$ denotes the probability of choosing action a in state s .

13.1.3 Expected Returns

In this section, we consider a Reinforcement Learning (RL) agent aiming to identify a policy $\pi(s, a) \in \Pi$ to optimize an expected return $V^\pi(s) : \mathcal{S} \rightarrow \mathbb{R}$ (also known as the V -value function). This can be expressed as follows:

$$V^\pi(s) = \mathbb{E} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, \pi \right] \quad (13.1)$$

Here:

- $r_t = \mathbb{E}_{a \sim \pi(s_t, \cdot)} R(s_t, a_t, s_{t+1})$
- $\mathbb{P}(s_{t+1} | s_t, a_t) = T(s_t, a_t, s_{t+1})$, where $a_t \sim \pi(s_t, \cdot)$

From the definition of expected returns, the optimal expected return is

$$V^*(s) = \max_{\pi \in \Pi} V^\pi(s) \quad (13.2)$$

Apart from the V -value function, we can introduce a few other functions of interest. The Q -value function $Q^\pi(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is defined as follows:

$$Q^\pi(s, a) = \mathbb{E} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi \right] \quad (13.3)$$

For Markov Decision Processes (MDP), this equation can be recursively reformulated using Bellman's equation:

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} T(s, a, s') (R(s, a, s') + \gamma Q^\pi(s', a = \pi(s'))) \quad (13.4)$$

Similar to the V -value function, the optimal Q -value function $Q^*(s, a)$ can be defined as follows:

$$Q^*(s) = \max_{\pi \in \Pi} Q^\pi(s, a) \quad (13.5)$$

A distinctive feature of the Q -value function, compared to the V -value function, is that the optimal policy can be directly derived from $Q^*(s, a)$:

$$\pi^*(s) = \arg \max_{a \in \mathcal{A}} Q^*(s, a) \quad (13.6)$$

The optimal V -value function $V^*(s)$ is the expected discounted reward when following the policy π^* for a given state s . The optimal Q -value function $Q^*(s, a)$ is the expected discounted reward for a given action a in state s when following policy π^* .

We can also define an advantage function:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \quad (13.7)$$

This quantity describes the relative advantage of action a compared to the expected return when directly following policy π . Note that a direct method to obtain estimates for $V^\pi(s)$, $Q^\pi(s, a)$, or $A^\pi(s, a)$ is through *Monte Carlo Methods*, that is, defining an estimate by performing several simulations starting from state s while following policy π . In practice, as we see, this is virtually impossible with limited data. Even if feasible, to improve computational efficiency, other methods are generally preferred.

13.1.4 Different Components and Setups for Learning Policies

An RL agent can comprise one or more of the following components:

- a *Value Function* representation, predicting the quality of each state or state/action pair,
- a direct representation of a *Policy* $\pi(s)$ or $\pi(s, a)$,
- an *Environment Model* (i.e., estimated transition and reward functions) combined with a planning algorithm.

The first two components are related to what is termed Model-free RL. When the last component is utilized, the algorithm is referred to

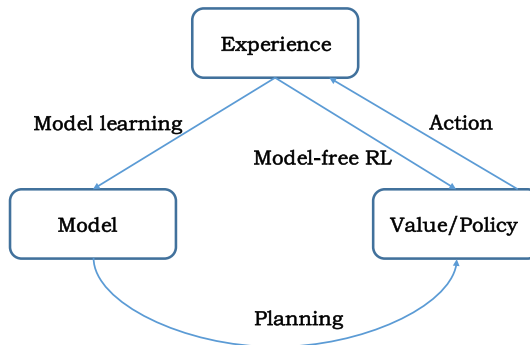


Figure 13.3. General schema of different RL methods. Direct methods employ representations of the value function or policy to act in the environment, while indirect methods leverage a model of the environment.

as Model-based RL. The combination of both, as well as the rationale for using this combination, is discussed in the following sections. Figure 13.3 offers an illustrative diagram containing all possible methods.

For most problems approaching real-world complexity, the state space is high-dimensional (and possibly continuous). To learn estimations of the model, value function, or policy, RL algorithms combined with deep learning have two primary advantages:

- ❑ Neural networks are apt at handling high-dimensional sensory inputs like time series and frames, and practically, they don't require exponential growth in data when adding additional dimensions to the state or action space.
- ❑ Moreover, they can be trained incrementally and leverage additional samples obtained during learning.

We now introduce key settings that can be employed in RL.

Offline and online learning: Sequential decision tasks emerge in two scenarios: (i) In offline learning, only limited data are available in a given environment; (ii) in online learning, an agent accumulates experience in the environment while learning. In both cases, the core learning algorithms we introduce next are fundamentally the same. The distinct feature of the batch setting is that the agent must learn from limited data without further interaction with the environment. In this scenario, enhancing generalization is considered.

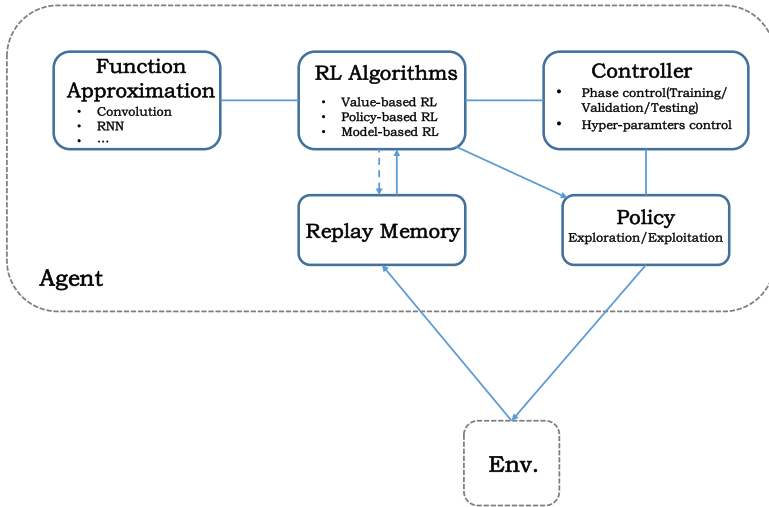


Figure 13.4. General schema of deep RL methods.

In the online setting, learning is more intricate and learning without the need for massive data (sample efficiency) is not only influenced by the learning algorithm's ability to generalize well from limited experience. Indeed, the agent can gather experience using an exploration/exploitation strategy. Furthermore, it can employ a replay memory to store its experiences for later reprocessing. In both batch and online settings, an additional consideration is computational efficiency, wherein efficiency depends on the efficacy of a given gradient descent step. All these elements are elaborated upon in subsequent sections. Figure 13.4 offers a general schema of different elements found in most deep RL algorithms.

On-policy and off-policy learning: As proposed by Sutton and Barto in 2018, on-policy methods aim to evaluate or improve the policy used for decision-making, while off-policy methods assess or refine a policy different from the one used to generate the data. In off-policy methods, learning is direct when using trajectories not necessarily obtained under the current policy but acquired from different behavioral policies $\beta(s, a)$. Under these circumstances, experience replay allows the reuse of samples from different behavioral policies. In contrast, on-policy methods usually introduce a bias when used with replay buffers because trajectories are typically not solely obtained

under the current policy π . As is discussed in the following sections, this makes off-policy methods sample-efficient as they can leverage any experience; in comparison, on-policy methods might introduce bias when using off-policy trajectories if not handled with care.

13.2 Introduction to Reinforcement Learning Algorithms

In this section, we present RL following the sequence in Figure 13.4: Value-based RL, Policy-based RL, and Model-based RL.

13.2.1 Value-Based RL

Value-based algorithms aim to construct a value function, which subsequently lets us define a policy. We discuss one of the simplest and most popular value-based algorithms, the Q -learning algorithm (proposed by Watkins (1989)) and its variant, Fitted Q -learning, which uses a parametrized function approximator (introduced by Gordon (1996)). We also delve into the key elements of the Deep Q -Network (DQN) algorithm (introduced by Mnih *et al.* (2015)), which employs a neural network as a function approximator and achieved superhuman performance in playing Atari games from pixels. We then review various improvements to the DQN algorithm and provide resources for further details. Toward the end of this chapter, we discuss the close connection between value-based and policy-based methods.

Q -learning algorithm: The fundamental version of Q -learning is a lookup table that stores the value $Q(s, a)$ (i.e., Equation 13.3) where each state–action pair has an entry. To learn the optimal Q -value function, the Q -learning algorithm leverages the Bellman equation for Q -values, whose unique solution is $Q^*(s, a)$:

$$Q^*(s, a) = (\mathcal{B}Q^*)(s, a) \quad (13.8)$$

where $\mathcal{B}$ is the Bellman operator that maps any function $K : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ to another function $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, defined as follows:

$$(\mathcal{B}K^*)(s, a) = \sum_{s' \in \mathcal{S}} T(s, a, s') (R(s, a, s') + \gamma \max_{a' \in \mathcal{A}} K(s', a')) \quad (13.9)$$

According to Banach's theorem, the fixed point for the Bellman operator $\mathcal{B}$ exists since it is a contraction mapping. In practice, under conditions where state-action pairs can be discretely represented and all actions are repeatedly sampled in all states (ensuring adequate exploration, thus eliminating the need to access the transition model), a general proof of convergence to the optimal value function is available (as per Watkins and Dayan's 1992 paper).

Due to high-dimensional (and potentially continuous) state-action spaces, this simple setting often isn't applicable. In these instances, a parametrized value function $Q(s, a; \theta)$ becomes essential, where θ denotes some parameters defining the Q -values.

13.2.2 Fitted Q -Learning Algorithm

Experience is collected and stored in the given dataset D as tuples $\langle s, a, r, s' \rangle$, where the state at the next time step s' is drawn from $T(s, a, \cdot)$, and the reward r is given by $r(s, a, s')$. In fitting the Q -learning algorithm, the algorithm starts from some random initialization of Q -values $Q(s, a; \theta_0)$, where θ_0 denotes the initial parameters (typically, initial Q -values should be close to 0 to prevent slow learning). The Q -value approximation $Q(s, a; \theta_k)$ in the k th iteration then updates toward the target value:

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta_k) \quad (13.10)$$

Here, θ_k refers to some parameters defining the Q -value in the k th iteration.

In the Neural Fitted Q -learning (NFQ) method, introduced by Riedmiller (2005), the state can be fed into a Q -network, yielding different outputs for every possible action. This provides an efficient structure, allowing for the advantage of obtaining $\max_{a' \in \mathcal{A}} Q(s', a'; \theta_k)$ in a single forward pass of the neural network for a given s' . Q -values can be parameterized by a neural network $Q(s, a; \theta_k)$, where parameters θ_k are updated through stochastic gradient descent (or its variants) by minimizing the squared loss:

$$L_{DQN} = (Q(s, a; \theta_k) - Y_k^Q)^2 \quad (13.11)$$

Thus, the update amount for Q -learning during parameter updates is as follows:

$$\theta_{k+1} = \theta_k + \alpha(Y_k^Q - Q(s, a; \theta_k)) \nabla_{\theta_k} Q(s, a; \theta_k) \quad (13.12)$$

where α is the neural network learning rate, also known as the step size. Note that the use of the squared loss isn't arbitrary. Indeed, it ensures $Q(s, a; \theta_k)$ converges to the expected value of the random variable Y_k^Q without bias. It thus ensures that $Q(s, a; \theta_k)$ should converge to $Q^*(s, a)$ after multiple iterations, assuming the neural network is well suited for the task and that experience collected in dataset D is sufficient. When updating weights, target values also change. Due to the generalization and inference abilities of neural networks, this approach can introduce substantial errors or divergence at different positions in the state-action space. As a result, the contraction mapping property of the Bellman operator, as in Equation (13.9), isn't sufficient to guarantee convergence. Experiments show that these errors may propagate with this update rule, making convergence slow or even unstable. Another related detrimental effect of using function approximators is that due to the max operator, Q -values tend to be overestimated. Given the risks of instability and overestimation, special care must be taken to ensure proper learning.

13.2.3 Deep Q -Networks

Leveraging the idea from NFQ, the DQN algorithm introduced by Mnih *et al.* (2015) managed to achieve strong performances in various Atari games in online settings by learning directly from pixels. It employs two heuristics to mitigate instability:

- The target Q -network in Equation (13.10) is replaced by $Q(s', a'; \theta_k^-)$, where parameters θ_k^- are updated only every $C \in \mathbb{N}$ iterations, assigned as $\theta_k^- = \theta_k$. This prevents rapid propagation of instability, reducing risks of divergence, as the target values Y_k^Q remain constant for C iterations. The idea of a target network can be seen as an instance of fitted Q -learning, where each period between target network updates corresponds to a single-fitted Q iteration.

- In online settings, a replay memory retains all information of the last $N_{\text{replay}} \in \mathbb{N}$ time steps, where experiences are gathered by following an ϵ -greedy policy. Updates then occur on a set of tuples $\langle s, a, r, s' \rangle$ randomly selected from the replay memory, referred to as a mini-batch. This technique allows for updates covering a broad range of the state space. Furthermore, the variance of updates from a mini-batch is smaller compared to those from a single tuple. Consequently, it provides the possibility for larger parameter updates, all while maintaining the efficiency of algorithm parallelization.

The schematic of the algorithm is depicted in Figure 13.5. Besides the target Q -network and replay memory, DQN also employs several essential heuristic methods. To keep the target values within a reasonable range and ensure proper learning in practice, rewards are constrained between -1 and $+1$. Clipping rewards limits the magnitude of the error derivatives and makes using the same learning rate across multiple games easier, though it introduces a bias. In games where the player has multiple lives, a strategy associates terminal states with life loss, prompting the agent to avoid these terminal states (in these states, the discount factor is set to 0). In DQN, various deep learning-specific techniques are also employed. Notably, input preprocessing steps are used to reduce input dimensionality, normalize the input (scaling pixel values to the range $[-1, 1]$), and handle specific features of the task. Additionally, convolutional layers are used for the first layer of the neural network function approximator, and the optimizer is a variant of Stochastic Gradient Descent (SGD) known as RMSprop.

13.2.4 Double DQN

The max operation in Q -learning (i.e., Equations (13.9) and (13.10)) uses the same values for both selecting and evaluating an action. This can lead to an overestimation of the chosen values, especially in situations where values are noisy or imprecise, resulting in overly optimistic value estimates. As a result, the DQN algorithm tends to produce upward biases. The double estimation method employs two estimates for each variable, ensuring that the estimator's choice and its value are decoupled. This allows for the removal of positive biases

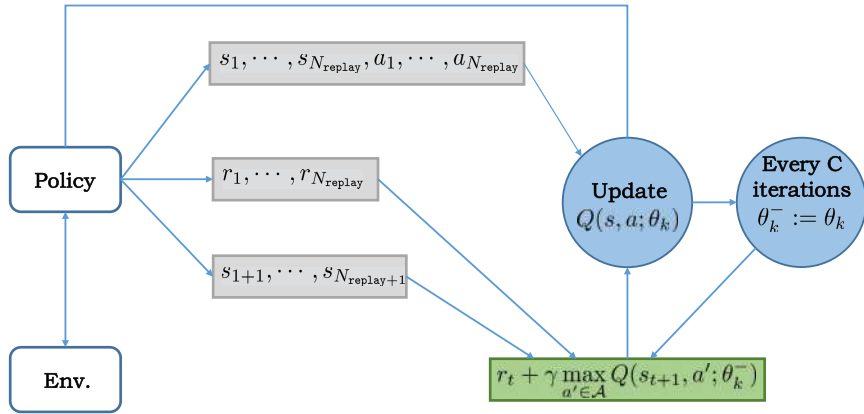


Figure 13.5. A schematic of the DQN algorithm. $Q(s, a; \theta_k)$ is initialized with random values (close to 0) throughout its domain, and the replay memory is initialized as empty. The target Q -network parameters, θ_k^- , are updated with the Q -network parameters, θ_k , only every C iterations, and θ_k^- remains unchanged between two updates. Updates use a mini-batch (e.g., 32 elements) of tuples $\langle s, a \rangle$ randomly fetched from the replay memory and the corresponding target values for the mini-batch of tuples.

when estimating action values, irrespective of whether the errors in estimating the Q -values arise due to randomness in the environment, function approximation, non-stationarity, or any other sources. In Double DQN, also known as DDQN (introduced by Van Hasselt *et al.* (2016)), the target value Y_k^Q is replaced by

$$Y_k^{DDQN} = r + \gamma Q(s', \arg \max_{a \in \mathcal{A}} Q(s', a; \theta_k); \theta_k^-) \quad (13.13)$$

This results in lesser overestimation of Q -learning values, improved stability, and hence enhanced performance. In comparison to DQN, the target network with weights θ_t^- is used to evaluate the current greedy action. Note that the policy is still selected based on values obtained with the current weights θ .

Summary: The original DQN algorithm can be integrated with the various variants discussed earlier, as researched by Hessel *et al.* in their 2018 paper titled “Rainbow”. Their experiments demonstrated that a combination of all the DQN extensions mentioned previously delivered state-of-the-art performance on the Atari 2600 benchmark,

both in terms of sample efficiency and final results. Overall, most Atari games could be tackled, allowing deep RL agents to surpass human-level performance. However, methods based on DQN have their limitations. Specifically, such algorithms struggle with large and/or continuous action spaces, and they cannot explicitly learn stochastic policies. Modifications addressing these limitations are discussed in the upcoming sections on policy-based approaches. Indeed, the following section reveals that value-based and policy-based methods can be viewed as two facets of the same model-free approach. Therefore, the limitations of discrete action spaces and deterministic policies pertain only to DQN. It should also be noted that methods, whether value-based or policy-based, do not utilize any environment model, limiting their sample efficiency.

13.2.5 Policy-Based RL

This section primarily focuses on RL algorithms that utilize policy gradient methods. Due to the stochastic gradient ascent relative to policy parameters, these methods optimize performance objectives (typically the expected cumulative return) by identifying effective policies, often parameterized as neural networks. Note that policy gradient methods fall under the broader category of policy-based methods, which also include evolutionary strategies. Such strategies use learning signals derived from sampled instances of policy parameters, developed for achieving superior returns, as showcased in a paper by Salimans *et al.* (2017). Herein, we introduce stochastic and deterministic gradient theorems which provide gradients for policy parameters to optimize performance objectives. Subsequently, various RL algorithms employing these theorems are detailed.

13.2.6 Stochastic Policy Gradient

From Equation (13.1), the expected return for a stochastic policy π starting from a given state s_0 can be represented as follows:

$$V^\pi(s_0) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \pi(s, a) R'(s, a) da ds \quad (13.14)$$

where $R'(s, a) = \int_{s' \in \mathcal{S}} T(s, a, s') R(s, a, s')$ and $\rho^\pi(s)$ denotes the discounted state distribution, defined as follows:

$$\rho^\pi(s) = \sum_{t=0}^{\infty} \gamma^t \Pr\{s_t = s | s_0, \pi\} \quad (13.15)$$

For differentiable policies π_w , a fundamental result of these algorithms is the policy gradient theorem:

$$\nabla_w V^{\pi_w}(s_0) = \int_{\mathcal{S}} \rho^{\pi_w}(s) \int_{\mathcal{A}} \nabla_w \pi_w(s, a) Q^{\pi_w}(s, a) da ds \quad (13.16)$$

This result allows us to adjust the policy parameters $w : \Delta w \propto \nabla_w V^{\pi_w}(s_0)$ from experience. This finding is particularly intriguing since the policy gradient doesn't rely on the gradient of the state distribution, which might be somewhat anticipated. The simplest method for deriving the policy gradient estimate, i.e., estimating $\nabla_w V^{\pi_w}(s_0)$ from experience, involves employing a score function gradient estimator, often referred to as the "REINFORCE" algorithm. The likelihood ratio trick can be utilized as follows, leading to a general method for estimating the gradient from expected values:

$$\begin{aligned} \nabla_w \pi_w(s, a) &= \pi_w(s, a) \frac{\nabla_w \pi_w(s, a)}{\pi_w(s, a)} \\ &= \pi_w(s, a) \nabla_w \log(\pi_w(s, a)) \end{aligned} \quad (13.17)$$

Considering the above equation, we can deduce

$$\nabla_w V^{\pi_w}(s_0) = \mathbb{E}_{s \sim \rho^{\pi_w}, a \sim \pi_w} [\nabla_w (\log \pi_w(s, a)) Q^{\pi_w}(s, a)] \quad (13.18)$$

It should be highlighted that, in practice, most policy gradient methods effectively use an undiscounted state distribution without impairing their performance.

Thus far, we've illustrated that policy gradient methods should encompass both policy evaluation and policy improvement. On one hand, policy evaluation estimates Q^{π_w} . On the other hand, policy improvement undertakes gradient steps concerning the value function estimate to optimize the policy $\pi_w(s, a)$. Intuitively, the policy improvement step enhances the probability of actions proportionate to the expected return.

The remaining question centers on how the agent executes the policy evaluation step, i.e., how it obtains estimates of $Q^{\pi_w}(s, a)$. The simplest method to estimate the gradient is to substitute the Q function estimator with the cumulative returns of an entire trajectory. Within Monte Carlo policy gradients, $Q^{\pi_w}(s, a)$ is estimated based on the environment's trajectory while adhering to the policy π_w . When combined with the backpropagation of neural network policies, the Monte Carlo estimator provides an unbiased and robust estimate as it assesses returns until the trajectory's conclusion (avoiding instability induced by bootstrapping). However, a major drawback is the requirement for rollouts consistent with the policy, which can exhibit significant variance. For a more accurate return estimate, multiple rollouts are typically needed. A more efficient method employs return estimates provided by value-based methods, as discussed in the forthcoming section on actor-critic approaches.

Two supplementary comments to add. First, to prevent policies from becoming deterministic, an entropy regularizer is commonly added to the gradient. With this regularizer, the learned policy remains stochastic, ensuring continuous exploration of the policy. Second, instead of using the value function Q^{π_w} in Equation (13.18), one can employ the advantage function A^{π_w} . While $Q^{\pi_w}(s, a)$ summarizes the performance of each action given a state under policy π_w , the advantage function $A^{\pi_w}(s, a)$ offers a comparative metric of the expected rewards for each action in state s , given by $V^{\pi_w}(s)$. Using $A^{\pi_w}(s, a) - Q^{\pi_w}(s, a)V^{\pi_w}(s)$ is typically smaller than $Q^{\pi_w}(s, a)$, helping reduce variance in the gradient estimator $\nabla_w V^{\pi_w}(s_0)$ during policy improvement without altering expectations. In other words, the value function $V^{\pi_w}(s)$ can be viewed as a baseline or control variable for the gradient estimator. When updating a policy's neural network, using such a baseline enhances numerical efficiency — achieving a given performance with fewer updates — because the learning rate can be larger.

13.2.7 Deterministic Policy Gradient

The policy gradient method can be extended to deterministic policies. The Neural Fitted Q Iteration with Continuous Actions (NFQCA) introduced by Hafner and Riedmiller (2011) and the Deep Deterministic Policy Gradient (DDPG) algorithm introduced a direct

representation of policy that extends the NFQ and DQN algorithms to overcome the limitations of discrete actions.

Let's denote $\pi(s)$ as the deterministic gradient $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$. In a discrete action space, policies are iteratively built using

$$\pi_{k+1}(s) = \arg \max_{a \in \mathcal{A}} Q^{\pi_k}(s, a) \quad (13.19)$$

where π_k is the policy in the k th iteration. In continuous action spaces, greedy policy improvement can be problematic, requiring global maximization at every step. We represent the differentiable deterministic policy as $\pi_w(s)$. In this context, an appealing and computationally attractive alternative is to move the policy in the direction of the gradient of Q , giving rise to the DDPG algorithm:

$$\nabla_w V^{\pi_w}(s_0) = \mathbb{E}_{s \sim \rho^{\pi_w}} [\nabla_w(\pi_w) \nabla_a(Q^{\pi_w}(s, a)) | a = \pi_w(s)] \quad (13.20)$$

This equation implies a dependency on $\nabla_a(Q^{\pi_w}(s, a))$, typically necessitating the use of an actor-critic approach.

13.2.8 Actor-Critic Method

As seen above, for both deterministic and stochastic scenarios, policies represented by neural networks can be updated through gradient ascent. In both cases, the policy gradient usually demands an estimate of the value function under the current policy. A popular approach employs the actor-critic architecture, introduced by Konda and Tsitsiklis (2000), composed of two parts: the actor (the policy) and the critic (an estimator for value functions, such as the Q -value function). In deep RL, both actor and critic can be represented by nonlinear neural network function approximators (see Mnih *et al.* (2016)). The actor utilizes the gradient derived from the policy gradient theorem to adjust policy parameters w . The critic, parameterized by θ , estimates an approximation of the value function for the current policy $\pi : Q(s, a; \theta) \approx Q^\pi(s, a)$.

Critic: The simplest off-policy method to estimate the critic is from tuples $\langle s, a, r, s' \rangle$, possibly sourced from a replay memory, using a pure bootstrapping algorithm, $TD(0)$. In each iteration, the current

value $Q(s, a; \theta)$ is updated towards

$$Y_k^Q = r + \gamma Q(s', a = \pi(s'); \theta) \quad (13.21)$$

While simple, this approach isn't computationally efficient due to its reliance on a pure bootstrapping technique, which tends to be unstable and slow in back-propagating over time. Ideally, one would want a structure that

- is sample-efficient, leveraging on-policy and off-policy trajectories (i.e., utilizing a replay memory),
- is computationally efficient, benefiting from the stability and rapid reward propagation of on-policy methods derived from samples gathered from the same behavioral policy.

Actor: For stochastic cases, the off-policy gradient during the policy improvement phase, as seen in Equation (13.18), is

$$\nabla_w V^{\pi_w}(s_0) = \mathbb{E}_{s \sim \rho^{\pi_\beta}, a \sim \pi_\beta} [\nabla_\theta (\log \pi_w(s, a)) \nabla_a (Q^{\pi_w}(s, a))] \quad (13.22)$$

Here, θ is typically a behavior policy different from π , making the gradient generally biased. This method performs well in practice, but the biased policy gradient estimator makes convergence analysis challenging without the GLIE assumption.

For actor-critic methods, an on-policy approach has been studied using asynchronous methods for performing policy gradient without experience replay. Here, multiple agents operate in parallel, undergoing asynchronous training. This parallelism ensures each agent experiences different sections of the environment at given time steps. In this scenario, n -step returns can be used without introducing bias. This straightforward idea can be applied to any learning algorithm requiring on-policy data, eliminating the need for a replay buffer. However, this asynchronous method isn't sample-efficient.

Another approach is to combine on-policy and off-policy samples, striking a balance between the sample efficiency of off-policy methods and the stability of on-policy gradient estimates. For instance, Q -Prop, introduced by Gu *et al.* (2017), uses a Monte Carlo on-policy gradient estimator while also reducing the variance of the gradient estimator by employing an off-policy critic as a control variate. A limitation of Q -Prop is that it requires on-policy samples to estimate the policy gradient.

Summary: Policy gradient is an effective technique for improving policies in RL environments. As we’ve seen, it typically involves estimating the value function of the current policy, and an efficient approach is to use an actor–critic architecture that can handle off-policy data.

Unlike DQN-based methods discussed earlier, these algorithms possess the following characteristics: They can operate in continuous action spaces, which is particularly intriguing in applications like robotics where forces and torques can assume continuous values. They can represent stochastic policies, which is valuable for constructing policies that explicitly explore. This is also useful when the optimal policy is a stochastic one, as in multi-agent scenarios where the Nash equilibrium can be a stochastic policy.

However, an alternative is to integrate policy gradient methods directly with off-policy Q -learning, as proposed by O’Donoghue *et al.* (2016). In specific settings, depending on the used loss function and entropy regularization, value-based and policy-based methods prove equivalent, as posited by Haarnoja *et al.* (2017). For example, when adding entropy regularization, Equation (13.18) can be written as follows:

$$\nabla_w V^{\pi_w}(s_0) = \mathbb{E}_{s,a}[\nabla_w(\log \pi_w(s,a))(Q^{\pi_w}(s,a))] + \alpha \mathbb{E}_s \nabla_w H^{\pi_w}(s) \quad (13.23)$$

where $H^\pi(s) = -\sum_a \pi(s,a) \log \pi(s,a)$. Consequently, we observe that the following policy is optimal: $\pi_w(s,a) = \exp(A^{\pi_w}(s,a)/\alpha - H^{\pi_w}(s))$. We can thus derive an estimate of the advantage function using the policy: $\tilde{A}^{\pi_w}(s,a) = \alpha(\log \pi_w(s,a) + H^\pi(s))$. Therefore, all model-free methods can be viewed as different facets of the same approach. A remaining limitation is that both value-based and policy-based methods are model-free; they don’t utilize any environment model. The following will detail model-based algorithmic approaches.

13.2.9 Model-Based RL

Previously, we discussed model-free methods that depend on either value-based or policy-based techniques. In this section, we introduce model-based methods, which rely on an environment model (both

dynamics and reward function) and planning algorithms. Lastly, we explore the advantages of both model-based and model-free methods, and how the two can be integrated.

The environment model is either explicitly provided (like in the game of Go, where all rules are known *a priori*) or derived from experience. To learn such a model, function approximators prove especially advantageous in high-dimensional (possibly partially observable) environments (see methods by Duchesne *et al.* (2017) and Nagabandi *et al.* (2018)). The model can then serve as a proxy for the real environment. When the environment model is available, planning consists of interacting with the model to recommend actions. In discrete action scenarios, lookahead search is typically accomplished by generating potential trajectories. For continuous action spaces, trajectory optimization with multiple controllers can be employed.

Lookahead search: Lookahead search in an MDP iteratively constructs a decision tree where the current state is the root node. It stores the gains obtained at nodes and focuses on promising potential trajectories. The main challenge of sampling trajectories is balancing exploration and exploitation. On one hand, the aim of exploration is to gather more information on parts of the search tree that have been simulated less (i.e., sections with high expected variance). On the other, exploitation refines the expected values of the most promising actions. The Monte Carlo Tree Search (MCTS) technique, introduced by Browne *et al.* (2012), is a popular lookahead search method. It gained widespread recognition due to its impressive achievements in the challenging domain of computer Go, as detailed by Silver *et al.* (2016). The idea is to sample multiple trajectories from the current state until a termination condition is met (e.g., a given maximum depth) (refer to Figure 13.6). Based on these simulated steps, the MCTS algorithm then recommends an action. Recent efforts have developed direct end-to-end learning models of policies and how to best utilize them without relying on explicit tree search techniques. Compared to isolated methods (simply learning the model and relying on it during planning), these techniques showcase improved sample efficiency, performance, and robustness to model misspecification.

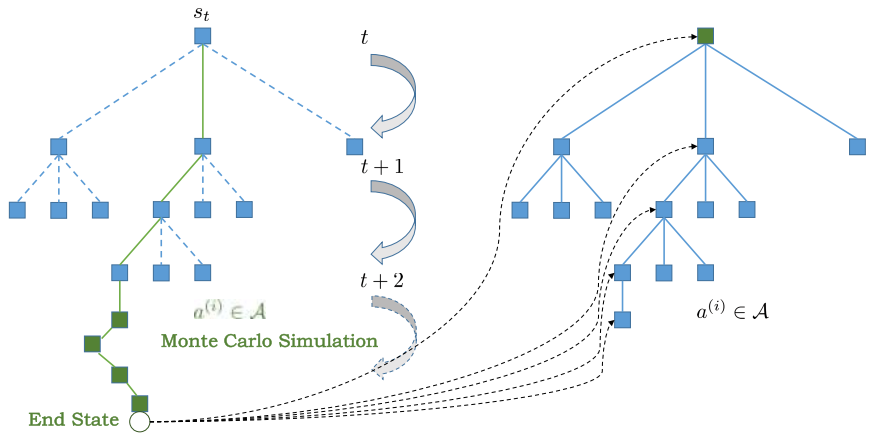


Figure 13.6. Illustration of how the MCTS algorithm performs Monte Carlo simulations and constructs a tree by updating statistics for various nodes. Based on the statistics gathered for the current node s_t , the MCTS algorithm selects an action to execute in the actual environment.

Trajectory optimization: While lookahead search techniques are confined to discrete actions, alternative techniques must be used for continuous actions. If the model is differentiable, we can compute analytical policy gradients by backpropagating rewards along the trajectory. For example, PILCO, introduced by Deisenroth and Rasmussen (2011), employs a Gaussian process to learn a probabilistic model of dynamics. It then explicitly uses uncertainty for planning and policy evaluation, achieving impressive sample efficiency. However, Gaussian processes still cannot reliably scale to high-dimensional problems.

One way to extend planning to higher dimensions is to leverage the generalization capacity of deep learning. For instance, Wahlström *et al.* (2015) proposed deep learning dynamics models (with autoencoders) and models in a latent state space. Model-predictive control can then find policies by repeatedly solving finite-horizon optimal control problems in this latent space. One can also build a probabilistic generative model in this latent space, aiming for it to have locally linear dynamics, which allows more efficient control execution. Another approach employs trajectory optimizers as teachers rather than demonstrators: Guided Policy Search (Levine and Koltun, 2013) uses sequences of actions suggested by another controller. It then

learns to adapt policies from these sequences. Trajectory optimization techniques have shown promise in many scenarios, such as in simulated 3D bipedal and quadrupedal creatures, as seen in the work of Mordatch *et al.* (2015).

Combining both approaches: The advantages of model-free and model-based approaches depend on various factors. First, the most appropriate method hinges on whether the agent has access to the environmental model. If not, learned models often carry inaccuracies that should be factored in. Note that through sharing neural network parameters, learned models can share hidden state representations with value-based methods.

Second, model-based methods necessitate collaboration with planning algorithms (or controllers), typically imposing high computational demands. Consequently, the time constraints of computing policy $\pi(s)$ — whether due to applications demanding real-time decisions or merely resource limitations — must be considered. Third, for some tasks, the structure of the policy (or value function) is more straightforward to learn, but for others, the model of the environment might be learned more effectively due to the task’s inherent structure (less complexity or more regularity).

Thus, the most effective approach depends on the structure of the model, strategy, and value function. Consider two examples for better clarity. In a fully observable maze, it’s evident how actions impact the next state, and the model’s dynamics can be succinctly summarized by the agent from just a few tuples (e.g., the agent being obstructed when attempting to navigate through maze walls). Once the model is known, planning algorithms can be employed efficiently. Now, consider a contrasting scenario where planning is more challenging: An agent needs to traverse a path where random events frequently occur. Suppose the optimal strategy is to move forward unless an object suddenly appears before the agent. Here, a model-free approach might easily capture the optimal strategy, while a model-based method might struggle, primarily because the model’s randomness results in numerous possible scenarios, even for a given sequence of actions.

We now outline how integrating learning and planning into an end-to-end training process yields an efficient algorithm in terms of performance (sample efficiency) and computational time, leveraging

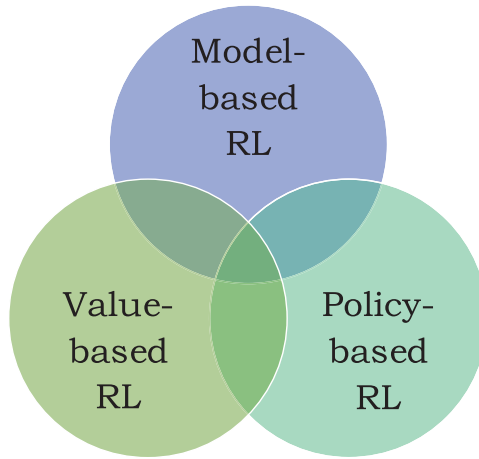


Figure 13.7. A Venn diagram of the space of common RL algorithms.

advantages from both aspects. Figure 13.7 illustrates a Venn diagram of different combinations.

When the model is available, a direct approach involves tree search techniques that utilize value and policy networks (e.g., as proposed by Silver *et al.* (2016)). When the model is unavailable and assuming the agent can only access a limited number of trajectories, a crucial attribute is a robust generalization algorithm. One possibility is constructing a model for generating additional samples for model-free RL algorithms. Another is using model-based techniques and controllers like MPC for basic tasks, refined with model-free fine-tuning for successful task execution.

Other methods develop neural network architectures, combining elements from both model-free and model-based methods. For instance, a value function can be amalgamated with steps back-propagated through the model (as proposed by Heess *et al.* (2015)). The VIN architecture (introduced by Tamar *et al.* (2016)) is a fully differentiable neural network featuring a planning module, which learns planning from a model-free objective (provided by a value function). It's suitable for planning reasoning tasks, like navigating from a starting to a target location, and has exhibited potent generalization across various domains.

Building on the same idea, the predictron aims to devise a more universally applicable algorithm, effective in the context of planning. It operates by implicitly learning an internal model in the abstract state space for policy evaluation. The predictron undergoes end-to-end training, deriving both direct rewards and value functions of multiple planning depths from the abstract state space. While the predictron architecture is confined to policy evaluation, this concept was extended to an algorithm that learns optimal strategies within the VPN architecture (proposed by Oh *et al.* (2017)). As VPN relies on n -step Q -learning, it necessitates on-policy data.

Merging model-free and model-based concepts is a pivotal research area for the future development of deep RL algorithms. Hence, we anticipate seeing more ingenious and intricate structures in this domain.

13.3 Distributed and Federated Reinforcement Learning

With technological advancements, agents now need to navigate an expansive state-policy space, requiring massive computational power or extended periods. If we can separate agents and environmental models across different machines for computation, distributed methods can address these challenges.

13.3.1 Distributed Reinforcement Learning

Most contemporary RL algorithms are distributed. OpenAI's Advantage Actor-Critic (A2C) algorithm is rooted in the conventional REINFORCE algorithm. In a standard A2C implementation, two processes are incessantly iterated: sample collection followed by a gradient update, using the standard formula, to produce a new policy. Due to computational constraints, a single-process A2C's sample collection efficiency is low.

A simple idea is to assign sample collection and model training to distinct modules. We introduce the concepts of Actor and Learner here. The Actor is responsible for interacting with the environment, producing training samples, while the Learner is in charge of model

training. DeepMind proposed the asynchronous advantage actor-critic architecture, abbreviated as A3C. In this framework, multiple Actor processes exist, each possessing a replica of the model to interact with the environment and produce training samples.

The Actor computes the gradient based on these samples and sends it to the Learner. The asynchrony here primarily lies in how the Learner utilizes the gradient. Upon receiving a gradient from a specific Actor, the Learner doesn't wait for gradients from other Actors but immediately updates the model, subsequently dispatching the latest model back to the original Actor. A3C achieves commendable results in games like Atari using only CPUs. However, as the model complexity grows, the required CPU resources substantially increase, leading to increased network overhead for transmitting gradients and model parameters. More critically, since A3C uses asynchronous updates, the gradients employed could be erroneous or outdated, limiting the further scalability of A3C.

In recent years, there hasn't been much change in the learners of distributed RL. PPO and IMPALA are two dominant techniques. The most significant innovation of IMPALA is the introduction of the V-trace algorithm, which offers corrections for off-policy phenomena. Within the IMPALA framework, each Actor has a model replica, sending training samples to the Learner. After updating the model, the Learner dispatches the new model to the Actor. Throughout this process, both Actor and Learner operate asynchronously. The Learner updates the model as soon as it receives training data, not waiting for all Actors, while the Actor continues sampling even during the Learner's model update, not waiting for the latest model.

Achieving exceptional results with policy gradient methods is challenging due to their sensitivity to step size choices. If the iteration step size is too small, training progresses at a snail's pace. However, if it's too large, the acquired signal gets severely disrupted by noise, leading to a dramatic performance drop. Additionally, these policy gradient methods are notoriously inefficient in terms of sample usage, requiring millions (or even billions) of time steps for mastering simple tasks. OpenAI introduced the Proximal Policy Optimization (PPO) algorithms to address these issues. PPO adeptly balances simplicity, sample complexity, and parameter tuning difficulty. It aims to compute an update at each iteration step to minimize the cost

function while ensuring a relatively small deviation from the prior policy.

Seeking further performance enhancement, major companies released frameworks for distributed RL, such as Google’s SEED RL and Menger, DeepMind’s Acme, Facebook’s Horizon, OpenAI’s Baselines, and UC Berkeley’s RLlib. These developments substantially bolstered the collaboration between hardware and software in distributed RL, maximizing hardware utility.

Contemporary RL has achieved remarkable outcomes across competitive domains. For instance, DeepMind’s AlphaGo algorithm defeated world champion Lee Sedol in the game of Go in 2016. OpenAI’s OpenAI Five demonstrated a competitive performance against professional teams in Dota2 matches. Moreover, in the StarCraft matches, DeepMind’s AlphaStar outperformed human players. RL increasingly showcases the potential to surpass human capabilities, heightening our anticipation for future advancements.

13.3.2 Federated Reinforcement Learning

Federated RL has a wide array of applications. For instance, in manufacturing, product creation might involve different factories, each responsible for different parts. Factories’ decision strategies are confidential, and there’s no inter-sharing. On the other hand, due to business constraints and the lack of reward models (for some factories), developing individual high-quality decision policies is often challenging. Therefore, federated learning of decision strategies can be beneficial without disclosing private data. Another example is constructing medical treatment strategies for patients in hospitals. Patients might receive treatments in various hospitals but may not provide feedback, meaning these hospitals cannot collect reward models based on patient outcomes or formulate treatment strategies. Additionally, patient data records are confidential, probably not shared across hospitals. Thus, federated learning of hospital treatment strategies becomes crucial.

Following the traditional categorization of data distribution and feature distribution in federated learning, we can classify federated RL into Horizontal Federated Reinforcement Learning (HFRL) and Vertical Federated Reinforcement Learning (VFRL).

Conventional distributed RL is parallel RL, where we assume multiple agents execute identical tasks (with corresponding rewards for states and actions). Agents might learn in diverse environments. It's worth noting that most parallel RL setups involve transferring agent experience memory or gradients. However, in practice, these data are sensitive, so direct data transfer is implausible. Researchers have thus proposed employing the HFRL approach to address these challenges, maintaining the parallel RL setting while ensuring protection of the original data and its gradient. Figure 13.8 illustrates a generic HFRL framework.

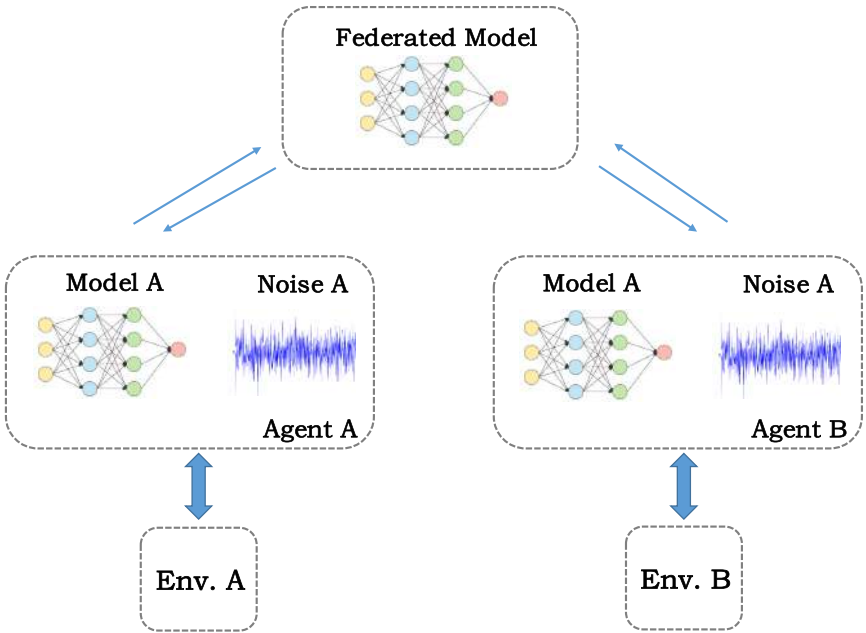


Figure 13.8. General HFRL framework. All RL agents locally train their models without exchanging individual experience data, model parameters, or gradients. Once trained, agents send encrypted model parameters to a server. The federated model at the server then updates its parameters by utilizing the received model parameters. The server subsequently dispatches the updated federated model to each agent. Agents then use these federated model parameters to update their local models. Here, encryption can adopt Gaussian differential privacy methods.

Researchers from Yonsei University have introduced a technically privacy-preserving distributed RL framework, dubbed Federated Reinforcement Distillation (FRD). The core idea is to exchange proxy experience memory composed of a set of predetermined states and time-averaged policies, thereby technically safeguarding the privacy of actual experiences. Built upon an A2C RL architecture, they numerically evaluated the efficacy of FRD, investigating how the structure of proxy memory and various memory exchange rules affect FRD's performance. Their procedure is as follows: (i) Each local experience memory is uploaded to a server, (ii) a global experience memory is constructed on the server, and (iii) each agent downloads and replays the global experience memory to train its local neural network model. However, FRD has certain limitations, such as coarse proxy states and questions surrounding the frequency of proxy experience memory exchanges within neural network architectures. Consequently, they proposed an enhanced version of FRD named MixFRD, which employs a mixed data augmentation algorithm to interpolate the globally collected proxy replay memories. Experiments demonstrated MixFRD's effectiveness in terms of task completion time and communication payload size.

VFRL aims to address a scenario where, for a single environmental model, multiple agents might be required to learn diverse strategies, or differing observations of the same environment. In this type of federated RL, only one agent possesses actions, states, and rewards, while the others only have actions and states. As this RL type is characterized by feature distribution, it's termed as VFRL. The main objective of this technique is to leverage the diverse knowledge learned by different agents within the same environment, training a more comprehensive agent. Throughout the training, any direct data transmission is prohibited. Figure 13.9 illustrates a possible framework for VFRL.

This approach enables each agent to learn a private Q -network strategy by sharing limited information (i.e., Q -network outputs) among agents. Information is encoded before sending and decoded upon receipt. All agents benefit from collaborative learning, building policy for decision-making.

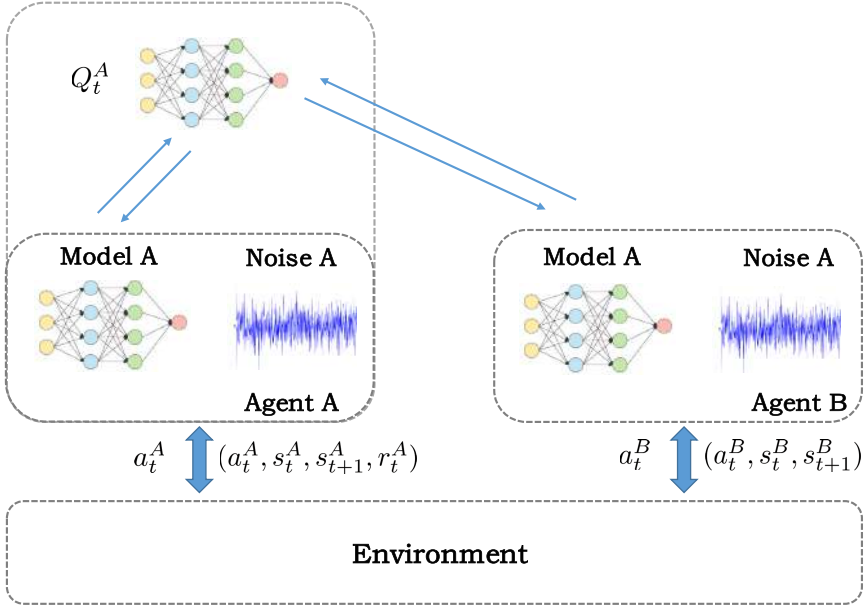


Figure 13.9. The generic framework of VFRL. Initially, each agent gathers Q -network output values from other agents, encrypted using Gaussian differential privacy. Moreover, it establishes a shared value network, such as an Multi-Layer Perceptron (MLP), using its own Q -network output and the encrypted values to compute the global Q -network output. Finally, updates are made to the shared value network and the agent's own Q -network based on the global Q -network output. Note that the MLP is shared among agents, and agents possessing a Q -network remain unknown to others, preventing any inferences based on the shared encrypted Q -network outputs during training.

13.4 Conclusion

Deep RL, a significant technique in the AI realm, has always been a pivotal method for addressing various strategy games. In this era prioritizing data privacy, the experiences, strategies, and data of individual agents have also become crucial privacy considerations. As technology continually advances, federated RL is bound to be an integral part of future innovations. Current federated RL faces challenges; for instance, some methods rely on model parameter exchange or Gaussian differential privacy for encryption. These methods can be unreliable when confronted with malicious attackers or

untrusted environments. Therefore, adopting more robust encryption techniques like secure multi-party computation and homomorphic encryption might be beneficial. Additionally, exploring new structural paradigms for federated RL, such as decentralization, can potentially accelerate training.

This page intentionally left blank

Part III

Federated Learning Systems

This page intentionally left blank

Chapter 14

Detailed Exploration of the FedLearn Federated Learning System

14.1 Open-Source Federated Learning Systems and Their Pain Points

Regarding federated learning systems, globally, there are many excellent open-source platforms addressing various challenges in federated learning. Renowned systems include FATE, TensorFlow Federated, Fedlearner, FedML, and pySyft. Some of these platforms focus on real-world industrial implementations of federated learning tasks, while others target the development of cutting-edge federated learning algorithms, each with its unique strengths.

14.1.1 Programming Language and Environment

Presently, nearly all open-source federated learning platforms are developed using Python. With its ease of use and extensibility, Python has cemented its indispensable role in building federated learning platforms. However, it's worth noting that in domestic online services, Java and its derivative service frameworks (e.g., Spring Boot, Apache Tomcat, and Redis) still dominate. (Refer to Figures 14.1 and 14.2). This landscape suggests that most companies' engineering environments are adapted to Java-based systems. Forcefully deploying a primarily Python framework in such an environment requires careful consideration of its impact on existing services and a significant engineering effort. Therefore, we believe a

Usage of programming languages by professional developers

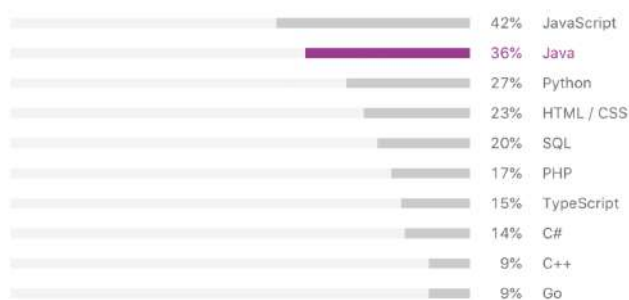


Figure 14.1. Language usage among professional developers in 2020.
Source: <https://blog.jetbrains.com/idea/2020/09/a-picture-of-java-in-2020/>.

Usage of Java by professional developers

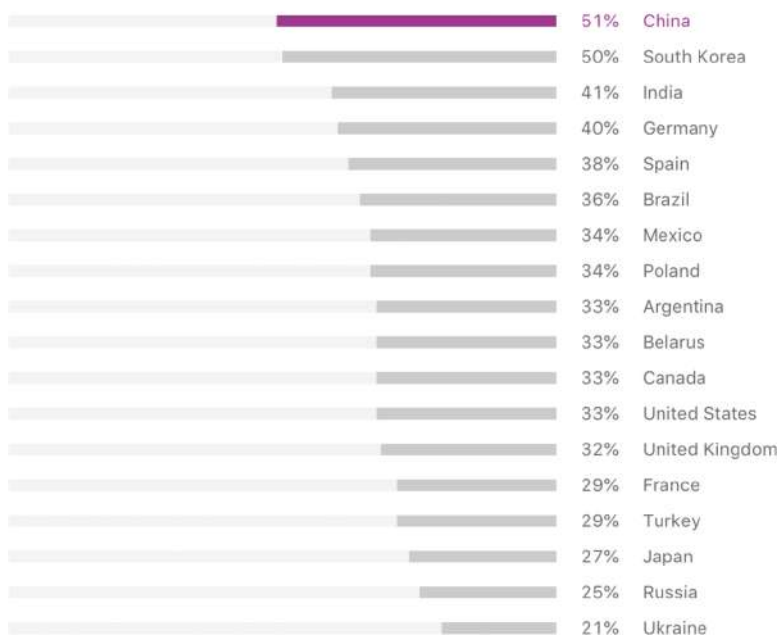


Figure 14.2. Proportion of Java use by professional developers in various countries in 2020.
Source: <https://blog.jetbrains.com/idea/2020/09/a-picture-of-java-in-2020/>.

native Java-based federated learning framework can assist companies in deploying and using federated learning services more effectively.

With these considerations, JD Technology designed a comprehensive federated learning system from the ground up — the FedLearn platform. At a systemic level, the FedLearn platform supports pure Java deployment and features a one-click deployment function. Moreover, FedLearn also offers solutions for Java+Python (Docker) containerized deployment to cater to the computational requirements of federated (deep) learning. Within this framework, the platform integrates various federated learning algorithms. For vertical federated learning, the platform encompasses both traditional machine learning algorithms and federated deep learning. For horizontal federated learning, the platform supports standard algorithms like FedAvg and has adapted to multiple popular deep learning frameworks, including TensorFlow, Pytorch, and Oneflow.

14.1.2 Big Data and Computational Efficiency

In modern machine learning tasks, big data is essentially a “standard feature”. Whether for traditional or deep learning models, training typically involves millions, tens of millions, or even billions of samples. Thus, effectively harnessing large-scale data is paramount. Federated learning, a branch of machine learning, is no exception. In practical applications, datasets usually scale to millions or more. Moreover, the unique demands of federated learning mean we often perform ciphertext calculations instead of plaintext, and ciphertext calculations are notably more complex. As a result, federated learning tasks demand significantly more computational power.

To address these challenges, the FedLearn platform has developed a distributed computing solution based on MapReduce. This solution is adaptable as the main computing engine in the current pure Java deployment environment and offers scalability to integrate with next-generation distributed computing engines, like Spark, catering to the high computational and data demands of federated learning. In real-world scenarios, the FedLearn platform has successfully and stably trained and inferred from datasets scaling to millions or tens of millions.

14.2 Advantages of the FedLearn Federated Learning System

Given the increasing emphasis on privacy protections domestically and internationally, and with the formulation and execution of relevant regulations and laws, federated learning becomes a pivotal technological direction. The challenge is to provide premium services to users while safeguarding their privacy, all in adherence to regional legal and regulatory requirements.

Considering the unique needs of JD Technology, which involve catering not only to internal use cases but also external commercial projects spanning risk control, government enterprises, finance, and more, specific commercial case considerations emerge. First, JD Technology's federated learning system must support a modeling platform dominated by JD Cloud that encompasses multiple partnering enterprises for collective modeling. Additionally, the system should support private deployment to facilitate collaborations led by governments, enhancing their oversight on business operations. Data interactions must be interpretable, verifiable, and assessable. Hence, to robustly cater to various business scenarios and meet multifaceted requirements, we need to devise a federated learning system distinctive to JD Technology, where there's a heightened demand for deployment agility, system compatibility, algorithm versatility, and data security.

JD Technology's self-developed federated learning platform is grounded in the philosophy of building core foundational capabilities in-house. Tailored to its unique use-cases, the platform has been meticulously designed from the ground up. It encompasses a comprehensive suite of systems and introduces a set of innovative, proprietary federated learning algorithms.

On the system level, Java is the language of choice. Based on this foundation, an entire mature system architecture has been developed. This architecture supports a wide range of federated learning models and offers a seamless one-click deployment. Structurally, the framework is divided into two main components: the Data Provider (Client) and the Data Aggregator (Master). Every algorithm is correspondingly bifurcated into these Client and Master components, enabling local data processing and cloud integration. Typically, the Client bears a more substantial load. It fetches data from local

databases and manages the local portion of the federated learning algorithms. At present, the Client consists of layers dedicated to data, computation, encryption, and networking. The networking layer facilitates asynchronous interactions, significantly enhancing computational and communication efficiency, thus allowing for full parallelism of computations and data transmission. To further ensure the immutability of data exchanges, blockchain technology has been integrated, mandating all data interactions to occur on-chain.

In the encryption layer, the system employs and refines a series of homomorphic and semi-homomorphic encryption algorithms. Ensuring data security, these methods also leverage parallel computing and CPU acceleration to optimize encryption computations. Algorithmically, the Client offers a robust map-reduce mechanism, enabling stable distributed execution and bypassing memory limitations associated with training data. A significant challenge for the Master component is to efficiently perform arithmetic operations under homomorphic encryption and to provide egalitarian training options. In scenarios where every training participant desires equal standing, ensuring no single aggregator dominates and possibly infringes on data privacy, a blockchain-based voting protocol has been established. This protocol ensures a randomized selection of the aggregator from all participants, safeguarding equal chances of representation across numerous training and inference rounds.

Algorithmically, besides offering traditional SecureBoost and federated linear methods, the platform introduces proprietary federated random forest, mixed XGB, and mixed linear federated learning algorithms. The distributed federated random forest model effortlessly surpasses memory constraints. And with the efficient processing of vast data volumes, unparalleled training precision becomes attainable. Using mixed algorithms, users can feed as much data as available. The algorithm, in turn, holistically determines the best regression approach, obviating the need for users to preprocess data or decide on vertical or horizontal federated learning. This substantially lowers the entry barrier for users.

The entire federated learning system is concise with minimal third-party dependencies. Both the Client and Master components can be deployed with a single click, making it ideal for private deployments and multi-party collaborative modeling. Simplified deployment, combined with the JVM's multi-OS compatibility, ensures

flexibility in hardware and software platform requirements, making deployment more cost-effective. The natural adaptability of various algorithms to multi-party collaborative modeling, supported by JD Technology’s robust technical foundation, paves the way for a multi-stakeholder data modeling ecosystem.

14.3 FedLearn System Architecture Design

14.3.1 Common Federated Learning System Architectures

As a distributed machine learning system with multi-party participation, typical federated learning system architectures include peer-to-peer, star, and hierarchical structures.

- *Peer-to-Peer*: Here, clients connect directly. This approach is efficient when there are few clients, especially in bi-party modeling scenarios where resource consumption is minimal. However, as the number of modeling clients increases, system interactions can exponentially grow, rendering the system unviable. Two-party and multi-party peer-to-peer interactions are illustrated in Figure 14.3.

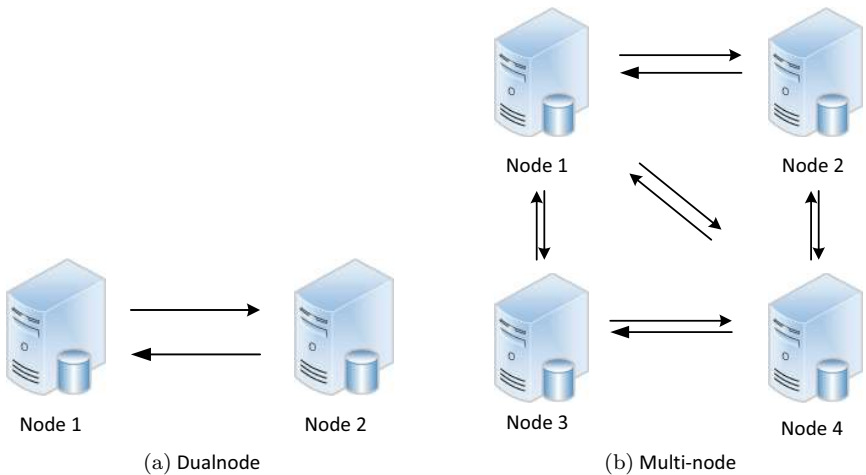


Figure 14.3. Peer-to-peer architecture.

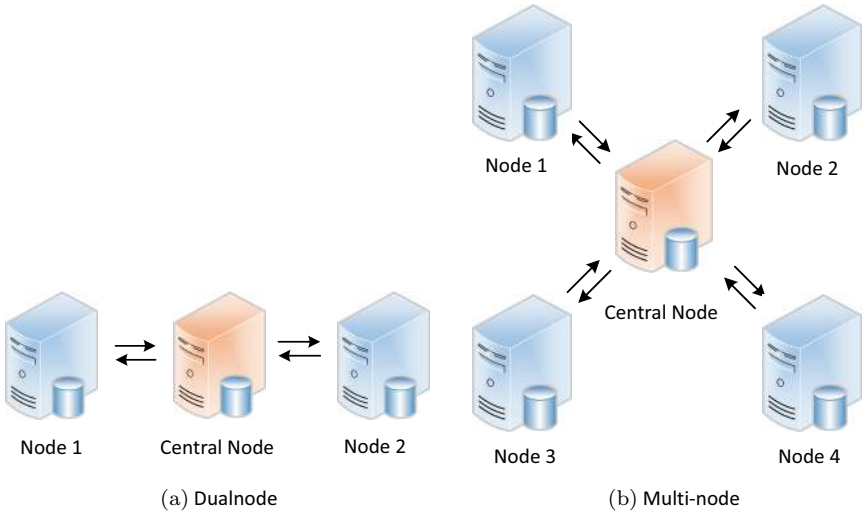


Figure 14.4. Star architecture.

□ *Star*: In this setup, there’s a centralized node responsible for data aggregation and message broadcasting. For two-party modeling scenarios, a central node still acts as a mediator, which can be cumbersome. But for multi-party collaborative modeling, the star architecture simplifies processes as each client interacts only with the central node. This is depicted in Figure 14.4.

14.3.2 Overview of FedLearn Architecture

Figure 14.5 illustrates the overall architecture of the FedLearn platform. At the system level, the primary components of the architecture can be categorized into the Coordinator and the Data and Computation Providers (i.e., Client and its subordinate Worker). The Coordinator is responsible for scheduling the algorithm, while the Data and Computation Providers handle specific data extraction, processing, and federated learning computations. The latter can be further divided into data, computation, encryption, and network layers. In the data layer, users can choose to integrate their SQL databases, traditional file systems, or distributed data storage like HDFS. In the computation layer, the FedLearn platform offers parallel computation extensions for various algorithms and

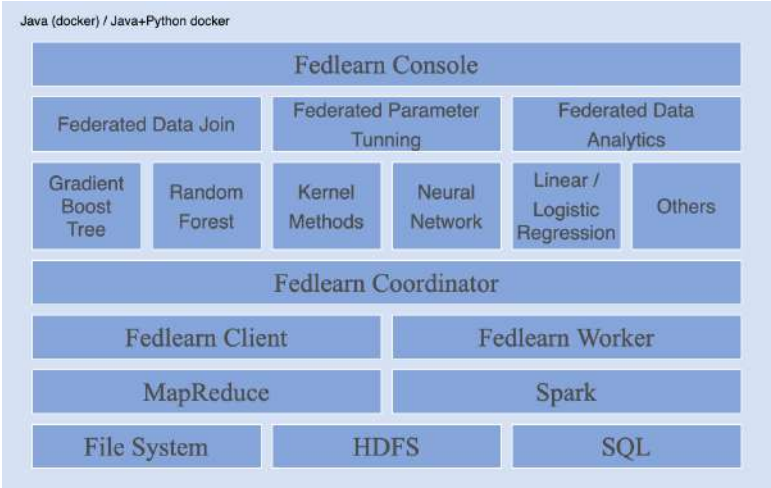


Figure 14.5. FedLearn architecture diagram.

distributed computing solutions for some highly parallelizable algorithms, like federated random forests. Considering the requirements of federated learning scenarios, the platform provides diverse encryption solutions, such as Secure Multi-Party Computation (SMPC) and homomorphic encryption. The encryption module has also been substantially improved, boosting encryption computation efficiency by over a hundredfold. In the network layer, FedLearn supports asynchronous interaction, significantly enhancing computational and network communication efficiency, thus improving federated learning modeling efficiency.

At the algorithmic level, the FedLearn platform not only provides traditional federated learning models like SecureBoost and federated linear models but also offers proprietary (distributed) federated random forest algorithms, federated kernel methods, federated deep learning algorithms, and various horizontal/hybrid federated learning approaches. These algorithms excel in different scenarios and data sizes, allowing users to select the most suitable federated learning method for modeling. Moreover, to reduce the complexity of federated learning modeling, FedLearn also offers auxiliary modeling tools. For instance, the federated data analysis module, while technically ensuring no privacy leaks, provides modelers with data insights, such as feature importance and correlation, visualizing these insights for

users. Additionally, a federated auto-tuning module suggests optimal parameters for the current training task based on past modeling results with the dataset.

14.3.3 FedLearn Standard Architecture Features

To support both bipartite and multiparty joint modeling scenarios, as well as features like private inference and secure multi-party computation, we designed a hybrid architecture. Our system divides into two main parts: the Coordinator and the Client:

- ❑ The Coordinator, deployed on any participating party or a neutral node, acts as the centralized node responsible for overall scheduling, broadcasting messages, and displaying and storing metadata. It also provides a unified logic for the interface and API, ensuring it never accesses users' raw data.
- ❑ The Client is deployed within each participant's internal network machine and is mainly responsible for data parsing and loading, local model training, intermediate data exchanges, and offering local convenience operations, such as dataset configurations.
- ❑ For bipartite modeling scenarios, we have made specific adjustments. The active participant's client possesses coordinating capabilities, responsible for initiating tasks and maintaining system status and operation. The passive participant's client utilizes a standard client setup.

The entire federated learning system structure is shown in Figure 14.6. It consists of an interface system, a Coordinator, and multiple Clients. Users can create and join tasks on the interface; system, reaching a consensus for joint training; set parameters to start training and view training metrics; perform simple inferences on the interface; or utilize the inference API to serve production systems.

Our federated learning system adopts a modular and hierarchical design, structured from the bottom up into data, scheduling, core, algorithm, and user interface layers, as depicted in Figure 14.7:

- ❑ The data layer primarily focuses on diverse data loading and storage capabilities, supporting sources, such as local files, MySQL, HDFS, and HTTP interfaces. It also facilitates model persistence, storing, and writing.

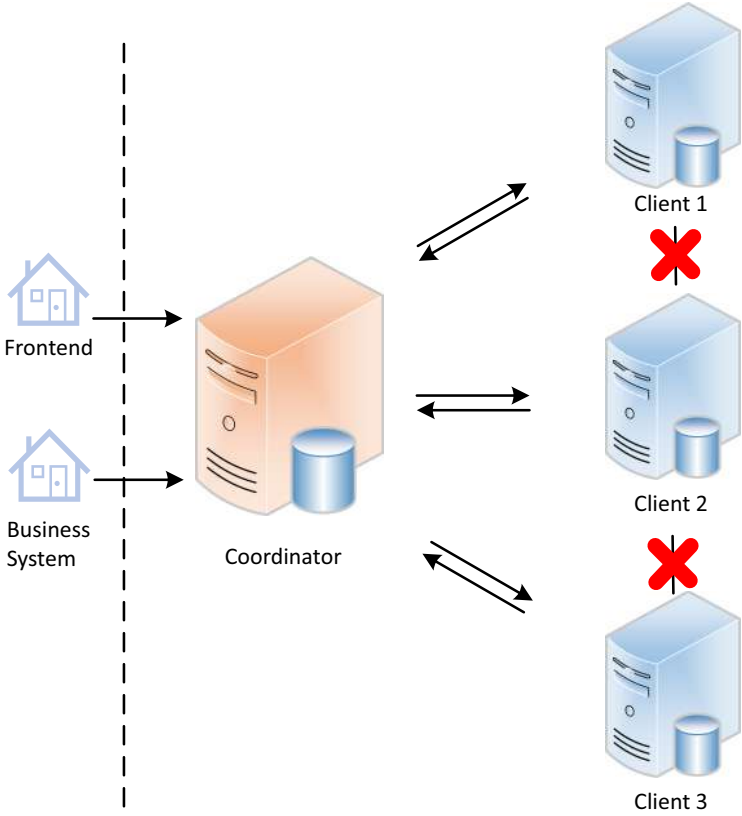


Figure 14.6. Standard FedLearn framework structure.

- ❑ The scheduling layer handles system scheduling and message exchanges between the Coordinator and various Clients.
- ❑ The core layer offers essential components for algorithmic design and can be managed by system scheduling.
- ❑ The algorithm layer encompasses intersection operations and the implementation of various federated learning algorithms.
- ❑ The user interface layer offers external calling interfaces, allowing users to input necessary parameters for model training and inference.

From the perspective of the operational process, the system is divided into several steps: consensus-building, training, and inference, as illustrated in Figure 14.8.

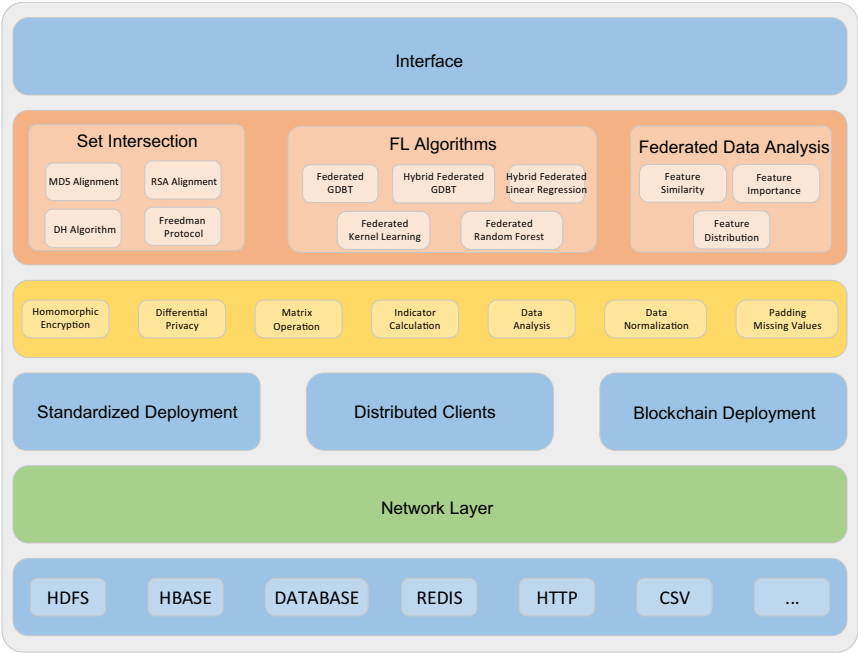


Figure 14.7. FedLearn component architecture.

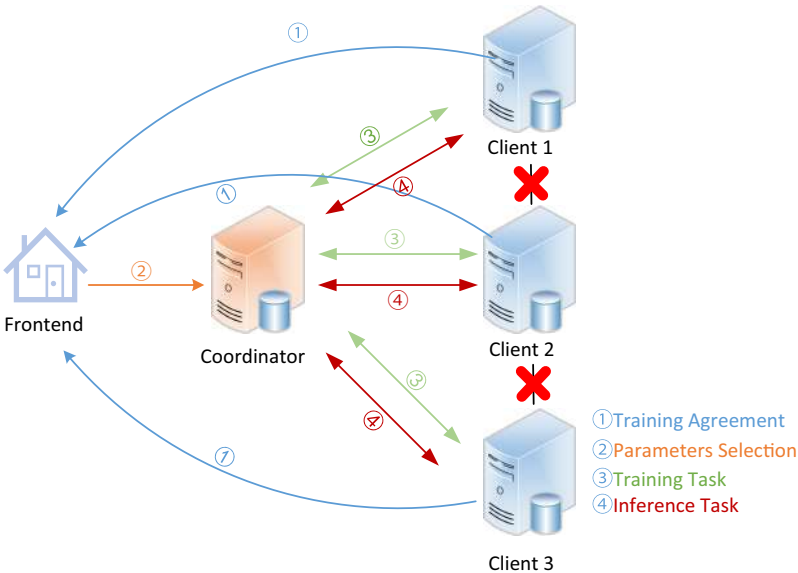


Figure 14.8. FedLearn training workflow.

1. **Consensus-building:** All participants agree on training objectives, the samples and features they provide. There's no need to disclose the actual features or field names; they can manually specify feature meanings to achieve consensus.
2. **ID alignment:** A privacy-preserving set intersection operation is used to align the users of vertical federated learning. Those who do not require alignment can skip this step.
3. **Parameter setting:** Users select algorithms on the frontend and set training parameters. There's also an option to use auto-tuning.
4. **Training and metric output:** During training, various metrics are updated, including precision, recall, AUC, F1, RMSE, and more.
5. **Model storage and inference:** Operations include model storage and retrieval, and inference using input users and features.

From the perspective of resource management and scheduling, both the Coordinator and Client have comprehensive scheduling and resource allocation systems. Each task possesses a distinct ID and resources, operating independently. The failure of a single task does not affect the entire system. Further details can be seen in Figure 14.9:

- ❑ Initially, when training or inference tasks are submitted to the coordinator, it allocates CPU and memory resources either based on user specification or default parameters.
- ❑ Once the task is initialized by the Coordinator, based on the algorithm's automation design, it requests resources from the Clients. Upon receiving the request, Clients similarly allocate necessary resources.
- ❑ If a task is paused, the system will serialize its corresponding data, releasing its resources and retaining only the task ID.
- ❑ Should a task be forcibly terminated, the system will erase all related information for that task.
- ❑ Upon the successful completion of a task, resource de-allocation requests are sent to both the Client and Coordinator.

14.3.4 Distributed Federated Learning

For situations where the data volume is too large and cannot be loaded on a single client, resulting in slow or impossible training,

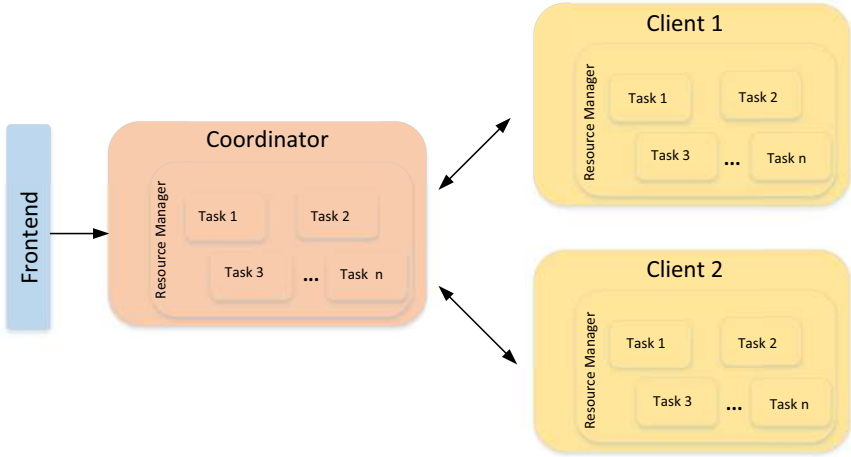


Figure 14.9. FedLearn resource management system.

we have designed distributed federated learning, transforming each client into a distributed system.

The distributed client follows the MapReduce design, consisting of a Manager unit and several Worker units. Refer to Figure 14.10, where a standard client is replaced by a distributed client.

- ❑ The Manager unit externally offers services consistent with a single client and internally handles scheduling for the entire distributed system, leading task decomposition and assignment.
- ❑ The Worker units load data segments as needed and complete the computing tasks assigned by the Manager, returning results accordingly.

In terms of system design, the distributed client’s overall design philosophy is inspired by the MapReduce framework. It is divided into Initialization, Computation, Aggregation, Termination, and other operations:

- ❑ Initialization (INIT Module) is for task analysis and DAG graph construction, facilitating subsequent task computations.
- ❑ Computation (MAP Module) is for distributed task calculations.
- ❑ Aggregation (REDUCE Module) integrates computational results.

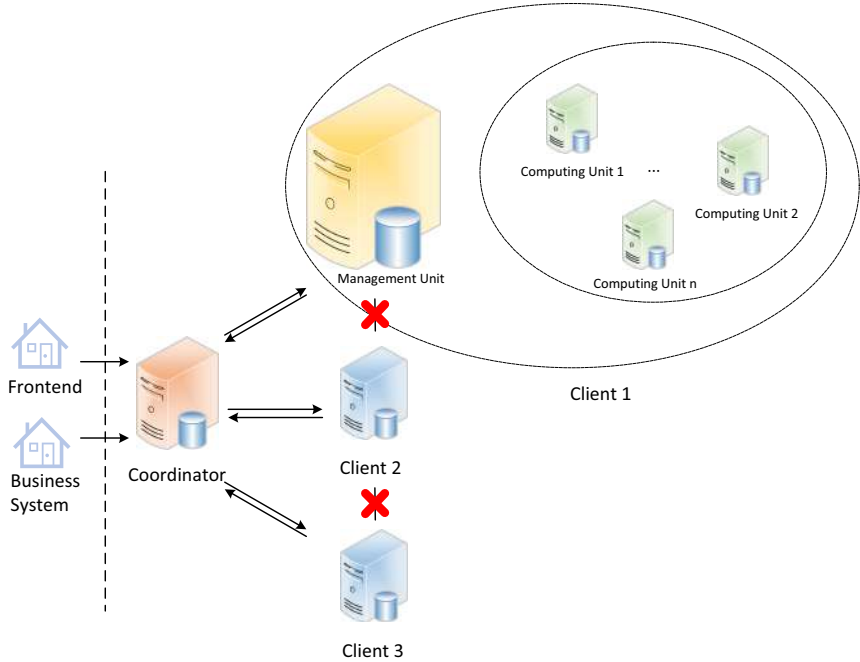


Figure 14.10. FedLearn distributed client architecture.

- ❑ Termination (FINISH Module) includes obtaining final results, cache cleaning, returning task status, and task outcomes.
- ❑ Miscellaneous (SINGLE Module) is to ensure compatibility with non-MapReduce tasks, supporting computations for modules without MapReduce functionality.

Through these five operations, basic distributed system operation is supported. Moreover, operations not yet modified for distributed processing are supported in the Miscellaneous (SINGLE Module). For more details, see Figure 14.11, and for the computation graph details, refer to Figure 14.12.

In addition, the distributed client also supports embedding within the Spark environment. An intermediate layer translates requests into Spark commands, which are then executed within the Spark system. Upon completion, results are translated back into the format of the federated learning system and returned to the coordinator. See Figure 14.13 for reference.

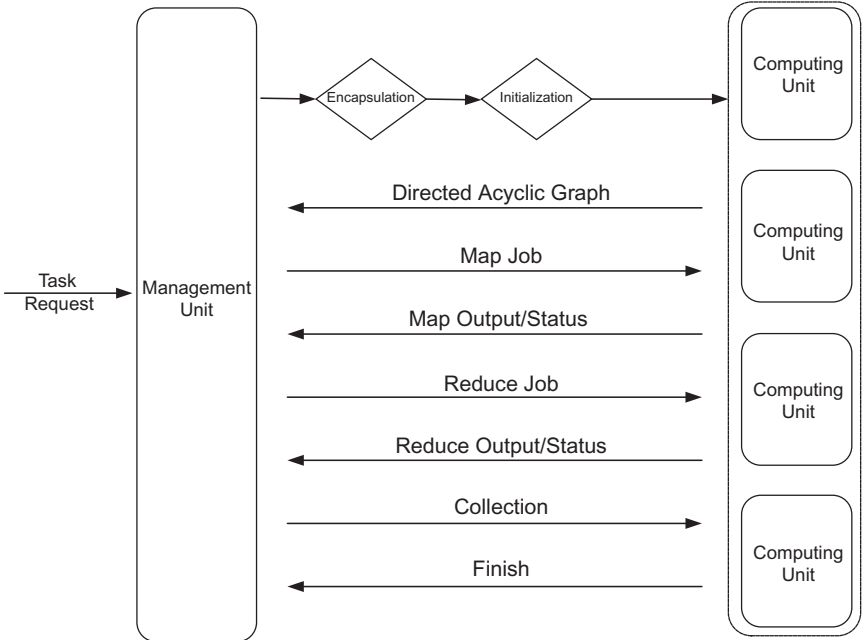


Figure 14.11. FedLearn distributed client computational workflow.

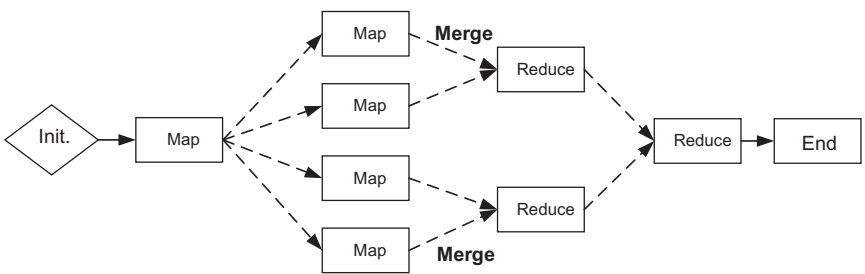


Figure 14.12. FedLearn distributed client computation diagram.

14.3.5 Blockchain Federated Learning Architecture

Based on the standard architecture, we’ve designed a blockchain version of federated learning by integrating blockchain technology at its core. By leveraging blockchain,

- unauthorized nodes can be prevented from joining via blockchain’s access control mechanism,

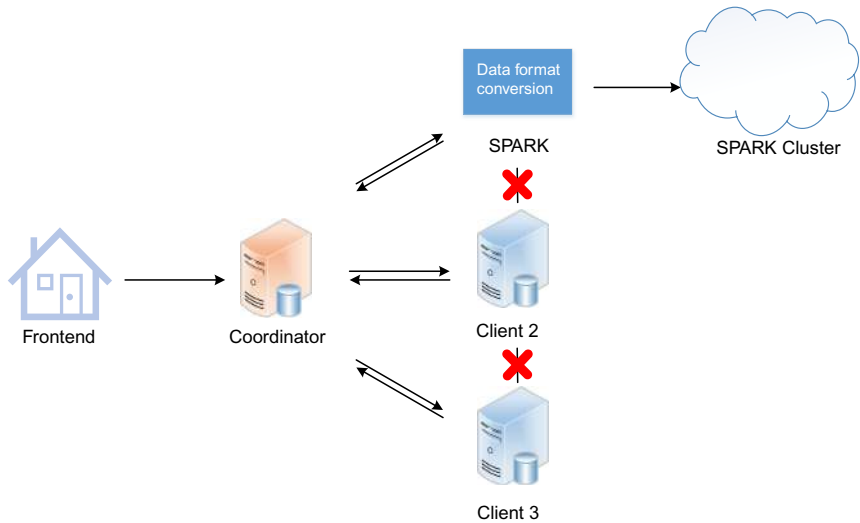


Figure 14.13. FedLearn spark client schematic.

- ❑ the coordinator can be randomly located on any node, eliminating the need for intermediary data storage on any single entity,
- ❑ furthermore, blockchain’s ledger system can fully record the history of data exchanges, facilitating traceability.

The federated learning system built upon blockchain is architecturally intricate. Every federated learning node relies on a blockchain node. The system architecture is illustrated in Figure 14.14, which encompasses a user interface, a coordinator, and several clients. Users can create or join tasks on the interface to achieve joint training consensus. The system flow is depicted in Figure 14.15, where the coordinator is randomly elected in each iteration.

14.4 Cross-Language Support in FedLearn

One notable feature of the FedLearn platform is its Java and Python joint deployment solution based on Docker. While integrating with the Java environment, it can also harness powerful third-party libraries in Python, enhancing federated learning model development efficiency and supporting federated deep learning. Within this joint

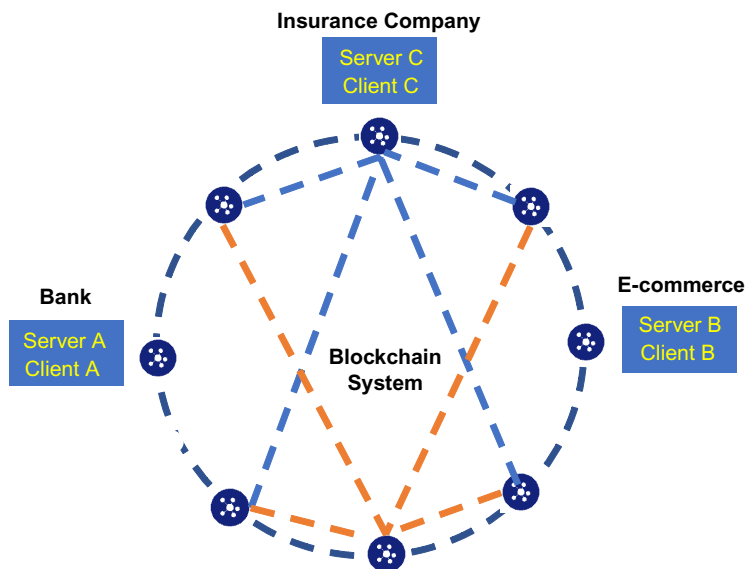


Figure 14.14. FedLearn blockchain federated learning system architecture.

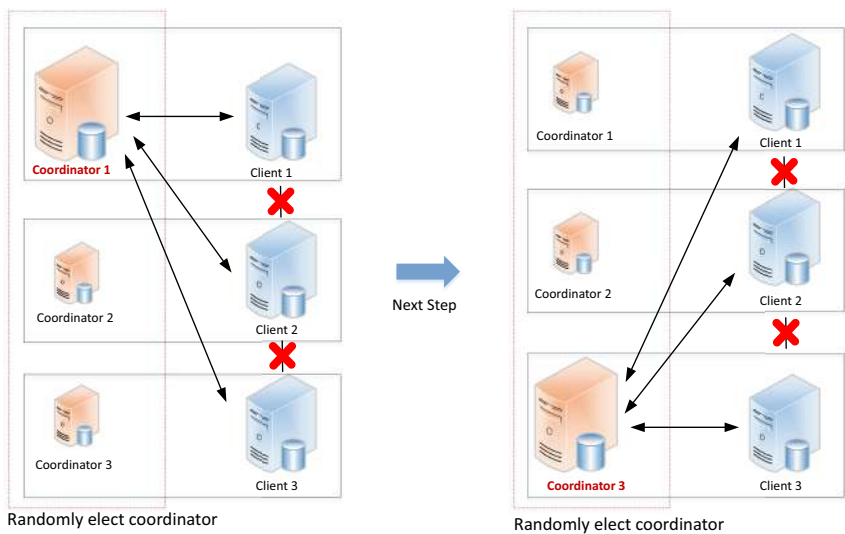


Figure 14.15. FedLearn blockchain federated learning system workflow.

deployment framework, a pivotal concern is addressing communication between Java and Python. We'll delve deeper into how FedLearn employs RPC technology to tackle this challenge.

14.4.1 Introduction to Remote Procedure Call Technology

Remote procedure call, commonly referred to as RPC, is a widely used technology in large-scale projects. RPC conveniently defines the procedure of function execution in various environments. Through standardized information transfer, function interface, request sequencing, and secure access, it facilitates invoking functions in remote environments just as if they were defined locally. This method allows the decomposing and decoupling of large-scale projects into standardized modules, streamlining development. Moreover, RPC technology, with its unified interface, enables these modules to interconnect rapidly and seamlessly, resulting in a cohesive project functionality.

14.4.1.1 Calling a local function

Before delving into RPC, let's consider a simple example illustrating the process of invoking a function defined locally. The following Python code defines a straightforward function, `Add()`, to compute the sum of two numbers. We intend to invoke this function to determine the sum of two variables, `a` and `b`:

```
def Add(x, y) {  
    answer = x + y  
    return answer  
}  
  
a = 10  
b = 20  
a_plus_b = Add(a, b)
```

When the `Add()` function is defined locally, the computer only needs to perform the following five steps in sequence to call and execute the `Add()` function when the program runs to compute `a_plus_b`:

1. **Variable push:** Push values of variables `a` and `b` to the stack. In this example, the values of `a` and `b` are 10 and 20, respectively.

2. **Parameter retrieval:** Enter the `Add()` function, extract values from the stack, and assign them to temporary variables `x` and `y`.
3. **Computation:** Calculate $x + y$ and store the result in the answer variable.
4. **Stack return:** Push the value of answer to the stack and then end the `Add()` function and return.
5. **Assign result:** Retrieve the returned value of 30 from the stack and assign it to `a_plus_b`.

This example is a basic function call paradigm. However, the essence of local function calls, no matter how complex the process, lies within this 5-step framework. In reality, the function call process can involve various optimizations. For instance, when implemented in C++ or Java, compilers can store parameters and return values with fewer variables directly in registers, eliminating the need for push and pop operations. They might also inline the function to enhance execution efficiency, but a deep dive into these details is beyond the scope of this discussion.

14.4.1.2 Remote procedure calls

Local functions can be perceived as the local node executing the function content. In contrast, remote function calls are like requesting another node to provide a specified service. The term “remote” indicates that the execution of the corresponding function will take place in the server’s remote environment, with the inherent characteristic that the “function body execution is invisible locally”.

How does the process of a RPC differ from that of a local call? Let’s imagine a remote procedure, `RemoteAdd()`, which performs the same task as `Add()` but within an RPC environment. When remotely invoking the `RemoteAdd()` function in an RPC environment, it involves two distinct entities: the client (which holds the parameters and requests the calculation) and the server (which performs the computation). The client’s workflow comprises the following steps:

1. **Function synchronization:** Initially, the client notifies the server about which function it needs to call. Here, a mapping exists between the function and a process ID. During remote invocation,

the client checks this function, mapping the call to a predefined Call ID on the server.

2. **Parameter preparation:** After confirming the target function, the client must send its local parameters to the remote environment. Unlike local calls, where a direct stack push is sufficient, remote calls do not share the same memory space. Hence, the client needs to convert parameters into a byte stream and send it to the server, which then translates this byte stream into its readable format. This involves a serialization and deserialization process, encapsulating the Call ID and input parameters into a binary package.
3. **Data transfer:** With data ready, how is it transferred? This step employs a network transport layer to send the serialized parameters and Call ID to the server. A common method is the HTTP2 protocol, which sends the data package from Step 2 to the designated ServerAddress.
4. **Await computation:** The client waits for the server to compute and return the results.
5. **Assign result:** If the server call succeeds, the result is deserialized, completing the assignment and updating of `a_plus_b`.

On the other side, the server, possessing the function body, undergoes the following process:

1. **Await request:** The server awaits a client request.
2. **Request parsing:** Upon receipt of a request, the server deserializes the data package to obtain the Call ID and parameters.
3. **Function retrieval:** By consulting a locally maintained hash map (`call_id_map`), which maps a string representation of Call ID to function pointers, the server identifies the corresponding function.
4. **Parameter retrieval:** Parameters for the function `a` and `b` are obtained by deserializing the params.
5. **Computation:** The server locally invokes the function pointed to by the Call ID and executes its code. In this example, the function should emulate `Add()` and produce the result.
6. **Result transmission:** The computed result, after serialization, is transmitted back to the client via the TCP layer.

14.4.1.3 New challenges with remote procedure calls

During remote invocation, the function body executes on a remote machine. Comparing the local call process, it's evident that remote function calls introduce several new challenges:

Call ID mapping: How do we convey to the remote machine that we want to invoke `RemoteAdd()` and not some other function? In local calls, the function body is directly specified using function pointers, and the compiler automatically addresses the respective pointer during the `Add()` call. However, in remote calls, due to distinct address spaces between processes, function pointers cannot be directly applied. In RPC, every function requires a unique ID. When clients perform RPC, they must attach this ID.

This necessitates maintaining a function $\leftrightarrow$ Call ID mapping (`call_id_map`) on both the client and server. Whether using function strings or integer IDs for Call ID mapping, the map typically takes the form of a hash table. Although maps on both ends needn't be identical, the Call IDs for the same functions must match. When the client needs to make a remote call, it checks the table to find the corresponding Call ID, and then sends it to the server. The server also checks the table to determine the function that the client needs to call and then executes the corresponding code of the function.

Serialization and deserialization: How does the client send parameter values to a remote function? In local calls, parameters are merely pushed onto the stack, allowing the function to read them directly. But in RPC, since the client and server are distinct processes (and sometimes even in different languages), direct memory transfer is infeasible. Hence, parameters are first serialized into a byte stream by the client and, after being sent to the server, are deserialized into a readable format. The reverse applies for values returned from the server. This serialization and deserialization can be custom-built or use protocols like Protocol buffers or FlatBuffers.

Network transmission: Since remote calls are typically networked, a transmission layer is essential. This layer needs to send the serialized parameters and Call ID to the server and retrieve the serialized results. While most RPC frameworks use the TCP protocol, there's

no specific constraint as long as the transmission is achieved. Other protocols like UDP, HTTP2, or Java's Netty can also be utilized. For network transmission, custom-built sockets or libraries like asio, ZeroMQ, etc., can be employed.

14.4.1.4 Do RPCs have advantages?

The following question arises: by contrasting with local function call implementations, we can conclude that invoking a function remotely via RPC incurs a number of additional costs. These costs include the transmission overhead of remote communication, the computational costs of serialization and deserialization, and the maintenance overhead for both the server and client. So, why do we prefer remote function calls over the more streamlined local function call approach? Let's delve into the benefits that remote function calls offer to large-scale projects.

Decentralized management and maintenance: First, the RPC paradigm is very conducive to decentralized management and maintenance by multiple parties. As illustrated in the earlier example:

1. From the client's perspective, the function's computation is a black box, with all the processing occurring on the server side, invisible to the client. Thus, the client's development process doesn't need to concern itself with the function's specific execution. It merely focuses on the required inputs to obtain desired outputs, designing appropriate functional modules accordingly.
2. From the server's side, details of when a function is invoked or executed are also invisible. The system's overall complexity and logical branches in the environment are irrelevant to the server. The focus of server-side development is on the efficiency, performance, and results of the function when executed.

It's evident that, apart from agreeing on input-output formats, dependencies between various parties in design, deployment, and service delivery are almost non-existent. Under this paradigm, the server responsible for function execution and the client responsible for function invocation operate almost independently. This is highly suitable for large projects where multiple teams work on distinct components. Moreover, cascading mechanisms can fully utilize deployed servers, meeting new requirements without redundant development, saving substantial manpower costs.

Optimized computational resources: Second, the RPC paradigm allows for targeted configuration and optimization of computational resources based on the function’s characteristics and needs, resulting in more efficient resource utilization.

For instance, executing all functions locally can be likened to a housekeeper performing the roles of a gardener, driver, nanny, and teacher simultaneously. Machines have limited resources and can only process tasks serially. On the one hand, this serial processing might not ensure efficiency. On the other, it’s unlikely for the jack-of-all-trades housekeeper to excel as much as a specialized nanny or teacher. Similarly, local computational resources might not satisfy varying performance optimization demands.

Remote function calls are akin to “letting professionals handle their specialties”. For instance, dedicated drivers are on standby for travel needs and are immediately available when required. When their services aren’t in demand, they’re free to assist others.

Depending on the specific computations within a function, servers maintaining these functions can adopt particular computational strategies for optimization, enhancing performance significantly. For example, if a function requires extensive matrix multiplications or dot product calculations, deploying corresponding servers in GPU-rich environments can markedly boost performance. If a function frequently interacts with databases with CRUD operations, the server might connect directly to distributed databases like SPARK or HBASE to amplify read-write efficiency. For functions with high concurrency but straightforward operations, deploying servers in high-performance cloud environments can double efficiency. These benefits are unattainable in entirely local computing environments.

Distributed architecture and cloud services: Nowadays, large-scale projects have mostly abandoned monolithic architectures. Even for medium- and small-scale projects, transitioning from monolithic to service-oriented distributed architectures is trending. RPC remote function calling is at the heart of distributed architectures, representing its innate advantage.

Based on the RPC response mode, remote function calls can be broadly divided into two types, each with extensive applications:

1. **Synchronous calls:** The client invokes a service method and waits until the service returns a result or times out before proceeding. Implementations of synchronous calls include WebService

and RMI. WebService operates on web containers using the HTTP protocol, making it suitable for interoperation between heterogeneous systems using different languages. RMI is essentially Java's RPC implementation, supporting the return of Java objects and primitive data types, making it apt for interoperation among Java-built systems.

2. **Asynchronous calls:** The client sends a message to middleware, doesn't wait for the server's response, and proceeds with its tasks. The Java version of asynchronous calls is Java Message Service (JMS). Popular open-source JMS middleware includes Apache's ActiveMQ and Kafka, as well as Alibaba's RocketMQ.

14.5 High-Performance Open-Source RPC Framework: gRPC

In the previous section, we briefly introduced the concept of RPC. However, when setting up real-world projects, project owners neither need nor can they feasibly begin from scratch, implementing essential lower-level features, such as network transport protocols, input/output definitions and sharing, and serialization/deserialization technologies. Therefore, selecting a highly efficient RPC framework becomes paramount. This section primarily introduces the high-performance open-source RPC framework, gRPC, developed by Google.

14.5.1 Unique Advantages of gRPC

gRPC is a RPC technology framework designed by Google, tailored for mobile application development and based on the HTTP/2 protocol standard. Being widely used within Google, gRPC has endured the tests of countless real-world projects. It combines stability, scalability, and standardization, making it a powerful and comprehensive open-source RPC framework. gRPC boasts several key advantages, facilitating more convenient and efficient RPC development and use:

14.5.1.1 Cross-language invocation

Thanks to Protocol Buffers, gRPC has established a unified standardized definition format. Functions from different programming environments can effortlessly share a single input/output format

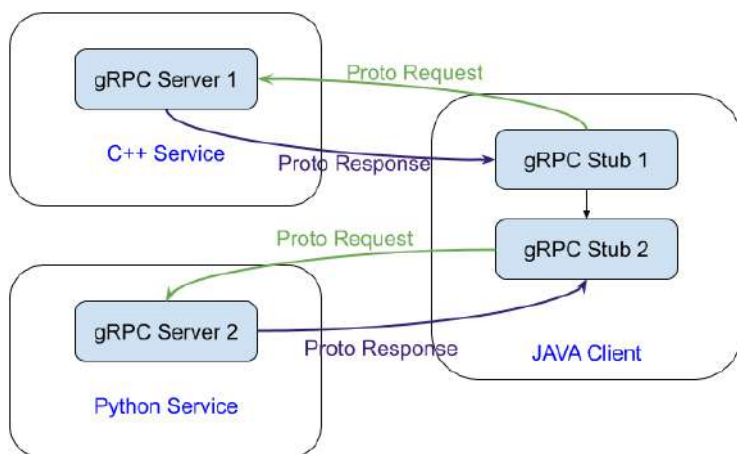


Figure 14.16. gRPC cross-language invocation example.

without much concern over cross-language compatibility issues. Serving as a nexus for various service calls within Google, gRPC supports a remarkable range of languages, from popular ones like C++, Java, C#, Node.js, Objective-C, PHP, Python, and Ruby, to more niche languages like Android Java, Dart, Go, JVM, and Web. This broad support gives gRPC its unique edge in cross-language invocation as shown in Figure 14.16.

Considering that function implementations in large projects are typically distributed across various development teams, this standardized format brings about immense convenience and efficient resource allocation. Teams can choose the most optimization-friendly programming language based on computational resource needs or community support scale. They can also bypass bottlenecks when developing specific functions in certain languages by leveraging pre-existing modules, reducing developmental challenges and redundant work.

14.5.1.2 Open source and standardization

Owing to its wide language support, gRPC's open-source community is notably vibrant. The serialization protocol of Protocol Buffers also ensures a structured and reliable development process, effectively catering to various expansions and improvements for distinct business scenarios. On one hand, the active open-source community

ensures that many issues encountered during development have been previously addressed by experienced developers, offering significant advantages in user experience. On the other hand, even today, gRPC continues to have numerous new features consistently being experimented with and incorporated. Despite rapid advancements in device capabilities and the emergence of novel deployment strategies in recent years, the community's developers can still meet diverse scenario demands, enhancing performance and fulfilling user expectations.

14.5.1.3 Communication security

A critical aspect of RPC is ensuring the security of communications and computational resources. Without adequate permission settings, RPC-based applications can be quite vulnerable. Whether it's due to hacker attacks or bugs in system requests, computational resources can be easily exhausted, paralyzing the server.

Originating from Google's internal frameworks, gRPC was open-sourced with high-security requirements in mind. It can be used in conjunction with various authentication mechanisms, ensuring that gRPC can be securely and easily used for communication with other systems. gRPC incorporates the following authentication methods:

1. **SSL/TLS:** gRPC integrates with SSL/TLS and promotes its use for server authentication and encryption of all data exchanged between clients and servers. Clients can optionally provide certificates for mutual authentication.
2. **ALTS:** For applications running on the Google Cloud Platform (GCP), gRPC supports ALTS as a transport security mechanism.
3. **Token-based authentication:** gRPC offers a universal authentication mechanism based on Google API implementations, attaching metadata-based credentials to requests and responses for authentication. These credentials can authenticate with Google across various deployment schemes via SSL/TLS. It's important to note that Google does not allow connections without SSL/TLS, and most gRPC language implementations forbid sending credentials over unencrypted channels. Google-specific credentials can only be used to connect to Google services; sending Google-issued OAuth2 tokens to non-Google services may result in token theft.

4. **Simple authentication API:** By extending the code, one can integrate their authentication system, providing all necessary authentication information as credentials to the server when creating channels or making calls.

14.5.2 Key Concepts of gRPC

In gRPC, a client can invoke predefined methods on gRPC servers located on other remote computers or environments as easily as invoking local functions. Similar to other RPC systems, gRPC is primarily centered around the idea of “service definition”, specifying which methods can be remotely called, along with their parameters and return types. On the server side, one simply needs to fill out and implement the corresponding service interface and then run a gRPC server to handle client requests. On the client side, a stub (or “Stub”) is retained, offering the same input/output format as the server functions (without being responsible for actual execution, akin to function signature declarations in JAVA or C++).

gRPC contains numerous standardized specifications and interface formats defined by Google. Not only have they been widely used in various business scenarios, but they’ve also expanded to encompass new, practical functionalities in response to diverse community requests. Familiarizing oneself with the essential concepts of gRPC will accelerate understanding and unveil the potential of the gRPC framework in specific project implementations.

14.5.2.1 Protocol buffers

Protocol Buffers, often referred to as “protobufs”, is a mature open-source mechanism developed by Google. It can serialize structured data independently or define data structures for message transmission in combination with data formats like JSON. Protocol Buffers serve as gRPC’s default interface definition language and establish the foundational standard for message exchanges. In defining RPCs, it’s essential to specify input and output types, formats, and security validations determining who and when the RPC can be invoked.

The first step with Protocol Buffers is defining serialized data structures in a .proto file. All data in protobufs must be constructed as “messages”, where each message represents a small logical record format comprising a series of key-value pairs called “fields”.

For instance, consider a remote function called `RemoteAdd()`. The message definitions required for input and output formats for this function are as follows:

```
message RemoteAddInput{
  int32 x = 1;
  int32 y = 2;
}

message RemoteAddOutput{
  int32 answer = 1;
}
```

Here, `RemoteAddInput` defines two `int32` inputs: `x` and `y`, while `RemoteAddOutput` consists of a single `int32` output named `answer`. This is analogous to declaring a function in C++ with two `int`-type parameters and one `int`-type return value.

After defining these structures, the Protocol Buffers compiler generates data access classes in the preferred language based on the protobuf definitions. For example, if C++ is the chosen language, the compiler will produce two classes: `RemoteAddInput` and `RemoteAddOutput`. In these classes, functions like `x()` can retrieve the value of the `x` field, `set_x()` sets its value, and utility functions like `ToByteArray()` serialize the entire data structure, while `parseFrom()` deserializes it from raw bytes.

14.5.2.2 gRPC services

A gRPC service is a collection of RPC functions — akin to a menu. Starting a gRPC service on a server is like opening a business for operation. The server then officially supports client-invoked RPCs listed on the “menu”. This menu format is also defined in a `.proto` file, using Protocol Buffers messages for RPC method parameters and return types. Here’s an example of a simple unary RPC service method:

```
service Calculator{
  rpc AddRemote (AddRemoteInput) returns (AddRemoteOutput);
}
```

gRPC supports four types of service methods, differing in the form of input and output:

1. **Unary RPC:** The client sends a single request to the server and receives a single response, much like invoking a typical C++ function.

```
rpc RemoteAdd (RemoteAddInput) returns (RemoteAddOutput);
```

2. **Server streaming RPC:** The client sends a request to the server and receives a stream of messages in return. The client reads from this returned stream until there are no more messages. gRPC preserves the order of messages within a single RPC call.

```
rpc RemoteAddManyOutputs (RemoteAddInput) returns (stream RemoteAddOutput);
```

3. **Client streaming RPC:** The client writes and sends a series of messages using a stream. Once all messages are sent, the client waits for the server to read them and respond. Again, gRPC retains the order of messages.

```
rpc RemoteAddManyInputs (stream RemoteAddInput) returns (RemoteAddOutput);
```

4. **Bidirectional streaming RPC:** Both parties use read-write streams to send a series of messages. The two streams operate independently, allowing the client and server to read and write in whatever order they prefer. The order of messages in each stream is preserved.

```
rpc RemoteAddBidi (stream RemoteAddInput) returns (stream RemoteAddOutput);
```

14.5.2.3 gRPC compilation and API content population

Once the input and output of gRPC service functions are determined, the Protocol Buffers compiler plugin provided by gRPC can be utilized. This will auto-generate corresponding code from .proto files, which includes API code for both gRPC clients and servers. gRPC typically invokes these APIs on the client side and implements them on the server side.

On the server side, the methods declared in the service are implemented and the gRPC server is run to handle client requests. The gRPC infrastructure decodes incoming requests, executes the service methods, and encodes the service response.

On the client side, there is a local object called a “stub” (in some languages, the preferred term is “Client”) that implements the same methods as the service. Clients can then call these methods on the local object, wrapping the call parameters in the appropriate Protocol Buffers message. gRPC handles the dispatching of the request to the server and returns the server’s Protocol Buffers response.

Of course, this API code merely represents the code framework specified in the .proto file for the input and output formats, analogous to a client placing an order from a menu. The actual content or body of the function, which represents the service being provided, needs to be filled in by developers. This can be likened to the process of a chef cooking a dish.

14.5.2.4 Standard RPC lifecycle

In most languages, gRPC programming APIs can generally be divided into synchronous and asynchronous call styles, catering to different application scenarios.

The ideal implementation sought by the RPC model is akin to local function synchronous calls. This means that an RPC call should remain blocked until a response is received from the server. Such an approach ensures efficient resource allocation and use while avoiding potential conflicts and risks.

However, in the vast landscape of RPC usage in networking, the nature is more asynchronous than synchronous. An RPC implemented as an ideal synchronous call might become inefficient and fault-prone if it blocks the current thread. Hence, developers often prefer the ability to make asynchronous RPC calls, which is crucial for real-world project deployment and optimization.

Depending on the RPC service method, the complete lifecycle of a gRPC client calling a gRPC server method may vary slightly.

1. **Unary RPC:** The client sends a single request and receives a single response. Once the client invokes a stub method, the server gets notified with client metadata, method name, and a specified deadline for the RPC. The server, application-specific, might

immediately send its initial metadata or wait for the client's request message. Once the server receives the client's request, it carries out all necessary actions to create and populate the response, returning it with status details, including a status code, optional status message, and optional trailing metadata. If successful, the response status is "OK", and the client receives the response, thereby concluding the RPC call.

2. **Server streaming RPC:** This is similar to Unary RPC, but the server responds with a stream of messages in reply to the client's request. Once all messages are sent, the server's status details are transmitted to the client, marking the completion of server-side processing. The client-side process finishes upon receiving all the server's messages.
3. **Client streaming RPC:** This is akin to Unary RPC, but the client sends a stream of messages to the server instead of a single message. The server typically sends its response message (along with its status details and optional trailing metadata) after receiving all the client's messages, but there can be exceptions.
4. **Bidirectional streaming RPC:** The call is initiated by the client invoking a method, with the server receiving client metadata, method name, and deadline, among others. The server can choose to send its initial metadata immediately or wait for the client to start streaming messages. Both streams (client and server) operate independently, allowing reading and writing of messages in any order by both parties. They can operate in a back-and-forth manner: The server waits for the client's message, sends a response, then waits for another client message based on the previous response, and so on.

14.5.2.5 Special RPC lifecycle

gRPC also accommodates special lifecycles to manage non-terminating or abnormally terminating RPC commands:

1. **Deadlines/Timeouts:** gRPC allows clients to specify how long they're willing to wait for an RPC to complete before it's terminated due to a `DEADLINE_EXCEEDED` error. On the server side, it can be checked whether a specific RPC has expired or how much time remains. Specifying a deadline or timeout is language-specific, with some APIs working based on timeouts (duration)

and others on deadlines (fixed points in time), with no default deadline.

2. **RPC termination:** In gRPC, both client and server acknowledge the successful execution of an RPC independently, yet their conclusions might not align. For instance, an RPC might succeed on the server side (“All responses sent!”) but fail on the client side (“Response arrived after my deadline!”). Typically, the server acts only after all client requests have been sent. However, in real-world scenarios to ensure robustness, the server might make decisions before the complete transmission of client requests, such as truncating overly lengthy content and returning an error message.
3. **RPC cancellation:** Both the client and server can cancel an RPC at any time. Cancellation terminates the RPC immediately, meaning no subsequent operations will execute. However, any actions preceding the cancellation will not be rolled back.
4. **Metadata verification:** Metadata, provided as key-value pairs, offers details about a specific RPC call, such as authentication specifics. In gRPC, keys are strings and values are often strings or binary data. Metadata remains opaque to gRPC itself, but it allows clients to supply the server with call-related information and vice versa. Access to metadata is language-dependent.
5. **Channel verification:** A channel in gRPC offers a connection to a gRPC server on a specified host and port. Channels are used when creating client stubs. Channel behavior, like enabling or disabling message compression, can be modified using channel parameters. Channel states include “connected” and “idle”. Monitoring or querying these states helps understand RPC command execution, with the way gRPC queries channel state or manages channel closure being language-specific.

14.5.2.6 Protocol buffers versioning

gRPC’s default version of Protocol Buffers is proto2. However, most of the open-source community users of Protocol Buffers opt for proto3. This version simplifies the syntax, introduces beneficial new features, and is compatible across all languages supported by gRPC. Currently, proto3 supports Java, C++, Dart, Python, Objective-C, C#, lite-runtime (Android Java), Ruby, and JavaScript. Using proto2 for client communication might pose compatibility issues, hence it’s recommended to use the proto3 version specification in gRPC.

14.6 Decoupling of FedLearn System Services and Algorithms

In the preceding sections, we introduced the system architecture of FedLearn and its design, which allows algorithm invocations through RPC. Given that the underlying system and algorithmic steps in real-life scenarios often face vastly different optimization objectives, both aspects interact via API interfaces, ensuring a thorough decoupling of the system and the algorithm.

Algorithm designers can implement a full algorithm within the algorithm package and carry out integrated testing without network dependencies, evaluating various algorithmic metrics and training speeds.

Simultaneously, a unified system framework ensures standardized optimization for network interactions and failure management, which consequently reduces both algorithmic and engineering development costs.

14.6.1 Phase Concept and Automata

Federated learning algorithms require multiple participants and numerous interactions to complete a single iteration. Thus, we've decomposed what would be a single iteration in conventional algorithms into several steps, with each step representing a phase. Transforming conventional algorithms into federated learning versions can significantly cut development costs. Therefore, we split a complete iteration into multiple phases and employ automata for state updates. All algorithms are presented through a unified interface, making external invocation seamless. From the system scheduling layer's perspective, only four elements are essential: termination conditions, request reception, parameter aggregation, and request dispatch. More details are shown in Figure 14.17.

14.6.2 Componentization

Various algorithms share many similarities in their implementation. As a result, we designed several generic components to support algorithm development:

- ❑ **Homomorphic encryption module:** The system offers additive homomorphic encryption based on Paillier and IterativeAffine, and RSA-based multiplicative homomorphic encryption.
- ❑ **Differential privacy module:** Provides differential privacy based on the exponential and Laplace mechanisms.
- ❑ **Mathematical operations module:** Offers fundamental mathematical computations supporting various scalar and vector operations.
- ❑ Multi-party gradient descent and derivation module.
- ❑ **Data analysis module:** Allows for analysis of raw user data, including feature correlation, validity, etc. The analysis can be visualized in chart form.
- ❑ **Metrics calculation:** Offers a module for metric calculations based on predicted and actual values, adapting to encountered real-world scenarios.
- ❑ **Data parsing and preprocessing:** Supports various types like two-dimensional arrays and sparse data parsing, alongside numerous inbuilt data preprocessing functions. Also offers a public interface for users to customize loading and parsing functions.
- ❑ **Various set intersection operations:** Currently supports intersection protocols based on MD5, RSA, Diffie-Hellman, Freedman (2009), among others.
- ❑ Parameter retrieval and generation.

Algorithm developers can leverage these common modules through a unified interface for various functionalities, with the added flexibility to custom-implement any module.

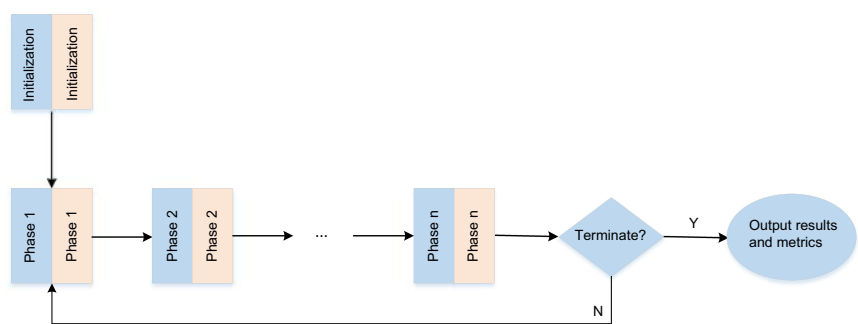


Figure 14.17. Structure of FedLearn’s algorithmic automaton.

14.6.3 Other System-Level Optimizations

- ❑ **Security authentication:** The system employs multiple security authentications to ensure overall security. First, every external API offered by the coordinator requires an identification code for invocation, with each code being unique. For interactions between the coordinator and the client, each request initiates an authentication process based on version number, unique identifier, source IP address, etc. Only upon successful authentication is the interaction allowed.
- ❑ **Training stability:** The task isolation mechanism ensures that tasks don't interfere with one another, and the failure of a single task doesn't compromise the entire system. Moreover, the resource allocator guarantees that resources used by any task do not exceed its allocation.
- ❑ **High-performance inference:** The system has been optimized for two main inference scenarios: rapid public network inference and large-scale offline inference. For the former, connection reuse and algorithmic optimizations reduce the request count, achieving public network online inference times below 200 milliseconds. For the latter, techniques such as batch-wise inference (to prevent memory overflow) and algorithm parallelization ensure faster batch inference speeds.

14.7 FedLearn Deployment and Use

14.7.1 System Components and Functionality

The federated learning system comprises Coordinator, Client, and Frontend components. The synergy between these components not only ensures system integrity but also its security.

14.7.1.1 Coordinator

The Coordinator serves as the backend control unit of the federated learning system, receiving requests from clients and facilitating encrypted communications between them. This ensures data remain invisible yet usable among clients, enabling the complete process of federated modeling. Acting as a scheduler, the Coordinator doesn't handle data directly. It ensures data security by encrypting

transmissions. Thus, it can be deployed on any participating node or even on a neutral one.

Its installation package consists of a startup script, configuration files, and dependency packages. Users can modify the configuration files before deploying using the script. The Coordinator supports both sqlite and mysql as data sources, allowing flexibility based on actual needs. Furthermore, it's imperative that there's network connectivity between the Coordinator and the Clients.

14.7.1.2 Client

The Client acts as the computational entity in the federated learning system, responsible for loading local data, processing models, and transmitting intermediate results to the Coordinator. Deployed within each participant's internal environment, it ensures data security through white-listed coordinators, only allowing access from authenticated Coordinators.

The Client's installation package includes startup scripts, configuration files, and dependency packages. Similar to the Coordinator, users can modify the configuration files before using the script for deployment. The configuration file allows various data source formats like csv, mysql, and hdfs, offering flexible data source switching for training and inference through interfaces.

14.7.1.3 Frontend

The Frontend offers a user interface for the federated learning system, enabling visual operations for both Coordinators and Clients. Participants can engage with the system to create, join, train, and infer tasks. Currently, the Frontend must be installed on the same server as the Coordinator.

The Frontend installation package consists of jar files and configuration files. To deploy, users need to modify database information in the yaml configuration file, ensuring it resides at the same directory level as the jar files.

14.7.2 Standard Deployment

There are no specific directory requirements for the java versions of Coordinator, Client, and Frontend. Using the `/home/fedlearn/app/` directory as an example, the deployment process is as follows:

14.7.2.1 Coordinator deployment

The Coordinator communicates with the Client using the HTTP protocol. Taking `/home/fedlearn/app/` directory as reference, the specific deployment steps are as follows:

- ❑ **Data initialization:** The Coordinator relies on a database to save persistent data. Thus, a database needs to be set up along with its structure. Our existing Python project can facilitate this setup. Depending on the metadata storage method, it currently supports both mysql and sqlite, with sqlite included in the system installation package.
- ❑ **Installation package download:** Obtain the corresponding version of the installation package in the `/home/fedlearn/app/` directory.
- ❑ Unzip the installation package and grant read-write permissions to the corresponding folders:

```
chmod 754 /home/fedlearn/app/*
unzip fedlearn-coordinator-assembly.zip -d /home/fedlearn/app
```

- ❑ Modify the Coordinator's configuration file, which resides in the `conf` directory of `fedlearn-coordinator`. Update the `master.properties` file's data source configuration. Currently, both mysql and sqlite are supported. Detailed settings are as follows:

```
# Application name, consistent with app in logback.xml
app.name=fedlearn-coordinator
# Configure server port
app.port=8092
# Actual path of the logback configuration file
log.settings=conf/logback.xml
# Metadata storage method, supports mysql and sqlite, and
sets the actual database file
db.type=sqlite
db.driver=org.sqlite.JDBC
db.url=jdbc:sqlite:conf/fl.db
# Whether to use HTTP multipart transmission during data
transfer
http.split=true
# Whether to compress during data transfer
http.zip=true
```

- ❑ Configure the service log path and grant it appropriate read-write permissions:

```
mkdir -d /home/fedlearn/log/fedlearn-coordinator
chmod 754 /home/fedlearn/log/fedlearn-coordinator/*
```


- ❑ **Starting the service:** The startup script is stored in the bin directory of fedlearn-coordinator. Based on your machine's configuration, you should modify the JAVA_MEM_OPTS in start.sh. Due to varying machine specifications, one of the two JVM options in JAVA_MEM_OPTS will take effect:

```
cd /home/fedlearn/app/fedlearn-coordinator/bin
sh start.sh -c master.properties
```

The -c flag points to the actual directory where the coordinator's configuration file is stored.

- ❑ **Check service status:** The following command helps determine if the service has successfully started:

```
ps -ef|grep fedlearn-coordinator
```

- ❑ **Stop the service:** The directory for this script is the same as the startup script:

```
cd /app/fedlearn-coordinator/bin
sh stop.sh
```

14.7.2.2 Client deployment

Each Client is deployed on a separate machine and interacts with the server using the HTTP protocol. Using the directory /home/fedlearn/app/ as an example, the specific deployment steps are as follows:

- ❑ **Download the installation package:** Retrieve the appropriate version of the installation package and place it in the '/home/fedlearn/app/' directory.
- ❑ **Unpack the Installation Package and Assign Read-Write Permissions:**

```
chmod 754 /home/fedlearn/app/*
unzip fedlearn-client-assembly.zip -d /home/fedlearn/app
```

- ❑ **Modify the client configuration file:** The configuration files are located in the "conf" directory of "fedlearn-client". Depending on your needs, adjust the "client.properties" file to set data source configurations:

```
# Application name used in the startup script; it should
# match the app name in logback.xml.
app.name=fedlearn-client
# Define the service's startup port.
app.port=8094
# Set the actual path for the log file.
log.settings=conf/logback.xml
# IP check for the master node. Only specified IPs are
# allowed to access this client service. Use commas for
# multiple IPs.
master.address=127.0.0.1
# Configure the training data source. It supports csv and
# mysql formats and allows multiple data source configurations.
train1.source=csv
train1.base=/home/fedlearn/Data/fedlearn-client/
train1.dataset=train1.csv
# Set the data source for inference. It supports formats like
# csv, mysql, hdfs, and more.
inference.data.source=csv
inference.base=/home/fedlearn/Data/fedlearn-client/
inference.dataset1=mo17ktest.csv
# Define the model saving directory.
model.dir=/home/fedlearn/Data/fedlearn-client/model/
# Specify the directory for storing prediction results.
predict.dir=/home/fedlearn/Data/fedlearn-client/predict/
```

- ❑ Set up the service log path and grant the appropriate read and write permissions:

```
mkdir -d /home/fedlearn/log/fedlearn-client
chmod 754 /home/fedlearn/log/fedlearn-client/*
```

- ❑ **Start the client service:** The startup script is located in the bin directory of fedlearn-client. Modify the JAVA_MEM_OPTS in start.sh according to your machine's configuration. Depending on the machine, one of two JVM options will be effective:

```
cd /home/fedlearn/app/fedlearn-client/bin
sh start.sh -c client.properties
```

The -c flag points to the actual directory where the client's configuration file is stored.

- ❑ **Check service status:** Determine whether the service is currently running:

```
ps -ef|grep fedlearn-client
```

- ❑ **Stop the service:** The directory for this operation is the same as the startup script:

```
cd /app/fedlearn-client/bin
sh stop.sh
```

14.7.2.3 Front-end deployment

The front-end provides a visual interface for the federated learning platform, facilitating user operations and optimizing federated learning model training and inference services. It is recommended to deploy the front end and the coordinator on the same machine. Using the directory `/home/fedlearn/app/` as an example, the detailed deployment steps are as follows:

- ❑ **Obtain the installation package:** Download the corresponding version of the front-end compressed package `federated-learning-front.jar`.
- ❑ **Configure the service log path:** Grant the required read and write permissions:

```
mkdir -d /home/fedlearn/log/federated-learning-front
chmod 754 /home/fedlearn/log/federated-learning-front/*
```

- ❑ *Start the Front-end Service:*

```
java -jar /home/fedlearn/app/federated-learning-front.jar
```

- ❑ **Check service status:** Verify if the service is currently running:

```
ps -ef|grep federated-learning-front
```

14.7.3 Deployment for Distributed Version

The deployment of the distributed version's coordinator and front end is consistent with the standard version and won't be reiterated here. The distributed client version comprises two parts: the management unit and the computational unit. These can be differentiated by roles. The specific deployment steps are as follows:

- ❑ **Obtain the installation package:** Download the blockchain version's compressed package named `fedlearn-distribution.zip`.

- ❑ **Unpack the installation package:** The management and computation units can be deployed by modifying the role in the configuration file. For example, to configure the manager, the settings are as follows:

```

role = manager
# Application name
app.name=fedlearn-distribution
# Start-up port
app.port=9094
# Log file path
log.settings=/home/fedlearn/fedlearn-distribution/conf/
logback.xml
# Addresses for the slave nodes, separated by semicolons
slaves.address=10.222.85.151:8094;10.222.85.152:8094
# Default slave
default.slave=http://10.222.85.151:8094
manager.address=10.222.85.151:9094
master.address=127.0.0.1
# Configuration for the training data source, supports
multiple data sources
train1.source=csv
train1.base=/home/fedlearn/data/house/
train1.dataset=reg2_train.csv
# Configuration for inference data source
inference.data.source=csv
inference.base=/home/fedlearn/data/house/
inference.dataset1=reg2_test.csv
# inference.table=user_click
# Allow prediction of UID in the training set
inference.allowTrainUid=True
# Model save path
model.dir=/home/fedlearn/data/model/
# Prediction result file directory
predict.dir=/home/fedlearn/data/predict/

```

- ❑ Configure the service log path and grant the required read and write permissions.

```

mkdir -d /home/fedlearn/log/fedlearn-distribution
chmod 754 /home/fedlearn/log/fedlearn-distribution/*

```

- ❑ The start and stop service commands for fedlearn-distribution can be referenced from the standard client deployment and will not be detailed here.
- ❑ For deploying the computation unit, simply set the role in the configuration file to worker. The rest of the operations are similar to the manager configuration to complete the deployment.

14.7.4 Container-Based Deployment

Docker is an open-source application container engine. It allows users to package source code along with its dependencies into a portable image. Algorithms like random forest, core algorithm, and horizontal federated learning from the federated learning platform are deployed on servers via Docker images. Here are the detailed deployment steps:

- ❑ **Install docker:** Refer to the official Docker documentation on www.docker.com to install and start Docker:

```
systemctl start docker
```

- ❑ **Obtain the federated learning algorithm's docker image:** Download the image archive `federated-learning-grpc-0.1.0.tar`. Use the following commands to load the image, view the existing images, and note the `image_id` corresponding to the federated learning algorithm:

```
docker load -i federated-learning-grpc-0.1.0.tar
docker image ls
```

- ❑ **Run the federated learning image in a docker container:** Execute the following command to run the image and start the `grpc` service:

```
docker run -ti -d=true --name federated-learninggrpc-p
8891:8891 (image_id)
bash -c "python3 /app/src/main/python/algorithm/server.py -P
8891
```

- ❑ **Terminate the image:** Use the following command to view existing containers and note the corresponding `container_id`:

```
docker container ls -a
docker stop (container_id)
```

14.7.5 User Interface Operations and API

A one-stop federated learning operation platform that includes online model training, online inference, and remote client-side inference:

- ❑ Both the client and the coordination sides are operated through a visual console. This includes initiating training and inference tasks, showcasing task details, and one-click stop/restart training. The client is controlled through directive operations, and the coordination process retains no data.
- ❑ **Coordination side:** Includes synchronous training and asynchronous online inference.
- ❑ Multiple model selection with flexible parameter configurations.
- ❑ **Client side:** Online selection to join the inference algorithm, supporting individual (API) or remote (CSV) file methods for predictions. It supports client-path methods, local file reading, reducing data network transmission, and enhancing interactive security.

14.7.5.1 System permissions module

System permissions refer to the differentiation of user rights by the system to ensure its security. Specifically, there are three types of users: super administrator, enterprise administrator, and regular user. The roles of each user type are as follows:

- ❑ **Super administrator:** The unique system super administrator, capable of creating new enterprises and their respective administrators, ensuring system security from the enterprise perspective.
- ❑ **Enterprise administrator:** Can create users and assign them permissions, ensuring their enterprise's data security within the federated learning system. Enterprise administrators can also participate in projects, training, and inference modules.
- ❑ **Regular user:** Can create and join tasks.

14.7.5.2 Project module

The project module includes task creation, task joining, and task details.

❑ *Task Initiation*

- **My initiated tasks:** Click on the “Create Task” button in the “My Initiated” interface to display all ongoing tasks on the platform, including task IDs, task names, and specific participants (including multiple parties). Supports fuzzy searches by task name or keyword.

- **Task details:** Click on a specific task name to view detailed information, displaying participants, client addresses, and feature information, which includes feature description, feature values, feature types, and user additions.
- **Create task:** Click on the “Create Task” button to navigate to the task details page.
- **Task information configuration:** Users can set the task password, visibility, inference rights, IP, port, protocol, training data, and other details.

□ *Joining a Task*

- **Tasks I’ve joined:** As a client, or if there’s no need to create a training task, you can directly click on the “Join Task” page on the left side. Currently, the platform does not have permission controls (future product development will be designed independently). You can directly view all tasks on the platform. The management page supports keyword or initiator account ID searches. To enhance readability, the currently logged-in account ID is divided into “Tasks I’ve Joined” and “Tasks I Haven’t Joined” tabs. Users can join tasks through the “Join Task” option or the “Tasks I Haven’t Joined” → “Join Task” route.
- Upon selecting “Join Task”, users are redirected to the task information entry page. A dropdown form allows quick task name selection. Enter the current IP and other details, click “Search”, select from the dropdown “Current User Participating Training Dataset”, and click “Confirm” to complete the joining process.

□ *Task Details*

Users can view the initiator and participants, as well as preview the feature set ID uploaded by each participant. For details, see Figure 14.18.

14.7.5.3 Training management module

The training management module primarily facilitates operations such as starting training, pausing training, restarting training, stopping training, and retraining for the tasks that have been created:

❑ *Training Task List*

- Front-end visualization of the training progress for all tasks, with percentage progress bars displaying completion. It supports online stopping and pausing actions. Detailed text descriptions are as follows:
- **Stop:** This stops the training task in its entirety. The task status is initialized, returning to 0%.
- **Pause:** Click on “Pause” to record the current task status, which can then be restarted to continue training.

❑ *Restart Training*

Click on “Restart” to resume a task that’s in a paused state and continue training.

❑ *Initiate Training*

Select the task name you want to train, for example, 1203.

- **Algorithm Selection:** The platform currently supports algorithms like FederatedGB, RandomForest, and more. Algorithm parameter descriptions can be found in Part 3. Training here uses FederatedGB as an example.
- **Pre-training preparation:** Select the ID alignment method and execute “ID Alignment”. Once ID alignment is complete, configure the parameter combinations for the selected algorithm.

Full parameter configuration can be viewed on the platform page.

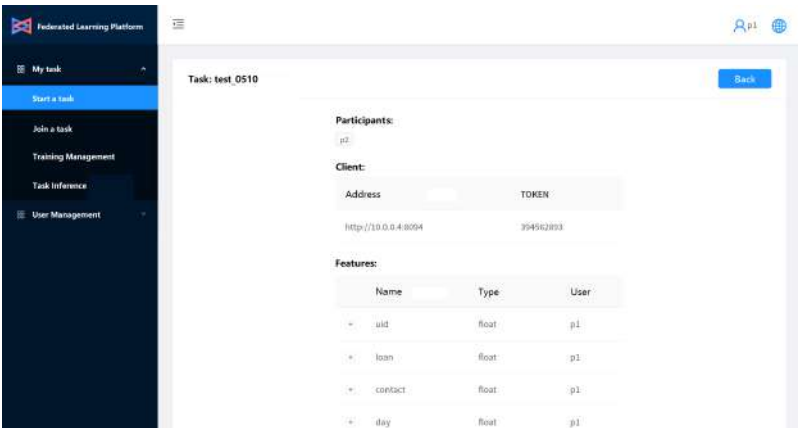


Figure 14.18. Task details.

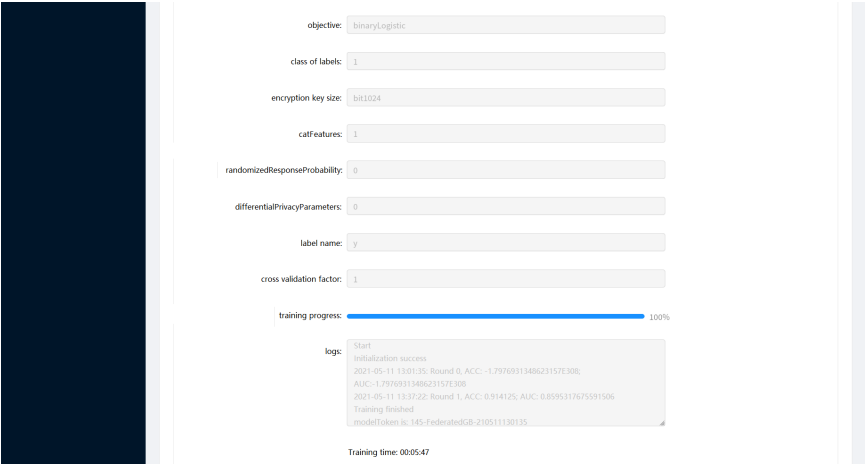


Figure 14.19. Training details.

- **Training Completion:** When the task status reaches 100%, the training is completed.
- **Task Details:** This includes model parameters, training parameters, training start time, overall progress, and the duration and data logs for each layer or tree during the training process. See Figure 14.19 for details.

❑ *Retraining*

Considering the reusability of each training task, a retraining function is provided to allow for retraining with the same task name, same algorithm, and same parameter settings.

Note: For tasks that are 100% trained, you can click on the “Inference” button to quickly proceed to the inference step.

14.7.5.4 Inference module

Currently, the inference supports three methods: manual inference, remote inference, and batch upload.

❑ *Manual Inference*

Users can input single or a small number of texts, call the real-time API for online inference, and get immediate results.

Clicking on the “?” icon on the right displays a manual inference example. See Figure 14.20 for details.

❑ Remote Inference

Taking into account remote deployments by customers, and in an effort to reduce transmission costs and ensure data interaction security, users can input the file path of the feature ID. The platform will directly read the data via the client and save the results in a local file on the client’s side.

❑ Batch Upload

For remote inference, the data to be predicted needs to be uploaded in advance to the corresponding client’s server. If a user has multiple files to predict, the convenience of the operation can be improved. An interface for batch uploading from the local computer is provided. Clicking on “Batch Upload” followed by “Upload File” will open a window that allows loading the required uid data files for prediction from the local machine, facilitating batch inference.

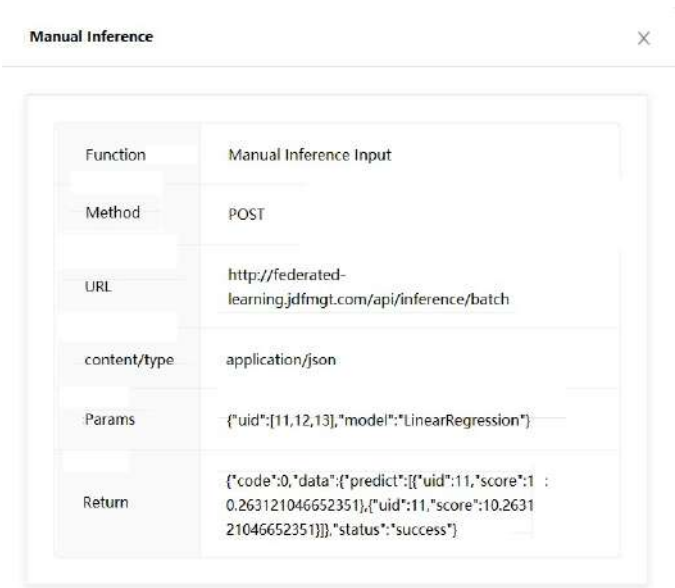


Figure 14.20. Manual inference example.

14.7.5.5 Model application module

The primary purpose of the model application is to migrate trained models to any inference service environment. This ensures that the training cluster and the inference cluster can be completely isolated, making it easier for model training and upgrades.

Chapter 15

Application of gRPC in FedLearn

As mentioned in the previous chapter, Remote Procedure Call (RPC) is a widely used technical solution in large projects. gRPC is a high-performance framework open-sourced by Google. This chapter delves into the application of gRPC in the FedLearn system using practical examples. We also explore the design rationale behind federated learning algorithms, engineering implementation, and security justification.

15.1 Application Example I: Vertical Federated Random Forest Algorithm

Next, we use the vertical random forest algorithm as an example, detailing what needs to be defined and filled within the gRPC framework. If you have questions about the principles and specific steps of the random forest algorithm, please refer to Chapter 4.

The overall architecture of the FedLearn federated learning platform is shown in Figure 15.1. Among them, we have the following:

Front is the user interface frontend responsible for visualization. It's the sole entry point for users, guiding them in interactions and operations.

Master is implemented in JAVA and serves as the controlling component. It oversees the overall framework's permission management,

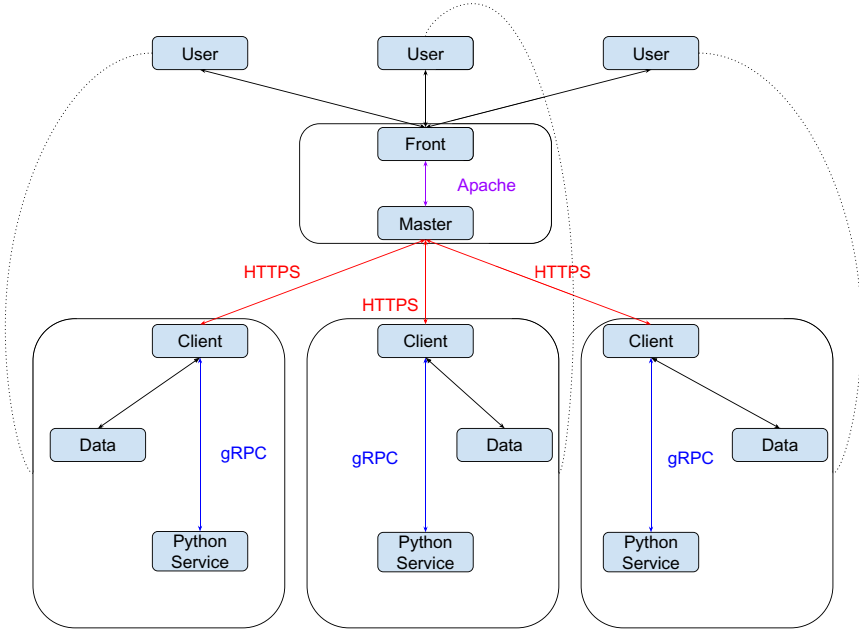


Figure 15.1. Simplified architecture of FedLearn.

external communication, and task scheduling. This includes user registration and login, task initiation and setup, connection management, database permissions, training execution, and communication between Clients.

Client is managed individually by each participating party and also implemented in JAVA. Its main responsibility is to manage local data settings and streamline the algorithm’s procedural execution.

Python Service is where the gRPC framework facilitates intra-machine cross-language invocation. Specifically, each deployed Client possesses a unique Python Service on their local machine. This service contains the implementation of most functions and facilitates calls to Python’s scientific computation tools, deep learning, etc. Therefore, most specific computational steps are not executed in the Client’s JAVA code but are performed by each Client through gRPC, invoking their respective Python Service. This module also offers runtime efficiency enhancement, CPU parallel optimization, and other

underlying performance improvements to cater to varying algorithmic and environmental demands.

Given the importance of gRPC, we next showcase its configuration, usage, and performance optimization processes using the vertical federated random forest algorithm as an illustration.

15.1.1 Overview of the Algorithm Process

Random forests are a popular machine learning algorithm that uses tree structures to partition feature spaces. By randomly selecting features, multiple independent decision trees are trained and used for predictions. Random forests can handle high-dimensional data without needing dimensionality reduction. They maintain model accuracy even when features are missing and can balance errors on imbalanced datasets, making them a widely applicable machine learning technique.

The vertical federated random forest algorithm introduced in this chapter retains all the benefits of a standalone random forest. To facilitate data exchange among multiple parties, the overall algorithm is partitioned into four sub-algorithmic steps executed across different parties, as illustrated in Figure 15.2.

15.1.1.1 Designing gRPC function interfaces

First, we need to define a ‘Calculate’ service to meet the needs of the four sub-algorithms in the random forest. For simplicity and generality, all gRPC invocations will utilize the same serialized data structure for input and output. This greatly simplifies the definition of the RPC function itself: All RPC functions will accept ‘InputMessage’ as



Figure 15.2. Overview of the FedLearn random forest algorithm process.

input and return ‘OutputMessage’ as output. Different stages only need to pick the corresponding names.

```
// All RPC functions are defined within the ‘Calculate’ Service.
// Based on the overview, we need to implement four different
  Algorithms, corresponding to the four input and output steps of
  the algorithm.
service Calculate {
  // Function calls for different Phases of the Random Forest
  algorithm
  // Phase1
  rpc RandomForestPhase1 (InputMessage) returns (OutputMessage) {}
  // Phase2
  rpc RandomForestPhase2 (InputMessage) returns (OutputMessage) {}
  // Phase3
  rpc RandomForestPhase3 (InputMessage) returns (OutputMessage) {}
  // Phase4
  rpc RandomForestPhase4 (InputMessage) returns (OutputMessage) {}
}
```

15.1.1.2 Interface scalability

Generally, the OutputMessage will have several different fields compared to the InputMessage. These extra fields might be used for Debug Information or Error Messages. However, theoretically, if we decide to eliminate debug information, the universal InputMessage format can be identical to the universal OutputMessage format, facilitating cascade calls.

```
// Universal input format
message InputMessage {
  repeated Matrix matrices = 1;
  repeated Vector vectors = 2;
  repeated double values = 3;
  repeated PaillierMatrix pailliermatrices = 4;
  repeated PaillierVector pailliervectors = 5;
  repeated PaillierValue pailliervalues = 6;
  PaillierKeyPublic paillierkeypublic = 7;
}
```

For most fields in InputMessage (excluding those declared with the “oneof” keyword or of the map type), we can use the repeated declaration to declare variables of the same format. Within gRPC, by declaring with repeated, variables of the same format become list-like, capable of storing any number of variables. This enhances

scalability when implementing different algorithms. At the same time, one can inquire about the number of same-format variables using the auto-generated `_size()` suffix function.

15.1.1.3 Plaintext and ciphertext transfers

The data content exchanged between multiple participants can be broadly classified into two categories: (1) data that don't require secrecy from other participants, such as user ID, sample dimensions, and the number of samples. Those data can be simply transmitted in plaintext. (2) Data like sample features across various participants and, crucially, the initiating party's sample labels. These are of paramount concern regarding privacy and must be encrypted during interactions. This ensures no participant possesses plaintext data they shouldn't have, preventing potential privacy breaches. In this instance, we utilize Paillier homomorphic encryption as our encryption choice during data transfers. In the interface definition, formats prefixed with Paillier are in encrypted form containing the public key.

```
// Format for Paillier homomorphic encryption public key
// The string type is used for easy conversion of large integers
message PaillierKeyPublic {
    string g = 1;
    string n = 2;
    string nsquare = 3;
}

// Format for ciphertext after Paillier encryption
message PaillierValue {
    string ciphertext = 2;
    string exponent = 3;
}
```

15.1.1.4 Data scale and dimensions

Common data dimensions in machine learning algorithms typically include zero dimensions (e.g., scalar values like sample counts), one dimension (e.g., sample labels), and two dimensions (e.g., feature matrices). In this instance, by leveraging the repeated keyword, we can expand predefined variable formats (like `paillierValue`) to

enhance support for serialized format dimensions. The related code content is as follows:

```
// Plaintext vector format
message Vector {
    repeated double values = 1;
}

// Plaintext matrix format
message Matrix {
    repeated Vector rows = 1;
}

// Encrypted vector format
message PaillierVector {
    repeated PaillierValue values = 1;
}

// Encrypted matrix format
message PaillierMatrix {
    repeated PaillierVector rows = 1;
}
```

15.1.1.5 Automated framework generation and logic populating

Upon defining the aforementioned interface format in the .proto file, we can utilize the io tools provided by gRPC¹ to automatically generate the RPC code framework. In this example, after executing the auto-generated code, we should get two files. One has the suffix pb2.py and contains all previously defined message types along with the implementation of related built-in functions. The other, with the suffix pb2_grpc.py, comprises the complete RPC function definitions and invocation methods.

Of course, what are auto-generated by the io tool are merely templates for the four function calls, from RandomForestPhase1 to RandomForestPhase4. The logic within these functions requires further elaboration; otherwise, they merely act as hollow functions that accept InputMessage as input and OutputMessage as output. Detailed implementation logic for each function can be referenced in Section 4.2.1.

¹<https://pypi.org/project/grpcio-tools/>.

15.2 Application Example II: Horizontal Federated Learning Scenario

15.2.1 Brief on Horizontal Federated Learning

The concept of federated learning was first introduced in 2017 and was initially tested on Gboard. Gboard is a quintessential smart recommendation application. One of its significant features is its ability to automatically suggest input content, enhancing the user's typing experience. Thus, a straightforward approach is to collaboratively use extensive user interaction information from numerous Android phones to train a shared global model. The pioneering federated learning technology was born under this premise, providing a technical solution for multi-party joint modeling that ensures privacy. It ensures that data from Android phone users, like keystrokes on Gboard, remain local, while the model continues to learn and update based on new user input data. In this modeling form, participating parties share the same feature space but different samples, hence termed horizontal federated learning. Two typical application scenarios are as follows: consumer-end data remain on devices (e.g., smartphone data) and business-end data stay within institutions (e.g., medical data).

The modeling method for horizontal federated learning generally includes two stages: (1) Each participant trains a local model based on their data and (2) a coordinating party (like data centers or parameter servers) integrates a shared global model based on local models received from participants and updates their local models. Efficient integration of a high-performance global model in horizontal federated learning has been extensively explored academically. In 2016, McMahan *et al.* proposed the FedAvg algorithm, where the coordinating party takes the size of each participant's dataset as weights and averages all local models to derive the global model. For a detailed theoretical proof of the convergence of the FedAvg algorithm, one can refer to the paper by Li *et al.* (2019). Sahu's FedProx algorithm introduced in 2018 added a proximal term to the local model's loss function. This term restricts the influence of local model updates on the global model, addressing, to some extent, the heterogeneity of datasets from multiple parties, making it more

suitable for scenarios where data from multiple parties are not identically distributed (Non-IID). Mohri *et al.*'s (2019) Agnostic Federated Learning (AFL) algorithm deduces an optimal multi-party data distribution through a composite analysis of participant data distributions. On the other hand, the FedMA algorithm by Wang *et al.* delves deeper into specific model network architectures, like convolutional neural networks and recurrent neural networks, updating one layer of the global neural network model at a time, reducing communication during the horizontal federated training process to enhance multi-party training efficiency.

15.2.2 Framework Design and Implementation of Horizontal Federated Learning in FedLearn

15.2.2.1 FedLearn horizontal federated learning framework

FedLearn has designed a unified horizontal federated learning framework with the aim to systematically integrate gradient-based linear and deep learning models, ensuring encryption. Figure 15.3

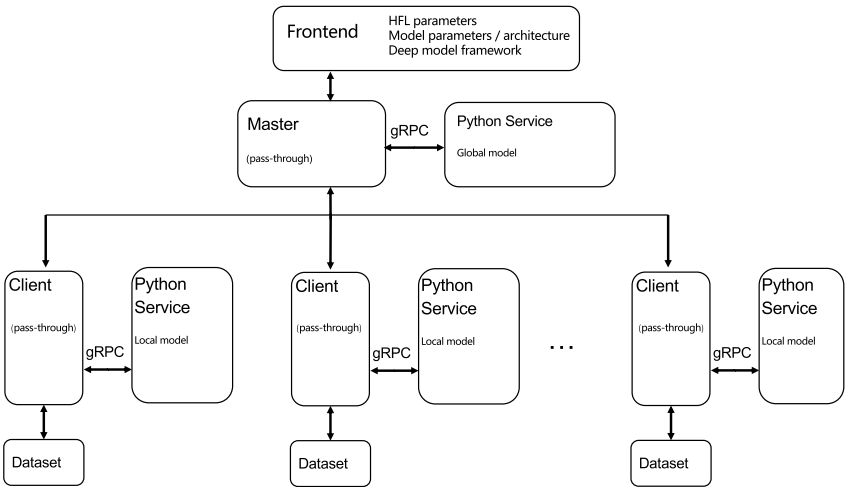


Figure 15.3. FedLearn horizontal federated learning framework.

illustrates the horizontal federated learning framework. This framework encompasses two types of Python services to accomplish the multi-party joint modeling in horizontal federated learning: (1) Training of the local model of the participants is done through Python services connected to the Participant Control End (Client) and (2) the integration of the global model by the Coordinating Control End (Master) is completed through its connected Python service. Under this framework, the entire process of horizontal federated learning multi-party joint modeling in FedLearn is as follows: Initially, the frontend configures the parameters of the horizontal federated algorithm, chooses the deep learning framework, sets the parameters and network architecture for participant models, and triggers multi-party joint modeling initiation by the Master. The Master relays the related parameters and configurations to the Clients, managing all participating parties. Clients load their local data and control their local Python service to train the local model through gRPC. After the local models of all participants are trained by their respective Client-side connected Python services, these models or gradients are passed to the Master through the Clients. The Master, connected to a Python computation service, then integrates the global model. After several iterations of local and global model training, when performance convergence is achieved, multi-party joint modeling is finalized.

The frontend configures parameters for the horizontal federated algorithm, selects the deep learning framework, sets participant model parameters and network architecture, and initiates multi-party joint modeling. The Master and Client implemented in Java mainly provide control scheduling, communication transmission, runtime monitoring, and data loading functions.

All algorithmic parts related to model training and inference are implemented within the Python service. In this service, the global and local models are relayed through the Master and Client. As most mainstream deep learning frameworks and machine learning libraries support Python, like TensorFlow, Pytorch, and sklearn, such an architectural design leverages Java's strengths in distributed system development and Python's prowess in machine learning and high-performance computing. This potentially supports a variety of models and frameworks seamlessly.

15.2.2.2 FedLearn horizontal federated learning control flow

The control flow diagram for FedLearn’s horizontal federated multi-party modeling training is presented in the following. The Java-implemented Master and Client communicate via the HTTPs protocol, ensuring reliable and secure remote data transfer between all participants. Java’s control over Python computation services is executed through the RPC gRPC. By leveraging the cross-platform advantages of RPC services, the system’s scalability, maintainability, and continuous delivery capability are enhanced, achieving high availability for the system.

Figure 15.4 illustrates the control flow of the entire FedLearn process for horizontal federated learning multi-party joint modeling. The specific steps are as follows: (1) The frontend triggers the Master to initiate multi-party joint modeling. The Master sends a TrainStart command to its controlled Python service. The Python service on the Master’s side initializes the global model, commences multi-party training, replies with the ModelTrain control command, and

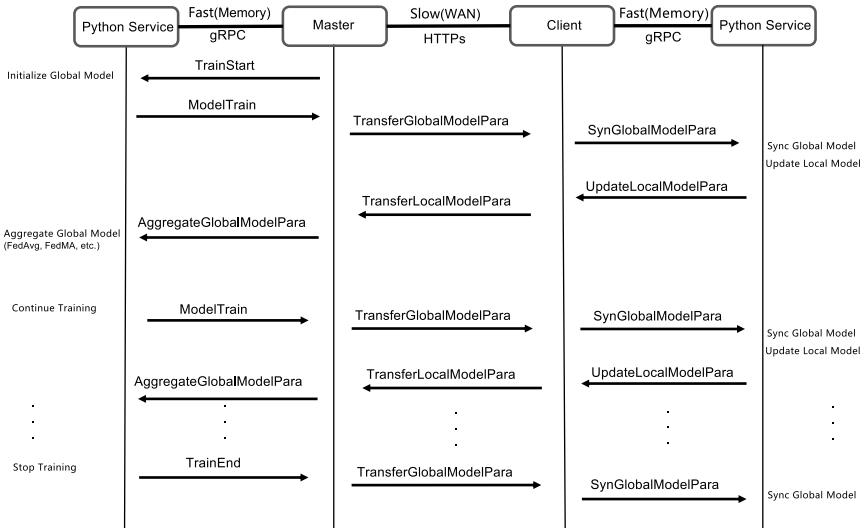


Figure 15.4. FedLearn horizontal federated learning training control flow.

and sends the global model to the Master. (2) Upon receiving the ModelTrain training command, the Master sends the TransferGlobalModelPara command to the Client, passing along relevant parameter configurations and the global model. (3) Once the Client receives the global model, it loads local data and sends the SynGlobalModelPara command to its controlled Python service. The Python service on the Client's side syncs the global model locally, uses the global model parameter weights to set up the local model, and trains a new local model using local data. (4) After training is complete on the participating parties' local models, the Python service sends the UpdateLocalModelPara command to the Client and transfers the local model. (5) The local model from the participants is then relayed to the Master through the Client using the TransferLocalModelPara control command. (6) After the Master receives the local models from the participants, they are sent to its Python service. The Master then issues the AggregatnGlobaeModelPara command to initiate the integration of the global model. The aggregation task is accomplished using the FedAvg algorithm, completing one round of multi-party joint training. (7) The Python service on the Master's side then decides whether to continue multi-party training or to halt based on certain criteria. If the training is to continue, Steps 2–6 are repeated. If the conditions to stop the training are met, such as the global model performance converging after multiple iterations or reaching the iteration limit, the Python service on the Master's side issues the TrainEnd command to halt the multi-party training, while also syncing the global model with all modeling participants.

Listed in the following are the main control commands used in the FedLearn horizontal federated multi-party modeling process. Both local and global models in the Python service are passed transparently through the Client and Master without any modifications. Thus, there are only two control commands for model transmission between the Master and Client: TransferGlobalModelPara and TransferLocalModelPara. Interactions between the Master/Client and the Python service involve various commands to fulfill two-phase multi-party modeling tasks. On the Client side, the main tasks are syncing the global model and updating the local model, resulting

in interaction commands like `SynGlobalModelPara` and `UpdateLocalModelPara`. On the Master side, the primary tasks are collecting individual local models, integrating the global model, and controlling the beginning and end of training. Thus, interaction commands with the Python service include `SynLocalModelPara`, `AggregatnGlobaeModelPara`, `TrainStart`, `ModelTrain`, and `TrainEnd`.

```
public enum HorizontalZooMsgType {
    /*Master <=> Client*/
    TransferGlobalModelPara("TransferGlobalModelPara"),
    TransferLocalModelPara("TransferLocalModelPara"),

    /*Master/Client => gRPC service*/
    TrainStart("TrainStart"),
    SynGlobalModelPara("SynGlobalModelPara"),
    SynLocalModelPara("SynLocalModelPara"),
    AggregatnGlobaeModelPara("AggregatnGlobaeModelPara"),

    /*gRPC service => Master/Client*/
    UpdateLocalModelPara("UpdateLocalModelPara"),
    ModelTrain("ModelTrain"),
    TrainEnd("TrainEnd ")
}
```

15.2.2.3 gRPC transmitted message payloads

Having introduced the control aspect of horizontal federated learning in FedLearn, we now move on to the data transmitted by FedLearn. In horizontal federated learning, all datasets of the participating parties remain local, ensuring that there's no exchange of training data between the parties and the coordinator. This technically guarantees data privacy. The Python service is deployed locally on both the Master and Client. Control over the Python computational service by the Master and Client is achieved through local gRPC. Data transmitted via gRPC has no specific privacy requirements. The following table lists the messages transmitted by gRPC in horizontal federated learning, including control commands, participant information, horizontal federated learning algorithm parameters, model name, model hyper-parameters, model parameter weights, dataset features, and dataset labels. The model parameter weights utilize the byte type (bytes) to support various network architectures and model parameter sizes uniformly through gRPC byte stream transfer.

```
//horizontal FL message
message HFLModelMessage {
    string CommandMsg = 1;           // Control Command
    string ModelName = 2;           // Model Name
    string ClientInfo = 3;          // Participant Information
    string HFLParameter = 4;        // Horizontal Federated Learning
        Algorithm Parameters
    string ModelHyperParameter = 5; // Model Hyperparameters
    bytes ModelBytes = 6;           // Model Parameter Weights
    Matrix matrices = 7;            // Dataset Features
    Vector vectors = 8;             // Dataset Labels
}
```

15.2.3 Utilizing gRPC to Support Diverse Model Types

To ensure the universality of the horizontal federated framework, one of the design objectives of FedLearn is to seamlessly integrate various linear models (such as linear classifiers and regressors) and deep neural network models (like convolutional neural networks and recurrent neural networks) within the horizontal federated learning framework using encryption-supported means and based on horizontal federated algorithms (e.g., FedAvg and FedMA). So, how does FedLearn support different types of machine learning models? Specifically, there are three design considerations to guarantee the generality of FedLearn's horizontal federated framework, making it compatible with multiple deep learning frameworks and machine learning libraries (like TensorFlow, Pytorch, sklearn, and Oneflow) and various model types: (1) uniform command dispatching; (2) designing inheritance relationships among diverse model classes; (3) harnessing gRPC byte stream transmissions to support model parameter transfers across different network frameworks.

15.2.3.1 Unified command dispatching on the Python side

To achieve the universality of the horizontal federated learning framework and support different models, a unified command dispatching and handling according to Figure 15.4 was implemented in Python, as depicted in Figure 15.5. Although models vary, the multi-party

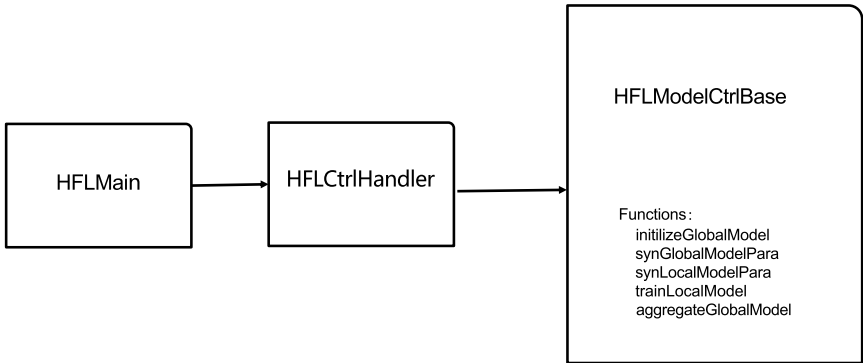


Figure 15.5. Unified command dispatching for FedLearn’s horizontal federated learning service in Python.

training process in horizontal federated learning remains consistent. Specifically, every model’s training control requires handling the same set of commands and providing identical command handling functions, such as initializing the global model, syncing the global model to the Client, syncing the local model to the Master, training the local model, and integrating the global model. This unified command dispatch and handling ensures FedLearn’s universality and scalability, enabling standardized integration of various models based on horizontal federated algorithms.

15.2.3.2 Inheritance relationships of models on the Python side

FedLearn serves as a universal and secure distributed machine learning platform, supporting a range of machine learning models (like linear classification and regression models) and deep learning models (such as multi-layer neural networks, convolutional neural networks, recurrent neural networks, and BERT). Figure 15.6 illustrates the inheritance relationships among models in FedLearn’s horizontal federated learning service in Python. These model control classes inherit from the foundational horizontal federated learning control class and override respective message handling functions, implementing specific command handling functions according to their own network structures and usage interfaces, including initialize the global model,

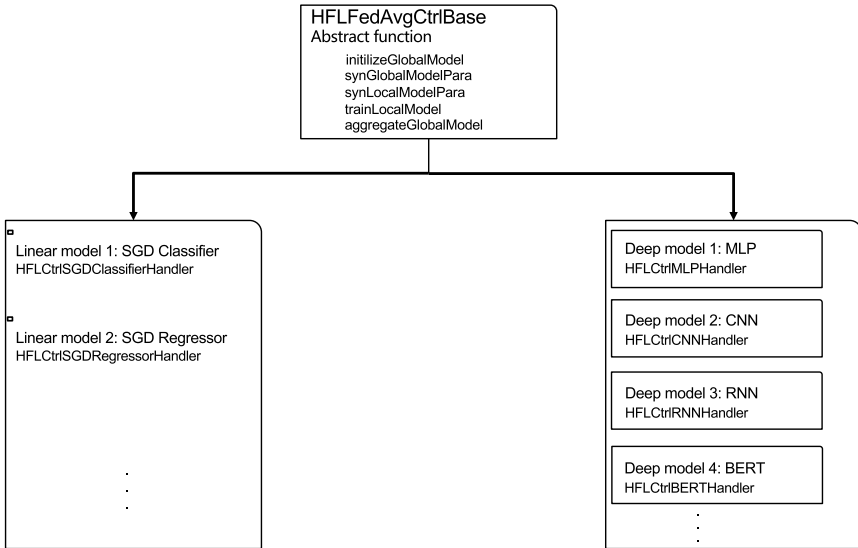


Figure 15.6. Inheritance relationships of models in FedLearn’s horizontal federated learning service in Python.

synchronize the global model, synchronize the local model, train the local model, and integrate the global model.

15.2.3.3 Supporting model parameter transmission across different network frameworks through gRPC byte streams

Given the variability in network structures and parameter quantities of different models — from simple linear models with a single layer to complex ones like BERT with hundreds of megabytes of parameters — this diversity manifests within the FedLearn’s Python service. For both the coordinating control side (Master) and participating control side (Client), varying models translate into byte streams of different sizes, as displayed in Figure 15.7. By serializing different model instance parameters into byte streams and utilizing gRPC’s cross-language byte stream transmission, the Python instances are transparently passed to the Java control side (both Master and Client), facilitating the transfer of local and global models between the coordinator and participant Python services.

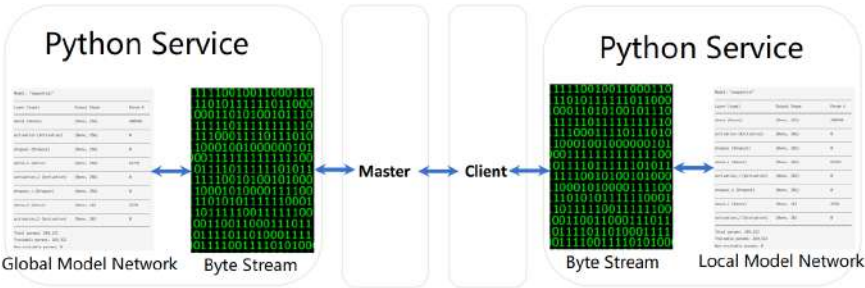


Figure 15.7. FedLearn’s horizontal federated learning transmitting various model parameters via gRPC byte streams.

FedLearn offers a universal and secure distributed machine learning platform. All computational services and model training within the FedLearn horizontal federated learning framework are executed on the Python side. This ensures that FedLearn’s horizontal federated framework is compatible with leading domestic and international deep learning frameworks (like TensorFlow, PyTorch, and OneFlow) and various machine learning libraries (such as sklearn). By employing unified command dispatching, designing various model class inheritances, and leveraging gRPC byte streams for transmitting model parameters across different network frameworks, FedLearn’s horizontal federated learning framework confidently supports a diverse range of models.

Chapter 16

Performance Optimization Practices in Real-World Scenarios

16.1 Introduction to FedLearn Business Scenarios

16.1.1 Precise Marketing Monitoring of Financial Products

Originating from the analysis of sales data of internet retailers, the “Long Tail Theory” suggests that traditionally businesses would focus on their major customers or significant matters, putting their main emphasis on mid to high-end clients in hopes of reducing operational costs and increasing profits. Banks often neglected services for small- and medium-sized enterprises, personal loans, and regular clients, thinking that allocating excessive resources and attention to these “long tail” customers would only increase their costs. However, with the maturation and application of technologies like artificial intelligence and big data, utilizing the vast amount of data and varied data types produced from daily human activities can assist banks in effectively identifying potential customers and enhancing the quality of marketing. By collaborating with large data companies, banks are pivoting toward a more data-driven and intelligent approach in new business development and maintaining existing customers, thus improving the quality of their “long tail” clientele.

Employing federated learning technology ensures data privacy at a technical level. This allows cross-domain learning algorithm

modules to make effective use of various types of data without the need for original data to be shared. For instance, banks can cooperate with risk control companies, align customer identification codes in encrypted forms, and identify mutual customers in this encrypted state. Then, extracting pertinent features, they can employ federated random forests mentioned previously for cross-domain learning and joint modeling without directly interacting with the raw data. This global perspective on multi-source data helps in model construction. Through such methods, banks can collaborate efficiently with various data institutions, addressing core challenges like high-quality enhancement costs for “long tail” customers, difficulty in tapping potential customers, and the problem of incomplete information making user profiling challenging.

16.1.2 Intelligent Credit Scoring

Internet consumer finance refers to small loans provided by online financial companies, allowing individual consumers to repay in installments for goods and services. When dealing with tens or even hundreds of thousands of loan applications, these platforms need to conduct comprehensive checks and evaluations of the borrower’s credit to mitigate lending risks. In the construction of a social credit system, the credit scoring of individuals and enterprises is paramount. By integrating data from various departments and achieving collaborative monitoring and assessment of trust-breaking behaviors, it plays a crucial role in refining the social credit atmosphere, ensuring the successful establishment and efficient functioning of the social credit system. Accurate and timely detection of credit anomalies in cities, understanding the reasons for these anomalies, predicting future trends, and guiding decision-making processes for governments, businesses, and individuals are essential for maximizing the value of the urban credit system.

The credit evaluation of individuals and enterprises is a significant part of building a social credit system. This evaluation, done by integrating data from commerce, judiciary, taxation, and consumption sectors and using historical credit records, extracts feature variables from various dimensions. AI models constructed

through data analysis techniques then compute these evaluations. The result — credit scores — is vital in promoting honesty and law-abiding behavior among individuals and businesses, fostering a trustful social atmosphere. Moreover, combining enterprise commercial data with external and proprietary data, one can unearth connections between businesses, and between businesses and individuals. These relationships, or paths of association between two companies or a company and an individual, can be used to investigate hidden risks and are applicable in lending scenarios like pre-loan audits, post-loan monitoring, and contact restoration. Algorithms that detect temporal and spatial anomalies can identify when and where credit anomalies appear in cities, allowing for an in-depth analysis of causes, aggregation of different timelines, and places of individual and corporate credit data to predict future trends. This maximizes the value of the urban credit system and guides the decision-making processes of governments, businesses, and individuals.

16.2 From 0-1: Practical Optimization of Federated Learning Algorithms

Homomorphic encryption inherently demands high computational overhead. Consequently, federated learning models that leverage homomorphic encryption as a privacy-preserving mechanism often require prolonged training durations. Given the data scale of our practical scenarios, a complete training process can span tens of hours, if not more. This duration doesn't even account for potential interruptions due to unexpected anomalies. Excessively long training times significantly impede the potential for practical applications of federated learning. On one hand, the cost of model tuning and optimal parameter selection skyrockets, often leading to suboptimal implementations. On the other, allocating personnel time becomes challenging, making it difficult to support scenarios requiring frequent model iterations. Given this backdrop, the training time for vertical federated learning algorithms becomes a crucial performance metric that warrants iterative optimization.

Next, using the federated random forest algorithm from the Fedlearn platform as an example — specifically, its application

in a real-world intelligent risk control scenario — we outline the challenges faced during implementation and the subsequent practices to optimize the algorithm’s performance.

Our model training data originates from a genuine credit risk control dataset (referred to hereafter as the ‘business dataset’). This dataset comprises approximately 500,000 user samples, with over 100 features describing aspects, such as credit behavior, purchase frequency, merchant distribution, and concurrent borrowing. These features are distributed among various stakeholders, ensuring no cross-disclosure. The dataset’s labels follow a binary 0-1 format, indicating user delinquency. To evaluate the model’s generalization capabilities, the dataset was partitioned into three subsets: Training (Train), Random Testing (Test), and Out-Of-Time testing (OOT). While the Train and Test samples hail from the same timeframe, used for model training and hyperparameter selection, OOT acts as a unique split for simulating ‘the future’ and gauging the model’s generalization capacity. Our primary concerns are twofold: assessing the quality of model predictions, utilizing metrics like Area Under the ROC Curve (AUC) and Kolmogorov–Smirnov Statistics (KS), and focusing on the model’s efficiency, i.e., the time taken for a complete training cycle. Tables 16.1 and 16.2 respectively illustrate the initial testing efficiency and inference metrics of the federated random forest algorithm on the business dataset.

Table 16.1. Initial training efficiency of the federated random forest algorithm on the business dataset. These results serve as benchmarks for subsequent algorithm iterations.

Parameters	Training time (Intranet ^a)	Training AUC	Training KS
Tree count = 50 Tree depth = 10 Samples per tree = 20,000 Features per tree = 20	35 Hours	0.635	0.196

^aThe data transmission speed is very fast in the intranet environment, so the training time part can be considered as not calculating the time required for data transmission, and the actual modeling time will vary according to the quality of network communication

Table 16.2. Inference metric results of the federated random forest algorithm on the business dataset. These results will serve as benchmarks for future algorithm iterations.

Dataset	Training		Testing		OOT	
Algorithm	AUC	KS	AUC	KS	AUC	KS
Federated random forest	0.635	0.196	0.615	0.167	0.612	0.161
SecureBoost	0.643	0.203	0.641	0.205	0.622	0.173

16.3 Performance Optimization

Performance bottlenecks were identified in the federated random forest algorithm. Two primary areas for optimization in time overhead emerged. First, common homomorphic encryption computational operations, where the speed of encrypted homomorphic addition is several orders of magnitude slower than its plaintext counterpart. This difference in computation speed significantly inflates the total time cost when federated learning algorithms are compared to conventional machine learning models. Second, in engineering services, computational resource utilization offers a substantial room for improvement, especially for parallelizable operations. Given that machine learning algorithms often entail distributed training — like the individual tree modeling in random forests that’s independent from other trees — distributed modifications during the modeling process can effectively harness machine resources, thereby enhancing training efficiency.

Considering the Paillier encryption scheme, a widely used homomorphic encryption method, as an example for adding two integers: Table 16.3 presents the time consumption when using a 1,024-bit key length with the Paillier protocol. As evident, plain addition merely takes 7–10 nanoseconds. In contrast, the Python-side addition takes about 39.8 microseconds, while the encrypted homomorphic addition and multiplication require time thousands or even tens of thousands of times more than the plaintext. Java-side addition computation reduces this time consumption to 12.8 microseconds, but the time taken for encrypted computations still remains hundreds to thousands of times greater than for plaintext operations.

Table 16.3. Comparison of time consumption for addition and scalar multiplication in plaintext and Paillier encrypted ciphertext states. Values in the table are derived from the average of 1,000 operations, with addition computed as $1+2$ and scalar multiplication as $10 * 1$ (plaintext) and $10 * [[1]]$ (ciphertext). Due to the unique characteristics of ciphertext computation, the size of plaintext numbers and the key length significantly influence the computational complexity and duration. Here, we use the simplest operations for illustrative purposes.

Operation	Avg time for 1 Python calculation	Avg time for 1 Java calculation
Plaintext addition	$7.15 \times 10^{-9}s$	$10.10 \times 10^{-9}s$
HE ciphertext addition	$3.98 \times 10^{-5}s$	$1.28 \times 10^{-5}s$
Time ratio (Ciphertext/Plaintext)	5.56×10^3s	2.38×10^2s
Plaintext scalar multiplication	$7.22 \times 10^{-9}s$	$1.05 \times 10^{-8}s$
HE ciphertext and plaintext scalar multiplication	$1.51 \times 10^{-4}s$	$6.87 \times 10^{-4}s$
Ratio (Ciphertext/Plaintext)	20,914	3,085

In the following, we conduct a brief analysis of the operational details related to homomorphic encryption, showcasing the bottlenecks within the computation process to propose optimization solutions. Based on the principles of the Paillier encryption scheme, the most crucial step in the ciphertext addition operation is multiplying two large integers together, and then taking the modulus of the result by another large integer. With a key length of 1,024 bits, the two large integers used for multiplication usually have a few hundred bits, while the modulus operation’s large integer is typically between 1,000 and 2,000 bits:

$$Enc(a + b) = (Enc(a) * Enc(b)) \mod n^2$$

Thus, the key to accelerating homomorphic addition lies in enhancing the computational speed of this operation. Similarly, analyzing the scalar multiplication operation of the Paillier encryption scheme reveals its computational bottleneck mainly stems from the following exponentiation and modulus operation:

$$Enc(a) * b = (Enc(a)^b) \mod n^2$$

Most programming languages have implemented this operation in their mathematical operators, like the commonly used *powmod* function in Python. However, our comparison reveals that Python's *powmod* implementation is relatively inefficient, as can be observed in Table 16.3. In the subsequent sections, we introduce high-performance computing libraries that can enhance the efficiency of these two computations.

16.3.1 GMP Computing Library

As inferred from the aforementioned analysis, cryptographic computations, including the encryption and decryption processes of homomorphic encryption and its addition operation, involve a plethora of high-precision multiplication, division, and modulus operations. In federated learning algorithms based on homomorphic encryption, a substantial portion of the training time is consumed by high-precision operations. Optimizing the efficiency of these high-precision operations can significantly reduce the training time for federated learning algorithms.

GMP, standing for the GNU Multiple Precision Arithmetic Library, is an open-source library designed for arbitrary precision arithmetic operations. It supports integers, rational numbers, and floating-point numbers of both positive and negative values. The precision of GMP is only limited by the available memory. It is predominantly used in fields like cryptographic applications and research, internet security applications, algebraic systems, and computational algebra research, among others. In this section, we briefly discuss how to leverage the GMP library to speed up homomorphic encryption operations under the Paillier encryption protocol.

16.3.2 Accelerating Homomorphic Encryption Computations with the GMP Library in Python

In the current Python ecosystem, GMP has open-source support via *gmpy2*. Hence, we can use the corresponding functions in *gmpy2* to carry out multiplication and modular exponentiation operations. It's worth noting that *gmpy2* does not provide a direct function for modular multiplication. We need to manually call multiplication and

modulo functions to achieve the modular multiplication, as demonstrated in Listing 16.1:

Listing 16.1. GMP modular multiplication.

```
1 import gmpy2
2 _USE_MOD_FROM_GMP_SIZE = (1 << (8*2))
3 // Define multimod function
4 def multimod(a, b, c):
5     if max(a, b, c) < _USE_MOD_FROM_GMP_SIZE:
6         return a * b % c
7     else:
8         return int(gmpy2.c_mod(gmpy2.mul(a, b), c))
```

For modular exponentiation, gmpy2 offers a direct interface via the powmod function, which is not explored in detail here.

Tables 16.4 and 16.5 display the computation times for homomorphic encryption ciphertext addition and scalar multiplication based on the Paillier encryption scheme after leveraging the GMP library. It is evident that the GMP library has significantly enhanced the computation speed.

16.3.2.1 Accelerating homomorphic encryption computations with the GMP library in Java

Given that GMP is developed in C, it can't be straightforwardly referenced in a Java environment. Here, we must exploit certain features of the Java language to invoke the GMP computational library.

Table 16.4. Time comparison for the ciphertext addition $[[1]] + [[2]]$ under the Paillier homomorphic encryption scheme in Python. By accelerating the modular multiplication, we've achieved about a 6-fold speedup for ciphertext addition, bringing the computation time down to roughly 14% of the original.

Operation	Avg time for 1 Python calculation	Time ratio (Ciphertext/ Plaintext)
Plaintext addition	$7.15 \times 10^{-9}\text{s}$	1
Homomorphic ciphertext addition	$3.98 \times 10^{-5}\text{s}$	5,566
Ciphertext addition with GMP	$4.10 \times 10^{-6}\text{s}$	811

Table 16.5. Time comparison for the ciphertext scalar multiplication $10^{*}[[1]]$ under the Paillier homomorphic encryption scheme in Python. By accelerating modular exponentiation, we’ve achieved about a four-fold speedup for ciphertext scalar multiplication, bringing the computation time down to roughly 20% of the original.

Operation	Avg time for 1 Python calculation	Time ratio (Ciphertext/ plaintext)
Plaintext scalar multiplication	$7.22 \times 10^{-9}\text{s}$	1
Homomorphic ciphertext scalar multiplication	$1.51 \times 10^{-4}\text{s}$	20,914
Ciphertext scalar multiplication with GMP	$2.80 \times 10^{-5}\text{s}$	3,919

One approach is to utilize the Java Native Interface (JNI) to interface with GMP. However, this approach requires Java developers to write some C code, making the process slightly cumbersome. Another method is to use Java Native Access (JNA). JNA is a community-developed library that enables Java applications to access native shared libraries without relying on the JNI. JNA is designed to provide native access with minimal effort and without the need for boilerplate or “glue” code. With JNA, Java developers can interface with GMP directly using Java code, eliminating the need for additional C language development.

Here, we showcase how to invoke the GMP interface in Java for modular exponentiation operations based on an open-source project named `jna-gmp` (<https://github.com/square/jna-gmp>) (see Listing 16.2 and 16.3):

Listing 16.2. Java invoking GMP utility class `LibGMP`.

```
1 import com.sun.jna.Native;
2 import com.sun.jna.NativeLibrary;
3 import com.sun.jna.NativeLong;
4 import com.sun.jna.Pointer;
5 public class LibGMP {
6     static class SizeT8 {
7         static native void _gmpz_import(mpz_t rop, int count, int order,
8             int size, int endian, int nails, Pointer buffer);
9         static native Pointer _gmpz_export(Pointer rop, Pointer countp,
10             int order, int size, int endian, int nails, mpz_t op);
```

```

11  }
12
13  public static void _gmpz_import(mpz_t rop, int count, int order,
14      int size, int endian, int nails, Pointer buffer) {
15      SizeT8._gmpz_import(rop, count, order, size, endian, nails,
16          buffer);
17  }
18
19  // Load the GMP library. Depending on the system and environment,
    different versions of the GMP library are loaded. For MAC, it
    is "gmp".
20  static {
21      loadlib("gmp");//Method for invoking the MAC version of GMP.
22      //loadlib("libgmp");//Method to call libgmp library installed
    on Linux.
23      //loadlib("libgmp-10");//Method for Windows users who have
    compiled the GMP library using mingw32.
24  }
25  private static void loadlib(String name) {
26      try {
27          NativeLibrary lib = NativeLibrary.getInstance(name,
28              LibGMP.class.getClassLoader());
29          Native.register(LibGMP.class, lib);
30          Native.register(SIZE_T_CLASS, lib);
31      } catch (Exception e) {
32          e.printStackTrace();
33      }
34  }
35
36  // GMP's powmod function.
37  public static native void _gmpz_powm(mpz_t rop, mpz_t base, mpz_t
    exp,
38      mpz_t mod);
39  public static native void _gmpz_powm_sec(mpz_t rop, mpz_t base, mpz_t
    exp,
40      mpz_t mod);
41  }

```

Listing 16.3. Java calls calculation class GMP.

```

1  import com.sun.jna.Native;
2  import com.sun.jna.NativeLibrary;
3  import com.sun.jna.NativeLong;
4  import com.sun.jna.Pointer;
5  public class GMP {

```

```

6  public static final ThreadLocal<GMP> INSTANCE = new ThreadLocal
    <GMP>() {
7      @Override protected GMP initialValue() {
8          return new GMP();
9      }
10 }
11
12 // Calculation function modPow
13 public static BigInteger modPow(BigInteger base, BigInteger exp,
14     BigInteger mod) {
15     return INSTANCE.get().modPowSecureImpl(base, exp, mod);
16 }
17 private BigInteger modPowSecureImpl(BigInteger base, BigInteger exp,
18     BigInteger mod) {
19     mpz_t basePeer = getPeer(base, sharedOperands[0]);
20     mpz_t expPeer = getPeer(exp, sharedOperands[1]);
21     mpz_t modPeer = getPeer(mod, sharedOperands[2]);
22     LibGMP._gmpz_powm_sec(sharedOperands[3], basePeer, expPeer,
23         modPeer);
24     int requiredSize = (mod.bitLength() + 7) / 8;
25     return new BigInteger(mpzSgn(sharedOperands[3]),
26         mpzExport(sharedOperands[3], requiredSize));
27 }

```

It's worth noting that for the Paillier encryption protocol, some open-source projects that implement the Paillier encryption protocol, such as javallier, have already integrated the use of JNA to invoke GMP.

Tables 16.6 and 16.7 list the computation times for homomorphic encryption ciphertext addition and scalar multiplication based on the

Table 16.6. Time comparison for homomorphic encryption ciphertext addition $[[1]] + [[2]]$ based on the Paillier homomorphic encryption scheme in a Java environment. By accelerating the modular multiplication, the computation time for ciphertext addition is approximately reduced to about 20% of the original, speeding up the operation by about four times.

Operation	Avg time for 1 Java calculation	Time ratio (Ciphertext/ Plaintext)
Plaintext addition	1.01×10^{-8} s	1
HE ciphertext addition	1.28×10^{-5} s	1,267
Ciphertext addition with GMP	2.41×10^{-6} s	238

Table 16.7. Time comparison for homomorphic encryption ciphertext scalar multiplication $10^{*}[[1]]$ based on the Paillier homomorphic encryption scheme in a Java environment. By accelerating the modular exponentiation, the computation time for ciphertext scalar multiplication is approximately halved, speeding up the operation by about 2 times.

Operation	Avg time for 1 Java calculation	Time ratio (Ciphertext/ Plaintext)
Plaintext scalar multiplication	$1.05 \times 10^{-8}s$	1
Homomorphic ciphertext scalar multiplication	$6.87 \times 10^{-5}s$	6,542
Ciphertext scalar multiplication with GMP	$3.24 \times 10^{-5}s$	3,085

Paillier encryption scheme in a Java environment after acceleration using the GMP library. Similar to the results in the Python environment, the GMP computational library also provides significant improvement in the Java environment.

After optimization with the GMP computational library, when modeling using the federated random forest algorithm on our business dataset, the modeling time was reduced from several tens of hours to approximately 7 hours and 12 minutes.

16.3.3 Homomorphic Encryption Computation Protocol Optimization

Given the results from the previous section, although the GMP significantly accelerates the computation of homomorphic encryption, the overall efficiency of algorithm training is still not up to par for everyday use. A primary reason is that the computation speed of the homomorphic encryption based on the Paillier protocol is relatively slow: On one hand, as a homomorphic encryption protocol, its homomorphic addition requires numerous large integer multiplications, exponentiations, and modular operations. On the other hand, because of the consistency of its keys, each encryption requires a computationally intensive obfuscation operation (Obfuscate) to ensure that encrypting the same plaintext yields different ciphertexts. This somewhat constrains the computational efficiency ceiling of the Paillier homomorphic encryption protocol.

Thus, choosing an efficient homomorphic encryption approach becomes crucial for further optimizing the federated learning algorithm based on homomorphic encryption. For example, in the 2020 paper by Fang *et al.* (2018), the authors proposed using the Okamoto–Uchiyama protocol (referred to hereafter as the OU protocol) to significantly reduce computation time. In this section, we briefly introduce the principles, implementation, and computational efficiency comparison of the RandomizedIterativeAffine protocol (abbreviated as RIAC) implemented on the FedLearn platform.

The RIAC protocol is based on the common affine cipher, which encrypts plaintext by linear transformation followed by modular operation and decrypts ciphertext using the multiplicative inverse for linear transformation followed by modular operation (as shown in Table 16.8). Due to its linear transformation nature, the affine cipher inherently supports homomorphic addition and scalar multiplication operations. As the affine cipher can be easily decrypted through frequency analysis, the RIAC enhances its security in two ways: by employing multi-round affine encryption, using different key lengths each time, and by introducing a random number in each encryption, ensuring that encrypting the same number yields different outcomes.

Tables 16.9 and 16.10 compare the time consumption of 1,000 random number homomorphic encryption computations using the Paillier protocol and the RIAC protocol in FedLearn. In the Python environment, the RIAC protocol consistently outperforms the Paillier protocol in terms of encryption, decryption, homomorphic addition, and scalar multiplication. In the Java environment, while encryption and decryption speeds between the RIAC and Paillier protocols are relatively close, the RIAC protocol still offers over a tenfold improvement in homomorphic addition.

Table 16.8. Encryption and decryption scheme of RIAC, where a and n are coprime large integers and a^{-1} is the multiplicative inverse of a modulo n , i.e., $1 = aa^{-1} \bmod n$.

Operation	Calculation
Encryption	$E(x) = (ax + b) \bmod n$
Decryption	$D(x) = a^{-1}(x - b) \bmod n$

Table 16.9. Computational efficiency comparison between Paillier and RIAC encryption protocols in a Python environment. The computation time is the total time for calculating 1,000 random numbers drawn from $N(0, 1e8)$, where homomorphic addition is the sum of 1,000 ciphertexts.

Operation	Paillier protocol	RIAC protocol	Time ratio (Paillier/RIAC)
Encryption	1.33×10^0 s	8.03×10^{-2} s	16.6
Decryption	3.73×10^{-1} s	2.09×10^{-2} s	17.8
Homomorphic addition	1.44×10^{-2} s	1.51×10^{-3} s	13.2
Homomorphic scalar multiplication	3.15×10^{-2} s	1.88×10^{-3} s	5.90

Table 16.10. Computational efficiency comparison between Paillier and RIAC encryption protocols in a Java environment. The computation time is the total time for calculating 1,000 random numbers drawn from $N(0, 1e8)$, where homomorphic addition is the sum of 1,000 ciphertexts.

Operation	Paillier protocol	RIAC protocol	Time ratio (Paillier/RIAC)
Encryption	1.30×10^{-2} s	2.88×10^{-2} s	0.45
Decryption	6.54×10^{-1} s	3.12×10^{-1} s	2.10
Homomorphic addition	2.00×10^{-2} s	1.62×10^{-3} s	12.35
Homomorphic scalar multiplication	1.11×10^{-2} s	2.16×10^{-3} s	5.14

Using the federated random forest algorithm on our business dataset, with algorithm parameters set at 50 trees, a depth of 10, and 20,000 samples randomly drawn for each decision tree, the overall modeling time was reduced from the previous 7 hours and 12 minutes to approximately 3 hours and 30 minutes. The RIAC encryption protocol provided about a twofold speed increase for the federated random forest algorithm based on homomorphic encryption on the FedLearn platform. Thus, the time required for the federated random forest algorithm modeling finally aligns with the usual working hours of an engineer, indicating preliminary feasibility for practical deployment.

16.4 Optimization of Engineering Service Performance

Optimization in engineering is an indispensable step when deploying machine learning models. On the one hand, machine learning models deal with the processing of massive data and complex computations. Inefficient engineering implementations can significantly reduce the efficiency of the entire machine learning workflow, affecting aspects, such as model tuning, iteration, and periodic retraining. On the other hand, almost all machine learning models inherently have the potential for parallel computation, including steps like parallelizable tree building in random forests and parallelizable matrix operations in deep learning. Proper engineering optimization, especially parallel optimization, can greatly enhance the productivity of algorithm researchers and business personnel. Federated learning, as a branch of distributed learning, also needs to account for the impact of data communication on the entire federated learning workflow. In this section, we use the federated random forest algorithm as an example to discuss how to optimize federated learning algorithms from an engineering perspective, focusing on parallelization and communication, to meet the requirements of better online services.

The FedLearn platform offers two deployment options: a performance-optimized version combining Java and Python, and a pure Java version. Therefore, we start with parallelization, discussing our parallel attempts in both Java environments and Python's gRPC services. We then introduce some optimizations in communication and conclude with potential directions for engineering enhancements that we believe can improve performance on the FedLearn platform.

16.4.1 Parallel Optimization

In the federated random forest algorithm, the modeling process for each decision tree is independent of others, allowing us to model each tree concurrently. For instance, when selecting the current splitting child nodes, we can pick a splitting node in each tree and then simultaneously compute the node's features and execute the splitting operation. Furthermore, many calculations in the algorithm, such as homomorphic encryption of all sample labels, can be performed in parallel. This corresponds to a viewpoint verified in many big data

projects: Parallel transformation can significantly enhance the algorithm's capability to handle large data.

16.4.1.1 Parallel optimization in Java

Parallelization in Java requires a certain finesse. Fortunately, the Stream API introduced in Java 8 aids us in easily performing some parallel operations. These operations generally meet the needs of federated learning model building. In the following, we provide an example from the FedLearn platform of using the Stream API for homomorphic encryption in the federated learning forest algorithm (see Listing 16.4). The specific usage of Streams is beyond the scope of this article, but interested readers can explore Java learning resources for further information.

Listing 16.4. Parallel RIAC homomorphic encryption in Java.

```
1 // Sequential encryption
2 public static IterativeAffineCiphertext[] IterativeAffineEncryptSingleProcess(
3     Vector vec, IterativeAffineKey key) {
4     int n = vec.getValuesCount();
5     IterativeAffineCiphertext[] result = new IterativeAffineCiphertext[n];
6     for (int i = 0; i < n; i++) {
7         result[i] = key.encrypt(vec.getValues(i));
8     }
9     return result;
10 }
11
12 // Parallel encryption
13 public static IterativeAffineCiphertext[] IterativeAffineEncrypt(Vector vec,
14     IterativeAffineKey key) {
15     int n = vec.getValuesCount();
16     IterativeAffineCiphertext[] result = new IterativeAffineCiphertext[n];
17     IntStream.range(0, n).parallel()
18         .forEach(index -> {
19             result[index] = key.encrypt(vec.getValues(index));
20         });
21     return result;
22 }
```

While the Stream API can significantly boost efficiency in a single-machine modeling process, there are still parts in Java-based

modeling that aren't easily accelerated using this API. Moreover, parallel computation for massive data is a critical consideration. The FedLearn platform is also experimenting with integrating distributed computation platforms, such as Spark clusters, and the outcomes of this integration are eagerly anticipated.

16.4.1.2 Parallel optimization in Python

As Python services are provided in the form of gRPC services, there are several choices when it comes to optimizing parallelism in Python services. If there are multiple nodes available to provide gRPC computing services, a load balancing approach can be employed. This involves deploying multiple gRPC services and leveraging load balancing for parallelized services (as shown in Figure 16.1). On the other hand, in a single-node scenario, Python's gRPC service doesn't natively support the parallel handling of requests (i.e., gRPC creating multiple processes to handle requests simultaneously). Hence, we can only resort to using Python's multiprocessing library for parallel computing within the service module (as illustrated in Figure 16.2).

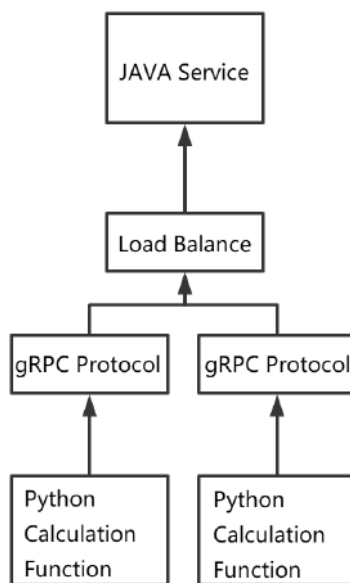


Figure 16.1. Deployment scheme of gRPC computation service across multiple nodes.

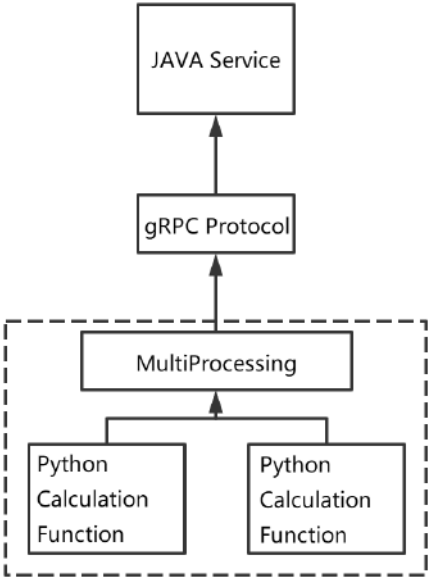


Figure 16.2. Deployment scheme of gRPC computation service on a single node.

16.5 Optimizing Information Transfer across Multiple Machines

16.5.1 Streamlining Data Transfer

As a branch of distributed learning, federated learning also involves the transfer of large amounts of data during model training. What sets it apart is that the size of encrypted data often vastly exceeds that of plain text in federated learning. For instance, after homomorphic encryption, transmitting a plaintext label (1 or 0) may require at most a single character. However, transmitting a label encrypted with a 1,024-bit key would need more than 500 characters, as shown in Table 16.11. Taking random forest as an example and based on business dataset parameters of constructing 50 trees with a sample size of 20,000 each, adding just one more depth level would require transmitting at least an additional 1GB of data. This indicates that the data volume required to be transferred in each federated learning step based on homomorphic encryption can be staggering.

Table 16.11. Length of serialized gRPC message of randomly generated floating-point numbers after encryption and estimated data packet size.

Float-point numbers	Serialized length of encrypted gRPC message	Estimated packet size
1	2136	2KB
1,000	638776	1.2MB
100,000	64508048	130MB

We briefly discuss some of the efforts and outcomes related to data minimization in the FedLearn platform’s federated learning algorithms. For federated learning algorithms based on homomorphic encryption, the bottleneck in data transfer unquestionably lies in transmitting the encrypted data. Hence, we should aim to minimize the transfer of large volumes of ciphertext during the modeling process. In the federated random forest algorithm, the primary ciphertext transfer arises when the active participant encrypts its labels and sends them to the passive participants for summation in quantile computation. Recognizing that the passive participants will inevitably encounter encrypted data during the training process, we have optimized this procedure as follows: During the training initialization phase, after the active participant obtains all the sampled data, it encrypts the labels of the sampled data and simultaneously sends them to the various passive participants. These passive participants retain the batch of encrypted labels throughout the training and destroy them upon completion. During the quantile computation and summation by the passive participants, the active participant no longer sends the encrypted label data but only sends sub-sample space information (i.e., sequence information used for sample positioning in that training). Upon receiving the sub-sample space information, the passive participants can directly select the corresponding encrypted labels from the retained encrypted labels.

This transfer method greatly reduces the data required for each computation. In practical use, federated random forest models using this method have reduced modeling time on business datasets from 3 hours and 30 minutes to 2 hours and 20 minutes. By using the same algorithm parameters — 50 trees, depth of 10, and 20,000 random samples for each tree — efficiency has been further enhanced.

16.5.2 Communication Message Compression and Asynchronous Communication

In the previous section, we transformed the data transmission content in the algorithm, changing the encrypted sample annotation to sub-sample space information. This change significantly enhanced the efficiency of the algorithm. In this section, we continue to optimize the sub-sample space, aiming to further compress the transmitted information.

Assuming we have sampled N examples from the dataset, the initiator needs to convey the position indexes of these samples to all the recipients. A straightforward transmission scheme would be to serialize the position indexes separated by delimiters. However, this method of transmitting sub-sample space information is still quite inefficient. Table 16.12 provides estimates on the size of the data packets needed to transmit different quantities of sub-sample spaces. For instance, when the initiator sends the sub-sample space composed of the first 100,000 samples (indexes 0–99999) to a recipient, the serialized length of the transmitted information is about 688,903, which corresponds to an approximate data packet size of 1.3 MB. For the federated random forest algorithm, the total transmission data volume is approximately the data packet of a single tree sample multiplied by the number of trees and the depth of the tree. Training will involve the transmission of a large amount of data when either the number of trees is large or the tree depth is profound. We briefly introduce the improvement of sub-space information transmission in the FedLearn platform’s federated random forest algorithm.

First, we can convert the sub-space information into a sequence of “0” and “1”, where “1” at the k th position implies that the k th sample is selected, and “0” means it is not selected. This way, we

Table 16.12. Estimation of the data transmission size required for the active participant to transmit sub-sample space information composed of the first N samples to the passive participants.

Sample ID count	Serialized message length	Estimated packet size
100,000	688,903	1.30 MB
500,000	3,888,903	7.80 MB
1,000,000	7,888,903	15.70 MB

get a binary string. By using a simple bit conversion method, we can convert this string into a number of any base, achieving data compression. For example, this can be converted into octal, generating an octal number that represents the initiator’s sample space information sent to the recipient (hereinafter referred to as octal compression). Additionally, when the selected samples are very few (often occurring near leaf nodes) or very many, we can compress consecutive “0” and “1” to further increase the compression rate of the sub-sample space (hereinafter referred to as the online version). Table 16.13 compares the efficiency of different information compression methods when transmitting sub-space information formed by sampling based on 1 million samples.

During the federated learning modeling process, if multiple recipients participate simultaneously (i.e., the total number of participants in federated learning is three or more), employing asynchronous communication during training will boost efficiency. That is, after the

Table 16.13. Comparison of time consumption for different information compression methods when transmitting sub-space information obtained from random sampling of 1 million samples.

Sampling ratio (%)	Metric	Original version	Octal compression	Online version
10	Serialized length	788,902	333,344	112,524
	Transmission time at 300KB/S	5.136s	2.170s	0.732s
	Total serialization + Transmission time	5.329s	2.340s	0.874s
	Time efficiency ratio	1.000	2.325	6.100
	Serialized length	3,944,458	333,344	112,524
50	Transmission time at 300KB/S	25.680s	2.170s	3.662s
	Total serialization + Transmission time	26.034s	2.296s	3.934s
	Time efficiency ratio	1.000	11.329	6.612
	Serialized length	7,888,903	333,344	2,125,022
	Transmission time at 300KB/S	51.360s	2.170s	13.834s
90	Total serialization + Transmission time	52.208s	2.272s	14.151s
	Time efficiency ratio	1.000	22.981	3.689

active participant sends the information to the first passive participant, it doesn't wait for the recipient's response but continues to send the information to the next passive participant. Once responses from all passive participants are received, the active participant proceeds with its modeling step. In the FedLearn platform, we adopt this asynchronous communication mechanism. To ensure stable transmission, data packets are segmented to prevent training failures due to transmission errors.

After these modifications of information compression and asynchronous transmission, the federated random forest algorithm, with parameters set at 50 trees, a depth of 10, and randomly extracting 20,000 samples per tree, saw the modeling time on the business dataset reduce from 2 hours 20 minutes to just 1 hour. With this, the federated random forest algorithm can now meet most of the daily modeling needs.

16.6 Real-Time Inference Optimization

In many federated learning scenarios, there's a frequent need for real-time inferences based on incoming requests. Often, the number of inference samples significantly exceeds the number of training samples, as seen in use-cases like user risk prediction. Taking the federated random forest algorithm as an example, it's not just efficient training we need. In many instances, there's also a pressing need for outstanding real-time inference capabilities, for instance, keeping the response time per individual inference under 200 ms. Typically, tree model inferences decide whether to activate the left or the right subtree by comparing a specific feature value with the threshold of the current node. In a federated learning context, this means fetching participant information for the current node and then sending a request to that participant for the node's decision result. After receiving the request, the participant compares the feature value with the received threshold and sends back which subtree (left or right) should be activated. During our online tests, we found that communication between participants usually takes about 50 ms. Therefore, assuming each tree in a federated random forest model infers in a fully parallel manner, the inference response time, in the most extreme case, would be roughly 50 ms multiplied by the tree's depth. To achieve a

Algorithm 16.1 Optimized inference for federated random forest

Step1: The active participant sends an inference request to the passive participants;

Step2: Upon receiving the request, the passive participants compare their local data with the inference model, calculates all nodes in the model, and sends the results back to the active party in a node: left/right format;

Step 3: Using the returned information, the active participant reassembles each decision tree and proceeds with the inference.

response time of 200 ms in such a network environment, the maximum depth of the tree in the federated random forest model can't exceed 4, which would significantly impact the model's performance. To address this, we've designed a more efficient inference approach, as shown in Algorithm 16.1. In this scheme, upon receiving an inference request, the passive side calculates all node decisions at once and sends back a packaged response to the active participant. The active side then reassembles each decision tree using this information and makes the local inference. Practical tests have shown that this new approach can compress the online inference latency of the random forest to under 100 ms.

16.6.1 Conclusion

Table 16.14 presents the optimization path for the federated random forest model on the FedLearn platform, from its initial version to its current stable service offering. Once the federated random forest modeling time met regular business demands, the model opened up the potential for parameter tuning. Continually aligning with business requirements, we utilized grid search for hyperparameter optimization in the federated random forest model. Table 16.15 showcases the final experimental results for the inference metrics of the federated random forest algorithm.

From our optimization practices in this chapter, it's clear that federated learning algorithms integrate a broader range of technologies compared to traditional machine learning algorithms. Their optimization paths encompass various areas ranging from encryption principles, computation units, system architecture to

Table 16.14. Optimization path and comparison before and after the optimization of the federated random forest algorithm, using a business dataset with 50 trees, depth 10, and each tree sampling 20,000 as an example.

Optimization	Pre-optimization metric	Post-optimization metric	Improvement
GMP calculation library	35.00h	7.20h	4.86×
Homomorphic encryption scheme	7.20h	3.50h	2.05×
Reduced transmission information	3.50h	2.33h	1.50×
Message compression and asynchronous communication	2.33h	1.00h	2.33×
Inference optimization	0.50s	0.10s	4.00×

Table 16.15. Final experimental results on the inference performance of the federated random forest algorithm.

Dataset	Training		Testing		OOT	
Algorithm	AUC	KS	AUC	KS	AUC	KS
Federated random forest (Before tuning)	0.635	0.196	0.615	0.167	0.612	0.161
SecureBoost	0.643	0.203	0.641	0.205	0.622	0.173
Federated random forest (After tuning)	0.658	0.238	0.656	0.237	0.638	0.195

network communication. Fortunately, the optimization outcomes have been encouraging. In the future, the FedLearn platform will continue exploring its current performance bottlenecks and functionality expansion. On one hand, the goal is to enhance module efficiency to support the overall operation of the FedLearn platform. On the other, there’s an aspiration to collaborate and optimize with new technologies (such as Flink, Spark, and various advanced deep learning frameworks) to cater to the platform’s needs for more data, more participants, and safer joint modeling.

Chapter 17

Federated Learning Based on Blockchain

17.1 Introduction to Blockchain

17.1.1 Overview of Blockchain

17.1.1.1 What is blockchain

The concept of *blockchain* originated with the advent of Bitcoin in early 2009. With the success of Bitcoin, financial institutions and tech companies shifted their focus to the mechanism behind Bitcoin, refining and improving upon it, leading to the emergence of blockchain technology. Essentially, blockchain is a decentralized distributed ledger database. As the ledger's data structure is composed of data blocks and "chains" connected in chronological order of transactions, it forms a chain-like data structure, hence the name "blockchain".

Technically, blockchain isn't a new technology in itself but a combination of various technologies. It ingeniously integrates pivotal technologies, such as peer-to-peer (P2P) networking, cryptographically secure shared ledgers, consensus mechanisms, and smart contracts, representing a technological mode of innovation. The most significant value of blockchain is its ability to establish trustworthy networks, enabling peer-to-peer value transfers without a centralized third party. This has expanded the application areas of blockchain across various sectors, leading to numerous disruptive

applications innovated around blockchain and paving new directions for industries.

17.1.1.2 Development of blockchain

The development of blockchain can be divided into three preliminary stages: Blockchain 1.0, represented by “currency”, with Bitcoin being its most successful application to date. Blockchain 2.0, symbolized by “smart contracts”. Ethereum, while supporting digital currency, also introduced the support for smart contracts. Blockchain 3.0, which offers potential for new applications beyond just digital currency or financial applications, targeting industry-specific applications.

17.1.1.3 Types of blockchains

Blockchains can be categorized based on their access permissions: *Public chains* are entirely open, allowing anyone to participate without any permission or identity verification. All data in such chains are transparent, and they are fully decentralized. For instance, Bitcoin operates on a public chain network. *Private chains* are closed systems, limited to organizational use, and require identity verification. They have a comprehensive permission management system. Essentially, private chains are centralized (or sometimes referred to as “weakly centralized”). *Consortium chains*, often used in environments with multiple member roles, have identity verification and permission settings. These chains are semi-decentralized or multi-centralized. For example, the Hyperledger Fabric project is a renowned consortium blockchain platform.

Blockchains can also be classified based on their connection type: *Single chain* refers to a blockchain system that can operate independently, such as Bitcoin’s main chain or consortium chains set up using Fabric in the Hyperledger project. *Side chain* pertains to a cross-chain technology for a blockchain system, aiming to integrate various blockchains, technically bridging isolated data islands and complementing each other. For instance, while the Bitcoin system doesn’t support smart contracts, Ethereum natively does. Therefore, the Ethereum blockchain can serve as a side chain to Bitcoin’s main chain, enhancing its functionalities. *Interconnected chain* refers to a broader ecological blockchain formed by interconnecting different blockchains. For example, domestic blockchains interconnected

through protocols with foreign blockchains can bring about a new level of intelligence to society.

17.1.2 Technical Features of Blockchain

Blockchains operate on a P2P decentralized network, employing “hash functions” to ensure the integrity and immutability of transaction data, utilizing “digital signature technology” for self-proving information, and leveraging “consensus algorithms” to establish consensus mechanisms in an untrusted network environment, forming a decentralized trust mechanism.

17.1.2.1 Cryptography

Cryptography is fundamental in ensuring the security of a blockchain system. The definition and construction of blocks and blockchains incorporate a plethora of modern cryptographic techniques, including cryptographic hash functions and public-key cryptography.

Hashing Algorithm: A hashing algorithm is a method that produces a fixed-length output for a given input data length. Its output is termed a hash or message digest. The primary characteristics of a hash algorithm include the following:

Characteristic 1: It generates a “fixed-length” “hash value” from the input data. This hash value, composed of numbers and letters, conceals the original input information.

Characteristic 2: If the input data remains unchanged, the output hash value will always be identical (as shown in Figure 17.1).

Characteristic 3: Even a slight modification in the input data will lead to inconsistent results (as shown in Figure 17.2).

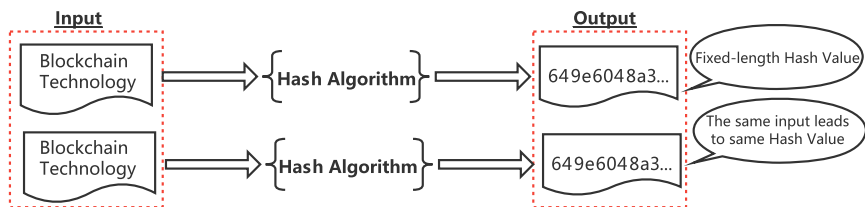


Figure 17.1. Same input, same hash value.

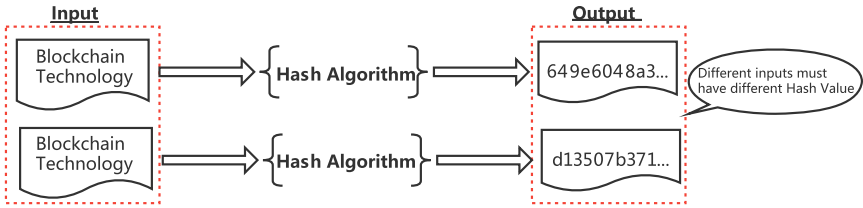


Figure 17.2. Changing a single character in input results in distinct outputs.

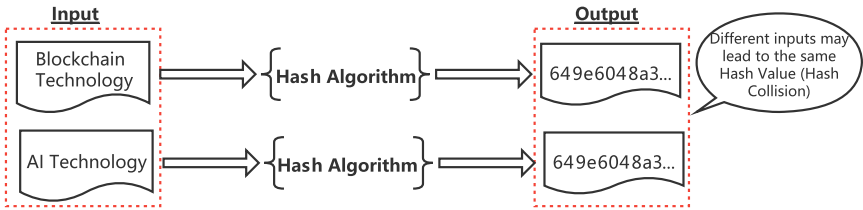


Figure 17.3. A hash collision: Different inputs but identical hash values.



Figure 17.4. Unidirectionality of a hash function.

Characteristic 4: Two entirely different inputs might produce the same hash value under the same hash function, leading to a hash collision (as shown in Figure 17.3). Since the input length can vary while the output length remains fixed, collisions are inevitable. The security of a hash function denotes that finding a collision is impractical with current computational resources (including time and space), giving hash algorithms their collision-resistant properties.

Characteristic 5: Hash functions are computationally irreversible. Based on the output of a hash function, it's challenging to deduce the input value (as shown in Figure 17.4).

Given the above characteristics, hash functions are widely used to protect the integrity of any type of data. Blockchain technology uses the “collision-resistant” feature of hashing algorithms to safeguard the integrity of transactional data.

Merkle tree: The Merkle tree, also known as a hash tree, is a quintessential binary tree structure consisting of a root node, a set of intermediate nodes, and a set of leaf nodes. The root node's content is the hash value (or digital digest) of all its child nodes. The structure is illustrated in Figure 17.5.

A prominent feature of the Merkle tree is its excellent traceability. Each transaction can be traced back to find its historical record. If there's any modification to the underlying transaction data, it would propagate to its parent node and successively up to the root of the Merkle tree. This results in a consensus validation failure between nodes, and tampering actions are detected. This is illustrated in Figure 17.6.

Specifically, if block data "002" is tampered with, it will directly affect the results of nodes D and A, eventually impacting the Merkle tree root result. This leads to block 3 and all subsequent blocks being inconsistent with other nodes, resulting in validation failure. Even if

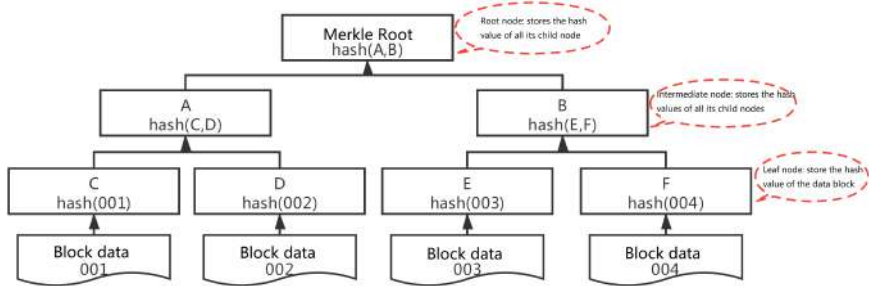


Figure 17.5. Structure of the Merkle tree.

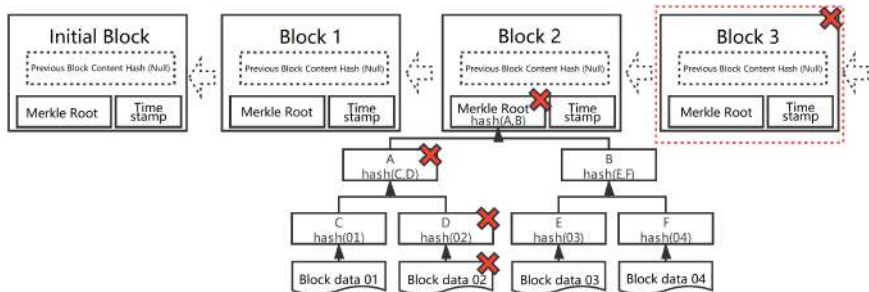


Figure 17.6. Tampering with transaction data.

one could modify all affected blocks on the chain, the content of the original block remains immutable, causing the final data comparison to still be inconsistent. Therefore, the blockchain’s chain structure effectively prevents data from being tampered with. Moreover, by comparing the Merkle tree roots stored on various nodes, it’s also easy to verify and trace which specific transaction was maliciously altered.

The Merkle tree is a crucial data structure in the blockchain, ensuring the completeness, traceability, and tamper-resistance of transaction data stored on the blockchain. Additionally, the data block records all transaction information captured during the block’s creation, facilitating audit and regulation.

Public-key cryptography: Public-key cryptography, also known as asymmetric cryptography, uses a pair of keys: a public key and a private key. The public key can be shared openly, while the private key must be kept confidential. The private key can derive the public key, but deriving the private key from the public key is challenging. Using public key cryptography, one can validate that the transaction data genuinely comes from the sender and hasn’t been altered or impersonated. It also ensures that only the message recipient can decode the valuable data. This is depicted in Figure 17.7.

Note: The message sender Alice uses the “public key” provided by the message recipient Bob to encrypt the message, forming a digital envelope. This digital envelope is then transmitted to Bob via the P2P network. Bob then uses his private key to decrypt the digital envelope, thus retrieving the message sent by Alice. During the

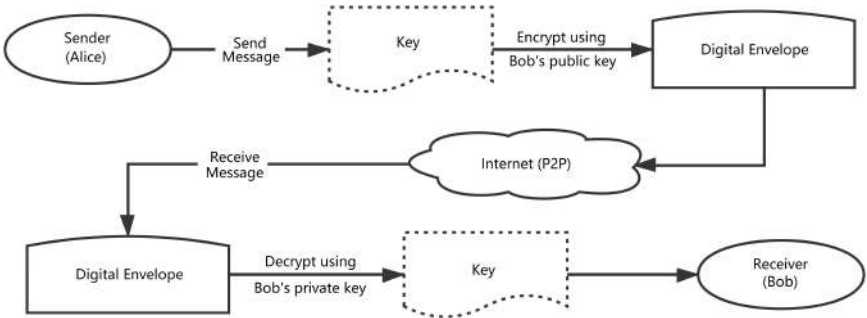


Figure 17.7. Public key cryptography, using public key for encryption and private key for decryption.

decryption process, only Bob's private key can unlock the digital envelope, ensuring its security. As such, the blockchain utilizes public key cryptography to safeguard key distribution and value exchange in network communications, ensuring the safety of blockchain data.

Digital signature: A digital signature refers to the sender of the information using their “private key to sign”, while the recipient uses the sender's public key to verify the signature, confirming that the information indeed originates from the sender. This is illustrated in Figure 17.8.

The transaction or block information sent by Alice is hashed to obtain the respective “digital digest”. Alice then uses her private key to sign this digital digest, generating the corresponding “digital signature”. The original data and its digital signature are transmitted to the message recipient Bob via the P2P network. Bob uses the public key provided by Alice to validate the digital signature. If the verification is successful, it proves that the digital signature genuinely belongs to Alice, and the information indeed came from her.

Blockchain technology uses digital signatures to ensure the non-repudiation of information.

Privacy protection algorithm: With the emergence and development of data sharing, how to effectively protect data privacy while satisfying jurisdictional laws and regulatory requirements, and ensuring data availability, poses a significant challenge in the field of information security. Blockchain's privacy protection needs to both conceal transaction details and verify the transaction's accuracy. Current blockchain privacy protection solutions include zero-knowledge proofs, homomorphic encryption, and ring signatures,

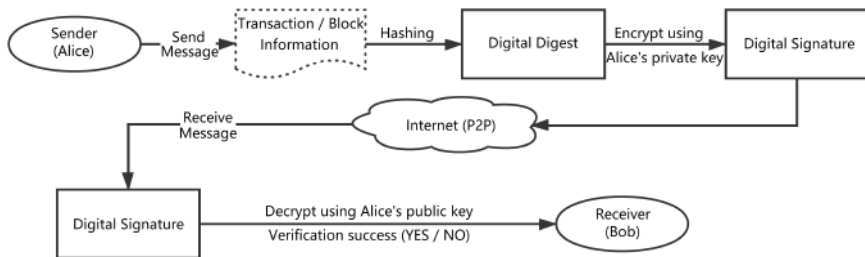


Figure 17.8. Digital signature, using private key for signing and public key for verification.

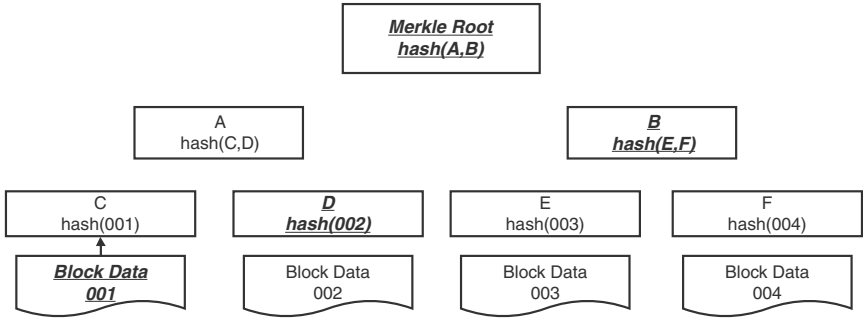


Figure 17.9. Application of the Merkle tree in zero-knowledge proofs.

among others. For instance, zero-knowledge proofs can validate the correctness of information without revealing the actual data. The application of the Merkle tree in zero-knowledge proofs is illustrated in Figure 17.9.

The Merkle tree offers a method allowing a party, such as Alice, to prove possession of certain data without transmitting the original data to another party. For instance, Alice only needs to disclose block data 001, D, B, and the Merkle tree root. Any user with block data 001 can, through calculations, obtain the same Merkle tree root value, indicating that Alice possesses block data 002, block data 003, and block data 004.

Blockchain technology uses zero-knowledge proofs to protect users' identity anonymity and the confidentiality of on-chain transaction data.

17.1.2.2 Consensus algorithm

Blockchain operates on a distributed architecture. To ensure a safe and reliable state consensus in an unreliable asynchronous network, distributed consensus algorithms are crucially applied to the blockchain. These consensus mechanisms address consistency issues in distributed systems. Under network timing assumptions, consensus mechanisms guarantee that all healthy nodes in the distributed network receive a consistent sequence of operations.

Consensus algorithms generally fall into categories of *crash fault-tolerant algorithms* (like Paxos and Raft), *Byzantine fault-tolerant consensus algorithms* (like PBFT and its variants, DumboBFT), and

probabilistic consensus algorithms (like Bitcoin's POW). Depending on the application scenario, consensus algorithms vary. For instance, the consensus algorithm for the Bitcoin public chain uses the non-strongly consistent POW proof-of-work consensus mechanism; the JD Chain project adopts the BFT-SMaRt consensus mechanism, and the Hyperledger Fabric project supports the Raft consensus mechanism.

Blockchains use consensus algorithm technology to solve decentralized multi-party trust issues.

17.1.2.3 Decentralized network

The goal of a decentralized network is to eliminate the central entity, decentralizing power originally vested in centralized roles and allowing users to conduct peer-to-peer transactions freely. A comparison between the traditional centralized server model and the P2P network model is illustrated in Figure 17.10.

The centralized server model can be understood as having just one server side. Multiple clients can only access resources, services, and content by connecting to the central server. In contrast, in the P2P decentralized network model, every participant (or node) in the network acts both as a server and a client. The idea is to distribute the center across every node, with data transmission and service implementations occurring between nodes without the need for an intermediary.

Blockchain technology's network layer communication protocol uses a peer-to-peer structure. In the P2P system, nodes operate

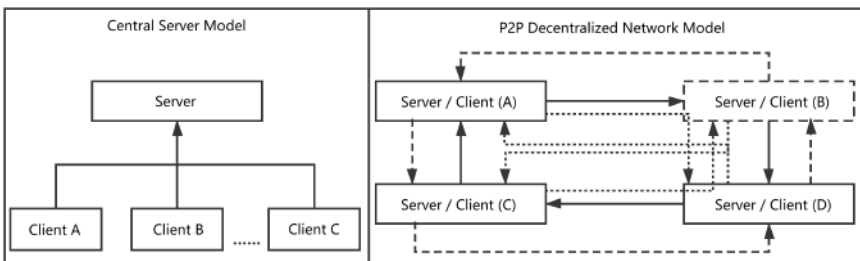


Figure 17.10. Comparison between the central server model and P2P decentralized network model.

equally and independently, not relying on other specific nodes, achieving decentralized control.

At its core, the blockchain is a decentralized distributed ledger database. The “decentralized” characteristic of the blockchain not only enhances system fault tolerance but also effectively avoids issues related to single-point failures and data monopolization or inequities introduced by centralization. This makes data more open and transparent, allowing data to directly generate value. At the same time, it strengthens the network’s resistance to attacks and prevents collusion between nodes. Leveraging the decentralized nature of blockchain establishes a trustworthy foundation for the internet, providing a range of reliable services.

17.1.2.4 Smart contracts

Smart contracts are computer programs that run on a replicable and shared ledger within the blockchain network environment. From a developer’s perspective, they can be seen as snippets of code; from a user’s standpoint, they are automatic assurance agreements that automatically execute relevant terms (like financial incentives or profit distribution) when specific conditions are met. Before the deployment of a contract, all its logical flows have been defined, and they’re signed using the private keys of all participating parties to ensure the contract’s validity. The signed smart contract is then disseminated and deployed on the blockchain via the P2P network. Once deployed on the blockchain network, the contract can self-execute and self-validate without any human intervention.

17.1.3 Integration of Blockchain Technology with Modern Technologies

With the rise of internet technologies in recent years, technologies like cloud computing, artificial intelligence (AI), big data, IoT, and blockchain have become hot topics in tech strategy. As these technologies become more integrated into business development, their importance in tech ecosystems is increasingly recognized. Moreover,

these technologies do not exist in isolation but are interwoven, providing combined technical support for business growth.

In the realm of cloud computing, the blockchain is seen as a particular manifestation of cloud technology. For instance, PaaS cloud services can support blockchain technology development, and IaaS can be used for deploying blockchain nodes. Additionally, given blockchain technology's features of transparency, immutability, and public accessibility, it's also an excellent solution for real-time data collection applications in edge computing.

In the big data sector, blockchain technology provides a unique, trustworthy, and complete source of data, aiding in the collection, storage, and analysis of big data, and technically breaking down data silos.

In the Internet of Things (IoT) domain, blockchain enables a decentralized model for IoT. IoT data collection typically uses Wireless Sensor Networks (WSNs). Blockchain technology can serve as the foundation for WSN data transmission, multi-hop forwarding, and aggregated uploading, ensuring data integrity and immutability.

In the AI sector, the combination of blockchain and AI can mutually empower each other at the data, algorithm, and computational capability levels. Technically, blockchain can ensure data reliability and enable data sharing in certain situations, providing trustworthy data to enhance AI modeling and application accuracy. At the algorithmic level, blockchain ensures that the AI engine's models and results remain unaltered, reducing the risk of models being compromised. In terms of computational capability, AI based on blockchain can realize decentralized intelligent joint modeling, offering users flexible computational power to meet their needs. Given that the blockchain is a distributed network, it enables decentralized computational power. Thus, blockchain assists in constructing a decentralized foundation platform for AI computational facilities.

Furthermore, we are currently exploring the application of blockchain technology to federated learning, providing it with a robust decentralized data structure, interaction mechanisms, and computational models. This amplifies the practicality of federated learning systems, enhancing their real-world value.

17.2 Innovative Integration of Federated Learning and Blockchain

The existence of AI has been long-established, and it has witnessed explosive growth in recent years, emerging as a new driving force behind socio-economic and technological advancement. The success of AI hinges on vast amounts of data. In its rapid growth phase, it inevitably gathered substantial data from various channels, often at the expense of data privacy, leading to numerous data breaches. As society progresses, the importance of data privacy has become increasingly evident. On the other hand, during collaborative training with shared data from multiple parties, disputes over data rights often arise, making it challenging to ensure equitable benefits. Recognizing concerns about such events, the concept of federated learning was introduced. It facilitates collaborative training across multiple parties without data leaving its source and can record contributions from each participant, reducing disputes and ensuring fairer benefit distribution.

Based on the previous section's introduction to blockchain, it's evident that blockchain offers unique advantages in data sharing, data traceability, and technical privacy protection. These are also the distinctive features that differentiate federated learning from traditional machine learning. By integrating federated learning with blockchain, blockchain's attributes can significantly enhance federated learning in areas, such as privacy protection, regulatory auditing, and decentralized governance.

Smart contracts are sets of predefined code logic that, depending on external inputs, execute automatically without human intervention or reliance on third-party institutions. The potential of smart contracts on the blockchain to replace centralized coordination services in traditional federated learning is apparent. In a federated learning platform, smart contracts generally provide the following capabilities:

Model update and propagation capability: Once smart contracts replace centralized coordination services, the initial challenge is distributing the primary model parameters and model update parameters from all participants while technically preventing data privacy breaches. It's crucial to negotiate a common data structure,

which can be masked using encryption algorithms, like homomorphic encryption and differential privacy. Two potential propagation methods are periodic pull and event-driven. In the former, participants periodically fetch model update data from the smart contract on the blockchain. In contrast, the event-driven approach requires participants to set up event listeners; the blockchain automatically pushes data to participants when new data becomes available. Comparatively, the event-driven approach is preferable.

Model update aggregation capability: Upon collecting sufficient model update data, the smart contract automatically triggers the aggregation rules for global model updates. This aggregation process remains impartial and isn't influenced by any participant. Following this, the aggregated model serves as the initial model for the next training round, perpetuating the cycle.

Incentive capability: While federated learning addresses data privacy needs, it also represents an avenue for participants to maximize data benefits. Determining the value contribution of each participant to the platform based on the limited interaction data during the federated learning process is essential. Generally, contributors' efforts can be calculated based on training time, dataset quantity, quality, and other metrics. Benefits are then dynamically allocated to participants based on incentive rules defined in the smart contract.

Regulatory audit capability: The traceable nature of blockchain naturally supports auditing, and its decentralized characteristic can easily integrate with multiple third-party regulatory bodies. To leverage this for regulatory auditing, smart contracts must provide the necessary capabilities. Broadly speaking, smart contracts need to offer appropriate data query interfaces.

17.2.1 Innovations in Architecture

Conventional federated learning platforms possess a centralized coordination service, which primarily facilitates the distribution and aggregation of model data. At the onset of model training, this service is responsible for distributing the initial model to all participants in federated learning. After each training round, it aggregates the disparate model updates from all participants and then

disseminates the updated model back to them. This process continues until training completion. While a centralized coordination service intrinsically doesn't have functional shortcomings, its existence as a centralized component in a multi-party federated learning platform poses potential single points of failure. Moreover, there's no clear establishment of its ownership, making it challenging to ensure absolute impartiality.

Integrating blockchain into the federated learning platform transitions the centralized coordination to decentralized blockchain coordination. The distribution and aggregation of model data no longer rely on centralized coordination. Instead, smart contracts deployed on the blockchain can autonomously aggregate model data and distribute it to all participants. Moreover, introducing the blockchain reduces the coupling between the federated learning participants. With a stable infrastructure, the admission and exit of participants become relatively simpler, further enhancing the enthusiasm and fairness among participants.

17.2.2 Innovations in Process

Federated learning is a process where participants independently carry out machine learning based on a shared model, mainly exchanging model data with each other.

The typical workflow of a federated learning platform with centralized coordination is as follows:

1. The task initiator sends the initial model data to the centralized coordination service.
2. The centralized coordination service distributes the initial model to all participants.
3. Each participant completes an iteration of model training independently and then uploads their model update to the centralized coordination service.
4. Upon receiving the model updates from all participants, the centralized service aggregates and updates the model parameters.
5. Repeat Steps 2–4 until the training task concludes.

As evident from the process, the centralized coordination service, while vital, is also a vulnerability. If this service experiences issues, the entire workflow halts. However, a blockchain-based federated

learning platform employs a decentralized structure to eliminate single points of failure. Additionally, it innovatively utilizes smart contracts to drive model updates, reinventing the federated learning process with a contract event mechanism. The optimized process can generally be outlined as follows:

1. The task initiator uploads the initial model data to the blockchain via a smart contract.
2. The smart contract broadcasts model data to all participants using an event mechanism.
3. Participants independently listen to the smart contract broadcast events to receive model data.
4. Each participant independently completes model iteration training, then uploads their model update to the blockchain via the smart contract.
5. The smart contract, following predefined logic, aggregates the model updates from all participants and updates the model parameters.
6. Repeat Steps 2–5 until the training task concludes.

With the enhancement of blockchain's smart contracts, federated learning participants can break free from the constraints of centralized coordination. Participants only interact with their local blockchain nodes, with all other data sharing details encapsulated within the blockchain network. This optimized process is more conducive for federated learning participants to focus on machine learning itself, without getting overly concerned about how data are exchanged among each other, thereby reducing collaborative costs for participants.

17.2.3 Data Support

Federated learning platforms, due to data privacy restrictions, only allow the exchange of model data among participants. At the start of federated learning, initial model data need to be distributed to each participant, termed as pre-data. During each iteration of training, model data are constantly updated and aggregated, referred to as process data. Once the training is complete, the trained model must be distributed to participants through the platform, termed as post-data.

In federated learning platforms without blockchain support, a centralized coordination service typically handles the distribution and aggregation of data. However, federated learning platforms backed by blockchain utilize smart contracts and event mechanisms to facilitate the circulation of pre-data, process data, and post-data among members. This method minimizes direct contacts between participants, enhancing the flexibility of the platform, ensuring that any unexpected departure of a participant doesn't disrupt the iterative learning process for others.

Regarding data security, conventional federated learning platforms already ensure data transmission security using two-way TLS channels and prevent reverse inference of model data through homomorphic encryption and differential privacy. On top of this, blockchain offers stability in data security. Once data enter the blockchain network, all subsequent transfers are recorded and immutable. While this doesn't directly elevate data security, it provides system-level support for tracing data flows. Furthermore, smart contracts, which operate within the blockchain execution engine and act as bridges between the blockchain and upper-layer business, support data privacy protection during execution through Secure Multi-Party Computation (SMC).

17.2.4 Incentive Support

The emergence of federated learning is partly due to innovative breakthroughs prompted by data privacy regulations, and also because participants hope to harness their data, collaborating with others to extract greater value and reap substantial rewards. To achieve this, effective incentive algorithms are crucial.

Accurate incentive algorithms require precise data. Using the value-determination ability of blockchain, one can register, validate, and distribute the value created by federated learning participants. As multiple participants engage in federated learning, the blockchain can log service interface calls, participants' contributions, and generated profits. Smart contracts can then automatically distribute profits according to pre-defined sharing rules and ratios to participant's blockchain accounts, and these can be pegged to real profits when appropriate. In case of disputes, the blockchain's data tracing capability facilitates reverse tracking.

With blockchain support, the incentive mechanism of federated learning strengthens the platform, ensuring its sustainable growth. The platform's fairness is guaranteed, motivating participant engagement. On the flip side, the platform can trace and penalize potential malicious attacks by participants, ensuring the continuous healthy operation of the federated learning alliance — a kind of reverse incentive support.

17.2.5 Regulatory and Audit Support

Due to data privacy regulations, federated learning platforms have always been a prime target for strict regulation. Conventional federated learning platforms must customize certain features to meet regulatory requirements. With blockchain support, leveraging its decentralization and traceability features, platforms can proactively embrace regulatory audits in a multi-party setting.

For regulation, platforms can introduce regulatory agencies as independent and impartial third-party read-only nodes into the blockchain federated learning network. Once integrated, regulators can monitor the federation's data and operations in real-time, promptly halting any non-compliant actions, ensuring the protection of public data privacy and the orderly, healthy functioning of the market economy.

For audits, blockchain's traceability makes it inherently audit-friendly. Any action or data update on the blockchain leaves a footprint. Should there be a need for auditing, one can simply backtrack by reading blocks in reverse, eliminating the need for additional system support.

17.3 Blockchain-Based Federated Learning Incentive Algorithms

During the federated learning process, malicious participants providing incorrect gradient parameter updates can sabotage the accuracy of model training. Additionally, lazy nodes that don't actively contribute model data can lead to training biases. A significant challenge in the industry is ensuring that all parties continuously and actively participate in the model data training process and that collaborations are conducted fairly and justly.

Blockchain consensus algorithms and smart contracts inherently have incentive properties. By leveraging these blockchain technological features, there is potential for encouraging active multi-party collaboration. The integration of blockchain and federated learning can pave the way for a blockchain-based federated learning algorithm, which holds promise in addressing this industry issue.

In this section, based on the different ways of federated learning incentives, we introduce several blockchain-based federated learning algorithms from various perspectives, including model quality assessment, weight value, and incentive distribution.

17.3.1 Model Quality Assessment Incentive Algorithm

Technically, federated learning aims to organize every data silo, incorporating them into the federated learning collaboration system. The first step is to preprocess local data, then feature the training data, followed by a quality assessment of the model data. The precision of model quality assessment during federated learning is a key determinant of the training outcome.

Metrics such as data volume, data validity, data information density, and data authenticity all play a role in data monitoring and quantitative assessment.

In centralized federated learning, a central aggregator consolidates all the uploaded local model data to generate a new global model. During this process, threats like model poisoning can emerge, potentially leading to training failures.

How can we ensure consistent model data contributions from participants? By deploying a reward task in a blockchain smart contract (see Figure 17.12). Based on the results of the model quality assessment algorithm, trainers who complete the task are rewarded accordingly. The smart contract sets up a model data transaction market for federated learning, allowing honest participants to profit from completing tasks. An effective profit distribution mechanism is vital for efficient collaboration among all parties. Here, we introduce an incentive method that distributes rewards based on the benefit generated by the participants. The benefit of a federated participant refers to the utility it generates when joining the federated training team.

Generally speaking, the periodic profit, denoted as $\hat{u}_i(t)$, that a participant i receives from the total budget $B(t)$ in a given profit-sharing round t can be represented as

$$\hat{u}_i(t) = \frac{u_i(t)}{\prod_{i=1}^N u_i(t)} B(t) \quad (17.1)$$

where $u_i(t)$ denotes the utility participant i generates for profit $B(t)$. Assuming the marginal profit each participant i makes for the collective (assuming the collective only consists of participant i), it is used to calculate its share of the profit. The numerical calculation method is

$$u_i(t) = v(\{i\}) \quad (17.2)$$

where $v(X)$ is a function that evaluates the utility of the collective X , with X representing the set of participants.

In this section, we use the Mean Squared Error (MSE) as a function to evaluate the utility of the collective X and introduce a model quality assessment incentive algorithm. MSE is a commonly used loss function in linear regression, with the objective to minimize this function during the regression process. It can also be employed for model evaluation comparisons. By calculating the ratio of the sum of squares of deviations between observed and actual values to the number of observations, we obtain the MSE for each model data accuracy assessment:

$$\text{MSE} = \frac{1}{m} \prod_{i=1}^m (f_i - y_i)^2 \quad (17.3)$$

where m represents the number of samples and the range of MSE values is $[0, +\infty)$. A smaller MSE value indicates a higher accuracy of the model data. When the predicted values perfectly match the actual values, the MSE equals 0.

Figure 17.11 depicts a smart contract deployed on the blockchain, which stores participants, training tasks, training status, and task rewards from the federated learning process. The smart contract releases model training tasks. During the federated learning process, after assessing the model data based on the model quality assessment algorithm, the accurate MSE value is updated in the smart contract.

```

/**
 * Precondition: the ids of the data owners are their addresses.
 */
contract SmartContract {
    // model Data where to assign validators, trainers, mse, status, federatedAggregator ...
    struct ModelData {
        string modelId;
        address[] trainers;
        address[] validators;
        uint[] msesByIter;
        mapping(address => uint[]) partialMsesByIter;
        uint currIter;
        uint improvement; // Percentage
        uint frozenPayment;
        mapping(address => uint) contributions; // Percentages
        Status status;
        address owner; // Model Buyer's are the owners
        address federatedAggregator; // federatedAggregator orchestrator
    }

    enum Status {INITATED, STOPPED, FINISHED}

    constructor() public {
        prices["LINEAR_REGRESSION"] = 1000000000000000;
    }

    /** EVENTS */
    event ModelCreationPayment(address owner, uint256 amount);
    event ContributionPayment(address receiver, uint256 amount);
    event ValidationPayment(address receiver, uint256 amount);
}

```

Figure 17.11. Model data structure in the smart contract.

Upon completing the current training round, the smart contract's pre-set events are triggered to initiate orderly transaction executions.

The entire federated learning process primarily encompasses three stages: data initialization, exchanging locally trained model data, and model data quality assessment with incentive distribution. These stages facilitate secure collaboration and effective incentives within the smart contract.

17.3.1.1 Initialization phase

The central aggregator invokes the `init()` function to initialize federated learning configuration parameters, test datasets, global model settings, and more (see Figure 17.13). The issuer of the federated learning task creates task rewards in the smart contract, specifying training models and aggregators. The aggregator sends the initial global model data task to all participants, ensuring encrypted and secure transmission of sensitive intermediary data.

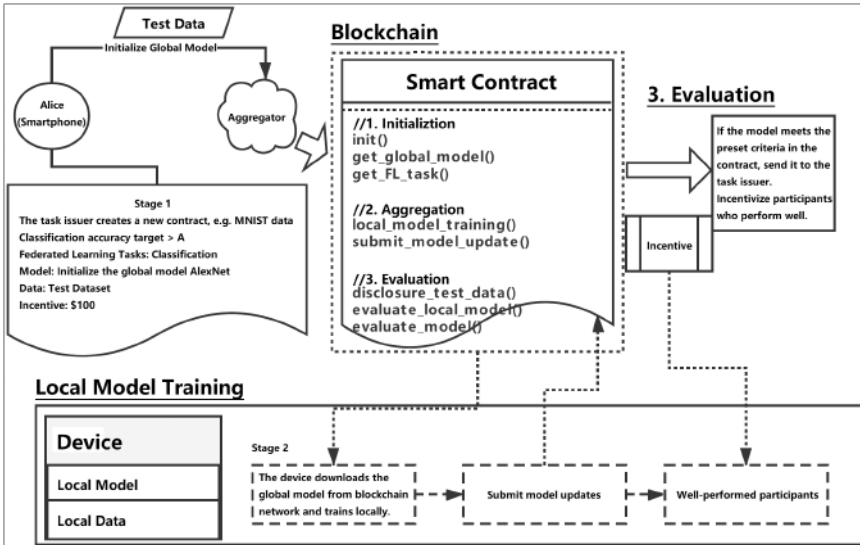


Figure 17.12. Model quality assessment incentive process.

17.3.1.2 Local training phase

Upon receiving a new model training task, each device runs the `localModelTraining()` function to train its local data. To prevent accuracy loss during the federated learning process, algorithms like gradient compression are typically employed to transfer local model data parameters.

Once a device completes its local training, it executes the `submitModelUpdate()` function to submit local model data updates to the aggregator, thereby earning the device an opportunity for profits in that round.

17.3.1.3 Evaluation phase

The central aggregator invokes the `evaluateModel()` function to assess data uploaded from the devices. If a device's model data passes the evaluation, the resulting MSE value from this assessment is stored in the smart contract as the basis for contribution. In the function `calculateContributions()` for determining this round's contributions, the improvement in model quality's precision serves as the criterion for marginal benefit rewards. In this manner, only devices with


```

/** ContractService() init phase: 1 deploy a smart contract 2 init model in SC**/
init():
    var SmartContract = artifacts.require("SmartContract");
    module.exports = function(deployer) {
        deployer.deploy(SmartContract);
    };

function initModel(string memory modelId, address[] memory validators, address[] memory trainers,
    address modelBuyer, address federatedAggregator) private pure returns(ModelData memory) {
    ModelData memory model = ModelData({
        trainers: trainers,
        validators: validators,
        modelId: modelId,
        msesByIter: new uint[](200),
        improvement: 0,
        curIter: 0,
        frozenPayment: 0,
        status: Status.INITATED,
        owner: modelBuyer,
        federatedAggregator: federatedAggregator,
    });

    return model;
}

# whole federated aggregator flow with python
app = create_app()
apl.init_app(app)
encryption_service = EncryptionService(is_active=app.config["ACTIVE_ENCRYPTION"])
data_owner_service = DataOwnerService()
contract_service = ContractService()
contract_service.init(app.config)
federated_aggregator = FederatedAggregator()
data_owner_service.init(encryption_service, app.config)
federated_aggregator.init(encryption_service=encryption_service, data_owner_service=data_owner_service,
    contract_service=contract_service, config = app.config)
logging.info("federated_aggregator running")

```

Figure 17.13. Initialization process.

high-quality models will earn rewards for this round; providers with insecure model data will be excluded, thereby losing the chance for earnings in this round.

When the model update count reaches its limit, the `ContributionPayment` event is triggered. The smart contract invokes the `finalizeContract()` function to reward the devices. Model purchasers distribute this round's reward $B(t)$ to model trainers based on the initial agreement in the smart contract. Subsequently, the central aggregator calls the `postEvaluation()` function to announce the results of this task. Ultimately, the task issuer obtains a higher-quality global model.

17.3.2 Weighted Incentive Algorithm

A centralized aggregator requires collaboration from numerous clients to complete aggregation operations and continuously broadcasts new global models to these clients. This demands

```

/**
 * finalize Contract to payment training contributors
 */
function finalizeContract(string memory modelId) public onlyModelBuyer isFinished(modelId) {
    ModelData storage model = models[modelId];

    _calculateContributions(modelId);

    // Pay trainers
    for (uint i = 0; i < model.trainers.length; i++) {
        address payable trainer = address(uint160(model.trainers[i]));
        _calculatePaymentForContribution(modelId, trainer);
    }
    // Return payments to model Buyer
    returnModelBuyerPayment(modelId, address(uint160(model.owner)));
}

/**
 * According to U(i) / N methods, pay for the Data Owner with his address
 * for his work done training the model.
 */
function _calculatePaymentForContribution(string memory modelId, address dataOwner)
private isFinished(modelId) view returns (uint) {
    uint paymentForImprov = (payments[modelId] * getImprovement(modelId)) / 100;
    uint paymentForImprovForTraining = (paymentForImprov * 70) / 100; // training contribution percent
    return paymentForImprovForTraining;
}

// core calculate contribution function
function _calculateContributions(string memory modelId) public isFinished(modelId) {
    ModelData storage model = models[modelId];
    uint iter = model.currIter; // current iterator of this training process
    model.improvement = calculateImprovement(model.msesByIter[0], model.msesByIter[iter]);

    // calculate the total contributions of the training
    uint contributionsSum = 0;
    for (uint i = 0; i < model.trainers.length; i++) {
        address trainer = model.trainers[i];
        model.contributions[trainer] = mseDifference(model.msesByIter[0], model.partialMsesByIter[trainer][iter]);
        contributionsSum = contributionsSum + model.contributions[trainer];
    }

    // update the contribution percent of every training
    for (uint i = 0; i < model.trainers.length; i++) {
        address trainer = model.trainers[i];
        model.contributions[trainer] = contributionPercentage(model.contributions[trainer], contributionsSum);
    }
}

```

Figure 17.14. MSE marginal benefit allocation method.

significant network bandwidth and is influenced by the stability of a single server. The centralized aggregation process might be biased toward certain clients; some might provide malicious or toxic models or even collect clients' private data from updated model data.

How can we establish a more secure, fair, and reliable incentive mechanism? A blockchain network can substitute for the centralized aggregator. By electing a blockchain consensus committee to verify and exchange local model updates from client devices, and subsequently storing it on the chain, this on-chain model data can then serve as a reference for subsequent incentive weights. This approach reduces the computational demands for consistency, defends against malicious model attacks, and facilitates more secure and efficient collaboration.

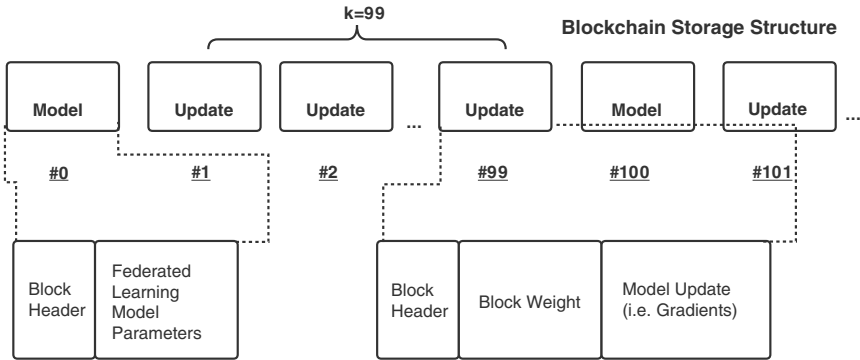


Figure 17.15. Block generation process.

17.3.2.1 Blockchain-based weighted incentive design

By placing federated learning model data on the chain, the consistency of data access among multiple collaborating parties is ensured, creating a transparent and public model data ledger. Each block is responsible for recording a qualified data evidence during the federated learning training process. Global and local model data are stored in different types of blocks, respectively.

Initially, the model is placed in Block #0, followed by the first round of training. Devices retrieve the current global model task for training and execute local training. When there are sufficient update blocks, the smart contract is triggered to begin aggregation. Devices elect a consensus committee to verify and exchange the model data they’ve submitted. Verification nodes upload the validated model data results to a new block. Once this round of training is completed, a new global model block is created and added to the blockchain (as shown in Figure 17.15). Federated learning relies only on the latest model block data and historical block data, with the latter being used for fault recovery or block verification purposes.

k : The total number of update blocks required for validation in each training round (e.g., 100).

t : The current training round ($t = 0, 1, \dots$)

Model block: A block that stores the global model data produced in the current round.

Formula: Block # $k \times (t + 1)$ is the model block produced in training round t .

Update Block: A block that stores the local model data submitted by devices.

Formula: Blocks $\# [k \times (t + 1) + 1, ((t + 1) \times (k + 1)) - 1]$ form the set of update blocks for training round t .

A description of the first 100 blocks, Blocks $\#[0, 100]$, on the blockchain is illustrated in the following:

Block number	Type	Formula	Description
#0	Model Block	$\# t \times (k + 1) = \#0$ When $k = 99, t = 0$	Block #0 is the first block on the blockchain It stores the initial federated learning model data, generated during the initialization of federated training
$\#[1, 99]$	Update Block	$\#[t \times (k + 1) + 1, (t + 1) \times (k + 1)) - 1] = \#[1, 99]$ When $k = 99, t = 0$	$\#[1, 99]$ represent blocks 1–99 on the blockchain (where k is the round and t is 100) They store updates approved during the federated learning process, submitted by devices
#100	Model Block	$\#t \times (k + 1) = \#100$ When $k = 99, t = 1$	#100 represents the 100-th block on the blockchain, generated after the first round of federated training

From an implementation standpoint, a Model Block should include the block header, round t , and global model data.

An Update Block should contain the round t , the locally updated model gradient data, and the weight score, among other fundamental federated training data.

The blockchain's chained structure ensures data determinacy. The key to consensus is appending the correct block to the chain based on a competitive mechanism. Considering computation, communication costs, and consensus reaching, an efficient and secure committee consensus mechanism is established. This mechanism verifies the device's model data before it's added to the chain. The committee consists of several honest consensus nodes responsible for verifying the data submitted by devices and coordinating block creation. The other nodes execute local training and send local updates to the committee. When a new round of training starts, the previous round's committee isn't re-elected. Instead, a new committee is elected based on scoring. Committee members evaluate and score the model data submitted by devices using a quality model evaluation algorithm. Only model data from devices that pass the evaluation are accepted and rewarded by the committee.

17.3.2.2 Weight value calculation

In the context of the blockchain committee-based federated learning scenario mentioned earlier, considering aspects, such as integrity and stability, one can assess the weight value based on each device's local learning accuracy or participation frequency. By comparing the scores in federated learning, we can update the global model data.

Here, we discuss the implementation of evaluating weight values based on learning accuracy.

A Peer is a device that participates in federated training, mainly responsible for training its local data. It then uploads the trained local model data (1, local model update) and sends it to the consensus nodes for model quality assessment (2, local model upload). After each round, a new block is generated through committee consensus (4, aggregator & block generate). The Peer downloads the latest global model (5, global model download) and proceeds with the next local update until the entire model training is completed.

A Node is a consensus committee node elected by all devices, responsible for verifying the model data uploaded by devices. After reaching a consensus with other nodes, it generates a new block to store the model data. In the specific federated learning process, when the Node receives model data from the device node, it verifies it using a given test dataset, discarding any toxic local model updates that might damage the global model. Once transaction validation passes, the Node broadcasts it with its signature to other consensus nodes for verification (3, cross validation). Only models with an accuracy above a given threshold are accepted, leading to the creation of a new, reliable global model for the next iteration (6, global model update). The entire interaction process is shown in Figure 17.16.

Using Mean Square Error (MSE) to evaluate model data as the basis for weight value assessment, the weight value calculation formula is as follows¹:

$$w^{(l)} = w^{(l-1)} + \prod_{i=1}^{N_D} \frac{N_i}{N_S} (w_i^{(l)} - w^{(l-1)}) \quad (17.4)$$

This formula represents the calculation between local weights and global weights in the last iteration of the l th epoch.

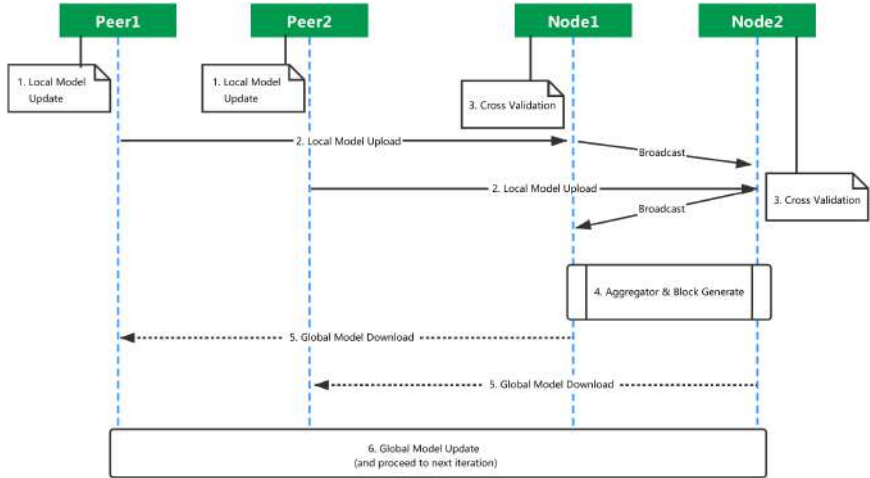


Figure 17.16. Model data update process.

¹Details can be found at <https://arxiv.org/abs/1808.03949>.

Here, w denotes weight, S is the data sample, D represents the device, N is the number of devices, and $SK \in S$. The weight value formula is based on the MSE function $f(w)$.

In the l th epoch iteration, the local model data update uploaded by the device node is

$$(w_i^{(l)}, \{\nabla f_k(w^{(l)})\}_{s_k \in S_i}) \quad (17.5)$$

After verifying all local results for this round, the global model update for the epoch is

$$(w^{(l)}, \nabla f(w^{(l)})) \quad (17.6)$$

Using the length of the local model data submitted by each device as the standard for weight value, once it passes the committee's verification, it's recorded in the update block (update) on the chain. At the end of this training round, the total device weight value equals the sum of the model data sizes in the validated update blocks. A new model block is generated, recording all relevant global model data updates and quantifying the contributions of each device based on weight values for incentives.

At the end of each training round, a new committee is elected to generate the next global data model block. In the distributed model training scenario, this committee's decision-making efficiency will significantly improve the overall performance of global model training.

17.3.3 Incentive Allocation Algorithm

In some federated learning scenarios, certain companies might already dominate the current market with a multitude of customer devices, amassing a wealth of high-quality data resources. During the construction of federated learning models, these resources from these companies become invaluable. However, when collaborating to train a joint global model, these market leaders could inadvertently aid their competitors. Since the federated learning model is shared among all participants, this could diminish the competitive edge of the market leaders. Therefore, any revenue distribution method designed for federated learning should take into account the cost of a participant joining the federation. An effective incentive mechanism is necessary:

one that encourages device updates to local model data, compensates contributors to the global model data, and establishes a fair profit-sharing economic incentive based on contribution.

Given these needs, and in combination with the model quality evaluation MSE algorithm and the marginal utility allocation method (see Figure 17.14 for code example) mentioned in Section 17.3.1, we can enhance this incentive distribution method by introducing device access fees and rewards for other participants, making it more aligned with real-world requirements:

□ Access Fees

Every device pays a fee to access the global model. Once this fee is paid to the model custodian, the device node can freely access the latest model. In the federated learning network, a device must first purchase a model access fee. Only after this purchase is the device trainer allowed to access the model tasks and participate in the global model training process, as shown in Figure 17.17.

```
function newModel(string memory modelId, address[] memory validators, address[] memory trainers,
    address modelBuyer) public onlyFederatedAggr {
    models[modelId] = initModel(modelId, validators, trainers, modelBuyer, msg.sender);
}

/**
 * Adds a new Data Owner to the DataOwners set.
 * @param doAddress the data owner address used as value in the mapping
 */
function setDataOwner(address doAddress) public {
    dataOwners[doAddress] = true;
}

/**
 * Adds a new setModelBuyer to the modelBuyer set.
 * @param doAddress the modelBuyerAddress address used as value in the mapping
 */
function setModelBuyer(address modelBuyerAddress) public {
    modelBuyers[modelBuyerAddress] = true;
}

/**
 * Called form ModelBuyer when ordering training of model.
 */
function payForModel(string memory modelId, uint256 pay) public payable onlyModelBuyer {
    // check parameter ....
    if (payments[modelId] == 0) {
        payments[modelId] = 0;
    }

    payments[modelId] += pay;
    emit ModelCreationPayment(address(this), pay);
}
```

Figure 17.17. Purchase of model access fee.

□ Profit Distribution

After each aggregator completes its round of aggregation, rewards are distributed to the devices based on the accuracy contribution of their submitted model data. Devices that frequently provide updates stand a chance to earn more rewards and continuously download updated global model data, thereby attracting more devices to participate. This incentive mechanism is highly scalable and adaptable to different practical applications.

If the cost incurred by participants is too high, the benefits from the federation might not adequately compensate. Therefore, federated learning might require staged payments to participants. This would further lead to an “interest” issue, as participants essentially lend their resources (like data) to the federation for profit generation.

To ensure the long-term stability of the data federation and gradually attract more high-quality participants in the future, a fair incentive mechanism tailored for the federated learning environment is necessary. Here, building on the incentive algorithm from Section 17.3.1, we have implemented a more optimized incentive allocation method (see Figure 17.18 for code example). This method allows for flexible solutions to the aforementioned profit-sharing issues within the federated learning environment. By maximizing sustainable business objectives, it dynamically distributes the allocated

```
/**
 * payment functions, according to contribution of trainers, validators, orchestrator by ModelBuyer
 */
function generateTrainingPayments(string memory modelId) public onlyModelBuyer isFinished(modelId) {
    ModelData storage model = models[modelId];
    _calculateContributions(modelId);

    // Pay trainers
    for (uint i = 0; i < model.trainers.length; i++) {
        address payable trainer = address(uint160(model.trainers[i]));
        executePayForContribution(modelId, trainer);
    }

    // Pay validators
    for (uint i = 0; i < model.validators.length; i++) {
        address payable validator = address(uint160(model.validators[i]));
        executePayForValidation(modelId, validator);
    }

    // Pay orchestrator
    executePayForOrchestration(modelId, address(uint160(model.federateAggregator)));
    // Return payments to model buyer
    return ModelBuyerPayment(modelId, address(uint160(model.owner)));
}
```

Figure 17.18. Incentive allocation design.

reward budget across all federation participants, minimizing inequalities among them.

To encourage participants to join federated learning, blockchain can ensure fairness through timeout checks and economic penalty mechanisms. Participants that fail to meet deadlines or submit erroneous entries would face financial penalties, with their pre-deposits being revoked or frozen and redistributed to honest participants. This approach further deters dishonest participation and promotes active participation for profit.

By utilizing this market-driven economic incentive model, a balance of interests among all stakeholders can be flexibly achieved within smart contracts. Task incentives, collateral, and timeout penalty mechanisms prevent participants from malicious activities like model attacks, fostering a healthy evolution in model data quality, activity, and effective collaboration.

17.4 Implementation of a Blockchain-Based Federated Learning System

Emerging technologies have been on the rise in recent years. AI, big data, IoT, blockchain, and cloud computing are foundational technologies widely applied across various business scenarios. Each new technology does not exist in isolation but can be organically combined to drive business implementation and innovation. The combination of federated learning and blockchain represents such an innovative attempt. Using the trustworthy, decentralized, immutable, traceable, and auditable characteristics of blockchain can promote the upgrading of traditional federated learning architectures.

17.4.1 System Model

Federated learning provides participants with the capability to collaboratively construct robust machine learning models, using technical measures to ensure data privacy. However, during the federated learning training process, the benefits of collaborative data processing and sharing are not evident. This might lead data owners to lose control over their data, significantly diminishing its utility value once it's out of their hands. Even though integrating data for training can

produce better model performance, how performance gains from such integration are distributed among participants remains uncertain. Federated learning thus requires a decentralized distributed system to technically ensure user privacy and security. Within this framework, effective machine learning must occur while ensuring data security, exchange, and training efficiency.

17.4.1.1 Trustworthy collaborative network

In the current traditional distributed federated learning systems, the model training phase necessitates the presence of a centralized server to aggregate and analyze data, coordinating participants to complete training tasks. Such centralized servers can show biases toward certain clients, potentially deviating from the global model. Moreover, the concentrated functionality of these servers poses risks: If attacked, the system's privacy and security can be compromised. The centralized service node has inherent privacy leak threats. Malicious central servers might tamper with or poison the model, even collecting private client data. Additionally, in the course of business implementation, it's challenging to persuade participants and auditing bodies that the centralized server is trustworthy, creating hurdles for federated learning technology adoption.

Blockchain can assist federated learning in constructing a credible, decentralized internet infrastructure platform. For instance, participants of federated learning can initially connect via a blockchain network. Instant model updates, generated by local training of participants, are synchronized to the blockchain. Once a "model update" is verified on the blockchain, it's broadcasted throughout the network and appended to the blockchain. At this point, the "model update" is distributed across various blockchain nodes and the data added to the blockchain are immutable. Based on this, it effectively circumvents issues presented by centralized servers, ensuring the authenticity, traceability, and auditability of "model updates" produced during federated learning. Using blockchain smart contracts to execute model aggregation can also be automated, triggering when certain predefined conditions are met, eliminating manual intervention. This increases the credibility of the "global model" derived from federated learning model aggregation. The aggregated model update is then stored on the blockchain for sharing among all participants.

Even when new participants join, they can access the latest federated learning global model.

17.4.1.2 Poisoning attacks

A significant security concern within federated learning mechanisms is the “poisoning attack”. As federated learning relies on collaboration between members for training, if a malicious party contaminates samples in the training set and submits poor-quality, unreliable models via “data poisoning”, it can lead to erroneous or failed model updates. Furthermore, during model aggregation, attackers can also introduce “model poisoning” by sending incorrect parameters or damaging the model, undermining the learning process during global aggregation. This reduces the learning task’s accuracy, prolongs the global model’s convergence time, and can significantly diminish the end model’s performance. Although several security protocols have been developed against poisoning attacks, there is still a need for a mechanism to proactively defend against malicious attacks in federated learning.

Blockchain can serve as an effective tool against malicious attacks in federated learning systems. First, blockchain, acting as a distributed storage alternative to centralized model aggregation servers, enhances the security of stored models and model update data. Second, model updates can only be appended to the blockchain when consensus nodes within the blockchain unanimously agree on the model update data from participants. The blockchain retains the history of all model updates, and its chain-like data structure ensures data integrity. Each model update can also be traced back to a specific participant, facilitating audits and detection of data tampering or malicious model substitution. Lastly, blockchain can introduce preset incentive mechanisms through smart contracts. If a node’s model data quality is noticeably below expectations, the system can automatically penalize or even exclude the malicious node from the network.

17.4.1.3 Inference attacks

While federated learning allows participants to train data locally, ensuring other participants can’t access one’s local data, this only offers relative privacy security. There’s still a risk of privacy leakage.

For instance, malicious participants might attempt to extract statistical information from trained models through reverse engineering, inferring sensitive information about others from shared parameters. This poses a serious threat to the privacy of all participants in federated learning.

Current defenses in federated learning use techniques like differential privacy, homomorphic encryption, and secret sharing to guard against reverse engineering of models. However, these defenses can only bolster the model's robustness under certain conditions. For example, adding Gaussian noise during model aggregation introduces differential privacy, resisting inference attacks. However, to enhance privacy protection, more noise must be added during the model training, reducing model usability. Moreover, differential privacy only provides single-point privacy protection. If records are interlinked, attackers can still launch inference attacks on algorithms protected by differential privacy.

Blockchain can bolster federated learning by auditing computation processes, monitoring malicious behaviors during model training, and technically enhancing privacy protection. For example, federated learning model parameters can be stored in the blockchain, ensuring the security and reliability of these parameters.

17.4.1.4 Model and workload evaluation

Model evaluation refers to the process of assessing the performance of one or more established models based on their respective categories, using various metrics. In essence, it determines if the constructed model is “useful”. During the evaluation process in federated learning, local model updates (i.e., local gradients) exceeding a predetermined threshold of high quality (or high precision) are sent to the aggregation server. This server then aggregates received model updates, possibly using the federated averaging algorithm, and sends the aggregated model updates back to the participants. Consequently, the key challenge in this evaluation phase is to strategically select suitable evaluation methods, swiftly identify issues in model selection or training, and iteratively optimize the model. For example, using the federated averaging algorithm might result in some devices being unfairly utilized.

To some extent, blockchain ensures the quality of federated learning model evaluations. It records all model updates produced by federated learning participants during training. These data are immutable and can be traced back to specific participants, ensuring the credibility of model update metadata. Simultaneously, smart contracts on the blockchain can be employed to assess the frequency and quality of contributions from each participant. For each model update submitted to the blockchain network, all nodes can use a pre-determined evaluation function within the smart contract to assess all uploaded model updates, generating an average quality score for each. Given the self-executing and self-verifying nature of smart contracts, they can ensure fairness and reliability in model evaluations, facilitating the extraction of high-quality models from a plethora. Additionally, smart contracts can utilize predefined reward and penalty mechanisms to penalize low-quality contributors and incentivize those with frequent and high-quality contributions, stimulating more users to contribute superior models.

17.4.1.5 Identity management

In federated learning, participants are typically geographically dispersed, making identity verification challenging. Blockchains, inherently decentralized, rely wholly on cryptographic public/private key pairs for identity authentication, with signatures serving as proofs. For instance, in public blockchain projects, the hash of the “public key” is used as the account address, ensuring a level of anonymity as there’s no direct link to the real identity. Participants can publicly disclose their “public keys”, allowing others to recognize them. As long as the “private key” remains secret, only the owner can control the assets, applicable across all network nodes. Moreover, nodes in a public blockchain can freely join or exit.

In contrast, consortium blockchain projects require identity verification of nodes through a membership management service; only authorized nodes may join. Thus, blockchains offer federated learning participants a trustworthy identity authentication mechanism. By leveraging the authorization mechanisms and identity management of consortium blockchains, mistrusting participants can be integrated, establishing a secure and trusted collaboration framework.

17.4.1.6 Introducing audit and regulation

A challenge in federated learning is ensuring that cross-site data transfers during interactions are manageable and auditable. Moreover, convincing auditing institutions about the trustworthiness of centralized model aggregation servers in federated learning is difficult.

Blockchains provide auditing capabilities for the entire federated learning model training process. For example, a blockchain ledger can record all “model update” historical data generated from the beginning to the end of model training. This chained data structure, stored in a distributed fashion across the network nodes, is tamper-proof. Additionally, the blockchain’s chained structure easily traces model updates back to specific participants. Thus, blockchains can adeptly address the challenges of auditing the security of information in federated learning. Furthermore, as a peer-to-peer network, blockchain nodes operate independently and can join or leave at any time without affecting the network’s operation, facilitating third-party audit institutions to join the blockchain network and supervise model update data produced during iterative training.

Hence, as a decentralized, traceable, and tamper-proof distributed shared ledger database, integrating blockchain with federated learning systems ensures data security, data consistency during model training, and protection against malicious attacks. Moreover, blockchain’s value-driven incentives can enhance participants’ enthusiasm in data provision and network model parameter updates.

17.4.2 System Architecture

17.4.2.1 Component architecture

At its core, the blockchain-based federated learning system still primarily hinges on federated learning, only employing blockchain as the intermediary connecting all participants. Given the inclusion of blockchain, this system also introduces features absent in generic federated learning systems, like inherent support for traceability and regulatory auditing.

The architecture of the blockchain federated learning system primarily comprises the Infrastructure Layer, Computation and Storage Layer, Smart Contract Layer, Task Scheduling and Execution Layer, Visualization Layer, and the P2P Network and Supervision Layer (Figure 17.19).

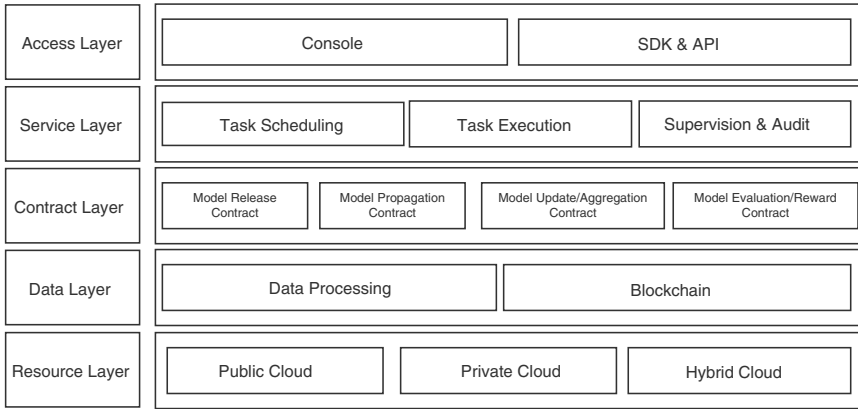


Figure 17.19. Component architecture.

Infrastructure layer: The system supports deployment in various formats, including physical machines, virtual machines, and containers. Container deployment is recommended due to its environment-agnostic nature, facilitating quick testing. For large-scale deployment in production environments, it is advisable to use Kubernetes for management.

Computation and storage layer: Primarily utilizes big data components for unified local data management and storage.

Smart contract layer: Offers functionalities for model dissemination, model update aggregation, model evaluation incentives, and model release.

Task scheduling and execution layer: This component manages the lifecycle of tasks within the federated learning system, providing initial model design services and ensuring state management and coordination across federated learning participants.

17.4.2.2 Process architecture

The learning process of the federated learning system has evolved with the integration of blockchain. In conventional federated learning systems, there is a centralized coordination component. This centralization can be problematic as it is susceptible to single point failures or sporadic betrayals, which can render the system unavailable or distort the predictive outcomes of the model. Additionally, centralized

services can't definitively assert their ownership, potentially leading to mistrust among the various participants and not fully mobilizing their enthusiasm. In contrast, blockchain-based federated learning platforms replace centralized coordination with decentralized architectures and smart contracts. Each participant acts as an equal and independent party in this decentralized framework. Data exchanges are conducted through smart contracts, eliminating the centralized single point of failure, and any malicious actions can be traced.

The process of the blockchain-based federated learning system is represented in Figure 17.20:

1. External users submit TASKs to any participant, who then prepares initial model data.
2. The participant uploads the initial model data to the blockchain and submits it to the smart contract.

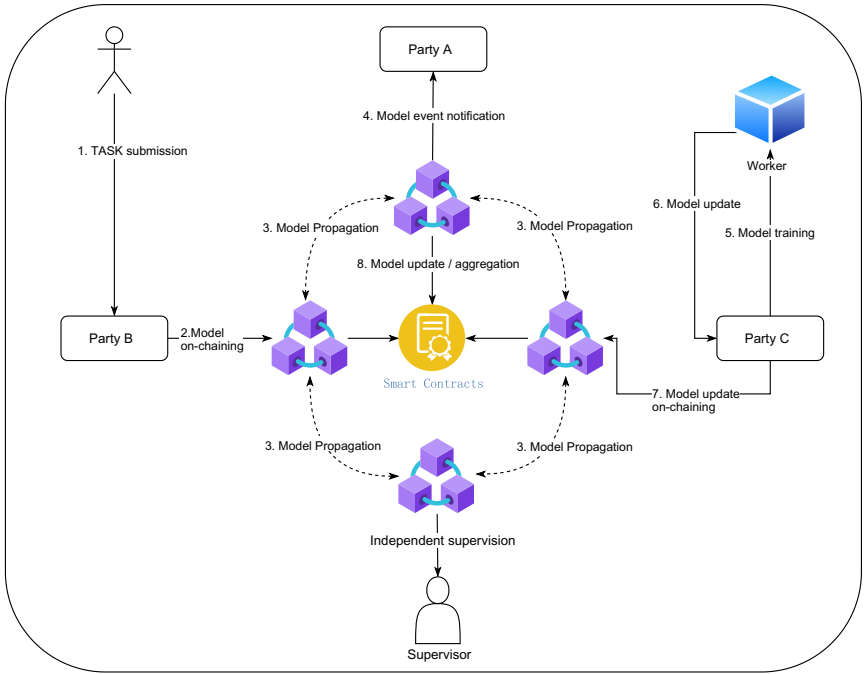


Figure 17.20. Blockchain federated learning system process.

3. The blockchain layer automatically disseminates the model data to all participant nodes.
4. Through decentralized control via the smart contract, model event notifications are automatically triggered, with participants continuously listening to contract events.
5. Once participants receive the contract event, or in other words, the model data, they can initiate local model training.
6. After local model training is completed, local model update data are produced.
7. Participants upload these local model update data to the blockchain.
8. The smart contract aggregates the individually uploaded local model update data from participants based on specific rules.
9. Continually iterate Steps 2–8 for model training until its completion.

The above provides a “bird’s-eye view” of the system process, giving a holistic understanding of how training tasks circulate within the system and the steps involved. Next, we delve into the coordination role of the blockchain from the perspective of smart contracts. Smart contracts, running atop the blockchain layer, serve as a bridge. They receive model update data from above and then, after logical processing, convert these data into a form suitable for blockchain diffusion and storage. Generally speaking, smart contracts should have the following features:

Compute node governance (Optional): The federated learning system and the blockchain’s foundational layer are two relatively independent systems. The bridge connecting them is the smart contract. Registering federated learning compute nodes with a smart contract to some extent logically ties these two systems, enhancing their coherence. On the other hand, registration and deregistration functionalities can be further expanded to manage the lifecycle of compute nodes, implementing features like admittance, expulsion, isolation, and voting for governance.

Random task initiator selection (Optional): The federated learning system is an alliance formed by multiple parties. Atop this

system, each party can independently build its control panel. Users from different panels have diverse tasks. While it's logical for tasks submitted to the federated learning system and blockchain to directly correspond to the initiating compute node, one of the benefits of a blockchain-based federated learning system is its impartiality. The task submitter and initiator can be designed to randomly match, significantly reducing predictability. In implementation, given the decentralized nature of blockchain, the same smart contract logic will be executed independently across multiple nodes. A pseudo-random algorithm should be used to ensure that the outcome is consistent across nodes. By mixing in unpredictable elements while ensuring consistency in the factors considered by each blockchain node, we can randomize task initiation and uphold fairness.

Model (Update) data dissemination: Once the task initiator submits a user training (prediction) request to the blockchain, the smart contract is responsible for disseminating the model (update) data to all compute nodes. This dissemination can follow either a PULL or PUSH mode. In the PULL mode, compute nodes periodically pull data from the blockchain. In contrast, in the PUSH mode, nodes listen to blockchain dissemination events and receive data as it becomes available. Comparatively, the PUSH mode is a more favorable approach.

Model update data aggregation: The blockchain replaces the centralized aggregation service component of traditional federated learning, relegating this function to the smart contract layer. When each computing node completes its local model training, it uploads the model data to the smart contract. The contract, based on predefined rules, aggregates multiple sets of data using federated learning aggregation algorithms once it receives a sufficient amount of local model update data (as defined by strategy). After producing the model data for the current iteration, it is then disseminated back to the computing nodes for the next round of iteration. This process of aggregation and dissemination is repeated in cycles.

Training task lifecycle management: Machine learning model training is not a single-step process. The training procedure requires N rounds of iterations, necessitating smart contracts to automatically determine when to end the training, whether it be reaching a preset number of iterations or when the model converges.

17.4.2.3 Deployment architecture

From the deployment perspective, the most significant change in the blockchain-based federated learning system is the replacement of the original centralized coordination service with the blockchain network, while everything else remains largely unchanged. Figure 17.21 provides a simplified depiction of the deployment architecture.

As the illustration shows, there are four participants: three enterprises and one regulatory authority that together form the federated learning platform. Each enterprise connects to its own blockchain peer node through a business gateway, with all data sharing and transmission occurring over the blockchain network. Within an enterprise, a server receives tasks from the business gateway, which then delegates the task to a worker for machine learning. The results are then sent back through the same route to be consolidated into the blockchain network. This cycle repeats to complete the iterative learning task.

From the perspective of the regulatory participant, they can directly monitor all activities within the system in real time through the regulatory gateway. In the event of any non-compliance, they can quickly intervene.

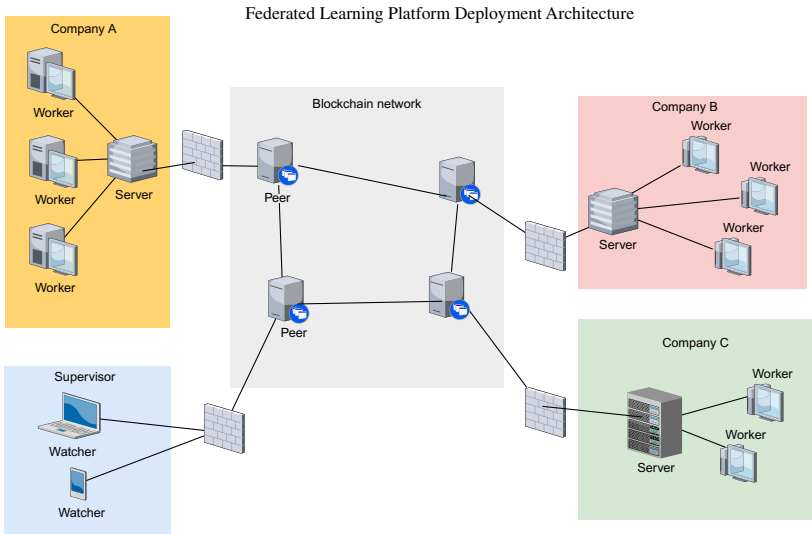


Figure 17.21. Deployment architecture.

This page intentionally left blank

Bibliography

- Abadi M, Barham P, Chen J, *et al.* Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016: 265–283.
- Acar A, Aksu H, Uluagac A S, *et al.* A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)*, 2018, 51(4): 1–35.
- Acharya J, Canonne C L, Tyagi H. Inference under information constraints I: Lower bounds from chi-square contraction. *IEEE Transactions on Information Theory*, 2020, 66(12): 7835–7855.
- Agarwal A, Duchi J C. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, 2011: 873–881.
- Agarwal A, Duchi J C. Distributed delayed stochastic optimization. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, 2012: 5451–5452.
- Agarwal A, Foster D P, Hsu D J, *et al.* Stochastic convex optimization with bandit feedback. In *Advances in Neural Information Processing Systems*, 2011: 1035–1043.
- Aggarwal N, Gupta C, Sharma I. Fully homomorphic symmetric scheme without bootstrapping. In *Proceedings of 2014 International Conference on Cloud Computing and Internet of Things*, 2014: 14–17.
- Agrawal R, Srikant R. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 2000: 439–450.
- Al-Rubaie M, Chang J M. Reconstruction attacks against mobile-based continuous authentication systems in the cloud. *IEEE Transactions on Information Forensics and Security*, 2016, 11(12): 2648–2663.

- Albrecht M R, Farshim P, Faugere J-C, *et al.* Polly cracker, revisited. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2011: 179–196.
- Alistarh D, Grubic D, Li J, *et al.* QSGD: Communication-efficient SGD via gradient quantization and encoding. arXiv preprint arXiv:1610.02132, 2016.
- Allen-Zhu Z, Hazan E. Variance reduction for faster non-convex optimization. In *International Conference on Machine Learning*, 2016: 699–707.
- Allen-Zhu Z, Li Y. Neon2: Finding local minima via first-order oracles. In *Advances in Neural Information Processing Systems*, 2018: 3720–3730.
- Allen-Zhu Z, Yuan Y. Improved SVRG for non-strongly-convex or sum-of-non-convex objectives. In *International Conference on Machine Learning*, 2016: 1080–1089.
- Alperin-Sheriff J, Peikert C. Practical bootstrapping in quasilinear time. In *Annual Cryptology Conference*, 2013: 1–20.
- Alperin-Sheriff J, Peikert C. Faster bootstrapping with polynomial error. In *Annual Cryptology Conference*, 2014: 297–314.
- Ang F, Chen L, Zhao N, *et al.* Robust federated learning with noisy communication. *IEEE Transactions on Communications*, 2020, 68(6): 3452–3464. <http://dx.doi.org/10.1109/TCOMM.2020.2979149>.
- Aono Y, Hayashi T, Phong L T, *et al.* Fast and secure linear regression and biometric authentication with security update. *IACR Cryptology ePrint Archive*, 2015, 2015: 692.
- Aono Y, Hayashi T, Phong L T, *et al.* Scalable and secure logistic regression via homomorphic encryption. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, 2016: 142–144.
- Aono Y, Hayashi T, Wang L, *et al.* Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 2017, 13(5): 1333–1345.
- Archer D W, Bogdanov D, Lindell Y, *et al.* From keys to databases — Real-world applications of secure multi-party computation. *The Computer Journal*, 2018, 61(12): 1749–1771.
- Ashizawa N, Yanai N, Cruz J P, *et al.* Eth2Vec: Learning contract-wide code representations for vulnerability detection on ethereum smart contracts. In *Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure*, 2021: 47–59.

- Aslett L, *et al.* HomomorphicEncryption: Fully Homomorphic Encryption. R package version 0.2. <http://www.louisaslett.com/HomomorphicEncryption>, 2014.
- Avron H, Druinsky A, Gupta A. Revisiting asynchronous linear solvers: Provable convergence rate through randomization. *Journal of the ACM (JACM)*, 2015, 62(6): 1–27.
- Bach F, Moulines E. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Neural Information Processing Systems (NIPS)*, 2011.
- Bach F, Perchet V. Highly-smooth zero-th order online optimization. In *Conference on Learning Theory*, 2016: 257–283.
- Bach F, Jenatton R, Mairal J, *et al.* Convex optimization with sparsity-inducing norms. *Optimization for Machine Learning*, 2011, 5.
- Badshah N. Facebook to contact 87 million users affected by data breach. *The Guardian*, 2018. <https://www.theguardian.com/technology/2018/apr/08/facebook-to-contact-the-87-million-users-affected-by-data-breach>.
- Bambauer J, Muralidhar K, Sarathy R. Fool’s gold: an illustrated critique of differential privacy. *Vanderbit Journal of Environment and Technology Law*, 2013, 16: 701.
- Baran P. On distributed communications networks. *IEEE Transactions on Communications Systems*, 1964, 12(1): 1–9.
- Barbosa P, Brito A, Almeida H. A technique to provide differential privacy for appliance usage in smart metering. *Information Sciences*, 2016, 370: 355–367.
- Barto A G, Sutton R S, Anderson C W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 1983(5): 834–846.
- Bater J, Elliott G, Eggen C, *et al.* smcql: Secure Querying for Federated Databases. *Proceedings of the VLDB Endowment*, 2017, 10(6).
- Beaver D. Efficient multiparty protocols using circuit randomization. In *Lecture Notes in Computer Science, Vol 576: Advances in Cryptology — CRYPTO ’91, 11th Annual International Cryptology Conference*, Santa Barbara, California, USA, August 11–15, 1991, Proceedings. Springer, 1991: 420–432. http://dx.doi.org/10.1007/3-540-46766-1_34.
- Beck A, Teboulle M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2009, 2(1): 183–202.

- Beck A, Tetruashvili L. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 2013, 23(4): 2037–2060.
- Beimel A. Secret-sharing schemes: A survey. In *International Conference on Coding and Cryptology*, 2011: 11–46.
- Bellman R. Dynamic programming. *Science*, 1966, 153(3731): 34–37.
- Ben-Or M, Goldwasser S, Wigderson A. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 1988: 1–10. <https://doi.org/10.1145/62212.62213>.
- Benaloh J. Dense probabilistic encryption. In *Proceedings of the Workshop on Selected Areas of Cryptography*, 1994: 120–128.
- Bent R, Van Hentenryck P. Online stochastic optimization without distributions. In *ICAPS*, Vol. 5, 2005: 171–180.
- Bergamaschi F. HELib. <https://github.com/homenc/HElib>, 2019.
- Berlinet A, Thomas-Agnan C. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Springer Science & Business Media, 2011.
- Bernstein J, Zhao J, Azizzadenesheli K, *et al.* signSGD with majority vote is communication efficient and fault tolerant. arXiv preprint arXiv:1810.05291, 2018.
- Bertsekas D P, *et al.* *Dynamic Programming and Optimal Control*, Vol. 1. Athena scientific Belmont, 2000.
- Bertsekas D P, Nedi A, Ozdaglar A E, *et al.* *Convex analysis and optimization*, 2003.
- Bianchi P, Fort G, Hachem W. Performance of a distributed stochastic approximation algorithm. *IEEE Transactions on Information Theory*, 2013, 59(11): 7405–7418.
- Blakley G. Safeguarding cryptographic keys. In *International Workshop on Managing Requirements Knowledge (MARK)*, 1979.
- Blondel M, Seki K, Uehara K. Block coordinate descent algorithms for large-scale sparse multiclass classification. *Machine Learning*, 2013, 93(1): 31–52.
- Boemer F, Lao Y, Cammarota R, *et al.* ngraph-he: A graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, 2019: 3–13.
- Boenisch F. Differential privacy: General survey and analysis of practicality in the context of machine learning.
- Bogdanov D, Laur S, Willemson J. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, 2008: 192–206.

- Bogdanov D, Kamm L, Laur S, *et al.* Privacy-preserving statistical data analysis on federated databases. In *Annual Privacy Forum*, 2014: 30–55.
- Bolte J, Sabach S, Teboulle M. Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *Mathematical Programming*, 2014, 146(1–2): 459–494.
- Bonawitz K, Ivanov V, Kreuter B, *et al.* Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017: 1175–1191.
- Boneh D, Franklin M. Efficient generation of shared RSA keys. *Journal of ACM*, 2001, 48(4): 702–722. <https://doi.org/10.1145/502090.502094>.
- Boneh D, Goh E-J, Nissim K. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography Conference*, 2005: 325–341.
- Bonte C, Vercauteren F. Privacy-preserving logistic regression training. *BMC Medical Genomics*, 2018, 11(4): 86.
- Bost R, Popa R A, Tu S, *et al.* Machine learning classification over encrypted data. In *NDSS*, Vol. 4324, 2015: 4325.
- Bottou L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, Springer, 2010: 177–186.
- Boyd S, Vandenberghe L. *Convex Optimization*. Cambridge University Press, 2004.
- Boţ R I, Csetnek E R, László S C. An inertial forward–backward algorithm for the minimization of the sum of two nonconvex functions. *EURO Journal on Computational Optimization*, 2016, 4(1): 3–25.
- Bradley J K, Kyrola A, Bickson D, *et al.* Parallel coordinate descent for l1-regularized loss minimization. arXiv preprint arXiv:1105.5379, 2011.
- Brakerski Z, Vaikuntanathan V. Fully homomorphic encryption from ring-LWE and security for key dependent messages. *Annual Cryptology Conference*, 2011: 505–524.
- Brakerski Z, Vaikuntanathan V. Efficient fully homomorphic encryption from (standard) LWE. *SIAM Journal on Computing*, 2014, 43(2): 831–871.
- Brakerski Z, Gentry C, Vaikuntanathan V. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 2014, 6(3): 1–36.
- Braverman M, Garg A, Ma T, *et al.* Communication lower bounds for statistical estimation problems via a distributed data processing inequality. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, 2016: 1011–1020.
- Brenner M, Moore T, Smith M. *Financial Cryptography and Data Security*, Springer, 2014.
- Brochu E, Cora V M, De Freitas N. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and

- hierarchical reinforcement learning. arXiv preprint arXiv:1012.2599, 2010.
- Browne C B, Powley E, Whitehouse D, *et al.* A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 2012, 4(1): 1–43.
- Brutzkus A, Elisha O, Gilad-Bachrach R. Low latency privacy preserving inference. arXiv preprint arXiv:1812.10659, 2018.
- Bubeck S, Cesa-Bianchi N, *et al.* Regret analysis of stochastic and non-stochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 2012, 5(1): 1–122.
- Bui D, Malik K, Goetz J, *et al.* Federated user representation learning. arXiv preprint arXiv:1909.12535, 2019.
- Byrd R H, Hansen S, Nocedal J, *et al.* A stochastic quasi-Newton method for large-scale optimization. *SIAM Journal on Optimization*, 2016, 26(2): 1008–1031.
- Caldas S, Konečný J, McMahan H B, *et al.* Expanding the reach of federated learning by reducing client resource requirements. arXiv preprint arXiv:1812.07210, 2018a.
- Caldas S, Wu P, Li T, *et al.* Leaf: A benchmark for federated settings. arXiv preprint arXiv:1812.01097, 2018b.
- Callebaut D. Generalization of the Cauchy-Schwarz inequality. *Journal of Mathematical Analysis and Applications*, 1965, 12(3): 491–494.
- Candès E J, Li X, Ma Y, *et al.* Robust principal component analysis?. *Journal of the ACM (JACM)*, 2011, 58(3): 11.
- Cao Q, Guo Z-C, Ying Y. Generalization bounds for metric and similarity learning. *Machine Learning*, 2016, 102(1): 115–132.
- Carlini N, Liu C, Kos J, *et al.* The secret sharer: Measuring unintended neural network memorization & extracting secrets. CoRR abs/1802.08232 (2018). arXiv preprint arXiv:1802.08232, 2018.
- Carpov S, Gama N, Georgieva M, *et al.* Privacy-preserving semi-parallel logistic regression training with Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2019, 2019: 101.
- Cha H, Park J, Kim H, *et al.* Federated reinforcement distillation with proxy experience memory. arXiv preprint arXiv:1907.06536, 2019.
- Cha H, Park J, Kim H, *et al.* Proxy experience replay: Federated distillation for distributed reinforcement learning. *IEEE Intelligent Systems*, 2020, 35(4): 94–101.
- Chai Z, Fayyaz H, Fayyaz Z, *et al.* Towards taming the resource and data heterogeneity in federated learning. In *2019 {USENIX} Conference on Operational Machine Learning (OpML 19)*, 2019: 19–21.
- Chai Z, Chen Y, Zhao L, *et al.* Fedat: A communication-efficient federated learning method with asynchronous tiers under non-iid data. arXiv preprint arXiv:2010.05958, 2020a.

- Chai Z, Ali A, Zawad S, *et al.* Tifl: A tier-based federated learning system. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, 2020b: 125–136.
- Challa R. Homomorphic encryption: Review and applications. In *Advances in Data Science and Management*. Springer, 2020: 273–281.
- Champagne D, Lee R B. Scalable architectural support for trusted software. In *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, 2010: 1–12.
- Chandra R. *Parallel Programming in OpenMP*. Morgan Kaufmann, 2001.
- Chandran N, Gupta D, Rastogi A, *et al.* EzPC: Programmable, efficient, and scalable secure two-party computation for machine learning. *ePrint Report*, 2017, 1109.
- Chang C-C, Lin C-J. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2011, 2: 1–27.
- Chapelle O, Zien A. Semi-supervised classification by low density separation. In *AISTATS*, 2005: 57–64.
- Chapelle O, Chi M, Zien A. A continuation method for semi-supervised SVMs. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006: 185–192.
- Chaturapruek S, Duchi J C, Ré C. Asynchronous stochastic convex optimization: the noise is in the noise and SGD don't care. *Advances in Neural Information Processing Systems*, 2015, 28: 1531–1539.
- Chen P-L, Tsai C-T, Chen Y-N, *et al.* A linear ensemble of individual and blended models for music rating prediction. In *Proceedings of the 2011 International Conference on KDD Cup 2011*, Vol. 18, 2011: 21–60.
- Chen L, Ben H, Huang J. An encryption depth optimization scheme for fully homomorphic encryption. In *2014 International Conference on Identification, Information and Knowledge in the Internet of Things*, 2014a: 137–141.
- Chen Z, Wang J, Zhang Z, *et al.* A fully homomorphic encryption scheme with better key size. *China Communications*, 2014b, 11(9): 82–92.
- Chen J, Pan X, Monga R, *et al.* Revisiting distributed synchronous SGD. arXiv preprint arXiv:1604.00981, 2016a.
- Chen T, Guestrin C. XGBoost: A scalable tree boosting system. In *KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2016b: 785–794. <https://doi.org/10.1145/2939672.2939785>.
- Chen H, Wang Y. Kernel-based sparse regression with the correntropy-induced loss. *Applied and Computational Harmonic Analysis*, 2018, 44(1): 144–164.

- Chen Y, Ning Y, Slawski M, *et al.* Asynchronous online federated learning for edge devices with non-iid data. arXiv preprint arXiv:1911.02134, 2019a.
- Chen M, Mao B, Ma T. Efficient and robust asynchronous federated learning with stragglers. In *Submitted to International Conference on Learning Representations*, 2019b.
- Chen M, Mathews R, Ouyang T, *et al.* Federated learning of out-of-vocabulary words. arXiv preprint arXiv:1903.10635, 2019c.
- Chen Y, Lan H, Du Z, *et al.* An instruction set architecture for machine learning. *ACM Transactions on Computer Systems (TOCS)*, 2019d, 36(3): 1–35.
- Cheng X, Fang L, Yang L, *et al.* Mobile big data: The fuel for data-driven wireless. *IEEE Internet of Things Journal*, 2017, 4(5): 1489–1516.
- Cheng K, Fan T, Jin Y, *et al.* Secureboost: A lossless federated learning framework. arXiv preprint arXiv:1901.08755, 2019.
- Cheon J H, Kim A, Kim M, *et al.* Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2017: 409–437.
- Cheon J H, Han K, Kim A, *et al.* Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2018a: 360–384.
- Cheon J H, Kim D, Kim Y, *et al.* Ensemble method for privacy-preserving logistic regression based on homomorphic encryption. *IEEE Access*, 2018b, 6: 46938–46948.
- Cheon J H, Kim D, Park J H. Towards a practical cluster analysis over encrypted data. In *International Conference on Selected Areas in Cryptography*, 2019: 227–249.
- Cho Y J, Wang J, Joshi G. Client selection in federated learning: Convergence analysis and power-of-choice selection strategies. arXiv preprint arXiv:2010.01243, 2020.
- Coates *et al.* “Deep learning with COTS HPC systems”, 2013.
- Conroy B, Sajda P. Fast, exact model selection and permutation testing for l2-regularized logistic regression. In *Artificial Intelligence and Statistics*, 2012: 246–254.
- Coron J-S, Mandal A, Naccache D, *et al.* Fully homomorphic encryption over the integers with shorter public keys. In *Annual Cryptology Conference*, 2011: 487–504.
- Coron J-S, Naccache D, Tibouchi M. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2012: 446–464.

- Coron J-S, Lepoint T, Tibouchi M. Scale-invariant fully homomorphic encryption over the integers. In *International Workshop on Public Key Cryptography*, 2014: 311–328.
- Cotter A, Shamir O, Srebro N, *et al.* Better mini-batch algorithms via accelerated gradient methods. In *Advances in Neural Information Processing Systems*, 2011: 1647–1655.
- Cramer R, Damgård I B, Nielsen J B. *Secure Multiparty Computation and Secret Sharing*, (1st edn) USA: Cambridge University Press, 2015.
- Crawford J L, Gentry C, Halevi S, *et al.* Doing real work with FHE: the case of logistic regression. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2018: 1–12.
- Dai B, Xie B, He N, *et al.* Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems*, 2014: 3041–3049.
- Dalcin L D, Paz R R, Kler P A, *et al.* Parallel distributed computing using Python. *Advances in Water Resources*, 2011, 34(9): 1124–1139.
- Damgård I, Jurik M. A generalisation, a simplification and some applications of paillier’s probabilistic public-key system. In *International Workshop on Public Key Cryptography*, 2001: 119–136.
- Dang Z, Li X, Gu B, *et al.* Large-scale nonlinear AUC maximization via triply stochastic gradients. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.
- De A. Lower bounds in differential privacy. In *Theory of Cryptography Conference*, 2012: 321–338.
- Dean J, Corrado G S, Monga R, *et al.* Large scale distributed deep networks, 2012.
- Defazio A, Bach F, Lacoste-Julien S. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, 2014: 1646–1654.
- Deisenroth M, Rasmussen C E. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011: 465–472.
- Dekel O, Gilad-Bachrach R, Shamir O, *et al.* Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 2012, 13(1).
- Demmler D, Schneider T, Zohner M. ABY-A framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.
- Deng L. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, 2014, 3.

- Dinur I, Nissim K. Revealing information while preserving privacy. In *Proceedings of the Twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2003: 202–210.
- Djatkiko M, Hardy S, Henecka W, *et al.* Privacy-preserving entity resolution and logistic regression on encrypted data. *Private and Secure Machine Learning (PSML)*, 2017.
- Dolev S, Gupta P, Li Y, *et al.* Privacy-preserving secret shared computations using mapreduce. *IEEE Transactions on Dependable and Secure Computing*, 2019.
- Dror G, Koenigstein N, Koren Y, *et al.* The Yahoo! Music Dataset and KDD-Cup’11. In *KDD Cup*, 2012: 8–18.
- Du W, Zhan Z. Building decision tree classifier on private data. In *Proceedings of the IEEE International Conference on Privacy, Security and Data Mining*, Vol. 14, 2002: 1–8.
- Du W, Atallah M J, *et al.* Privacy-preserving cooperative scientific computations. In *CSFW*, Vol. 1, 2001: 273.
- Du W, Han Y S, Chen S. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, 2004: 222–233.
- Duchesne L, Karangelos E, Wehenkel L. Machine learning of real-time power systems reliability management response. In *2017 IEEE Manchester PowerTech*, 2017: 1–6.
- Duchi J, Singer Y. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 2009, 10(Dec): 2899–2934.
- Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011, 12(7): 2121–2159.
- Duchi J C, Bartlett P L, Wainwright M J. Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization*, 2012, 22(2): 674–701.
- Dudik M, Harchaoui Z, Malick J. Lifted coordinate descent for learning with trace-norm regularization. In *Artificial Intelligence and Statistics*, 2012: 327–336.
- Dwork C. An ad omnia approach to defining and achieving private data analysis. In *International Workshop on Privacy, Security, and Trust in KDD*, 2007: 1–13.
- Dwork C. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, 2008: 1–19.
- Dwork C. A firm foundation for private data analysis. *Communications of the ACM*, 2011, 54(1): 86–95.

- Dwork C, Naor M. On the difficulties of disclosure prevention in statistical databases or the case for differential privacy. *Journal of Privacy and Confidentiality*, 2010, 2(1).
- Dwork C, Mcsherry F, Nissim K, *et al.* Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, 2006: 265–284.
- Dwork C, Mcsherry F, Nissim K, *et al.* Differential privacy — A primer for the perplexed. In *Joint UNECE/Eurostat Work Session on Statistical Data Confidentiality*, 2011: 11.
- Dwork C, Roth A, *et al.* The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 2014, 9(3–4): 211–407.
- Elgamal T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 1985, 31(4): 469–472.
- Emekci F, Agrawal D, Abbadi A E, *et al.* Privacy preserving query processing using third parties. In *22nd International Conference on Data Engineering (ICDE'06)*, 2006: 27–27.
- Erkin Z, Veugen T, Toft T, *et al.* Generating private recommendations efficiently using homomorphic encryption and data packing. *IEEE Transactions on Information Forensics and Security*, 2012, 7(3): 1053–1066.
- Erlingsson Ú, Pihur V, Korolova A. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014: 1054–1067.
- Espeholt L, Soyer H, Munos R, *et al.* Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, 2018: 1407–1416.
- EU. Regulation (EU) 2016/679 of the European Parliament and of the Council on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). <https://eur-lex.europa.eu/eli/reg/2016/679/oj>, 2016.
- Evans D, Kolesnikov V, Rosulek M. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2017, 2(2–3).
- Falkner S, Klein A, Hutter F. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *International Conference on Machine Learning*, 2018: 1436–1445.
- Fan J, Li R. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 2001, 96(456): 1348–1360.

- Fang C, Li C J, Lin Z, *et al.* Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator. In *Advances in Neural Information Processing Systems*, 2018: 689–699.
- Fellows M, Koblitz N. Combinatorial cryptosystems galore!. *Contemporary Mathematics*, 1994, 168: 51.
- Feng J, Jain A K. Fingerprint reconstruction: from minutiae to phase. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010, 33(2): 209–223.
- Feng Y, Huang X, Shi L, *et al.* Learning with the maximum correntropy criterion induced losses for regression. *Journal of Machine Learning Research*, 2015, 16: 993–1034.
- Feyzmahdavian H R, Aytekin A, Johansson M. An asynchronous mini-batch algorithm for regularized stochastic optimization. In *2015 54th IEEE Conference on Decision and Control (CDC)*, 2015: 1384–1389.
- Feyzmahdavian H R, Aytekin A, Johansson M. An asynchronous mini-batch algorithm for regularized stochastic optimization. *IEEE Transactions on Automatic Control*, 2016, 61(12): 3740–3754.
- Fouque P-A, Poupard G, Stern J. Sharing decryption in the context of voting or lotteries. In *International Conference on Financial Cryptography*, 2000: 90–104.
- Franceschi L, Donini M, Frasconi P, *et al.* Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70, 2017: 1165–1173.
- Franceschi L, Frasconi P, Salzo S, *et al.* Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, 2018: 1563–1572.
- Frank M, Wolfe P. An algorithm for quadratic programming. *Naval Research Logistics (NRL)*, 1956, 3(1–2): 95–110.
- François-Lavet V, Henderson P, Islam R, *et al.* An introduction to deep reinforcement learning. arXiv preprint arXiv:1811.12560, 2018.
- Fredrikson M, Jha S, Ristenpart T. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015: 1322–1333.
- Freedman D A. *Statistical Models: Theory and Practice*. Cambridge University Press, 2009.
- Friedman J, Hastie T, Tibshirani R. *The Elements of Statistical Learning*, Vol. 1. Springer series in statistics, New York, 2001.
- Fung B C, Wang K, Chen R, *et al.* Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (Csur)*, 2010, 42(4): 1–53.

- Gabriel E, Fagg G E, Bosilca G, *et al.* Open MPI: Goals, concept, and design of a next generation MPI implementation. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, 2004: 97–104.
- Garber D, Hazan E. Faster rates for the frank-wolfe method over strongly-convex sets. arXiv preprint arXiv:1406.1305, 2014.
- Garber D, Hazan E. Faster rates for the Frank-Wolfe method over strongly-convex sets. In *ICML*, Vol. 15, 2015: 541–549.
- Gascón A, Schoppmann P, Balle B, *et al.* Secure linear regression on vertically partitioned datasets. *IACR Cryptology ePrint Archive*, 2016, 2016: 892.
- Gascón A, Schoppmann P, Balle B, *et al.* Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies*, 2017, 2017(4): 345–364.
- Geng X, Gu B, Li X, *et al.* Scalable semi-supervised SVM via triply stochastic gradients. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 2019: 2364–2370.
- Gentry C. *A Fully Homomorphic Encryption Scheme*, Vol. 20. Stanford University, Stanford, 2009.
- Gentry C. Toward basing fully homomorphic encryption on worst-case hardness. In *Annual Cryptology Conference*, 2010: 116–137.
- Gentry C, Halevi S. Implementing gentry's fully-homomorphic encryption scheme. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2011: 129–148.
- Gentry C, Halevi S, Peikert C, *et al.* Ring switching in BGV-style homomorphic encryption. In *International Conference on Security and Cryptography for Networks*, 2012: 19–37.
- Gentry C, Sahai A, Waters B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Annual Cryptology Conference*, 2013: 75–92.
- Geurts P, Ernst D, Wehenkel L. Extremely randomized trees. *Machine Learning*, 2006, 63(1): 3–42.
- Geyer R C, Klein T, Nabi M. Differentially private federated learning: A client level perspective. arXiv preprint arXiv:1712.07557, 2017.
- Ghadimi S, Lan G. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 2013, 23(4): 2341–2368.
- Ghadimi S, Lan G. Accelerated gradient methods for nonconvex nonlinear and stochastic programming. *Mathematical Programming*, 2016, 156(1–2): 59–99.

- Ghafoor A, Muftic S, Schmölzer G. CryptoNET: Design and implementation of the secure email system. In *2009 Proceedings of the 1st International Workshop on Security and Communication Networks*, 2009: 1–6.
- Gilad-Bachrach R, Dowlin N, Laine K, *et al.* Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, 2016: 201–210.
- Goldfarb D, Iyengar G, Zhou C. Linear convergence of stochastic Frank-Wolfe variants. arXiv preprint arXiv:1703.07269, 2017.
- Goldreich O, Goldwasser S, Halevi S. Public-key cryptosystems from lattice reduction problems. In *Annual International Cryptology Conference*, 1997: 112–131.
- Gong Y, Fang Y, Guo Y. Private data analytics on biomedical sensing data via distributed computation. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2016, 13(3): 431–444.
- Goodfellow I, Bengio Y, Courville A, *et al.* *Deep Learning*, Vol. 1. MIT Press Cambridge, 2016.
- Goodman B, Flaxman S. European Union regulations on algorithmic decision-making and a “right to explanation”. *AI magazine*, 2017, 38(3): 50–57.
- Gordon G J. Stable fitted reinforcement learning. In *Advances in Neural Information Processing Systems*, 1996: 1052–1058.
- Graham R L, Woodall T S, Squyres J M. Open MPI: A flexible high performance MPI. In *International Conference on Parallel Processing and Applied Mathematics*, 2005: 228–239.
- Grama M, Musat M, Muñoz-González L, *et al.* Robust aggregation for adaptive privacy preserving federated learning in healthcare. arXiv preprint arXiv:2009.08294, 2020.
- Grcar J F. A matrix lower bound. *Linear Algebra and Its Applications*, 2010, 433(1): 203–220.
- Gropp W, Gropp W D, Lusk E, *et al.* *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, Vol. 1[M]. MIT Press, 1999.
- Gu B, Ling C. A new generalized error path algorithm for model selection. In *International Conference on Machine Learning*, 2015: 2549–2558.
- Gu B, Huo Z, Huang H. Asynchronous stochastic block coordinate descent with variance reduction. arXiv preprint arXiv:1610.09447, 2016.
- Gu S, Lillicrap T, Ghahramani Z, *et al.* Q-prop: Sample-efficient policy gradient with an off-policy critic. arXiv preprint arXiv:1611.02247, 2016.
- Gu B, Liu G, Huang H. Groups-keeping solution path algorithm for sparse regression with automatic feature grouping. In *KDD*, 2017: 185–193.

- Gu B, Wang D, Huo Z, *et al.* Inexact proximal gradient methods for non-convex and non-smooth optimization. In *AAAI*, 2018: 3093–3100.
- Gu B, Xin M, Huo Z, *et al.* Asynchronous doubly stochastic sparse kernel learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- Gu B, Huo Z, Huang H. Asynchronous doubly stochastic group regularized learning. In *International Conference on Artificial Intelligence and Statistics*, 2018: 1791–1800.
- Gu B, Huo Z, Deng C, *et al.* Faster Derivative-Free Stochastic Algorithm for Shared Memory Machines. In *International Conference on Machine Learning*, 2018: 1807–1816.
- Gu B, Dang Z, Li X, *et al.* Federated doubly stochastic kernel learning for vertically partitioned data. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining on Applied Data Science Track*, 2020.
- Gu B, Xu A, Deng C, *et al.* Privacy-preserving asynchronous federated learning algorithms for multi-party vertically collaborative learning. arXiv preprint arXiv:2008.06233, 2020.
- Guo B, Wang Z, Yu Z, *et al.* Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm. *ACM Computing Surveys (CSUR)*, 2015, 48(1): 1–31.
- Guo J, Liu Z, Lam K-Y, *et al.* Secure weighted aggregation in federated learning. arXiv preprint arXiv:2010.08730, 2020.
- Haarnoja T, Tang H, Abbeel P, *et al.* Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, 2017: 1352–1361.
- Hafner R, Riedmiller M. Reinforcement learning in feedback control. *Machine Learning*, 2011, 84(1–2): 137–169.
- Halevi S, Shoup V. Algorithms in helib. In *Annual Cryptology Conference*, 2014: 554–571.
- Halevi S, Shoup V. Helib. <https://github.com/shaih/HElib>, 2014.
- Halko N, Martinsson P-G, Tropp J A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 2011, 53(2): 217–288.
- Hallman R A, Laine K, Dai W, *et al.* Building applications with homomorphic encryption[C] //Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018: 2160–2162.
- Han K, Hong S, Cheon J H, *et al.* Efficient logistic regression on large encrypted data. *IACR Cryptology ePrint Archive*, 2018a, 2018: 662.
- Han Y, Özgür A, Weissman T. Geometric lower bounds for distributed parameter estimation under communication constraints. In *Conference on Learning Theory*, 2018b: 3163–3188.

- Hao T, Huang Y, Wen X, *et al.* Edge AIBench: towards comprehensive end-to-end edge computing benchmarking. In *International Symposium on Benchmarking, Measuring and Optimization*, 2018: 23–30.
- Hard A, Rao K, Mathews R, *et al.* Federated learning for mobile keyboard prediction. arXiv preprint arXiv:1811.03604, 2018.
- Hardt M, Talwar K. On the geometry of differential privacy. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, 2010: 705–714.
- Hardy S, Henecka W, Ivey-Law H, *et al.* Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. arXiv preprint arXiv:1711.10677, 2017.
- Hartmann F, Suh S, Komarzewski A, *et al.* Federated learning for ranking browser history suggestions. arXiv preprint arXiv:1911.11807, 2019.
- Hazan E, Kale S. Beyond the regret minimization barrier: optimal algorithms for stochastic strongly-convex optimization. *The Journal of Machine Learning Research*, 2014, 15(1): 2489–2512.
- Hazan E, Luo H. Variance-reduced and projection-free stochastic optimization. In *International Conference on Machine Learning*, 2016: 1263–1271.
- He R, Zheng W-S, Hu B-G. Maximum correntropy criterion for robust face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011, 33(8): 1561–1576.
- He Y, Hu X, He J, *et al.* A review of privacy and security issues in machine learning systems. *Journal of Computer Research and Development*, 2019, 56(10): 2049.
- Heess N, Wayne G, Silver D, *et al.* Learning continuous control policies by stochastic value gradients. arXiv preprint arXiv:1510.09142, 2015.
- Hessel M, Modayil J, Van Hasselt H, *et al.* Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32, 2018.
- Hino H, Shen H, Murata N, *et al.* A versatile clustering method for electricity consumption pattern analysis in households. *IEEE Transactions on Smart Grid*, 2013, 4(2): 1048–1057.
- Hinton G, Srivastava N, Swersky K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 2012, 14(8).
- Ho Q, Cipar J, Cui H, *et al.* More effective distributed ML via a stale synchronous parallel parameter server. *Advances in Neural Information Processing Systems*, 2013, 2013: 1223.
- Hoffstein J, Pipher J, Silverman J H. NTRU: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*, 1998: 267–288.

- Hofmann T, Lucchi A, Lacoste-Julien S, *et al.* Variance reduced stochastic gradient descent with neighbors. In *Advances in Neural Information Processing Systems*, 2015: 2305–2313.
- Hong M. A distributed, asynchronous, and incremental algorithm for non-convex optimization: An ADMM approach. *IEEE Transactions on Control of Network Systems*, 2017, 5(3): 935–945.
- Hong M, Wang X, Razaviyayn M, *et al.* Iteration complexity analysis of block coordinate descent methods. *Mathematical Programming*, 2017, 163(1–2): 85–114.
- Horvath S, Ho C-Y, Horvath L, *et al.* Natural compression for distributed deep learning. arXiv preprint arXiv:1905.10988, 2019.
- Hsieh C-J, Olsen P A. Nuclear norm minimization via active subspace selection. In *ICML'14: Proceedings of the 31st International Conference on International Conference on Machine Learning*, 2014: 575–583.
- Hsieh C-J, Chang K-W, Lin C-J, *et al.* A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the 25th International Conference on Machine Learning*, 2008: 408–415.
- Hsieh C-J, Yu H-F, Dhillon I S. Passcode: Parallel asynchronous stochastic dual co-ordinate descent. arXiv preprint, 2015.
- Hsieh C-J, Natarajan N, Dhillon I S. PU learning for matrix completion. In *ICML*, 2015: 2445–2453.
- Hsieh K, Harlap A, Vijaykumar N, *et al.* Gaia: Geo-distributed machine learning approaching {LAN} speeds. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017: 629–647.
- Hu Y, Niu D, Yang J, *et al.* FDML: A collaborative machine learning framework for distributed features. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019: 2232–2240.
- Huang X, Shi L, Suykens J A. Ramp loss linear programming support vector machine. *The Journal of Machine Learning Research*, 2014, 15(1): 2185–2211.
- Huang F, Chen S, Huang H. Faster stochastic alternating direction method of multipliers for nonconvex optimization. In *ICML*, 2019: 2839–2848.
- Huang F, Gao S, Pei J, *et al.* Accelerated zeroth-order momentum methods from mini to minimax optimization. arXiv preprint arXiv:2008.08170, 2020.
- Huo Z, Huang H. Asynchronous stochastic gradient descent with variance reduction for non-convex optimization. arXiv preprint arXiv:1604.03584, 2016.
- Huo Z, Huang H. Distributed asynchronous dual free stochastic dual coordinate ascent. arXiv preprint arXiv:1605.09066, 2016.

- Huo Z, Huang H. Asynchronous mini-batch gradient descent with variance reduction for non-convex optimization. In *AAAI*, 2017: 2043–2049.
- Huo Z, Gu B, Huang H. Decoupled asynchronous proximal stochastic gradient descent with variance reduction. arXiv preprint arXiv:1609.06804, 2016.
- Huo Z, Gu B, Liu J, *et al.* Accelerated method for stochastic composition optimization with nonsmooth regularization. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- IBM. IBM-FHE-ToolKit. <https://github.com/IBM/fhe-toolkit-android/>, 2019.
- Ishai Y, Paskin A. Evaluating branching programs on encrypted data. In *Theory of Cryptography Conference*, 2007: 575–594.
- Jacob L, Obozinski G, Vert J-P. Group lasso with overlap and graph lasso. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009: 433–440.
- Jagannathan G, Wright R N. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 2005: 593–599.
- Jaggi M. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *ICML* (1), 2013: 427–435.
- Jain P, Meka R, Dhillon I S. Guaranteed rank minimization via singular value projection. In *Advances in Neural Information Processing Systems*, 2010: 937–945.
- Jain P, Tewari A, Kar P. On iterative hard thresholding methods for high-dimensional m-estimation. In *Advances in Neural Information Processing Systems*, 2014: 685–693.
- Jalali A, Johnson C C, Ravikumar P K. On learning discrete graphical models using greedy methods. In *Advances in Neural Information Processing Systems*, 2011: 1935–1943.
- Jamieson K G, Nowak R, Recht B. Query complexity of derivative-free optimization. In *Advances in Neural Information Processing Systems*, 2012: 2672–2680.
- Jenatton R, Mairal J, Bach F R, *et al.* Proximal methods for sparse hierarchical dictionary learning. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010: 487–494.
- Jiang X, Kim M, Lauter K, *et al.* Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018: 1209–1222.

- Johnson R, Zhang T. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, 2013: 315–323.
- Juvekar C, Vaikuntanathan V, Chandrakasan A. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018: 1651–1669.
- Jäschke A, Armknecht F. Unsupervised machine learning on encrypted data. In *International Conference on Selected Areas in Cryptography*, 2018: 453–478.
- Kairouz P, McMahan H B, Avent B, *et al.* Advances and open problems in federated learning. arXiv preprint arXiv:1912.04977, 2019.
- Kaissis G A, Makowski M R, Rückert D, *et al.* Secure, privacy-preserving and federated machine learning in medical imaging. *Nature Machine Intelligence*, 2020, 2(6): 305–311.
- Kang J, Xiong Z, Jiang C, *et al.* Scalable and communication-efficient decentralized federated edge learning with multi-blockchain framework. In *International Conference on Blockchain and Trustworthy Systems*, 2020: 152–165.
- Kar P, Karnick H. Random feature maps for dot product kernels. In *Artificial Intelligence and Statistics*, 2012: 583–591.
- Karr A F, Lin X, Sanil A P, *et al.* Privacy-preserving analysis of vertically partitioned data using secure matrix products. *Journal of Official Statistics*, 2009, 25(1): 125.
- Kawachi A, Tanaka K, Xagawa K. Multi-bit cryptosystems based on lattice problems. In *International Workshop on Public Key Cryptography*, 2007: 315–329.
- Khaled A, Mishchenko K, Richtárik P. First analysis of local gd on heterogeneous data. arXiv preprint arXiv:1909.04715, 2019a.
- Khaled A, Richtárik P. Gradient descent with compressed iterates. arXiv preprint arXiv:1909.04716, 2019b.
- Khandani A E, Kim A J, Lo A W. Consumer credit-risk models via machine-learning algorithms. *Journal of Banking & Finance*, 2010, 34(11): 2767–2787.
- Kim M, Song Y, Wang S, *et al.* Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR Medical Informatics*, 2018, 6(2): e19.
- Kim H, Park J, Bennis M, *et al.* Blockchained on-device federated learning. *IEEE Communications Letters*, 2019, 24(6): 1279–1283.
- Kingma D P, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

- Kleinbaum D G, Dietz K, Gail M, *et al.* *Logistic Regression*, Springer, 2002.
- Kloft M, Brefeld U, Laskov P, *et al.* Efficient and accurate lp-norm multiple kernel learning. In *Advances in Neural Information Processing Systems*, 2009: 997–1005.
- Konda V R, Tsitsiklis J N. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, 2000: 1008–1014.
- Konečný J, Richtárik P. Semi-stochastic gradient descent methods. arXiv preprint arXiv:1312.1666, 2013, 2(2.1): 3.
- Konečný J, Richtárik P. Randomized distributed mean estimation: Accuracy vs. communication. *Frontiers in Applied Mathematics and Statistics*, 2018, 4: 62.
- Konečný J, McMahan H B, Ramage D, *et al.* Federated optimization: Distributed machine learning for on-device intelligence. arXiv preprint arXiv:1610.02527, 2016a.
- Konečný J, McMahan H B, Yu F X, *et al.* Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492, 2016b.
- Koskela A, Honkela A. Learning rate adaptation for federated and differentially private learning. arXiv preprint arXiv:1809.03832, 2018.
- Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012: 1097–1105.
- Kumar R, Khan A A, Kumar J, *et al.* Blockchain-federated-learning and deep learning models for covid-19 detection using ct imaging. *IEEE Sensors Journal*, 2021.
- Kungurtsev V, Egan M, Chatterjee B, *et al.* Asynchronous stochastic subgradient methods for general nonsmooth nonconvex optimization. arXiv preprint arXiv:1905.11845, 2019.
- Kurth T, Zhang J, Satish N, *et al.* Deep learning at 15pf: supervised and semi-supervised classification for scientific data. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017: 1–11.
- Lacoste-Julien S. Convergence rate of Frank-Wolfe for non-convex objectives. arXiv preprint arXiv:1607.00345, 2016.
- Lacoste-Julien S, Jaggi M. On the global linear convergence of Frank-Wolfe optimization variants. In *Advances in Neural Information Processing Systems*, 2015: 496–504.
- Lacoste-Julien S, Jaggi M, Schmidt M, *et al.* Block-coordinate Frank-Wolfe optimization for structural SVMs. In *ICML 2013 International Conference on Machine Learning*, 2013: 53–61.

- Laine K. Simple encrypted arithmetic library 2.3. 1. *Microsoft Research* <https://www.microsoft.com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1.pdf>, 2017.
- Lan G. The complexity of large-scale convex programming under a linear optimization oracle. arXiv preprint arXiv:1309.5550, 2013.
- Lan G, Lee S, Zhou Y. Communication-efficient algorithms for decentralized and stochastic optimization. *Mathematical Programming*, 2020, 180(1): 237–284.
- Larsen R M. Lanczos bidiagonalization with partial reorthogonalization. *DAIMI Report Series*, 1998, 27(537).
- Lawson C L, Hanson R J, Kincaid D R, *et al.* Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software (TOMS)*, 1979, 5(3): 308–323.
- Leblond R, Pedregosa F, Lacoste-Julien S. ASAGA: Asynchronous Parallel SAGA. In *Artificial Intelligence and Statistics*, 2017: 46–54.
- Leblond R, Pedregosa F, Lacoste-Julien S. ASAGA: Asynchronous parallel saga. arXiv preprint arXiv:1606.04809, 2016.
- Leblond R, Pedregosa F, Lacoste-Julien S. Asaga: Asynchronous parallel Saga. In *20th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017*, 2017.
- Lee S-I, Lee H, Abbeel P, *et al.* Efficient l 1 regularized logistic regression. In *Proceedings of the National Conference on Artificial Intelligence*, Vol. 21, 2006: 401.
- Lee J, Clifton C. How much is enough? choosing ε for differential privacy. In *International Conference on Information Security*, 2011: 325–340.
- Lei L, Ju C, Chen J, *et al.* Non-convex finite-sum optimization via SCSG methods. In *Advances in Neural Information Processing Systems*, 2017: 2348–2358.
- Leroy D, Coucke A, Lavril T, *et al.* Federated learning for keyword spotting. In *ICASSP 2019–2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019: 6341–6345.
- Levine S, Koltun V. Guided policy search. In *International Conference on Machine Learning*, 2013: 1–9.
- Levy-dit Vehel F, Perret L. A Polly Cracker system based on satisfiability. In *Coding, Cryptography and Combinatorics*, Springer, 2004: 177–192.
- Levy-dit Vehel F, Marinari M G, Perret L, *et al.* A survey on polly cracker systems. In *Gröbner Bases, Coding, and Cryptography*, Springer, 2009: 285–305.
- Li M, Andersen D G, Smola A. Distributed delayed proximal gradient methods. In *NIPS Workshop on Optimization for Machine Learning*, Vol. 3, 2013: 3.

- Li M, Zhang T, Chen Y, *et al.* Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014: 661–670.
- Li H, Lin Z. Accelerated proximal gradient methods for nonconvex programming. In *Advances in Neural Information Processing Systems*, 2015: 379–387.
- Li X, Arora R, Liu H, *et al.* Nonconvex sparse learning via stochastic optimization with progressive variance reduction. arXiv preprint arXiv:1605.02711, 2016.
- Li Y, Jiang X, Wang S, *et al.* Vertical grid logistic regression (vertigo). *Journal of the American Medical Informatics Association*, 2016, 23(3): 570–579.
- Li G, Hari S K S, Sullivan M, *et al.* Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017: 1–12.
- Li T, Sahu A K, Zaheer M, *et al.* Federated optimization in heterogeneous networks. arXiv preprint arXiv:1812.06127, 2018.
- Li X, Yang W, Wang S, *et al.* Communication efficient decentralized training with multiple local updates. arXiv preprint arXiv:1910.09126, 2019, 5.
- Li X, Huang K, Yang W, *et al.* On the convergence of fedavg on non-iid data. arXiv preprint arXiv:1907.02189, 2019.
- Li J, Khodak M, Caldas S, *et al.* Differentially private meta-learning. arXiv preprint arXiv:1909.05830, 2019.
- Li Y, Chen C, Liu N, *et al.* A blockchain-based decentralized federated learning framework with committee consensus. *IEEE Network*, 2020, 35(1): 234–241.
- Li T, Sahu A K, Talwalkar A, *et al.* Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 2020, 37(3): 50–60.
- Li L, Fan Y, Tse M, *et al.* A review of applications in federated learning. *Computers & Industrial Engineering*, 2020: 106854.
- Li M, Deng C, Li T, *et al.* Towards transferable targeted attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020: 641–649.
- Li X, Zhao T, Arora R, *et al.* An improved convergence analysis of cyclic block coordinate descent-type methods for strongly convex minimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, 2016: 491–499.

- Li X, Zhao T, Arora R, *et al.* On faster convergence of cyclic block coordinate descent-type methods for strongly convex minimization. *Journal of Machine Learning Research*, 2017, 18: 184:1–184:24.
- Li Y, Yang M, Zhang Z. A survey of multi-view representation learning. *IEEE Transactions on Knowledge and Data Engineering*, 2019, 31(10): 1863–1883. <http://dx.doi.org/10.1109/TKDE.2018.2872063>.
- Lian X, Huang Y, Li Y, *et al.* Asynchronous parallel stochastic gradient for nonconvex optimization. arXiv preprint arXiv:1506.08272, 2015.
- Lian X, Zhang H, Hsieh C-J, *et al.* A comprehensive linear speedup analysis for asynchronous stochastic parallel optimization from Zeroth-order to first-order. arXiv preprint arXiv:1606.00498, 2016.
- Lian X, Zhang C, Zhang H, *et al.* Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. arXiv preprint arXiv:1705.09056, 2017.
- Lian X, Zhang W, Zhang C, *et al.* Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, 2018: 3043–3052.
- Liang Y, Liu C, Luan X-Z, *et al.* Sparse logistic regression with a $L_{1/2}$ penalty for gene selection in cancer classification. *BMC Bioinformatics*, 2013, 14(1): 198.
- Lillicrap T P, Hunt J J, Pritzel A, *et al.* Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.
- Lin Q, Lu Z, Xiao L. An accelerated proximal coordinate gradient method. In *Advances in Neural Information Processing Systems*, 2014: 3059–3067.
- Lin Y, Han S, Mao H, *et al.* Deep gradient compression: Reducing the communication bandwidth for distributed training. arXiv preprint arXiv:1712.01887, 2017.
- Liu Y, Ma Z, Liu X, *et al.* Boosting privately: Privacy-preserving federated extreme boosting for mobile crowdsensing. *CoRR*, 2019, abs/1907.10218.
- Liu J, Wright S J, Ré C, *et al.* An asynchronous parallel stochastic coordinate descent algorithm. *Journal of Machine Learning Research*, 2015, 16(285–322): 1–5.
- Liu J, Wright S J. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. *SIAM Journal on Optimization*, 2015, 25(1): 351–376.
- Liu F, Ng W K, Zhang W. Encrypted gradient descent protocol for outsourced data mining. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, 2015: 339–346.

- Liu J, Juuti M, Lu Y, *et al.* Oblivious neural network predictions via mini-ONN transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017: 619–631.
- Liu Y, Kang Y, Zhang X, *et al.* A communication efficient vertical federated learning framework. arXiv preprint arXiv:1912.11187, 2019.
- Liu Y, Liu Y, Liu Z, *et al.* Federated forest. arXiv preprint arXiv:1905.10053, 2019.
- Liu D, Dligach D, Miller T. Two-stage federated phenotyping and patient representation learning. arXiv preprint arXiv:1908.05596, 2019.
- Liu Y, Zhang X, Wang L. Asymmetrically vertical federated learning. arXiv preprint arXiv:2004.07427, 2020.
- Liu L, Zhang J, Song S, *et al.* Client-edge-cloud hierarchical federated learning. In *ICC 2020–2020 IEEE International Conference on Communications (ICC)*, 2020: 1–6.
- Liu D, Miller T. Federated pretraining and fine tuning of BERT using clinical notes from multiple silos. arXiv preprint arXiv:2002.08562, 2020.
- Liu Y, Huang A, Luo Y, *et al.* Fedvision: An online visual object detection platform powered by federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34, 2020: 13172–13179.
- Liu D, Chen X, Zhou Z, *et al.* HierTrain: Fast hierarchical edge AI learning with hybrid parallelism in mobile-edge-cloud computing. *IEEE Open Journal of the Communications Society*, 2020, 1: 634–645.
- Liu Y, Chen C, Zheng L, *et al.* Privacy preserving pca for multiparty modeling. arXiv preprint arXiv:2002.02091, 2020.
- Liu Y, Peng J, Kang J, *et al.* A secure federated learning framework for 5G networks. *IEEE Wireless Communications*, 2020, 27(4): 24–31.
- Liu J, Huang J, Zhou Y, *et al.* From distributed machine learning to federated learning: A survey. arXiv preprint arXiv:2104.14362, 2021.
- Liu J, Meng X. A review of privacy protection research in machine learning. *Journal of Computer Research and Development*, 2020, 57(2): 346.
- Lu Z, Xiao L. On the complexity analysis of randomized block-coordinate descent methods. *Mathematical Programming*, 2015, 152(1–2): 615–642.
- Lu R, Jin X, Zhang S, *et al.* A study on big knowledge and its engineering issues. *IEEE Transactions on Knowledge and Data Engineering*, 2018, 31(9): 1630–1644.
- López-Alt A, Tromer E, Vaikuntanathan V. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, 2012: 1219–1234.
- Maclaurin D, Duvenaud D, Adams R. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, 2015: 2113–2122.

- Mairal J, Ponce J, Sapiro G, *et al.* Supervised dictionary learning. In *Advances in Neural Information Processing Systems*, 2009: 1033–1040.
- Malkin M, Wu T, Boneh D. Experimenting with Shared Generation of RSA keys. In *In Proceedings of the Internet Society's 1999 Symposium on Network and Distributed System Security (SNDSS)*, 1999: 43–56.
- Mandal K, Gong G. PrivFL: Practical privacy-preserving federated regressions on high-dimensional data over mobile networks. In *CCSW'19: Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*. New York, NY, USA: Association for Computing Machinery, 2019: 57–68. <https://doi.org/10.1145/3338466.3358926>.
- Mania H, Pan X, Papailiopoulos D, *et al.* Perturbed iterate analysis for asynchronous stochastic optimization. arXiv preprint arXiv:1507.06970, 2015.
- Mania H, Pan X, Papailiopoulos D, *et al.* Perturbed iterate analysis for asynchronous stochastic optimization. *SIAM Journal on Optimization*, 2017, 27(4): 2202–2229.
- Marquardt D W, Snee R D. Ridge regression in practice. *The American Statistician*, 1975, 29(1): 3–20.
- McKeen F, Alexandrovich I, Berenzon A, *et al.* Innovative instructions and software model for isolated execution. *Hasp@ isca*, 2013, 10(1).
- McMahan H B, Moore E, Ramage D, *et al.* Communication efficient learning of deep networks from decentralized data. arXiv preprint arXiv:1602.05629, 2016a.
- McMahan H B, Moore E, Ramage D, *et al.* Federated learning of deep networks using model averaging. arXiv preprint arXiv:1602.05629v3, 2016b.
- McMahan B, Moore E, Ramage D, *et al.* Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, 2017a: 1273–1282.
- McMahan H B, Ramage D, Talwar K, *et al.* Learning differentially private recurrent language models. arXiv preprint arXiv:1710.06963, 2017b.
- Mcsherry F D. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 2009: 19–30.
- Mei S, Bai Y, Montanari A, *et al.* The landscape of empirical risk for nonconvex losses. *The Annals of Statistics*, 2018, 46(6A): 2747–2774.
- Mendes R, Vilela J P. Privacy-preserving data mining: methods, metrics, and applications. *IEEE Access*, 2017, 5: 10562–10582.
- Meng Q, Chen W, Yu J, *et al.* Asynchronous stochastic proximal optimization algorithms with variance reduction. arXiv preprint arXiv:1609.08435, 2016.

- Meng Q, Chen W, Yu J, *et al.* Asynchronous stochastic proximal optimization algorithms with variance reduction. In *AAAI*, 2017: 2329–2335.
- Micciancio D, Regev O. Lattice-based cryptography. In *Post-quantum Cryptography*, Springer, 2009: 147–191.
- Michel F. How many photos are uploaded to flickr every day and month?. Papadimitriou, S., & Sun, J. (2008). Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining, 2012.
- Minar J. libfhe. <https://github.com/rdancer/fhe/tree/master/libfhe>, 2010.
- Mishra P K, Rathee D, Duong D H, *et al.* Fast secure matrix multiplications over ring-based homomorphic encryption. *IACR Cryptology ePrint Archive*, 2018, 2018: 663.
- Mnih V, Kavukcuoglu K, Silver D, *et al.* Human-level control through deep reinforcement learning. *Nature*, 2015, 518(7540): 529–533.
- Mnih V, Badia A P, Mirza M, *et al.* Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016: 1928–1937.
- Moharrer A, Ioannidis S. Distributing Frank-Wolfe via map-reduce. In *2017 IEEE International Conference on Data Mining, ICDM 2017*, New Orleans, LA, USA, November 18–21, 2017, 2017: 317–326.
- Mohassel P, Rindal P. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018: 35–52.
- Mohassel P, Zhang Y. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, 2017: 19–38.
- Mohri M, Sivek G, Suresh A T. Agnostic federated learning. In *International Conference on Machine Learning*, 2019: 4615–4625.
- Mokhtari A, Ozdaglar A, Jadbabaie A. Efficient nonconvex empirical risk minimization via adaptive sample size methods. In *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019: 2485–2494.
- Montgomery P L. A survey of modern integer factorization algorithms. *CWI Quarterly*, 1994, 7(4): 337–366.
- Moore II D H. Classification and regression trees, by Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. Brooks/Cole Publishing, Monterey, 1984, 358 pages, \$27.95. *Cytometry*, 1987, 8(5): 534–535. <https://onlinelibrary.wiley.com/doi/abs/10.1002/cyto.990080516>.
- Mordatch I, Lowrey K, Andrew G, *et al.* Interactive control of diverse complex characters with neural networks. *Advances in Neural Information Processing Systems*, 2015, 28: 3132–3140.

- Naccache D, Stern J. A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM conference on Computer and Communications Security*, 1998: 59–66.
- Nagabandi A, Kahn G, Fearing R S, *et al.* Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018: 7559–7566.
- Nagar A. Privacy-preserving blockchain based federated learning with differential data sharing. arXiv preprint arXiv:1912.04859, 2019.
- Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 2008: 21260.
- Narayanan A, Shmatikov V. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, 2008: 111–125.
- Natarajan B K. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 1995, 24(2): 227–234.
- Necoara I, Clipici D. Efficient parallel coordinate descent algorithm for convex optimization problems with separable constraints: Application to distributed MPC. *Journal of Process Control*, 2013, 23(3): 243–253.
- Nedic A, Ozdaglar A. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 2009, 54(1): 48–61.
- Nemirovski A, Juditsky A, Lan G, *et al.* Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 2009, 19(4): 1574–1609.
- Nesterov Y, Spokoiny V. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 2011: 1–40.
- Nesterov Y. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 2012, 22(2): 341–362.
- Nesterov Y, Spokoiny V. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 2017, 17(2): 527–566.
- Nguyen N, Needell D, Woolf T. Linear convergence of stochastic iterative greedy algorithms with sparse constraints. arXiv preprint arXiv:1407.0088, 2014.
- Nguyen L M, Liu J, Scheinberg K, *et al.* SARAH: A novel method for machine learning problems using stochastic recursive gradient. In *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70, 2017a: 2613–2621.
- Nguyen L M, Liu J, Scheinberg K, *et al.* Stochastic recursive gradient algorithm for nonconvex optimization. arXiv preprint arXiv:1705.07261, 2017b.

- Nguyen H T, Sehwag V, Hosseinalipour S, *et al.* Fast-convergent federated learning. arXiv preprint arXiv:2007.13137, 2020.
- Nikolaenko V, Weinsberg U, Ioannidis S, *et al.* Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*, 2013: 334–348.
- Nishide T, Sakurai K. Distributed paillier cryptosystem without trusted dealer. In *International Workshop on Information Security Applications*, 2010: 44–60.
- Nitanda A. Stochastic proximal gradient descent with acceleration techniques. In *Advances in Neural Information Processing Systems*, 2014: 1574–1582.
- Niu F, Recht B, Ré C, *et al.* Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. arXiv preprint arXiv:1106.5730, 2011.
- Nocedal J, Wright S J. *Numerical Optimization*. (2nd edn). New York, NY, USA: Springer, 2006.
- Nock R, Hardy S, Henecka W, *et al.* Entity Resolution and Federated Learning get a Federated Resolution. arXiv preprint arXiv:1803.04035, 2018.
- Norris J R, Norris J R. *Markov Chains*. Cambridge University Press, 1998.
- O’Donoghue B, Munos R, Kavukcuoglu K, *et al.* Combining policy gradient and q-learning. arXiv preprint arXiv:1611.01626, 2016.
- Ogura N, Yamamoto G, Kobayashi T, *et al.* An improvement of key generation algorithm for Gentry’s homomorphic encryption scheme. In *International Workshop on Security*, 2010: 70–83.
- Oh J, Singh S, Lee H. Value prediction network. arXiv preprint arXiv:1707.03497, 2017.
- Ohrimenko O, Schuster F, Fournet C, *et al.* Oblivious multi-party machine learning on trusted processors. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016: 619–636.
- Okamoto T, Uchiyama S. A new public-key cryptosystem as secure as factoring. In *International Conference on the Theory and Applications of Cryptographic Techniques*, 1998: 308–318.
- Oliveira F P, Tavares J M R. Medical image registration: A review. *Computer Methods in Biomechanics and Biomedical Engineering*, 2014, 17(2): 73–93.
- Ouyang H, He N, Tran L, *et al.* Stochastic alternating direction method of multipliers. In *International Conference on Machine Learning*, 2013: 80–88.
- Ozfatura E, Ozfatura K, Gunduz D. Time-correlated sparsification for communication-efficient federated learning. arXiv preprint arXiv:2101.08837, 2021.
- Paillier P. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT’99: Proceedings of the 17th International*

- Conference on Theory and Application of Cryptographic Techniques*. Berlin, Heidelberg: Springer-Verlag, 1999: 223–238.
- Paine T, Jin H, Yang J, *et al*. Gpu asynchronous stochastic gradient descent to speed up neural network training. arXiv preprint arXiv:1312.6186, 2013.
- Passerat-Palmbach J, Farnan T, Miller R, *et al*. A blockchain-orchestrated federated learning architecture for healthcare consortia. arXiv preprint arXiv:1910.12603, 2019.
- Pedregosa F. Hyperparameter optimization with approximate gradient. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, Vol. 48, 2016: 737–746.
- Pedregosa F, Leblond R, Lacoste-Julien S. Breaking the nonsmooth barrier: A scalable parallel method for composite optimization. In *Advances in Neural Information Processing Systems*, 2017: 56–65.
- Peteiro-Barral D, Guijarro-Berdiñas B. A survey of methods for distributed machine learning. *Progress in Artificial Intelligence*, 2013, 2(1): 1–11.
- Pham N H, Nguyen L M, Phan D T, *et al*. ProxSARAH: An efficient algorithmic framework for stochastic composite nonconvex optimization[J]. arXiv preprint arXiv:1902.05679, 2019.
- Phuong T T, *et al*. Privacy-preserving deep learning via weight transmission. *IEEE Transactions on Information Forensics and Security*, 2019, 14(11): 3003–3015.
- Pouyanfar S, Sadiq S, Yan Y, *et al*. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)*, 2018, 51(5): 1–36.
- Que J, Jiang X, Ohno-machado L. A collaborative framework for distributed privacy-preserving support vector machine learning. In *AMIA Annual Symposium Proceedings*, Vol. 2012, 2012: 1350.
- Rahimi A, Recht B. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2007: 1177–1184.
- Rahimi A, Recht B. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2008: 1177–1184.
- Rahimi A, Recht B. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems*, 2009: 1313–1320.
- Ramanan P, Nakayama K. Baffle: Blockchain based aggregator free federated learning. In *2020 IEEE International Conference on Blockchain (Blockchain)*, 2020: 72–81.
- Ramaswamy S, Mathews R, Rao K, *et al*. Federated learning for emoji prediction in a mobile keyboard. arXiv preprint arXiv:1906.04329, 2019.
- Razaviyayn M, Hong M, Luo Z-Q, *et al*. Parallel successive convex approximation for nonsmooth nonconvex optimization. In *NIPS*, 2014.

- Razaviyayn M, Hong M, Luo Z-Q, *et al.* Parallel successive convex approximation for nonsmooth nonconvex optimization. In *Advances in Neural Information Processing Systems*, 2014: 1440–1448.
- Recht B, Re C, Wright S, *et al.* Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, 2011: 693–701.
- Reddi S J, Hefny A, Sra S, *et al.* On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems*, 2015: 2647–2655.
- Reddi S J, Sra S, Póczos B, *et al.* Fast stochastic methods for nonsmooth nonconvex optimization. arXiv preprint arXiv:1605.06900, 2016.
- Reddi S J, Sra S, Póczos B, *et al.* Stochastic frank-wolfe methods for nonconvex optimization. In *Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference On*, 2016: 1244–1251.
- Reddi S J, Hefny A, Sra S, *et al.* Stochastic variance reduction for nonconvex optimization. In *International Conference on Machine Learning*, 2016: 314–323.
- Regev O. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 2009, 56(6): 1–40.
- Regulation P. Regulation (EU) 2016/679 of the European Parliament and of the Council. *Regulation (EU)*, 2016, 679: 2016.
- Riazi M S, Weinert C, Tkachenko O, *et al.* Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018: 707–721.
- Richtárik P, Takáč M. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 2016, 156(1–2): 433–484.
- Riedmiller M. Neural fitted Q iteration—first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, 2005: 317–328.
- Rieke N, Hancox J, Li W, *et al.* The future of digital health with federated learning. *NPJ Digital Medicine*, 2020, 3(1): 1–7.
- Rivest R L, Adleman L, Dertouzos M L, *et al.* On data banks and privacy homomorphisms. *Foundations of Secure Computation*, 1978, 4(11): 169–180.
- Rivest R L, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 1978, 21(2): 120–126.
- Rizk E, Vlaski S, Sayed A H. Federated learning under importance sampling. arXiv preprint arXiv:2012.07383, 2020.

- Roth V, Fischer B. The group-lasso for generalized linear models: uniqueness of solutions and efficient algorithms. In *Proceedings of the 25th International Conference on Machine Learning*, 2008: 848–855.
- Rouhani B D, Riazi M S, Koushanfar F. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, 2018: 1–6.
- Roux N L, Schmidt M, Bach F R. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in NIPS*, 2012: 2663–2671.
- Rudin W. *Fourier Analysis on Groups*, Vol. 121967. Wiley Online Library, 1962.
- Rudin W, *et al.* *Principles of Mathematical Analysis*, Vol. 3. McGraw-Hill, New York, 1964.
- Ruszczynski. A P. *Nonlinear Optimization: Nonlinear Optimization*, Princeton University Press, 2006.
- Ryffel T, Trask A, Dahl M, *et al.* A generic framework for privacy preserving deep learning. arXiv preprint arXiv:1811.04017, 2018.
- Ryffel T, Tholoniati P, Pointcheval D, *et al.* Ariann: Low-interaction privacy-preserving deep learning via function secret sharing. arXiv preprint arXiv:2006.04593, 2020.
- Régim J-C. Generalized arc consistency for global cardinality constraint. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Vol 1, 1996: 209–215.
- Sahu A K, Li T, Sanjabi M, *et al.* On the convergence of federated optimization in heterogeneous networks. arXiv preprint arXiv:1812.06127, 2018a, 3.
- Sahu A K, Li T, Sanjabi M, *et al.* Federated optimization for heterogeneous networks. arXiv preprint arXiv:1812.06127, 2018b, 1(2): 3.
- Salimans T, Ho J, Chen X, *et al.* Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864, 2017.
- Sanderson C, Curtin R. Armadillo: A template-based C++ library for linear algebra. *Journal of Open Source Software*, 2016, 1(2): 26.
- Sanil A P, Karr A F, Lin X, *et al.* Privacy preserving regression modelling via distributed computation. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004: 677–682.
- Sarathy R, Muralidhar K. Evaluating Laplace noise addition to satisfy differential privacy for numeric data. *Transaction on Data Privacy*, 2011, 4(1): 1–17.

- Sattler F, Wiedemann S, Müller K-R, *et al.* Robust and communication-efficient federated learning from non-iid data. *IEEE Transactions on Neural Networks and Learning Systems*, 2019, 31(9): 3400–3413.
- Schmidt M, Roux N L, Bach F. Convergence rates of inexact proximal-gradient methods for convex optimization. arXiv preprint arXiv:1109.2415, 2011.
- Schmidt M, Roux N L, Bach F. Minimizing finite sums with the stochastic average gradient. arXiv preprint arXiv:1309.2388, 2013.
- Schmidt M, Roux N L, Bach F. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 2017, 162(1–2): 83–112.
- Scholl P, Smart N P. Improved key generation for Gentry’s fully homomorphic encryption scheme. In *IMA International Conference on Cryptography and Coding*, 2011: 10–22.
- Schulman J, Wolski F, Dhariwal P, *et al.* Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.
- Shafi G, Silvio M. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *14th ACM STOC*, Vol. 365, 1982.
- Shalev-Shwartz S, Zhang T. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. In *ICML*, 2014: 64–72.
- Shalev-Shwartz S. SDCA without duality, regularization, and individual convexity. In *Proceedings of the 33rd International Conference on Machine Learning*, ICML 2016, New York City, NY, USA, June 19–24, 2016, 2016: 747–754.
- Shamir A. How to Share a Secret. *Communication of the ACM*, 1979, 22(11): 612–613. <https://doi.org/10.1145/359168.359176>.
- Shayan M, Fung C, Yoon C J, *et al.* Biscotti: A ledger for private and secure peer-to-peer machine learning. arXiv preprint arXiv:1811.09904, 2018.
- Shen X, Alam M, Fikse F, *et al.* A novel generalized ridge regression method for quantitative genetics. *Genetics*, 2013, 193(4): 1255–1268.
- Shen J, Li P. A Tight bound of hard thresholding. arXiv preprint arXiv:1605.01656, 2016.
- Shi W, Gu B, Li X, *et al.* Quadruply stochastic gradients for large scale nonlinear semi-supervised AUC optimization. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 2019: 3418–3424.
- Shokri R, Shmatikov V. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015: 1310–1321.
- Shokri R, Stronati M, Song C, *et al.* Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, 2017: 3–18.

- Shu J, Jia X, Yang K, *et al.* Privacy-preserving task recommendation services for crowdsourcing. *IEEE Transactions on Services Computing*, 2018.
- Silver D, Huang A, Maddison C J, *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature*, 2016, 529(7587): 484–489.
- Silver D, Hasselt H, Hessel M, *et al.* The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, 2017: 3191–3199.
- Sirb B, Ye X. Consensus optimization with delayed and stochastic gradients on decentralized networks. In *2016 IEEE International Conference on Big Data (Big Data)*, 2016: 76–85.
- Skillicorn D B, McConnell S M. Distributed prediction from vertically partitioned data. *Journal of Parallel & Distributed Computing*, 2008, 68(1): 16–36.
- Smart N P, Vercauteren F. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *International Workshop on Public Key Cryptography*, 2010: 420–443.
- Smart N P, Vercauteren F. Fully homomorphic SIMD operations. *Designs, Codes and Cryptography*, 2014, 71(1): 57–81.
- Smith V, Chiang C-K, Sanjabi M, *et al.* Federated multi-task learning. arXiv preprint arXiv:1705.10467, 2017.
- Smola A J, Schölkopf B. *Learning with Kernels*, Vol. 4. Citeseer, 1998.
- Snoek J, Larochelle H, Adams R P. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012: 2951–2959.
- So J, Guler B, Avestimehr A S, *et al.* CodedPrivateML: A fast and privacy-preserving framework for distributed machine learning. *CoRR*, 2019, abs/1902.00641.
- Soria-Comas J, Domingo-Ferrer J. Big data privacy: challenges to privacy principles and models. *Data Science and Engineering*, 2016, 1(1): 21–28.
- Sprague M R, Jalalirad A, Scavuzzo M, *et al.* Asynchronous federated learning for geospatial applications. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2018: 21–28.
- Steinwandt R. A ciphertext-only attack on Polly Two. *Applicable Algebra in Engineering, Communication and Computing*, 2010, 21(2): 85–92.
- Stich S U. Local SGD converges fast and communicates little. arXiv preprint arXiv:1805.09767, 2018.
- Sun Z, Strang K D, Pambel F. Privacy and security in the big data paradigm. *Journal of Computer Information Systems*, 2018.

- Suresh A T, Felix X Y, Kumar S, *et al.* Distributed mean estimation with limited communication. In *International Conference on Machine Learning*, 2017: 3329–3337.
- Sutton R S, Barto A G. *Reinforcement Learning: An Introduction*, MIT Press, 2018.
- Sutton R S, McAllester D A, Singh S P, *et al.* Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, Vol. 99, 1999: 1057–1063.
- Suykens J A, Vandewalle J. Least squares support vector machine classifiers. *Neural Processing Letters*, 1999, 9(3): 293–300.
- Swan M. *Blockchain: Blueprint for a New Economy*. O'Reilly Media, Inc., 2015.
- Takác M. Randomized coordinate descent methods for big data optimization, 2014.
- Tamar A, Wu Y, Thomas G, *et al.* Value iteration networks. arXiv preprint arXiv:1602.02867, 2016.
- Tang H, Lian X, Qiu S, *et al.* Deepsqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression, arXiv preprint arXiv:1907.07346, 2019.
- Tappenden R, Richtárik P, Gondzio J. Inexact coordinate descent: Complexity and preconditioning. *Journal of Optimization Theory and Applications*, 2016: 144–176.
- Taskar B, Chatalbashev V, Koller D, *et al.* Learning structured prediction models: A large margin approach. In *Proceedings of the 22nd International Conference on Machine Learning*, 2005: 896–903.
- Taylor L, Floridi L, Van der Sloot B. *Group Privacy: New Challenges of Data Technologies*, Vol. 126. Springer, 2016.
- Tibshirani R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1996, 58(1): 267–288.
- Tibshirani R, Saunders M, Rosset S, *et al.* Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2005, 67(1): 91–108.
- Tourky D, Elkawagy M, Keshk A. Homomorphic encryption the “holy grail” of cryptography. In *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, 2016: 196–201.
- Tropp J A, Gilbert A C. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 2007, 53(12): 4655–4666.
- Truex S, Baracaldo N, Anwar A, *et al.* A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019: 1–11.

- Truex S, Liu L, Chow K-H, *et al.* LDP-Fed: Federated learning with local differential privacy. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, 2020: 61–66.
- Tseng P. On accelerated proximal gradient methods for convex-concave optimization. *SIAM Journal of Optimization*, 2008.
- Tseng P, Yun S. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 2009, 117(1–2): 387–423.
- Vaidya J, Clifton C. Privacy preserving association rule mining in vertically partitioned data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge discovery and Data Mining*, 2002: 639–644.
- Vaidya J, Clifton C. Privacy-preserving k-means clustering over vertically partitioned data. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003: 206–215.
- Vaidya J, Clifton C. Privacy preserving naive bayes classifier for vertically partitioned data. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, 2004: 522–526.
- Vaidya J, Yu H, Jiang X. Privacy-preserving SVM classification. *Knowledge and Information Systems*, 2008, 14(2): 161–178.
- Van dijk M, Gentry C, Halevi S, *et al.* Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2010: 24–43.
- Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30, 2016.
- Van Ly L. Polly Two: A new algebraic polynomial-based public-key scheme. *Applicable Algebra in Engineering, Communication and Computing*, 2006, 17: 267–283.
- Vapnik V. *The Nature of Statistical Learning Theory*. Springer Science & Business Media, 2013.
- Vapnik V N, Vapnik V. *Statistical Learning Theory*, Vol 1. Wiley, New York, 1998.
- Verbraeken J, Wolting M, Katzy J, *et al.* A survey on distributed machine learning. *ACM Computing Surveys (CSUR)*, 2020, 53(2): 1–33.
- Veugen T, Attema T, Spini G. An implementation of the Paillier crypto system with threshold decryption without a trusted dealer, 2019, Cryptology ePrint Archive.
- Villa S, Salzo S, Baldassarre L, *et al.* Accelerated and inexact forward-backward algorithms. *SIAM Journal on Optimization*, 2013, 23(3): 1607–1633.

- Wagh S, Gupta D, Chandran N. SecureNN: Efficient and private neural network training. *IACR Cryptology ePrint Archives*, 2018, 2018: 442.
- Wagh S, Tople S, Benhamouda F, *et al.* Falcon: Honest-majority maliciously secure framework for private deep learning. arXiv preprint arXiv:2004.02229, 2020.
- Wahlström N, Schön T B, Deisenroth M P. From pixels to torques: Policy learning with deep dynamical models. arXiv preprint arXiv:1502.02251, 2015.
- Wainwright M J, Jordan M I. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 2008, 1(1–2): 1–305.
- Wan L, Ng W K, Han S, *et al.* Privacy-preservation for gradient descent methods. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007: 775–783.
- Wang G, Hoiem D, Forsyth D. Learning image similarity from flickr groups using fast kernel machines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012, 34(11): 2177–2188.
- Wang F, Wang K, Li B. LWE-based FHE with better parameters. In *International Workshop on Security*, 2015: 175–192.
- Wang Y-X, Sadhanala V, Dai W, *et al.* Parallel and distributed block-coordinate Frank-Wolfe algorithms. In *International Conference on Machine Learning*, 2016: 1548–1557.
- Wang X, Ma S, Goldfarb D, *et al.* Stochastic quasi-Newton methods for non-convex stochastic optimization. *SIAM Journal on Optimization*, 2017, 27(2): 927–956.
- Wang J, Joshi G. Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms. arXiv preprint arXiv:1808.07576, 2018.
- Wang Z, Ji K, Zhou Y, *et al.* SpiderBoost: A class of faster variance-reduced algorithms for nonconvex optimization. arXiv preprint arXiv:1810.10690, 2018.
- Wang Z, Song M, Zhang Z, *et al.* Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, 2019: 2512–2520.
- Wang J, Sahu A K, Yang Z, *et al.* MATCHA: Speeding up decentralized SGD via matching decomposition sampling. In *2019 Sixth Indian Control Conference (ICC)*, 2019: 299–300.
- Wang T, Zhao J, Yu H, *et al.* Privacy-preserving crowd-guided AI decision-making in ethical dilemmas. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019: 1311–1320.

- Wang Z, Ji K, Zhou Y, *et al.* SpiderBoost and momentum: Faster variance reduction algorithms. In *Advances in NIPS*, 2019: 2403–2413.
- Wang H, Yurochkin M, Sun Y, *et al.* Federated learning with matched averaging. arXiv preprint arXiv:2002.06440, 2020.
- Wang J, Wang S, Chen R-R, *et al.* Local averaging helps: Hierarchical federated learning and convergence analysis. arXiv preprint arXiv:2010.12998, 2020.
- Wang Y, Cheng Y, Jia Z, *et al.* A secure multi-party blockchain auditable signature scheme. *Journal of Computer Applications*, 2020, 40(9): 2639–2645.
- Watkins C J C H. Learning from delayed rewards, 1989.
- Watkins C J, Dayan P. Q-learning. *Machine Learning*, 1992, 8(3–4): 279–292.
- Wei K, Li J, Ding M, *et al.* Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 2020, 15: 3454–3469.
- Wei K, Yang M, Wang H, *et al.* Adversarial fine-grained composition learning for unseen attribute-object recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, 2019a: 3741–3749.
- Wei Y, Wang Z, Zhou S, *et al.* Federated visualization: A new privacy-preserving visualization model. *Chinese Journal of Intelligent Science and Technology*, 2019b, 1(4): 415–420.
- Wendland H. *Scattered Data Approximation*, Vol. 17. Cambridge University Press, 2004.
- Weng J, Weng J, Zhang J, *et al.* Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive. *IEEE Transactions on Dependable and Secure Computing*, 2019.
- Wu S, Teruya T, Kawamoto J, *et al.* Privacy-preservation for stochastic gradient descent application to secure logistic regression. In *The 27th Annual Conference of the Japanese Society for Artificial Intelligence*, Vol. 27, 2013: 1–4.
- Wu X, Zhu X, Wu G-Q, *et al.* Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering*, 2014, 26(1): 97–107.
- Wu L, Sun P, Fu Y, *et al.* A neural influence diffusion model for social recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019: 235–244.
- Wu Q, He K, Chen X. Personalized federated learning for intelligent IoT applications: A cloud-edge based framework. *IEEE Open Journal of the Computer Society*, 2020, 1: 35–44.

- Xiao L, Zhang T. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 2014, 24(4): 2057–2075.
- Xie B, Liang Y, Song L. Scale up nonlinear component analysis with doubly stochastic gradients. In *Advances in Neural Information Processing Systems*, 2015: 2341–2349.
- Xie C, Koyejo S, Gupta I. Asynchronous federated optimization. arXiv preprint arXiv:1903.03934, 2019.
- Xing E P, Ho Q, Xie P, *et al.* Strategies and principles of distributed machine learning on big data. *Engineering*, 2016, 2(2): 179–195.
- Xu C, Tao D, Xu C. A survey on multi-view learning. *CoRR*, 2013, abs/1304.5634. <http://arxiv.org/abs/1304.5634>.
- Xu Y, Zhu S, Yang S, *et al.* Learning with non-convex truncated losses by SGD. arXiv preprint arXiv:1805.07880, 2018.
- Xu R, Baracaldo N, Zhou Y, *et al.* Hybridalpha: An efficient approach for privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019: 13–23.
- Yan Y, Chen M, Shyu M-L, *et al.* Deep learning for imbalanced multimedia data classification. In *2015 IEEE International Symposium on Multimedia (ISM)*, 2015: 483–488.
- Yan F, Ruwase O, He Y, *et al.* SERF: Efficient scheduling for fast deep neural network serving via judicious parallelism. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016: 300–311.
- Yan Y, Chen M, Sadiq S, *et al.* Efficient imbalanced multimedia concept retrieval by deep learning on spark clusters. *International Journal of Multimedia Data Engineering and Management (IJMDEM)*, 2017, 8(1): 1–20.
- Yang H-M, Xia Q, Wang X-F, *et al.* A new somewhat homomorphic encryption scheme over integers. In *2012 International Conference on Computer Distributed Control and Intelligent Environmental Monitoring*, 2012: 61–64.
- Yang J, Sindhwani V, Fan Q, *et al.* Random laplace feature maps for semi-group kernels on histograms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014: 971–978.
- Yang Y, Wu L, Yin G, *et al.* A survey on security and privacy issues in Internet-of-Things. *IEEE Internet of Things Journal*, 2017, 4(5): 1250–1258.
- Yang T, Andrew G, Eichner H, *et al.* Applied federated learning: Improving google keyboard query suggestions. arXiv preprint arXiv:1812.02903, 2018a.

- Yang Y, Feng Y, Suykens J A. Correntropy based matrix completion. *Entropy*, 2018b, 20(3): 171.
- Yang Q, Liu Y, Chen T, *et al.* Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2019a, 10(2): 1–19.
- Yang K, Fan T, Chen T, *et al.* A Quasi-Newton method based vertical federated learning framework for logistic regression. arXiv preprint arXiv:1912.00513, 2019b.
- Yang X, Deng C, Wei K, *et al.* Adversarial learning for robust deep clustering. *Advances in Neural Information Processing Systems*, 2020a, 33.
- Yang M, Deng C, Yan J, *et al.* Learning unseen concepts via hierarchical decomposition and composition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020b: 10248–10256.
- Yao A C-C. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 1986: 162–167.
- Ye T, Xiao P, Sun R. DEED: A general quantization scheme for communication efficiency in bits. arXiv preprint arXiv:2006.11401, 2020.
- Yen I E-H, Lin T-W, Lin S-D, *et al.* Sparse random feature algorithm as coordinate descent in Hilbert space. In *Advances in Neural Information Processing Systems*, 2014: 2456–2464.
- You Y, Lian X, Liu J, *et al.* Asynchronous parallel Greedy coordinate descent. In *Advances in Neural Information Processing Systems*, 2016: 4682–4690.
- You Y, Buluç A, Demmel J. Scaling deep learning on gpu and knights landing clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017: 1–12.
- Yu H, Vaidya J, Jiang X. Privacy-preserving svm classification on vertically partitioned data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2006: 647–656.
- Yu S. Big privacy: Challenges and opportunities of privacy study in the age of big data. *IEEE Access*, 2016, 4: 2751–2763.
- Yu H, Yang S, Zhu S. Parallel restarted SGD for non-convex optimization with faster convergence and less communication. arXiv preprint arXiv:1807.06629, 2018, 2(4): 7.
- Yu H, Yang S, Zhu S. Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, 2019: 5693–5700.

- Yuan X, Li P, Zhang T. Gradient hard thresholding pursuit for sparsity-constrained optimization. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014: 127–135.
- Yun H, Yu H-F, Hsieh C-J, *et al.* NOMAD: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. *Proceedings of the VLDB Endowment*, 2014, 7(11): 975–986.
- Zhang T. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the Twenty-first International Conference on Machine Learning*, 2004: 116.
- Zhang H, Xu Y, Zhang J. Reproducing kernel Banach spaces for machine learning. *Journal of Machine Learning Research*, 2009, 10(Dec): 2741–2775.
- Zhang T. Analysis of multi-stage convex relaxation for sparse regularization. *Journal of Machine Learning Research*, 2010, 11(Mar): 1081–1107.
- Zhang C-H, *et al.* Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics*, 2010, 38(2): 894–942.
- Zhang Y, Duchi J C, Jordan M I, *et al.* Information-theoretic lower bounds for distributed statistical estimation with communication constraints. In *NIPS*, 2013: 2328–2336.
- Zhang R, Kwok J. Asynchronous distributed ADMM for consensus optimization. In *International Conference on Machine Learning*, 2014: 1701–1709.
- Zhang X, Xu C, Jin C, *et al.* Efficient fully homomorphic encryption from RLWE with an extension to a threshold encryption scheme. *Future Generation Computer Systems*, 2014, 36: 180–186.
- Zhang Z. Revisiting fully homomorphic encryption schemes and their cryptographic primitives, 2014.
- Zhang R, Zheng S, Kwok J T. Fast distributed asynchronous SGD with variance reduction. arXiv preprint arXiv:1508.01633, 2015.
- Zhang W, Zhao P, Zhu W, *et al.* Projection-free distributed online learning in networks. In *International Conference on Machine Learning*, 2017: 4054–4062.
- Zhang G-D, Zhao S-Y, Gao H, *et al.* Feature-distributed SVRG for high-dimensional linear classification. arXiv preprint arXiv:1802.03604, 2018.
- Zhang Q, Huang F, Deng C, *et al.* Faster stochastic Quasi-Newton methods. arXiv preprint arXiv:2004.06479, 2020.
- Zhang Y. Analysis of “federated learning” and its application in the financial field. *Rural Finance Research*, 2020, 12(52–58).

- Zhang Q, Gu B, Deng C, *et al.* Secure bilevel asynchronous vertical federated learning with backward updating. arXiv preprint arXiv:2103.00958, 2021.
- Zhao T, Yu M, Wang Y, *et al.* Accelerated mini-batch randomized block coordinate descent method. In *Advances in Neural Information Processing Systems*, 2014a: 3329–3337.
- Zhao J, Jung T, Wang Y, *et al.* Achieving differential privacy of data disclosure in the smart grid. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, 2014b: 504–512.
- Zhao H, Li X, Zhan L, *et al.* Sampling robot data protection method based on blockchain technology. *Journal of Huazhong University of Science and Technology: Natural Science Edition*, 2015(S1): 216–219.
- Zhao S-Y, Li W-J. Fast asynchronous parallel stochastic gradient descent: A lock-free approach with convergence guarantee. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30, 2016.
- Zhao Y, Zhao J, Jiang L, *et al.* Privacy-preserving blockchain-based federated learning for IoT devices. *IEEE Internet of Things Journal*, 2020, 8(3): 1817–1829.
- Zheng W, Popa R A, Gonzalez J E, *et al.* Helen: Maliciously secure cooperative learning for linear models. *CoRR*, 2019, abs/1907.07212. <http://arxiv.org/abs/1907.07212>.
- Zhong L W, Kwok J T. Efficient sparse modeling with automatic feature grouping. *IEEE Transactions on Neural Networks and Learning Systems*, 2012, 23(9): 1436–1447.
- Zhou Z, Mertikopoulos P, Bambos N, *et al.* Distributed asynchronous optimization with unbounded delays: How slow can you go?. In *International Conference on Machine Learning*, 2018: 5970–5979.
- Zhou Y, Wang Z, Ji K, *et al.* Momentum schemes with stochastic variance reduction for nonconvex composite optimization. arXiv preprint arXiv:1902.02715, 2019.
- Zhu C, Han S, Mao H, *et al.* Trained ternary quantization. arXiv preprint arXiv:1612.01064, 2016.
- Zhu T, Li G, Zhou W, *et al.* Differentially private data publishing and analysis: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2017, 29(8): 1619–1638.
- Zhu J, Fu Y. Blockchain application research progress. *Science and Technology Review*, 2017, 35(13): 70–76.
- Zhu L, Gao F, Shen M, *et al.* A review of blockchain privacy protection research. *Journal of Computer Research and Development*, 2017, 54(10): 2170.

- Zhuang W, Wen Y, Zhang X, *et al.* Performance optimization of federated person re-identification via benchmark analysis. In *Proceedings of the 28th ACM International Conference on Multimedia*, 2020: 955–963.
- Zhuo H H, Feng W, Xu Q, *et al.* Federated reinforcement learning. arXiv preprint arXiv:1901.08277, 2019, 1.
- Zinkevich M, Smola A J, Langford J. Slow learners are fast. In *NIPS*, 2009.
- Zou H, Hastie T. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2005, 67(2): 301–320.

Index

A

- asynchronous algorithms, 179, 203–207, 209–210, 213, 215, 231, 242
- asynchronous approaches, 287
- asynchronous communication, 436–438
- asynchronous distributed algorithms, 183
- asynchronous federated algorithms, 287
- asynchronous federated learning, 205
- asynchronous methods, 208, 292, 337
- asynchronous model, 204
- asynchronous parallel algorithms, 275
- asynchronous settings, 205
- asynchronous training, 179, 288, 337
- asynchronous vertical algorithms, 205

B

- blockchain federated learning system, 367, 476
- blockchain-based federated learning, 478

C

- collaborative modeling, 212

D

- data mining, 78, 151–152, 162, 178, 245–247, 249–250, 254, 262
- deep learning, 16, 45, 79, 120, 233–234, 237, 242–243, 321, 324, 326, 331, 340, 355, 368, 402, 431
- deep learning frameworks, 409, 413, 416
- deep learning models, 355, 409, 414
- distributed federated learning, 364–365
- distributed machine learning algorithms, 303
- distributed machine learning system, 358
- distributed reinforcement learning, 343

E

- encryption, 91

F

- federated deep learning, 355
- federated kernel methods, 360
- federated learning, 3, 6–7, 11–14, 16, 21, 23, 26, 35, 40, 43, 56, 64, 109, 111, 114, 122–124, 127, 132, 136, 141, 148, 151–154, 160, 162, 164,

168–169, 172, 178–183, 185–187,
192, 201, 210–212, 214, 220,
233–234, 239, 243, 257, 262–263,
267, 270–271, 278–279, 281–283,
285–287, 289–293, 295–298, 302,
308–312, 314, 319, 345, 353,
355–357, 359–362, 364, 367–368,
385, 394, 401, 407–410, 412, 414,
417, 419, 431–432, 434, 437–438,
451–458, 460–461, 464, 466–477,
479–480
federated learning algorithms, 421,
423, 429, 431, 435, 439, 458
federated learning framework, 409,
413, 416
federated learning incentive
algorithms, 457
federated learning modeling, 437
federated learning models, 419, 432,
464, 468
federated learning platforms,
456–457
federated learning process, 459
federated learning system, 358, 366,
368, 387–388, 395, 473, 476–477,
480–481
federated reinforcement learning, 345

H

homomorphic addition, 83, 90,
102–103, 248, 257, 304, 421–422,
428–429
homomorphic computation, 82–83
homomorphic decryption, 94
homomorphic encryption, 53, 79–80,
82–85, 89–90, 99, 103–106, 112,
114, 123–124, 128, 130, 136,
138–139, 141, 245–248, 250,
254–255, 270, 300, 310–311, 314,
318–319, 357, 360, 386, 405, 419,
421–424, 427–432, 434, 447, 456,
474

homomorphic mathematical
operation, 212
homomorphic multiplication, 90
homomorphic operations, 82–83,
89–90, 92–93, 99, 249–251, 258–259
homomorphic properties, 86, 88–89,
91, 93, 96, 98, 140

K

kernel methods, 120, 154–157, 159,
172, 174, 176, 178

M

machine learning, 6, 16, 18, 23,
25–27, 45, 49, 53, 55, 109–110, 121,
127–128, 151, 156, 162, 178, 181,
183, 185, 192, 212, 233–235,
238–239, 243, 246, 269–271,
289–290, 309, 311, 313, 321, 355,
403, 405, 409, 413, 416, 431, 452,
454, 472, 481
machine learning algorithms, 421, 439
machine learning models, 413–414,
421, 431, 471, 480
multi-agent, 338
multi-party, 179–180, 212, 454
multi-party collaborative modeling,
211, 357–359, 458
multi-party communication, 136, 314
multi-party computation, 53, 56,
68–69, 78, 246, 270, 317, 349,
360–361, 456
multi-party decryption, 305
multi-party generation, 145
multi-party joint modeling, 407,
409–410
multi-party joint training, 411
multi-party key generation, 319
multi-party modeling tasks,
309, 411
multi-party participation, 358
multi-party secure computing, 311

multi-party training, 408, 410–411,
414
multi-party trust issues, 449

P

privacy protection, 33–34, 49–50, 53,
56–58, 60, 63–64, 67–68, 99, 106,
162, 219–220, 233, 356, 447, 452,
456, 474
privacy protection algorithm, 447

R

reinforcement learning, 321–322, 324,
328, 345
respectively, 122

S

secure data mining, 247, 250
secure multi-party computation,
318
synchronous procedures, 164