FEDERATED LEARNING IN MOBILE EDGE COMPUTING: OPTIMIZATION, PRIVACY, AND APPLICATIONS FOR CYBERSECURITY

by

Hengrun Zhang

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

	Dr. Kai Zeng, Dissertation Director
	Dr. Songqing Chen, Committee Co-Chair
	Dr. Yue Cheng, Committee Member
	Dr. Daniel A. Menascé, Committee Member
	Dr. David S. Rosenblum, Department Chair

Date: 04/25/2022

Spring Semester 2022
George Mason University
Fairfax, VA

Federated Learning in Mobile Edge Computing: Optimization, Privacy, and Applications for Cybersecurity

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Hengrun Zhang
Master of Science
George Mason University, Fairfax, VA, 2018
Master of Science
Shanghai Jiao Tong University, Shanghai, China, 2015
Bachelor of Science
East China University of Science and Technology, Shanghai, China, 2012

Director: Dr. Kai Zeng, Professor
Department of Computer Science

Spring Semester 2022
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my wife for her consistent support and encouragement during my entire Ph.D. journey. This dissertation is also dedicated to my parents who always stand behind me and support me both physically and mentally.

Acknowledgments

I would like to express my deepest gratitude to my Ph.D. dissertation director, Dr. Kai Zeng, for all his support. This whole Ph.D. program is quite a long journey for me with numerous hurdles and obstacles, especially during the last two years with the pandemic. I could not reach the destination without his helpful guide. Dr. Zeng's passion on research always influences me. The discussion with him is always informative, and his constructive feedback helps me to reach each research goal. As an international student, life in a foreign country is not always easy. His patience, warm heart, and support greatly relieve me both physically and mentally. I thank him from the bottom of my heart.

I would like to extend my appreciation to my dissertation committee members, Dr. Songqing Chen, Dr. Yue Cheng, and Dr. Daniel A. Menascé, for their help, guidance, and suggestions during my whole Ph.D. journey. I neither can reach the destination without their hands. I also would like to thank all the other professors and specialists who helped me in some other ways: Dr. Liang Zhao, Dr. Shucheng Yu, Dr. Hakan Aydın, Mr. Ryan Lucas, Ms. Cecelia Kimes, etc. Although some of them may not quite remember my face, I will never forget their help.

I also would like to appreciate my lab-mates in Wireless Innovation and Cybersecurity Lab (WICL): Monireh Dabaghchian, Amir Alipour-Fanid, Long Jiao, Jie Tang, Shuai Lin, Ning Wang, Pu Wang, Junqing Le, Weiwei Li, Songlin Chen, Yaqi He, Zhihao Li, Guohua Sun. The moments with them are always joyful. I will never forget their support and friendship in my whole Ph.D. study. Some of them have left the lab to start a new journey, while others are still on the way. I wish all of them the best luck in the future.

Last but not least, I also want to thank all my family members for their support, understanding, and sacrifice during my entire Ph.D. study. They are always my hardest backup who sweep all my worries behind.

I would like to also acknowledge that some works in my dissertation have been partially supported by the Commonwealth Cyber Initiative (CCI) and its Northern Virginia (NOVA) Node.

Table of Contents

List of Tables

List of Figures

Abstract

FEDERATED LEARNING IN MOBILE EDGE COMPUTING: OPTIMIZATION, PRIVACY, AND APPLICATIONS FOR CYBERSECURITY

Hengrun Zhang, PhD

George Mason University, 2022

Dissertation Director: Dr. Kai Zeng

In the big data era, compared with traditional centralized machine learning, federated learning can greatly reduce data collection time, relieve computation burdens of local clients, and preserve data privacy. Recently, communication overhead in federated learning begins to attract increasing attentions. At first, federated learning needs several communication rounds between local devices and public servers for model aggregation, which are mostly through wireless networks. Besides, since uploaded model parameters in federated learning are still vulnerable to inference attacks, extra privacy masks are needed for them, such as Secret Sharing (SS) and secure Multi-Party Computation (MPC), which can further aggravate communication. In addition, the number of communication rounds can soar if adversaries try to slow convergence of federated learning by poisoning local data or uploaded model parameters.

In this dissertation, we provide comprehensive analysis about optimizing communication overhead in federated learning from the above three aspects. In detail, we discuss the following four research projects: 1) We consider communication overhead reduction through convergence performance optimization in federated learning via introducing centralized machine learning-based adaptive learning strategies to the model parameter update rule. Con-

vergence upper bounds under our optimization scheme are derived after each communication round with a certain number of local iterations, and after a given number of communication rounds. Through comparison with the bounds of original federated learning, we theoretically analyze how those strategies should be tuned to help federated learning effectively optimize convergence performance and reduce overall communication overhead; 2) We propose a privacy-preserving task scheduling strategy based on (2,2) SS and mobile edge computing to reduce data processing latency, in which locally-learned model parameters are separated into two portions before uploaded to public edge servers for parameter aggregation based on MPC. We show that the related privacy constraint can be enforced through constructing a pairwise Markov chain. We further formulate the whole task scheduling problem as a stochastic latency minimization problem and solve it by converting it into a linear programming problem; 3) We further extend the (2,2) case to (R,L) case, and propose a communication-aware secret share placement strategy to optimize communication overhead by minimizing weighted transmission hop counts in a hierarchical edge computing architecture. We show that the constructed optimization problem is NP-hard, and efficient heuristic algorithms can be applied to find sub-optimal solutions. We respectively evaluate two traditional heuristics, i.e. Genetic Algorithm (GA) and Particle Swarm Optimization (PSO), introduce two basic heuristics, i.e. top-down and bottom-up heuristic, and further propose an advanced algorithm, called Bottom-Up Top-Down ($BUTD$) heuristic. Based on comparison, we find that our proposed $BUTD$ heuristic can outperform all the other four heuristics when communication among different shares of the same secret is comparable to that among different secrets; 4) Finally, we talk about combating data poisoning attacks through developing a federated self-learning intrusion detection system, which is based on a practical application of federated learning in cybersecurity. This application targets Controller Area Network (CAN bus) and is based on Graph Neural Network (GNN). We show that different driving scenarios and vehicle states will impact sequence patterns and data contents of CAN messages. In this case, we develop a federated learning architecture to accelerate the learning process while preserving data privacy.

Chapter 1: Introduction

1.1 Background and Motivation

Traditional centralized machine learning usually faces the following two challenges: 1) Collecting enough valid data for model training can take extravagant time; 2) Model training can require excessive computational resources. Data explosion in recent years has further made machine learning and data mining tasks almost impossible and impractical to be processed in a single local device, especially considering that mobile phones and tablets are becoming primary computing devices for most people [1, 2]. In this situation, cloud computing or mobile edge computing become the saviors for all such tasks.

However, cloud computing or mobile edge computing can bring a myriad of privacy concerns with the raw data upload requirement. Recently, a new distributed learning algorithm, called federated learning [3], has been proposed to counter the challenges of centralized learning while preserving the data privacy. An overview of federated learning process is shown in Fig. 1.1. Instead of directly uploading raw data to central servers, in federated learning, each end device can train its own data locally, and only upload learned model parameters for global model parameter aggregation, which greatly improves the privacy level. On the other hand, communication overhead in federated learning has recently become one of its major concerns. At first, federated learning needs several communication rounds between local devices and public servers for model aggregation, which are mostly through wireless networks. Besides, since uploaded model parameters in federated learning are still vulnerable to inference attacks, extra privacy masks are needed for them, such as Secret Sharing (SS) and secure Multi-Party Computation (MPC), which can further aggravate communication. In addition, the number of communication rounds can soar if adversaries try to slow convergence of federated learning by poisoning local data or uploaded model parameters.

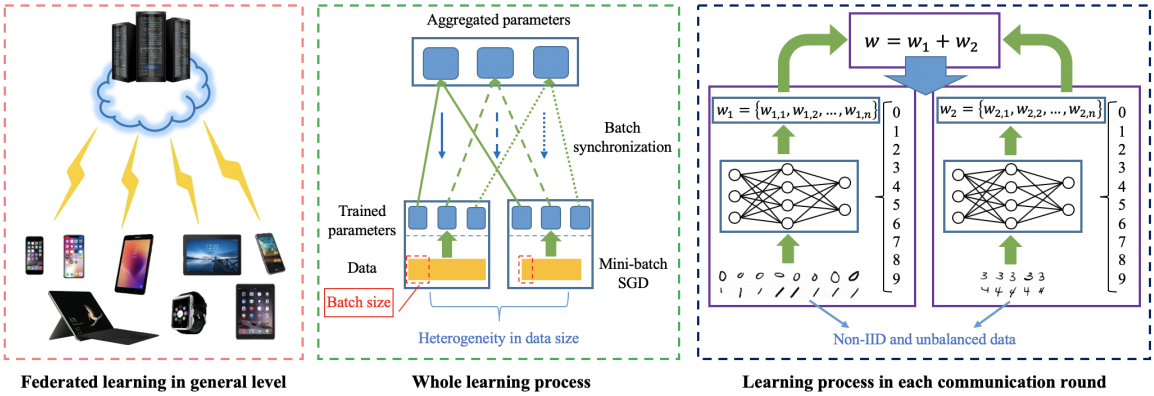


Figure 1.1: The overview of federated learning

Considering the above observations, we need to develop communication overhead reduction strategies from different aspects to improve the applicability of federated learning.

1.2 Problem Statement

In this dissertation, we aim at solving *how to optimize communication overhead during the whole process of federated learning*, which can be further divided into the following three detailed research problems:

- How can those adaptive learning strategies originally proposed in centralized machine learning be adapted to federated learning to reduce communication overhead through accelerating convergence rate and improving convergence performance?
- After introducing SS and MPC to further protect uploaded model parameters, and mobile edge computing to relieve communication concerns between public servers and local devices, how can we minimize computation latency and communication overhead among those edge servers?
- How can we develop intrusion detection systems to filter out those compromised local devices and attacks, which can negative impact the convergence performance of federated learning, and further raise the number of necessary communication rounds?

1.3 Research Projects in This Dissertation

The authors in [3] have shown that federated learning can realize proper convergence accuracy based on mini-batch Stochastic Gradient Descent (SGD) for unbalanced and non-Independent-and-Identically-Distributed (non-IID) datasets. However, considering that several rounds of global model parameter aggregation are needed during the whole learning process, communication overhead in federated learning largely depends on its convergence performance, which can be further impacted by system and data statistical heterogeneity [4]. In the real world, different end devices can have different computation capabilities, communication conditions and power levels (left block in Fig. 1.1), which introduce system heterogeneity. These issues can cause some end devices (usually called stragglers) not able to finish enough iterations timely for local model parameter update, thus uploading outdated model parameters. On the other hand, different end devices may have local datasets with different sizes (middle block in Fig. 1.1) and non-IID distributions (right block in Fig. 1.1 for training, which are considered as data statistical heterogeneity. Derived local model parameters in the current communication round can greatly drift away from previous global model parameters because of system and data statistical heterogeneity, which causes unexpected fluctuations or even divergences. Targeting the above two kinds of heterogeneity, many existing works have proposed convergence performance optimization strategies through end device synchronization [5–7], end device selection [8–10], cross-client variance reduction [11–13], etc.

However, all the above federated learning optimization algorithms are developed based on the original federated learning. In centralized machine learning, many adaptive learning strategies [14–16] are developed to optimize convergence performance, which in general consider momentum and adaptive learning rate. These adaptive strategies can be introduced to federated learning before any further optimization targeting heterogeneity issues. In [17], the authors try to generalize relevant algorithms and apply them in federated learning, such as *FedAdagrad*, *FedAdam*, and *FedYogi*. However, all these algorithms are only simple

extensions of corresponding adaptive learning algorithms. Although convergence guarantee is provided in this paper, they do not give rigorous analysis about the potential solution bias and the impact of momentum and adaptive learning rate on federated learning. With the above observations, we carefully design an adaptive Federated learning algorithm targeting the model parameter Update Rule, called *FedUR*, which is based on comprehensive analysis of the impact of those adaptive learning strategies on federated learning. Detailed discussion is in Chapter 2.

Although federated learning can greatly improve the privacy level, recent works have shown that model parameters are still vulnerable to inference attacks [18–21], and extra privacy masks are needed for those uploaded model parameters. Currently, Secret Sharing (SS) [22,23], secure Multi-Party Computation (MPC) [24,25] and Differential Privacy (DP) [26,27] are the most commonly-adopted techniques to protect uploaded model parameters. Therein, SS and MPC are usually considered together. Let's take (R,L) SS as an example. Locally-learned model parameters will be randomly split into L portions before uploaded to public servers for aggregation based on MPC. A (R,L) SS scheme has a privacy requirement that a single public server can obtain at most $R-1$ secret shares at the same time. Such a split actually acts as a means of encryption. Compared with MPC based on homomorphic encryption [28], SS has lower computation complexity and higher robustness to user dropouts, which make it more suitable to model parameter aggregation in federated learning, since local devices, such as smart phones and tablets, usually have limited computation capacity, and communication is usually through wireless networks. DP is another light-weighted privacy-preserving technique, which protects uploaded model parameters by adding artificial noise.

DP usually induces the tradeoff between convergence performance and uploaded model parameter privacy in federated learning while SS and MPC raise concerns in data processing latency and communication overhead. In this dissertation, we consider the problems brought by SS and MPC. In federated learning, aggregating M sets of locally-learned model parameters will incur $O(M)$ of communication overhead between public servers and local

devices. With the introduction of SS, an extra $O(ML)$ communication overhead will be caused among local devices if each secret is separated to L shares. This high communication overhead poses great challenges in communication latency and transmission power management. Recently, mobile edge computing [29, 30] is proposed to replace conventional cloud computing, which can effectively lower communication overhead by offloading some computation tasks to nearby edge nodes instead of remote cloud servers. On the other hand, different from cloud computing, edge nodes are distributed across the whole edge network, and each edge node is usually believed to have limited computation and storage capacity, which can cause data processing latency. Based on the above observations, we propose two privacy-preserving task scheduling strategies in mobile edge computing.

The first strategy targets (2,2) SS. A pairwise Markov chain is constructed to minimize data processing latency on the edge nodes. Once edge nodes receive uploaded model parameter portions, they will decide whether to process those portions themselves or assign portions to other nodes based on the constructed Markov chain. Considering the aforementioned privacy constraint, we cannot consider each node separately. Instead, each two nodes will be considered in pairs, and this is why we call our Markov chain "pairwise". The Markov chain will be constructed carefully with possible pair transitions. The second strategy is an extension of the first one. At first, the (2,2) SS is generalized to (R,L). Besides, the strategy is developed in a hierarchical mobile edge computing architecture, which has been widely considered to provide flexibility in balancing traffic loads in the whole edge network and bring improved quality of service (QoS) [31–34]. Furthermore, both data processing latency and communication overhead are considered in this strategy. Since the constructed optimization problem is NP-hard, we try to find the sub-optimal solutions based on two traditional heuristics (GA and PSO) and three proposed heuristics (top-down, bottom-up and $BUTD$ heuristic). Based on comparison, we find that our proposed $BUTD$ heuristic can outperform all the other four heuristics when communication among different shares of the same secret is comparable to that among different secrets. Detailed discussion is in Chapter 3 and Chapter 4.

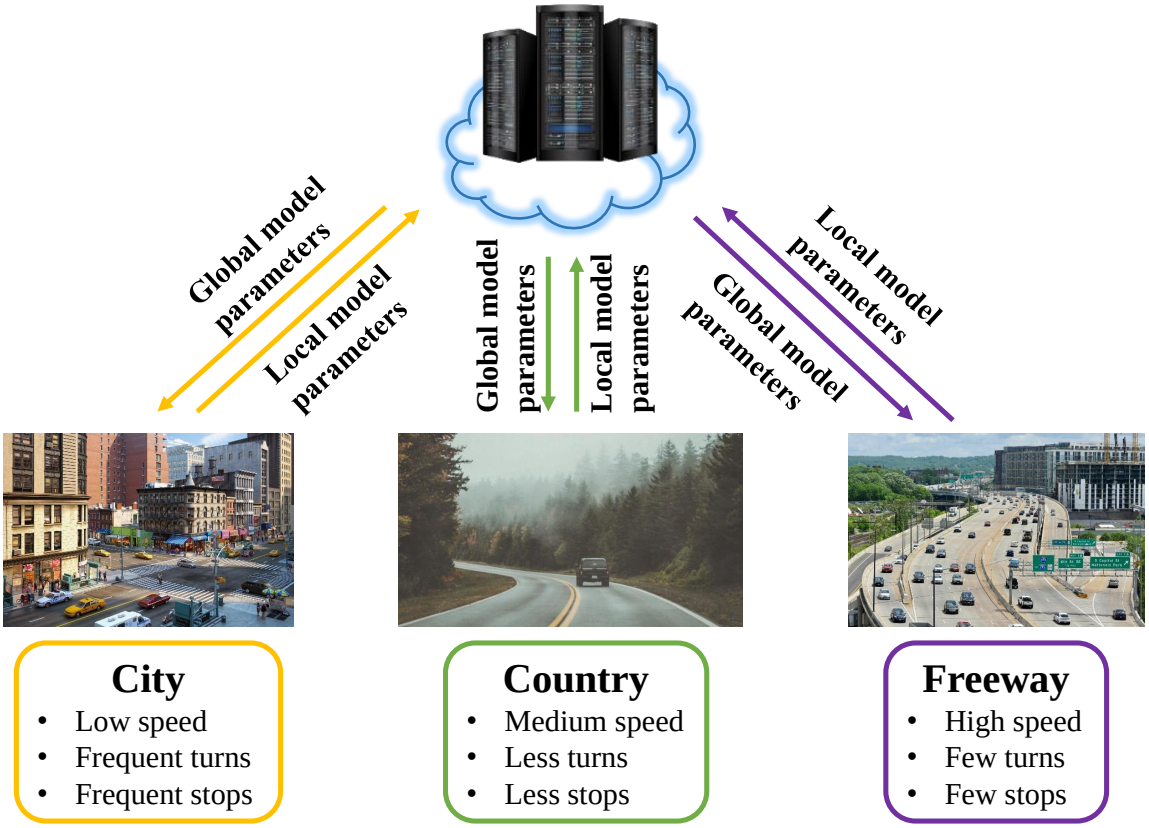


Figure 1.2: Vehicles Driving in Different Scenarios

Federated learning can be seen as an Internet-of-Things (IoT) system, which usually faces numerous security problems. Attackers can try to compromise local devices, and poison raw data or uploaded model parameters [35–38], which negatively impacts convergence performance of federated learning, and further increases the number of necessary communication rounds between local devices and public servers. When developing robust model parameter aggregation strategies, federated learning can be conversely relied on to design self-learning intrusion detection systems for combating those security issues. In Chapter 5, targeting the CAN bus system in the vehicle network, we propose a federated intrusion detection system for data poisoning attacks. As is illustrated in Fig. 1.2, vehicles can drive in different scenarios (e.g. city, country, and freeway) with different features. Some vehicle states happening frequently in one scenario may be seldom seen in another. For example,

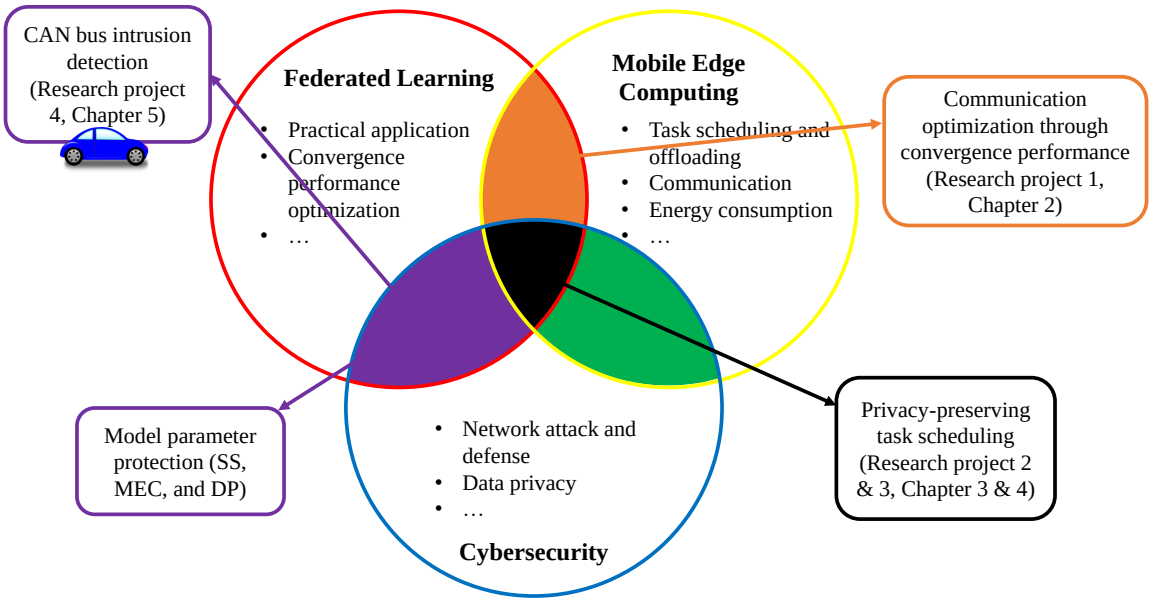


Figure 1.3: Dissertation Architecture

a driving speed of 60 mph can be seen quite often in freeways while is almost impossible to happen in cities. Changes in vehicle states can further lead to variations in message sequences of CAN bus communication. Therefore, the intrusion detection model trained on one vehicle will be constrained by its limited driving scenarios and vehicle states (e.g., a taxi may mostly drive in cities with low speed and a lot of stops), so cannot be applied to other vehicles with different driving scenarios and vehicle states (e.g., an intercity bus may mostly drive on freeways with high speed and few stops). To take advantage of crowd-sourcing while protecting user data privacy, we can adopt a federated learning framework to train a universal model that covers a wide range of driving scenarios and vehicle states.

The general architecture of this dissertation can be described in Fig. 1.3.

Chapter 2: FedUR: Federated Learning Optimization through Adaptive Centralized Learning Optimizers

In this chapter, we aim at reducing communication overhead in federated learning by solving the first detailed research problem in Section 1.2. One paper related to this research project is currently under review [39].

2.1 Introduction

Considering that several rounds of global model parameter aggregation are needed during the whole learning process, communication overhead in federated learning largely depends on its convergence performance, which can be further impacted by system and data statistical heterogeneity [4]. In the real world, different end devices can have different computation capabilities, communication conditions and power levels (left block in Fig. 1.1), which introduce system heterogeneity. These issues can cause some end devices (usually called stragglers) not able to finish enough iterations timely for local model parameter update, thus uploading outdated model parameters. On the other hand, different end devices may have local datasets with different sizes (middle block in Fig. 1.1) and non-IID distributions (right block in Fig. 1.1 for training, which are considered as data statistical heterogeneity. Derived local model parameters in the current communication round can greatly drift away from previous global model parameters because of system and data statistical heterogeneity, which causes unexpected fluctuations or even divergences. Targeting the above two kinds of heterogeneity, many existing works have proposed convergence performance optimization strategies through end device synchronization [5–7], end device selection [8–10], cross-client variance reduction [11–13], etc.

However, all the above federated learning optimization algorithms are developed based on the original federated learning. In centralized machine learning, many adaptive learning strategies [14–16] are developed to optimize convergence performance, which in general consider momentum and adaptive learning rate. These adaptive strategies can be introduced to federated learning before any further optimization targeting heterogeneity issues. In [17], the authors try to generalize relevant algorithms and apply them in federated learning, such as *FedAdagrad*, *FedAdam*, and *FedYogi*. However, all these algorithms are only simple extensions of corresponding adaptive learning algorithms. Although convergence guarantee is provided in this paper, they do not give rigorous analysis about the potential solution bias and the impact of momentum and adaptive learning rate on federated learning. With the above observations, we carefully design an adaptive federated learning strategy, called *FedUR*, which is based on comprehensive analysis of the impact of those adaptive learning strategies on federated learning. The main contributions of this research project are as follows:

- We fully investigate the change of surrogate update rule after introducing adaptive learning strategies. We show that the surrogate update rules of current adaptive federated learning algorithms are equivalent to the objective update rules only when each communication round has a single local iteration.

- We further consider convergence performance when each communication round has multiple local iterations. We respectively analyze the convergence upper bound after each communication round and a given number of communication rounds. Through comparison with those bounds of the original federated learning, we theoretically analyze the impact of momentum and adaptive learning rate on federated learning and how to tune them to effectively optimize convergence performance.

- We evaluate our proposed adaptive federated learning strategy, i.e. *FedUR*, through extensive experiments based on several real datasets and machine learning models. Experimental results show that compared to existing adaptive federated learning algorithms,

FedUR can in general increase final convergence accuracy by 5.5-17.4%, and accelerate the whole learning process by 21.9-47.2% with lower communication overhead.

The rest of this chapter is organized as follows. Related works will be summarized in Section 2.2. Preliminaries and basics in this research project will be introduced in Section 2.3. Convergence performance and communication overhead will be analyzed in Section 2.4, which is followed by solving two application issues of *FedUR* in Section 2.5. Experimental results for validation and evaluation will be provided in Section 2.6. Some future directions on federated learning optimization will be discussed in Section 2.7. Finally, conclusions will be drawn in Section 2.8.

2.2 Related Work

Federated learning, firstly proposed by Google [3] and usually called *FedAvg*, is a new distributed learning algorithm, which can protect data privacy during model training. Communication overhead is an important factor in federated learning. At first, the communication links from the end devices, such as smart phones and sensors, to the cloud are mostly wireless. Besides, several rounds of communication are needed between end devices and central servers for global model parameter aggregation. Intuitively, communication overhead can be diminished by reducing the uploaded data size in each communication round or the necessary number of communication rounds in the whole learning process.

As to reducing the uploaded data size, [40–42] consider compressing uploaded model parameters. Recently, *FedPAQ* [43] proposes to upload a quantized version of local model parameters, and also analyzes the impact on convergence performance. *ASTW_FedAVG* [44] develops a layerwise local model parameter update strategy, in which parameters in deep layers of the learning model are uploaded less frequently than those in the shallow layers. Partial local device participation in each communication round (similar to SGD) is another strategy. Convergence performance can further get ameliorated via intelligent end device selection. *LAG* [45] selects end devices based on model update contributions, in

which local clients approaching local optima do not need to upload their model parameters. *FAVOR* [8] uses deep reinforcement learning to select a subset of end devices achieving the best test accuracy. *FOLB* [9] aims at a near-optimal device selection distribution to maximize the loss decrease. Communication conditions are further considered in [46, 47] during end device selection. *AAFL* [7] explores the best number of selected end devices in each communication round based on current network conditions and deep reinforcement learning.

Wang *et al.* [48] systematically analyze convergence upper bounds of gradient descent under *FedAvg* and non-IID data distributions, which are further optimized under the constraint in communication rounds. On the other hand, convergence performance optimization of federated learning can also reduce the necessary number of communication rounds, which is usually about tackling those heterogeneity issues. According to [4], heterogeneity issues can be further divided into system and data statistical heterogeneity. System heterogeneity has been considered since the era of distributed learning. Different end devices may have different computation capacities, communication conditions and power levels, which usually cause synchronization problems. Up to now, there are in general three types of systems to tackle those synchronization problems, i.e. Bulk Synchronous Parallel (BSP) systems [49–51], ASynchronous Parallel (ASP) systems [52–54], and Stale Synchronous Parallel (SSP) systems [5, 55, 56]. These three kinds of systems respectively completely wait, completely do not wait, and partially wait for all selected end devices to upload local updates before each global aggregation. Recently, *FedNova* [6] further investigates the impact of asynchronous model aggregation on convergence performance in federated learning, and proposes normalized weighted gradients to alleviate objective inconsistency. *PSP* [57] provides flexibility in local model parameter aggregation through a hierarchical mobile edge computing structure.

Data statistical heterogeneity is considered after federated learning is proposed. *CABS* [58], *BA-SGD* [59], and *Adaptive-B* [10] try to balance the global data distribution through batch size optimization. *FedProx* [11] and *FedDANE* [60] introduce a proximal term to the

original loss function for local model parameter update, which makes updated parameters more similar across different end devices with non-IID data distributions. Such a strategy can provide significantly more stable and accurate convergence performance compared with *FedAvg*. *VRL-SGD* [12] and *SCAFFOLD* [13] focus on modifying gradients to mitigate cross-client variance. *FedPD* [61] expands the discussion to non-convex loss functions, and optimizes federated learning from the primal-dual perspective.

Recently, some works have tried to generalize those centralized learning-based adaptive strategies [14–16] to federated learning [17, 62, 63], and propose relevant algorithms, such as *FedAdagrad*, *FedAdam*, and *FedYogi*. However, the impact of these adaptive strategies on federated learning has not been well studied, nor the convergence performance has been optimized. In this research project, we fully investigate the impact of momentum and adaptive learning rate on federated learning, and analyze how to tweak these two factors to optimize convergence performance and communication overhead.

2.3 Preliminaries and Basics

When it comes to evaluating a machine learning algorithm, the final convergence accuracy is usually an important metric. On the other hand, as has been discussed in the previous section, communication overhead also needs to be considered for federated learning, which further depends on the data size of transmitted model parameters, the number of selected end devices, and the number of communication rounds (or convergence rate). In this research project, we aim to reduce the number of communication rounds through optimizing convergence performance from the aspect of the update rule. In the view of local iterations, we consider the following two factors:

- **Communication frequency:** the number of local iterations before each global model parameter aggregation, denoted by τ.
- **Convergence rate:** the total number of local iterations in the whole learning process, denoted by T.

Then, the total number of communication rounds C can be calculated by $C = T/\tau$. Based on this formula, we can see that a smaller T and larger τ will result in a reduced C. *FedUR* is also orthogonal to those aforementioned federated learning optimization methods targeting system and data statistical heterogeneity, which can be integrated with *FedUR* to further increase convergence rate and reduce communication overhead. The core notations in this research project are summarized in Table 2.1.

Table 2.1: Core Notations

Notations	Description
N_k, N, K	Batch size of the k-th end device, the whole population, and total number of selected end devices.
$f_n(\boldsymbol{w})$, $F_k(\boldsymbol{w})$, $F(\boldsymbol{w})$	Loss function of the n-th data sample, the k-th end device, and global loss function.
$\boldsymbol{G}_k(t)$, $\boldsymbol{G}(t)$	Mini-batch gradient of the k-th end device, and global mini-batch gradient in the t-th iteration.
β_1, β_2	First and second-order exponential decay rate, $\beta_1, \beta_2 \in [0, 1)$.
$\boldsymbol{m}_k(t)$, $\boldsymbol{m}(t)$	Biased first raw moment estimate of the k-th end device, and the whole population.
$\boldsymbol{u}_k(t)$, $\boldsymbol{u}(t)$	Biased second raw moment estimate of the k-th end device, and the whole population.
$\hat{\boldsymbol{m}}_k(t)$, $\hat{\boldsymbol{m}}(t)$	Bias-corrected first raw moment estimate of the k-th end device, and the whole population.
$\hat{\boldsymbol{u}}_k(t)$, $\hat{\boldsymbol{u}}(t)$	Bias-corrected second raw moment estimate of the k-th end device, and the whole population.
$\boldsymbol{\eta}_k(t)$, $\boldsymbol{\eta}(t)$	Learning rate vector of the k-th end device, and the whole population in the t-th iteration.
$\boldsymbol{w}_k(t)$, $\boldsymbol{w}(t)$	Learned model parameter vectors of the k-th end device, and the whole population.
$\boldsymbol{w}^F(t)$	Aggregated global model parameter vectors in the t-th iteration.

2.3.1 Federated Learning

Federated learning can be seen as a special type of distributed learning, but has no control of data distributions. Each end device owns a portion of the whole dataset for local training, but those data portions are unbalanced and non-IID. With a given type of loss function $f(\cdot)$, federated learning can derive the loss function for each data sample $\boldsymbol{x}_n$ in each end device, i.e. $f_n(\boldsymbol{w})$. Under mini-batch SGD, the mini-batch loss function of the k-th end device can be calculated by averaging loss functions across all selected data samples:

$$F_k(\boldsymbol{w}) = \frac{1}{N_k} \sum_{n \in P_k} f_n(\boldsymbol{w}) \tag{2.1}$$

where P_k is the set of selected data samples in the k-th end device.

When it comes to local model parameter update in the k-th end device, the corresponding gradient $\boldsymbol{G}_k(t)$ in each local iteration t can be derived by computing partial derivatives of $F_k(\boldsymbol{w})$ on the aggregated global model parameter vector $\boldsymbol{w}^F(t)$. Then, the i-th model parameter in the k-th end device will be updated as follows:

$$w_{k,i}(t+1) \leftarrow w_i^F(t) - \eta_{k,i}(t) G_{k,i}(t) \tag{2.2}$$

Afterwards, all the K selected end devices will upload their learned model parameters to the central server for aggregation:

$$\boldsymbol{w}^F(t+1) = \frac{1}{N} \sum_{k=1}^{K} N_k \boldsymbol{w}_k(t+1) \tag{2.3}$$

The result is considered as the globally learned model parameters after the t-th iteration. Finally, $\boldsymbol{w}^F(t+1)$ will be broadcast to all end devices for the next iteration.

Formula (2.2) and (2.3) together construct the surrogate model update rule of federated learning. In [3], the authors show its equivalence to the objective update rule $w_i(t+1) =$

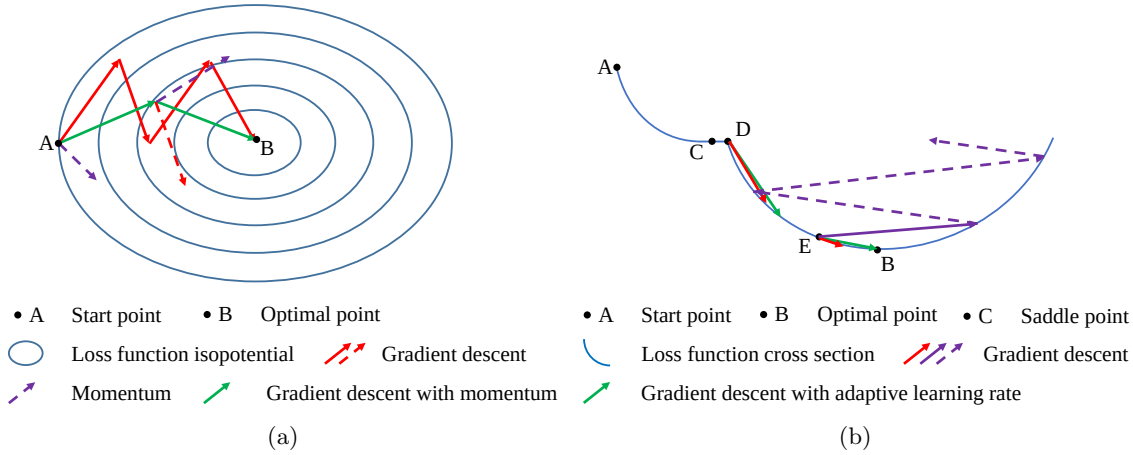


Figure 2.1: Effect of (a) momentum and (b) adaptive learning rate

$w_i(t) - \eta G_i(t)$ under a static learning rate ($\eta_{k,i}(t) = \eta$ for all i, k, and t) and one local iteration per communication round.

2.3.2 Adaptive Federated Optimization

All the centralized learning-based adaptive algorithms are in general based on accumulated gradients, which are used to define momentum and adaptive learning rate. These two terms can help to optimize convergence performance through tweaking learning directions and step sizes respectively, which are illustrated in Fig. 2.1.

In this subsection, we give the analysis based on *Adam* [15]. All the other algorithms have similar strategies. With the introduction of momentum and adaptive learning rate, the i-th model parameter will be updated as follows:

$$w_i(t+1) \leftarrow w_i(t) - \eta_i(t) m_i(t) \tag{2.4}$$

with

$$m_i(t) \leftarrow \beta_1 m_i(t-1) + (1-\beta_1) G_i(t) \tag{2.5a}$$

$$\eta_i(t) \triangleq \frac{\eta_0}{\sqrt{u_i(t)} + \epsilon} \tag{2.5b}$$

Here, η_0 is a predefined global learning rate. ϵ is a small constant added to prevent the denominator being zero. $u_i(t)$ is updated as follows:

$$u_i(t) \leftarrow \beta_2 u_i(t-1) + (1-\beta_2) G_i^2(t) \tag{2.6}$$

Note that all the first and second moment estimates are initialized to 0, which can cause biases. Therefore, the following corrections are further applied to tackle them.

$$\hat{m}_i(t) = \frac{m_i(t)}{1-\beta_1^t} \qquad \hat{u}_i(t) = \frac{u_i(t)}{1-\beta_2^t} \tag{2.7}$$

All the above formulae denote the objective model update rule. In [17], the authors extend those adaptive algorithms to federated learning, and let's still take *FedAdam* as an example, which is summarized in Algorithm 1.

End devices in *FedAdam* will upload derived local gradients instead of local model parameters. In this case, those global moment estimates and model parameters can be directly derived in the central server based on the aggregated global gradients. However, as is shown in line 22 of Algorithm 1, each end device actually uploads **accumulated local gradients** when each communication round has multiple local iterations. Let's take the momentum in the first communication round as an example. In the objective update rule, if we fully expand Formula (2.5a), we can get:

$$\boldsymbol{m}(\tau) = (1-\beta_1) \sum_{t=1}^{\tau} \beta_1^{\tau-t} \boldsymbol{G}(t) \tag{2.8}$$

Algorithm 1: *FedAdam*

Input : N: global batch size. K: number of selected end device. N_k: local batch size, $k = 1, 2, \cdots, K$. τ: number of local iterations. β_1, β_2: decay rate. ϵ: small constant.

Output: $\boldsymbol{w}^{F^*}$: best aggregated global model parameters.

Initialize $t \leftarrow 0$, $\boldsymbol{w}^F(0) \leftarrow$ random seed, $\boldsymbol{w}^{F^*} \leftarrow \boldsymbol{w}^F(0)$ $\boldsymbol{m}(0) \leftarrow \mathbf{0}$, $\boldsymbol{u}(0) \leftarrow \mathbf{0}$;

for $j = 0, 1, 2, \cdots$ **do**

  $S \leftarrow$ random set of K end devices;

  for *each end device* k *in* S **in parallel do**

    $\boldsymbol{G}_k((j+1)\tau) \leftarrow$ `LocalUpdate(`$\boldsymbol{w}^F(j\tau), k$`)`;

  $\boldsymbol{G}((j+1)\tau) \leftarrow \frac{1}{N}\sum_{k=1}^{K} N_k \boldsymbol{G}_k((j+1)\tau)$;

  Compute $m_i((j+1)\tau)$ and $\hat{m}_i((j+1)\tau)$ based on Formula (2.5a) and (2.7);

  Compute $u_i((j+1)\tau)$ and $\hat{u}_i((j+1)\tau)$ based on Formula (2.6) and (2.7);

  $\boldsymbol{w}^F((j+1)\tau)) \leftarrow \boldsymbol{w}^F(j\tau) - \eta_0 \frac{\hat{m}_i((j+1)\tau)}{\sqrt{\hat{u}_i((j+1)\tau)+\epsilon}}$;

  if $F(\boldsymbol{w}^F((j+1)\tau)) < F(\boldsymbol{w}^{F^*})$ **then**

    $\boldsymbol{w}^{F^*} \leftarrow \boldsymbol{w}^F((j+1)\tau)$

  if $F(\boldsymbol{w}^{F^*})$ *is less than a threshold* **then**

    break ; `//Learning process ends.`

Function `LocalUpdate(`$\boldsymbol{w}, k$`):`

  for *each local iteration* t **do**

    $t \leftarrow t + 1$;

    $S_L(t) \leftarrow$ random set of N_k data samples;

    Compute the gradient $\boldsymbol{g}_{\boldsymbol{d}}(t)$ for each selected data sample $\boldsymbol{d} \in S_L(t)$;

    $\boldsymbol{G}_k(t) \leftarrow \frac{1}{N_k}\sum_{\boldsymbol{d} \in S_L(t)} \boldsymbol{g}_{\boldsymbol{d}}(t)$;

    $\boldsymbol{w}_k(t) \leftarrow \boldsymbol{w}_k(t-1) - \eta_l \boldsymbol{G}_k(t)$;

    if $t \mod \tau = 0$ **then**

      return $\boldsymbol{w} - \boldsymbol{w}_k(t)$;

Under *FedAdam*, the surrogate update rule will become:

$$\boldsymbol{m}(\tau) = (1 - \beta_1) \sum_{t=1}^{\tau} \boldsymbol{G}'(t) \tag{2.9}$$

Based on the above comparison, we can see the following two problems: 1) Directly using accumulated local gradients will remove the effect of exponential decay; 2) The global gradient $\boldsymbol{G}'(t)$ in Formula (2.9) is not equal to $\boldsymbol{G}(t)$, since in each local iteration t, the local gradient $\boldsymbol{G}_k(t)$ is calculated based on $\boldsymbol{w}_k(t-1)$ instead of $\boldsymbol{w}^F(t-1)$. For adaptive learning rate, we can have similar observations. Such two problems can cause unnecessary fluctuations and extra communication rounds in the learning process.

In [17], the authors do not consider the above two problems. Instead, they focus on ensuring convergence by introducing a local learning rate η_l (line 20 of Algorithm 1), which is not necessary based on our analysis in the next section. Besides, the convergence upper bounds in [17] are in terms of global gradients, which are actually comparative results. They cannot guarantee that the learning process will finally converge to a global optimum. Our *FedUR* is based on the analysis about these two problems. We show that properly tweaking hyper-parameters can effectively alleviate their impact on convergence performance. All the derived convergence upper bounds are in reference to the global optimum.

2.4 Convergence Performance Analysis

In this section, we will derive convergence upper bounds under accumulated local gradients to show that the two problems discussed in Section 2.3.2 will not cause divergence in the learning process. In [48], the authors investigate the convergence performance under *FedAvg* and accumulated local gradients. Through comparison with those convergence upper bounds, we will analyze how to tune relevant hyper-parameters to improve convergence performance. A lower convergence upper bound under the same number of local iterations can further infer lower communication overhead under the same convergence requirement. As has been discussed in Section 2.3, communication frequency and convergence rate together determine the overall communication overhead. Correspondingly, we will derive two upper bounds for comparison.

2.4.1 Basic Assumptions and Definitions

For a proper comparison, our theoretical analysis setting is similar to that in [48]. *FedUR* is based on synchronized model parameter aggregation. Assume when the whole learning process finally converges, the total number of local iterations is T, which is further separated into J intervals, with each interval containing τ local iterations and corresponding to one communication round in federated learning. Then, we use the shorthand $[j]$ to denote the j-th iteration interval during the whole learning process, i.e. the interval $[(j-1)\tau, j\tau]$ with $j = 1, 2, \cdots, J$. From the perspective of centralized learning, we define the objective model parameter update rule for each iteration interval $t \in [j]$ as follows:

$$w_{[j],i}(t+1) \leftarrow w_{[j],i}(t) - \frac{\eta_0}{\sqrt{\hat{u}_{[j],i}(t)} + \epsilon} \hat{m}_{[j],i}(t) \tag{2.10}$$

Compared with *FedAdam*, our *FedUR* offloads momentum calculation back to local iterations, through which we can resume the effect of exponential decay. Besides, we move the predefined global learning rate η_0 to local updates. In this case, for the k-th end device and in each iteration interval $t \in [j]$, we have:

$$w_{[j],k,i}(t+1) \leftarrow w_{[j],k,i}(t) - \eta_0 \hat{m}_{[j],k,i}(t) \tag{2.11}$$

Afterwards, locally-learned model parameters instead of local gradients will be uploaded to the central server for aggregation. In this case, the central server does not need to keep track of those moment estimates except aggregated global parameters. The adaptive learning rate $\boldsymbol{\eta}_{[j]}(t)$ is introduced in global model parameter aggregation, and the surrogate model parameter update rule becomes:

$$w^F_{[j],i}(j\tau) = \frac{1}{N} \sum_{k=1}^{K} N_k [w^F_{[j],i}((j-1)\tau) - \frac{w^F_{[j],i}((j-1)\tau) - w_{[j],k,i}(j\tau)}{\sqrt{\hat{u}_{[j],i}} + \epsilon}] \tag{2.12}$$

Here, we approximate $\boldsymbol{\eta}_{[j]}(t)$ based on the expected global bias-corrected second moment estimate in the iteration interval $[j]$, with each element denoted by $\hat{u}_{[j],i}$. With such a design, in each iteration interval, our algorithm can be considered as having a static learning rate $\boldsymbol{\eta}_{[j]}$, with the i-th element:

$$\eta_{[j],i} = \frac{\eta_0}{\sqrt{\hat{u}_{[j],i}} + \epsilon} \tag{2.13}$$

$\hat{u}_{[j],i}$ can be derived in the central server based on $w^F_{[j],i}((j-1)\tau)$. More details will be given in Section 2.5.1.

Similar to [48], we assume in theoretical analysis that the loss function $F(\boldsymbol{w})$ in this research project is *convex*, *ρ-Lipschitz* and *β-smooth*. In this case, with a slight abuse of concepts, we will have:

$$|F(\boldsymbol{w}) - F(\boldsymbol{w'})| \leq \rho ||\boldsymbol{w} - \boldsymbol{w'}|| \tag{2.14a}$$

$$||\boldsymbol{G}(\boldsymbol{w}) - \boldsymbol{G}(\boldsymbol{w'})|| \leq \beta ||\boldsymbol{w} - \boldsymbol{w'}|| \tag{2.14b}$$

where $||\cdot||$ means the $\mathcal{L}^2$ norm. Based on the same model parameter vector $\boldsymbol{w}$, we further define an upper bound δ_k to denote gradient difference derived from the k-th end device and the whole population. In other words, we have:

$$||\boldsymbol{G}_k(\boldsymbol{w}) - \boldsymbol{G}(\boldsymbol{w})|| \leq \delta_k \tag{2.15}$$

with $\delta \triangleq \frac{\sum_{k=1}^{K} N_k \delta_k}{N}$.

2.4.2 Communication Frequency

In communication frequency analysis, we will consider each iteration interval separately. The convergence upper bound compares *FedUR* with centralized learning in each iteration interval. At the beginning of each iteration interval $[j]$, we assume $\boldsymbol{w}_{[j]}((j-1)\tau) \triangleq \boldsymbol{w}^F_{[j]}((j-$

$1)\tau) = \boldsymbol{w}_{[j],k}((j-1)\tau)$ and $\boldsymbol{m}_{[j]}((j-1)\tau) \triangleq \boldsymbol{m}_{[j],k}((j-1)\tau)$ for $\forall j, k$. As has been described in Formula (2.13), since *FedUR* can be seen as having a constant learning rate in each iteration interval, we can decouple momentum and adaptive learning rate.

With the introduction of momentum, we give the following convergence upper bound related to the number of local iterations before a global model parameter aggregation.

Theorem 2.4.1. *For each iteration interval* $[j]$ *and* $t \in [j]$*, under a constant learning rate* η*, we have:*

$$|F(\boldsymbol{w}^F_{[j]}(t)) - F(\boldsymbol{w}_{[j]}(t))| \leq \rho[h_1(t-(j-1)\tau) - h_2(t-(j-1)\tau)] \tag{2.16}$$

where:

$$h_1(x) \triangleq \frac{\delta}{\beta}((\eta\beta+1)^x - 1) - \eta\delta x \tag{2.17a}$$

$$h_2(x) \triangleq \frac{\eta\delta\beta_1[(\eta\beta+1)^x - 1]}{\eta\beta+1-\beta_1} - \frac{\eta^2\delta\beta\beta_1(1-\beta_1^{x-1})}{(\eta\beta+1-\beta_1)(1-\beta_1)} \tag{2.17b}$$

for any $x = 1, 2, \cdots$.

Formula (2.16) is derived based on the ρ-*Lipshitz* assumption described in Formula (2.14a). $h_1(x)$ is the corresponding upper bound derived in *Theorem 1* of [48], which is based on *FedAvg*. $h_2(x)$ shows how much the upper bound can get improved with the introduction of momentum. As has been discussed in [48], when x becomes large, the exponential term $(\eta\beta - 1)^x$ will become dominant. Besides, since $\beta_1 < 1$ by definition, with the increase of x, the second term in Formula (2.17b) will become a constant. In this case, the value of $h_2(x)$ largely depends on its first term. For the first term, at first we can conclude that it is bigger than 0. Furthermore, under a constant learning rate η, the only parameter that can be manually tuned is β_1 in the first term. With the increase of β_1, the numerator of the first term will become larger while its denominator will become smaller, which makes the value of the first term larger, and further increases the value of $h_2(x)$. A formal proof

is detailed in Appendix A.1.

Based on the above observations, we can find that **the introduction of momentum can reduce the upper bound.** In other words, if we set a threshold describing the difference requirement between aggregated model parameters and globally learned model parameters, SGD with momentum can endure more local iterations before global aggregations (larger τ), which reduces necessary communication frequency. Besides, in order to further reduce communication frequency, **a decay rate β_1 very close to 1** should be put on the momentum term in model parameter update, shown in Formula (2.5a).

On the other hand, with the introduction of adaptive learning rate described in Formula (2.13), we also derive the convergence upper bound as follows.

Theorem 2.4.2. *For any interval $[j]$ and $t \in [j]$, under the adaptive learning rate given in Formula* (2.13)*, we have:*

$$|F(\boldsymbol{w}^F_{[j]}(t)) - F(\boldsymbol{w}_{[j]}(t))| \leq \rho h_3(t - (j-1)\tau) \tag{2.18}$$

where:

$$h_3(x) \triangleq \frac{\delta}{\beta}((\bar{\eta}_{[j]}\beta + 1)^x - 1) - \bar{\eta}_{[j]}\delta x \tag{2.19}$$

for any $x = 1, 2, \cdots$.

Here, we use $\bar{\eta}_{[j]}$ to represent a scalar-form learning rate, which can replace all $\eta_{[j],i}$ of Formula (2.13) in this theorem. The definition of $\bar{\eta}_{[j]}$ and relevant proof are provided in Appendix A.2. Based on Formula (2.19), we can see that **the upper bound largely depends on $\bar{\eta}_{[j]}$,** considering the dominant term $(\bar{\eta}_{[j]}\beta + 1)^x$, which further refers to the adaptive learning rate $\eta_{[j],i}$. The adaptiveness of $\eta_{[j],i}$ with accumulated gradients can suppress the effect of β at the beginning of learning process, which is very helpful since gradient changes can be very sharp and induce large $h_3(x)$. Theoretically, we can try to shrink the upper bound through continuously reducing η_0 in $\eta_{[j],i}$. However, a small η_0 will

further induce a small learning rate. As has been discussed in Section 2.3.2, if the learning rate becomes too small, the learning process will slow down or even converge to a local optimum or saddle point. We will analyze in Section 2.5.2 how to balance communication frequency and convergence rate by tuning η_0. In addition, the accuracy of this upper bound depends on the extent to which $\hat{u}_{[j],i}$ can approximate $\hat{u}_i(t)$ in each interval $[j]$. Such approximation is affected by variances of gradients. In this case, we need **a large exponential decay rate β_2, i.e. as close to 1 as possible,** to reduce their impact. Note that our adaptive federated learning algorithm also needs to **meet the convergence requirement in adaptive centralized learning** [15], i.e. $\frac{\beta_1^2}{\sqrt{\beta_2}} < 1$.

2.4.3 Convergence Rate

In the last subsection, we derived two convergence upper bounds to discuss the difference between *FedUR* and centralized learning. With those bounds, we can claim that the two problems discussed in Section 2.3.2 will not cause any divergence in each iteration interval $[j]$. However, the whole learning process cannot be seen as conventional centralized learning, since we assume model parameter synchronization at the beginning of each iteration interval. In this subsection, we will compare this **partial** centralized learning with the global optimal model parameters $\boldsymbol{w}^*$, through which we analyze the convergence rate of our adaptive federated learning algorithm.

According to relevant analysis given in [15], corrections made in Formula (2.7) will generate an expected ratio as follows:

$$|\frac{\mathbb{E}[\hat{m}_i(t)]}{\sqrt{\mathbb{E}[\hat{u}_i(t)]}}| = |\frac{\mathbb{E}[G_i(t)]}{\sqrt{\mathbb{E}[G_i^2(t)]}}| \leq 1 \tag{2.20}$$

Such an approximation will become more accurate with gradients getting more stable. Considering the above formula together with Formula (2.4) and (2.5b), we can further derive that in the centralized view, the step size of model parameter update largely depends on

the predefined global learning rate η_0. In other words, in each iteration interval $[j]$, the equivalent partial centralized learning will have:

$$|w_{[j],i}(t+1) - w_{[j],i}(t)| \leq \eta_0 \tag{2.21}$$

From this formula, we can see that the whole learning process will finally converge as long as η_0 is bounded.

When we investigate the convergence rate analysis in [48], we will notice that most of the related proof can be inherited in this research project, which is based on convexity properties of loss functions. The main difference caused by the adaptiveness of our algorithm is in Lemma 5 of [48]. When the loss function $F(\boldsymbol{w})$ is β-smooth, according to Lemma 3.4 in [64] and Formula (2.21), for each iteration interval $[j]$ and an I-dimensional model parameter vector, we can have:

$$\begin{aligned}
&F(\boldsymbol{w}_{[j]}(t+1)) - F(\boldsymbol{w}_{[j]}(t)) \\
\leq &\boldsymbol{G}_{[j]}^T(t)(\boldsymbol{w}_{[j]}(t+1) - \boldsymbol{w}_{[j]}(t)) + \frac{\beta}{2}||\boldsymbol{w}_{[j]}(t+1) - \boldsymbol{w}_{[j]}(t)||^2 \\
\leq &\eta_0 \sum_{i=1}^{I} G_{[j],i}(t) + \frac{\beta I \eta_0^2}{2}
\end{aligned} \tag{2.22}$$

This upper bound is the basis of analyzing convergence rate in [48] for *FedAvg* (Formula (26) of [48]). The smaller this upper bound is, the higher convergence rate we can achieve. Let's denote this upper bound in *FedAvg* and *FedUR* as $\theta_{1,[j]}(t)$ and $\theta_{2,[j]}(t)$, respectively. Then, with further processing, we will have:

$$\theta_{1,[j]}(t) = \frac{\beta \sum_{i=1}^{I} G_{[j],i}^2(t)}{2}(\eta - \frac{1}{\beta})^2 - \frac{\sum_{i=1}^{I} G_{[j],i}^2(t)}{2\beta} \tag{2.23a}$$

$$\theta_{2,[j]}(t) = \frac{\beta I}{2}(\eta_0 + \frac{\sum_{i=1}^{I} G_{[j],i}(t)}{\beta I})^2 - \frac{(\sum_{i=1}^{I} G_{[j],i}(t))^2}{2\beta I} \tag{2.23b}$$

Since the root-mean square is greater than or equal to arithmetic mean, we should have $\min_\eta \theta_{1,[j]}(t) \leq \min_{\eta_0} \theta_{2,[j]}(t)$. However, based on the above two equations, the real achieved upper bound largely depends on η and η_0, which are pre-defined and fixed. Furthermore, β will become a dynamic hyper-parameter and highly related to gradients if it is considered separately in each communication round. In this case, $-\frac{\sum_{i=1}^{I} G_{[j],i}(t)}{\beta I}$ will be more stable than $\frac{1}{\beta}$, and η_0 will be more likely to perform consistently than η.

If we define a ratio $\gamma \triangleq \theta_{2,[j]}(t)/\theta_{1,[j]}(t)$, then based on Lemma 2 in [48], we can derive the following convergence rate upper bound:

Theorem 2.4.3. *If all the following assumptions hold under a given $\varepsilon > 0$ and a given total number of local iterations T:*

1. *η_0 is bounded*
2. *$\phi - \frac{\rho h(\tau)}{\tau \varepsilon^2} > 0$*
3. *$F(\boldsymbol{w}_{[j]}(t)) - F(\boldsymbol{w}^*) \geq \varepsilon$ for any j*
4. *$F(\boldsymbol{w}(T)) - F(\boldsymbol{w}^*) \geq \varepsilon$*

where we define $\phi \triangleq \gamma\eta\omega(1 - \frac{\beta\eta}{2})$ and $\omega \triangleq \min_j \frac{1}{||\boldsymbol{w}_{[j]}((j-1)\tau) - \boldsymbol{w}^*||}$, then we have:

$$F(\boldsymbol{w}(T)) - F(\boldsymbol{w}^*) \leq \frac{1}{T(\phi - \frac{\rho h(\tau)}{\tau \varepsilon^2})} \tag{2.24}$$

Here, η represents the learning rate in *FedAvg*. $h(\tau) \triangleq h_1(\tau) - h_2(\tau)$ with the learning rate η replaced by $\bar{\eta}_{[j]}$. Compared with *FedAvg*, this upper bound improvement is determined by γ and $h(\tau)$. γ depends on $\theta_{1,[j]}(t)$ and $\theta_{2,[j]}(t)$, and $\theta_{2,[j]}(t)$ is more likely to approach the minimum during the whole learning process. As has been discussed in Section 2.4.2, $h(\tau)$ will be reduced with the introduction of momentum and adaptive learning rate,

which further decreases the upper bound. In general, this upper bound is more likely to get depressed under *FedUR*. In other words, compared with *FedAvg*, *FedUR* needs a smaller T to reach the same convergence accuracy.

2.5 Application Issues

Up to now, we have in general constructed our adaptive federated learning model and theoretically analyzed its communication overhead improvement from the perspective of both communication frequency and convergence rate. During our algorithm deployment, two hyper-parameters still need to be fixed in each iteration interval $[j]$, i.e. the expected global bias-corrected second raw moment estimate $\hat{\boldsymbol{u}}_{[j]}$ and the global learning rate η_0.

2.5.1 Computation of $\hat{\boldsymbol{u}}_{[j]}$

Typically, the computation of those moment estimates needs gradient information. In *FedUR*, we can derive an expected $\hat{\boldsymbol{u}}_{[j]}$ in each iteration interval $[j]$ only based on the aggregated global model parameter vector $\boldsymbol{w}^F_{[j]}((j-1)\tau)$. The general idea is as follows. Formula (2.5a) gives us the way to derive the global gradient $G_i(t)$ from the biased first raw moment estimate in two consecutive iterations. In each iteration t, we can at first infer $m_i(t)$ from $\hat{m}_i(t)$ based on Formula (2.7). Since we can derive the relationship described by Formula (2.25), we can get $\hat{m}_i(t)$ through $\hat{m}_{k,i}(t)$.

$$\hat{m}_i(t) = \frac{1}{N}\sum_{k=1}^{K} N_k \hat{m}_{k,i}(t) \tag{2.25}$$

Finally, we can calculate $\hat{m}_{k,i}(t)$ via the local model parameter update rule described by Formula (2.26), in which $w_i^F(t)$ was already derived in the last global aggregation, and

$w_{k,i}(t+1)$ is the uploaded local model parameter.

$$w_{k,i}(t+1) \leftarrow w_i^F(t) - \eta_0 \hat{m}_{k,i}(t) \tag{2.26}$$

When it comes to the situation that each communication round has multiple local iterations, we need to compute the expected $\hat{u}_{[j],i}$ in each iteration interval $[j]$ shown in Formula (2.13). Suppose each communication round includes τ local iterations. We at first modify Formula (2.26), and calculate the corresponding expected $\hat{m}_{[j],k,i}$ as follows:

$$\hat{m}_{[j],k,i} = \frac{w_{[j],i}^F((j-1)\tau) - w_{[j],k,i}(j\tau)}{\eta_0 \tau} \tag{2.27}$$

Afterwards, we can directly use Formula (2.25) and (2.7) to compute the expected $\hat{m}_{[j],i}$ and $m_{[j],i}$. t in Formula (2.7) will be defined as $(j-1)\tau + \tau/2$. Finally, we will modify Formula (2.5a) as follows to derive gradient information, in which $m_i(t)$ will be replaced with $m_{[j],i}$:

$$G_{[j],i} = \frac{m_{[j],i} - \beta_1 m_{[j],i}((j-1)\tau)}{1-\beta_1} \tag{2.28}$$

Note that $G_{[j],i}$ is the expected accumulated gradient within the interval $[j]$, which we can directly use to compute $\hat{u}_{[j],i}$ based on Formula (2.6) and (2.7).

2.5.2 Computation of η_0

Based on Theorem 2.4.2, we know that η_0 should be set as small as possible to achieve a low communication frequency. In Section 2.4.3, we further notice that the convergence rate depends on $\theta_{2,[j]}(t)$, which reaches the minimum when $\eta_0 = -\frac{\sum_{i=1}^{I} G_{[j],i}(t)}{\beta I}$. In this case, there will be a trade-off between communication frequency and convergence rate from the perspective of η_0. During application, η_0 will be updated in the central server after

each model parameter aggregation. In this case, we can generate the following optimization problem to derive the best η_0:

$$\arg\min_{\eta_0} \zeta(\eta_0) = \alpha\theta_{2,[j]}(j\tau) + (1-\alpha)\eta_0 \tag{2.29}$$

where $\alpha \in (0, 1]$ is a hyper-parameter to balance the above two factors. α cannot be 0 since we need to guarantee that the whole learning process will finally converge. $\boldsymbol{G}_{[j]}(j\tau)$ will be directly computed based on the newly aggregated global model parameter vector $\boldsymbol{W}^F(j\tau)$. With a slight modification of $\zeta(\eta_0)$, we get:

$$\zeta(\eta_0) = \frac{\alpha\beta I}{2}[\eta_0 + \frac{\alpha\sum_{i=1}^{I} G_{[j],i}(j\tau) + (1-\alpha)}{\alpha\beta I}]^2 - \frac{[\alpha\sum_{i=1}^{I} G_{[j],i}(j\tau) + (1-\alpha)]^2}{2\alpha\beta I} \tag{2.30}$$

where we can derive the best η_0 as:

$$\eta_0 = -\frac{\alpha\sum_{i=1}^{I} G_{[j],i}(j\tau) + (1-\alpha)}{\alpha\beta I} \tag{2.31}$$

Note that the above formula can be negative. When this situation happens, we will set η_0 to 0.001 to ensure proper convergence. Besides, we also need to set an upper bound towards η_0 to satisfy the first assumption in Theorem 2.4.3, which is $\eta_0 \leq 0.1$ in this research project.

Now, we can give our adaptive federated learning algorithm, which is summarized in Algorithm 2. In the algorithm design, we assume that the number of local iterations τ is given in each communication round, which can be optimized under given resource constraints based on the control algorithm in [48]. In Section 2.6.3, we will further compare our algorithm with *FedAvg* and *FedAdam* under different τ to show its advantages in saving communication overhead. Besides, from Formula (2.31), we can see that η_0 is related to β, which will be estimated in each communication round. The global estimation is an aggregated result of all from selected end devices. To get distinguished from exponential

Algorithm 2: *FedUR*

Input : N: global batch size. K: number of selected end device. N_k: local batch size, $k = 1, 2, \cdots, K$. τ: number of local iterations. α: trade-off weight. β_1, β_2: decay rate. ϵ: small constant.

Output: $\boldsymbol{w}^{F^*}$: best aggregated global model parameters.

Initialize $t \leftarrow 0$, $\boldsymbol{w}^F(0) \leftarrow$ random seed, $\boldsymbol{w}^{F^*} \leftarrow \boldsymbol{w}^F(0)$ $\boldsymbol{m}_{[0]} \leftarrow \boldsymbol{0}$, $\boldsymbol{u}_{[0]} \leftarrow \boldsymbol{0}$, $\eta_0 \leftarrow 0.001$;

Initialize $\boldsymbol{m}_k(0) \leftarrow \boldsymbol{0}$, $\boldsymbol{u}_k(0) \leftarrow \boldsymbol{0}$ for $k = 1, 2, \cdots, K$;

for $j = 0, 1, 2, \cdots$ **do**

  $S \leftarrow$ random set of K end devices;

  for *each end device* k *in* S **in parallel do**

    $(\boldsymbol{w}_k((j+1)\tau), \hat{\beta}_k) \leftarrow$ `LocalUpdate(`$\boldsymbol{w}^F(j\tau), \eta_0, k$`)`;

  Compute $\hat{\boldsymbol{m}}_{[j+1],k}$ based on Formula (2.27) for each end device k, derive the aggregated $\hat{\boldsymbol{m}}_{[j+1]}$ via Formula (2.25), and estimate $\boldsymbol{m}_{[j+1]}$ through Formula (2.7);

  Compute $\boldsymbol{G}_{[j+1]}$ based on Formula (2.28);

  Update $\boldsymbol{u}_{[j+1]} \leftarrow \beta_2 \boldsymbol{u}_{[j]} + (1-\beta_2)\boldsymbol{G}_{[j+1]}$, and compute $\hat{\boldsymbol{u}}_{[j+1]}$ based on Formula (2.7);

  Compute $\boldsymbol{w}^F((j+1)\tau)$ based on Formula (2.12);

  if $F(\boldsymbol{w}^F((j+1)\tau)) < F(\boldsymbol{w}^{F^*})$ **then**

    $\boldsymbol{w}^{F^*} \leftarrow \boldsymbol{w}^F((j+1)\tau)$

  if $F(\boldsymbol{w}^{F^*})$ *is less than a threshold* **then**

    break ; `//Learning process ends.`

  Estimate $\hat{\beta} \leftarrow \frac{1}{N}\sum_{k=1}^{K} N_k \hat{\beta}_k$;

  Update η_0 based on Formula (2.31);

Function `LocalUpdate(`$\boldsymbol{w}, \eta_0, k$`)`**:**

  $\boldsymbol{w}_k(t) \leftarrow \boldsymbol{w}$;

  Compute the gradient under $\boldsymbol{w}$, denoted as $\boldsymbol{G}(\boldsymbol{w})$;

  for *each local iteration* t **do**

    $t \leftarrow t + 1$;

    $S_L(t) \leftarrow$ random set of N_k data samples;

    Compute the gradient $\boldsymbol{g}_{\boldsymbol{d}}(t)$ for each selected data sample $\boldsymbol{d} \in S_L(t)$;

    Aggregate $\boldsymbol{G}_k(t) \leftarrow \frac{1}{N_k}\sum_{\boldsymbol{d} \in S_L(t)} \boldsymbol{g}_{\boldsymbol{d}}(t)$;

    $\boldsymbol{m}_k(t) \leftarrow \beta_1 \boldsymbol{m}_k(t-1) + (1-\beta_1)\boldsymbol{G}_k(t)$;

    $\hat{\boldsymbol{m}}_k(t) \leftarrow \boldsymbol{m}_k(t)/(1-\beta_1^t)$;

    $\boldsymbol{w}_k(t) \leftarrow \boldsymbol{w}_k(t-1) - \eta_0 \hat{\boldsymbol{m}}_k(t)$;

    if $t \mod \tau = 0$ **then**

      $\hat{\beta}_k \leftarrow ||\boldsymbol{G}_k(t) - \boldsymbol{G}(\boldsymbol{w})||/||\boldsymbol{w}_k(t) - \boldsymbol{w}||$;

      return $(\boldsymbol{w}_k(t), \hat{\beta}_k)$;

decay rates, we denote these estimations as $\hat{\beta}$ and $\hat{\beta}_k$ here.

2.6 Experimental Results

2.6.1 Experimental Settings

In this section, we will make comprehensive comparison among our proposed *FedUR*, *FedAvg*, and *FedAdam* in terms of final convergence accuracy, communication frequency and convergence rate to validate the improvement in convergence performance and communication overhead. The reason for considering *FedAdam* as a representative of existing adaptive federated learning algorithms is that according to the experimental results in [17], *FedAdam* in general achieves the best convergence performance. All the algorithms are evaluated based on three machine learning models, i.e. logistic regression, multi-layer neural network, and convolutional neural network (CNN). The loss function is generally based on cross entropy with difference on non-linear transforms. Logistic regression and multi-layer neural networks consider *sigmoid* function, while *REctified Linear Unit* (*ReLU*) function is adopted in CNN. We conduct simulations based on two real-world datasets, i.e. MNIST [65] and CIFAR-10 [66].

In detail, an *L2* regularized multi-class logistic regression is applied here. The multi-layer neural network used in this research project has an input layer, a hidden layer and an output layer, which are all densely connected. The hidden layer totally has 1,000 units, and the output layer has a *softmax* activation. For CNN, we applied a network with two convolution layers, two max pooling layers, a dense layer and an output layer. Therein, the first and second convolution layer respectively have 32 and 64 channels. Both of these two layers utilize a 5×5 kernal, a 1×1 stride and a *ReLU* activation. The two max pooling layers both conduct a 2×2 pool size. The dense layer has 1,000 units, concatenated by a *softmax* output layer.

For those hyper-parameters, we initialize $\eta_0 \leftarrow 0.001$. Besides, we have $\beta_1 \leftarrow 0.9$ and $\beta_2 \leftarrow 0.999$ to make them as close to 1 as possible and meet the requirement $\frac{\beta_1^2}{\sqrt{\beta_2}} < 1$.

The small constant ϵ is set to 10^{-7}. In *FedAvg*, the static learning rate η is set to 0.01. In *FedAdam*, the global learning rate η_0 is always set to 0.001.

For the federated learning setting in this research project, we consider synchronous model parameter aggregation under data statistical heterogeneity. In other words, datasets across all end devices are unbalanced and non-IID. The MNIST dataset is composed of 70,000 gray-scale images of handwritten digits (60,000 for training and 10,000 for test), while the CIFAR-10 dataset contains 50,000 color images for training and 10,000 test color images. These two datasets both have 10 classes. In this case, we at first sort training images according to those classes, and then partition them to 100 shards, which can be seen as a scenario that 100 end devices participate in a federated learning task. Since our adaptive federated learning algorithm is evaluated from communication frequency and convergence rate, the whole training process will stop after 1,000 local iterations instead of a certain number of communication rounds. For comprehensive comparison, we consider the scenario with 10, 20, and 50 selected end devices in each communication round, with each selected end device randomly picking up 10 data samples (batch size) in each local iteration.

2.6.2 Convergence Accuracy

In this subsection, we compare our adaptive federated learning algorithm with two baselines, i.e. *FedAvg* and *FedAdam* on convergence performance. The final convergence accuracy in terms of test sets are considered as the metric. Note that we fix the trade-off weight α to 0.5 in this subsection, and leave the discussion about it in the Section 2.6.4.

Fig. 2.2, 2.3, and 2.4 respectively show convergence performance comparison under logistic regression, multi-layer neural network, and CNN. Therein, logistic regression and multi-layer neural network are both based on the MNIST dataset, while CNN is analyzed under the CIFAR-10 dataset. For *FedAvg* and *FedAdam*, we consider the highest communication frequency in the evaluation, i.e. $\tau = 5$. Our adaptive federated learning scheme further evaluates the scenario with $\tau = 10$ and $\tau = 20$. Since the learning process needs to

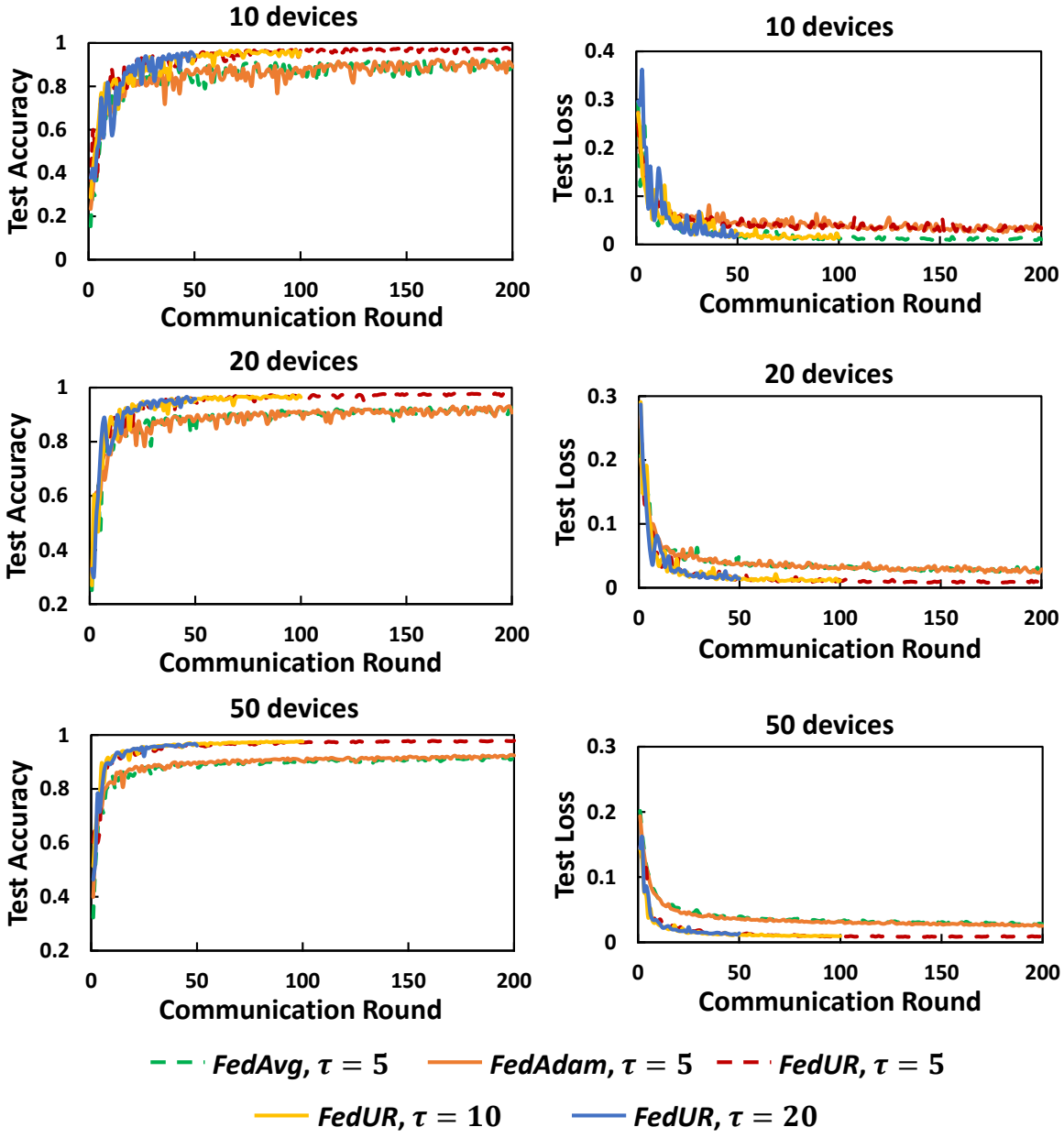


Figure 2.2: Comparison results for logistic regression

stop after 1,000 local iterations, $\tau = 5$, $\tau = 10$, and $\tau = 20$ respectively infer 200, 100, and 50 communication rounds. Note that the x-axes in all these figures are communication round. Both accuracy and loss are illustrated based on the test sets in those two datasets

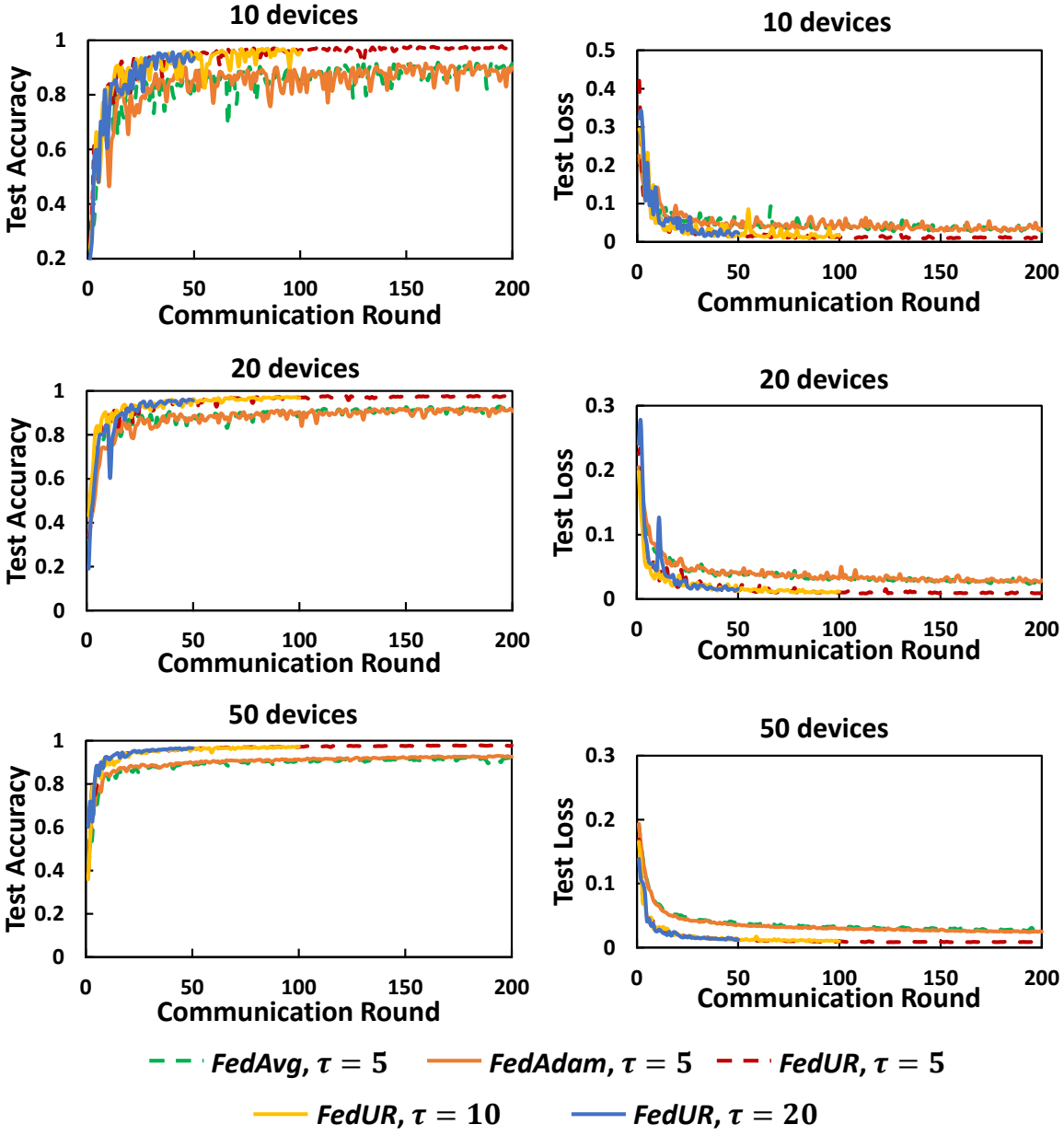


Figure 2.3: Comparison results for multi-layer neural network

and the aggregated global model parameters after each communication round.

From Fig. 2.2, we can see that generally all of the three algorithms will finally converge to over 90% of test accuracy. Besides, with more end devices selected in each communication

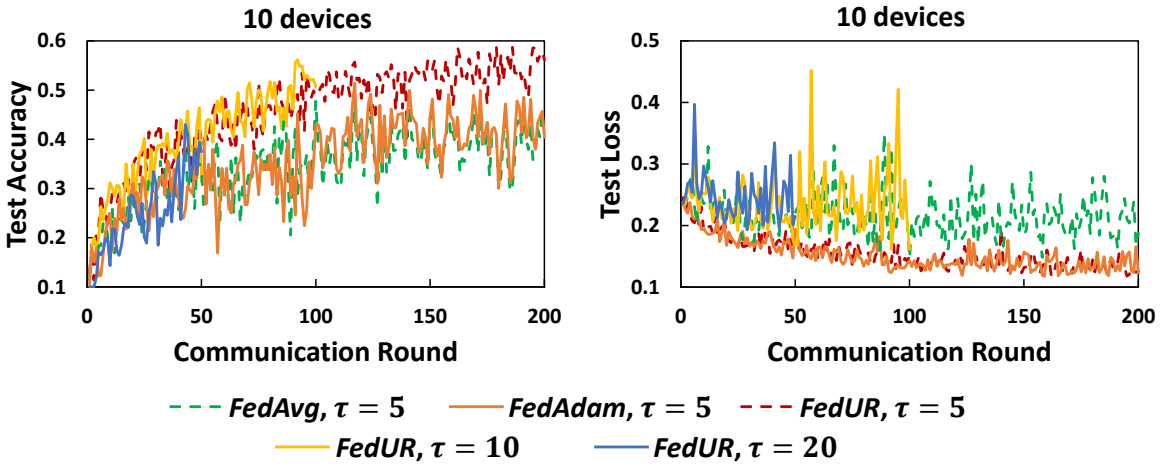


Figure 2.4: Comparison results for CNN

round, more stable convergence process will be achieved. On the other hand, our proposed *FedUR* can achieve much higher test accuracy than *FedAvg* and *FedAdam*, which is due to the equivalence reconstruction to centralized learning in our algorithm. Furthermore, *FedUR* can achieve similar convergence performance under all the three evaluated communication frequencies, which infers the robustness of our proposed algorithm.

Similar trends can also be found in the other two models, i.e. multi-layer neural network and CNN, which are shown in Fig. 2.3 and 2.4. Note that we only illustrate the scenario with 10 selected end devices in each communication round for CNN. We further list the achieved best test accuracy of all the three algorithms under $\tau = 5$ in Table 2.2. Besides CNN, LR and MLP respectively refer to logistic regression and multi-layer neural network. From this table, we can see that in all cases, our proposed algorithm has the best performance. In general, compared with *FedAdam*, our *FedUR* can achieve a test accuracy improvement of 5.5%, 5.8%, and 17.4% in logistic regression, multi-layer neural network, and CNN.

2.6.3 Communication Frequency and Convergence Rate

We further make communication frequency and convergence rate comparison here. Our general idea is that if our proposed adaptive federated learning algorithm can achieve a

Table 2.2: Best Achieved Test Accuracy

		FedAvg	*FedAdam*	*FedUR*
MNIST 10 devices	LR	0.9244	0.9263	0.9782
	MLP	0.9198	0.9194	0.9784
MNIST 20 devices	LR	0.9281	0.9311	0.9787
	MLP	0.9294	0.9258	0.9774
MNIST 50 devices	LR	0.9195	0.9264	0.9792
	MLP	0.9211	0.9302	0.9798
CIFAR-10	CNN	0.4828	0.5142	0.6038

similar convergence rate and accuracy to those two baselines under a lower communication frequency or a larger τ, the overall communication overhead is definitely saved. From the previous subsection, we already know that even if we enlarge τ from 5 to 20, our proposed algorithm still outperforms the two baselines in the final convergence accuracy. In this subsection, we will further compare the convergence rate under different communication frequencies.

Similar to the previous subsection, the two baselines still adopt the most frequent communication, i.e. $\tau = 5$. Besides, α is still fixed to 0.5 here. We compare the number of necessary communication rounds among *FedAvg*, *FedAdam*, and our adaptive federated learning to reach an 80% test accuracy for the MNIST dataset or 30% test accuracy for the CIFAR-10 dataset. Some typical experimental results are shown in Fig. 2.5. In order to clearly reflect the trend, we applied a 5th-order one-dimensional mean filter towards raw experimental results. From this figure, we can see that for logistic regression and multi-layer neural network, our proposed algorithm achieves the highest convergence rate under all the three different communication frequencies. For CNN, our method performs the best under $\tau = 5$ and $\tau = 10$, and has similar performance when $\tau = 20$.

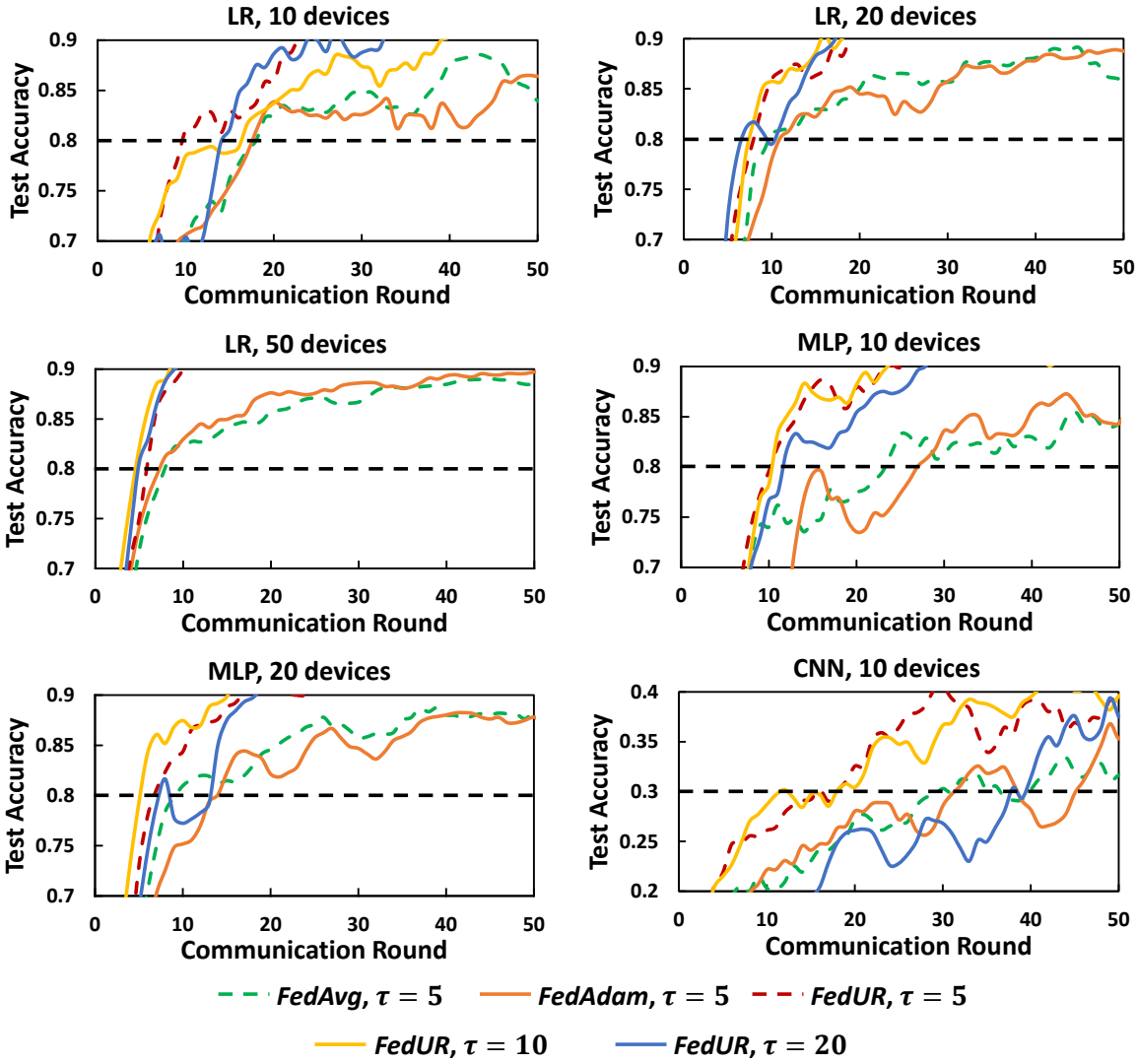


Figure 2.5: Comparison result among different communication frequencies

We further list in Table 2.3 the specific number of necessary communication rounds to reach 80% or 30% test accuracy for each case in Fig. 2.5. Therein, LR-10 refers to logistic regression plus 10 selected devices in each communication round, and so on. $\tau = 5$, $\tau = 10$, and $\tau = 20$ are all under our proposed algorithm. In general, compared with *FedAdam*, our method can respectively increase the convergence rate by 47.2%, 41.0%, and 21.9% under $\tau = 5$, $\tau = 10$, and $\tau = 20$.

In general, *FedUR* can achieve better final convergence accuracies and higher convergence rates even under lower communication frequencies, which will greatly reduce communication overhead during the whole learning process.

Table 2.3: Number of Necessary Communication Rounds to Reach 80% (MNIST) or 30% (CIFAR-10) Test Accuracy

	FedAvg	*FedAdam*	$\tau = 5$	$\tau = 10$	$\tau = 20$
LR-10	18	18	10	17	16
LR-20	10	11	8	8	7
LR-50	8	8	6	5	5
MLP-10	24	27	11	11	12
MLP-20	10	14	7	6	8
CNN-10	32	32	16	18	38

2.6.4 Learning Process Optimization

The value of the hyper-parameter α in Formula (2.31) will determine η_0, and further impact the whole convergence performance. In this subsection, we compare the convergence performance of our adaptive federated learning algorithm under $\alpha = 0.1$, 0.3, 0.5, 0.7, and 0.9. Both logistic regression and multi-layer neural network are considered here.

Fig. 2.6 compares those scenarios under different communication frequencies. 50 end devices are randomly selected in each communication round. From this figure, we can see that our algorithm performs similarly under different α and logistic regression, while some difference can be noticed for multi-layer neural network. In detail, there is an obvious performance degradation when $\tau = 20$ and $\alpha = 0.1$. Based on the theoretical analysis in Section 2.5.2, we know that a small α means that much emphasis is put on communication frequency, which will induce a small η_0. A small η_0 will further cause a small learning rate. If the learning rate is too small, the convergence will slow down or even stop unexpected in a local optimal or saddle point. Such an impact seems to get exaggerated with the

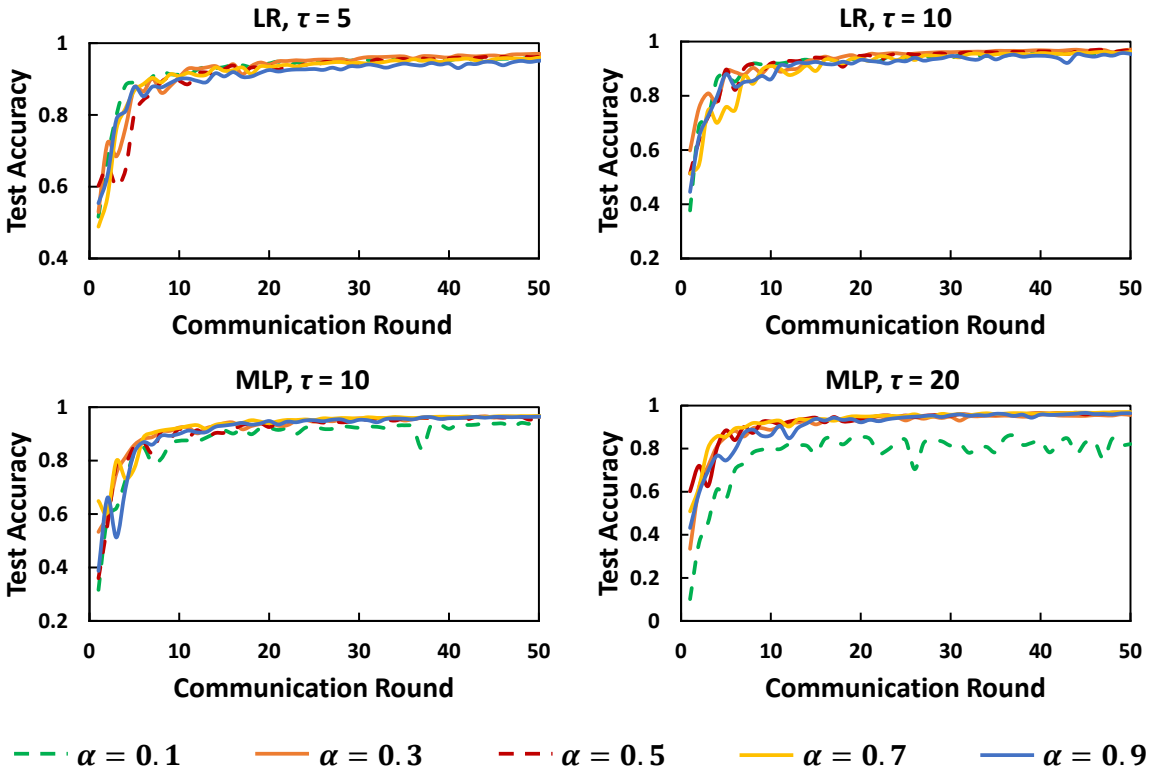


Figure 2.6: Convergence performance with different communication frequencies

decrease of communication frequency. Furthermore, the complexity of a machine learning model also plays a role therein. On the other hand, the convergence is less stable when $\alpha = 0.9$, especially at the beginning of the whole learning process. A too large α will induce uncertainty to local iterations, and further cause oscillations in the whole learning process, especially at the beginning.

We also evaluate convergence performance of our algorithm under different numbers of selected end devices in each communication round, which is illustrated in Fig. 2.7. τ is set to 5 here. From this figure, we can see that the negative impact of a too small α is not significant, which is probably due to frequent communication, i.e. $\tau = 5$. On the other hand, oscillations caused by a too large α is still noticeable in the whole learning process.

The value of η_0 in each communication round is shown in Fig. 2.8 with $\tau = 5$. All the three machine learning models are considered here. In Formula (2.31), we know that both

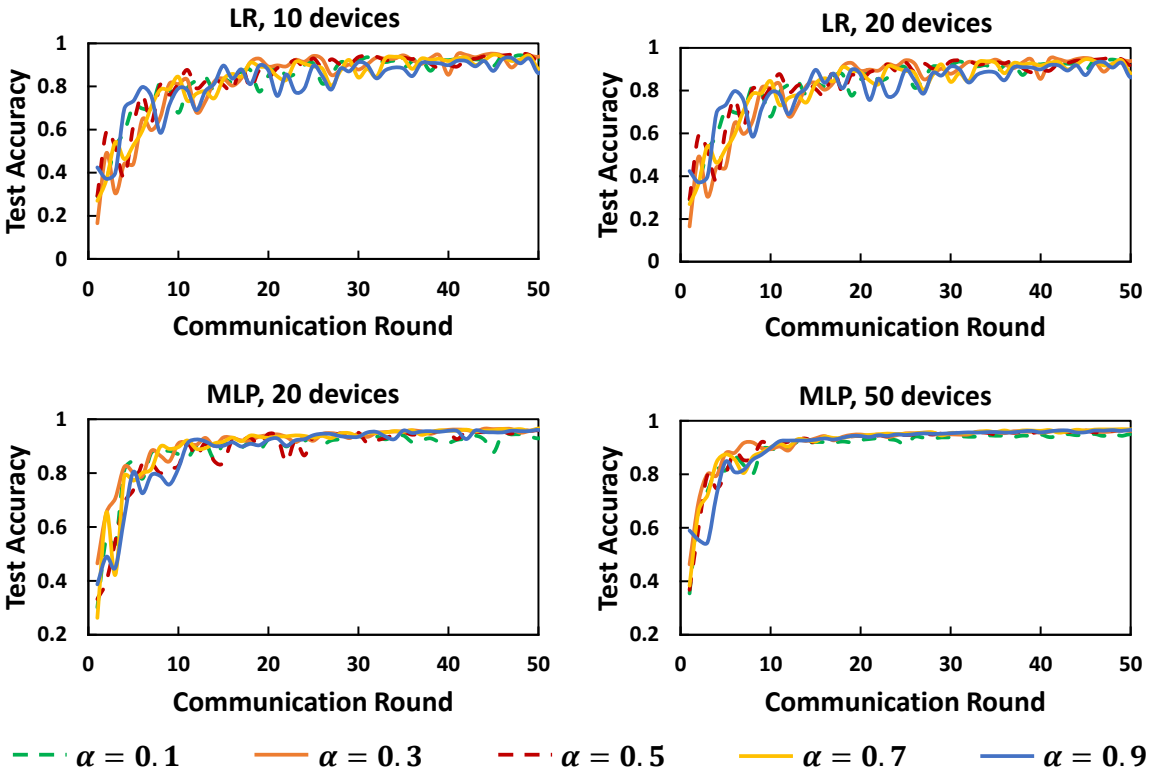


Figure 2.7: Convergence performance with different numbers of selected end devices in each communication round

gradients and β should decrease as the learning process goes. These two trends may get cancelled when we consider η_0. That is the main reason why the *Adam* optimizer adopts a constant global learning rate η_0. However, based on Fig. 2.8, we observe that η_0 should have an increase trend at the beginning, and then gradually flattens, when applied in federated learning. In addition, η_0 should be set differently under different machine learning models and numbers of selected end devices in each communication round. Actually, many other factors in federated learning, such as communication frequency, can also cause difference in the η_0 setting, which will not be shown here due to the page limit.

In Section 2.4.3, we defined a ratio γ as a metric to evaluate the convergence rate improvement of our adaptive federated learning algorithm compared with *FedAvg*. In Fig. 2.9, we illustrate the value of this ratio under $\tau = 5$, and in different machine learning

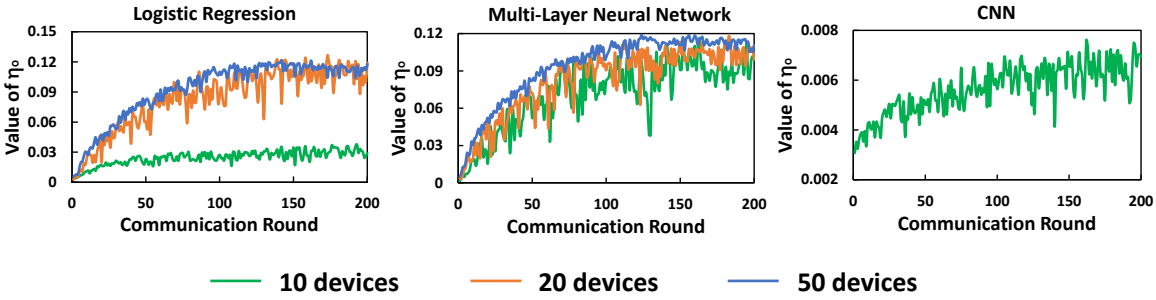


Figure 2.8: Value of η_0 with $\tau = 5$, different machine learning models, and different numbers of selected end devices

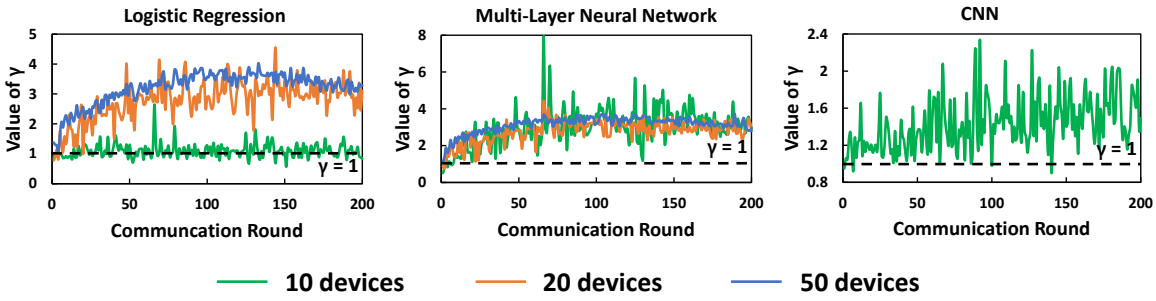


Figure 2.9: Value of γ with $\tau = 5$, different machine learning models, and different numbers of selected end devices

models plus selected end devices in each communication round. We also draw a dotted line in this figure to illustrate $\gamma = 1$. We can easily figure out that the value of γ is greater than 1 in most situations, which means that our proposed algorithm can generally outperform *FedAvg* in convergence rate. Besides, we also notice that γ is close to 1 at the beginning, and increases afterwards, which further validates our discussion in Section 2.4.3 that η_0 is more likely to perform consistently than η.

2.7 Future Directions on Federated Learning Optimization

2.7.1 Federated Learning Optimization Scheme Integration

System and data statistical heterogeneity are widely considered when it comes to convergence performance optimization in federated learning. For system heterogeneity, it is usually

about end device synchronization [5–7]. As to data statistical heterogeneity, existing works have considered battling unbalanced and non-IID data distributions across different end devices through end device selection (*FAVOR* [8] and *FOLB* [9]), batch size optimization (*CABS* [58], *BA-SGD* [59], and *Adaptive-B* [10]), and cross-client variance reduction (*FedProx* [11]), *VRL-SGD* [12], and *SCAFFOLD* [13]). Our proposed algorithm is orthogonal to all the above federated learning optimization strategies, which are developed based on the original federated learning or *FedAvg*. A proper integration with those strategies can further improve convergence performance.

If we further expand the scope to communication overhead optimization, uploaded model parameter compression has recently been considered [40,41,43]. Therein, a quantized version of local model parameters are uploaded to the central server for aggregation, which will bring new challenges to central servers on deriving global momentum and adaptive learning rate. In addition, communication management schemes, such as [48], can be integrated into our algorithm to further optimize communication between end devices and central servers if resource constraints exist.

2.7.2 Tradeoff among Convergence Performance, Communication Overhead, and Privacy Protection Level

Recent works have shown that uploaded model parameters in federated learning are still vulnerable to inference attack [18–21]. An even higher privacy-preserving level can be reached by protecting uploaded model parameters based on secret sharing [22, 23], secure multi-party computation [24, 25], or differential privacy [26, 67]. However, the first two strategies can further increase communication overhead if local model parameters need to be separated into several pieces and uploaded to different central servers. The recently-proposed mobile edge computing can relieve such a concern by offloading model aggregation tasks from remote cloud servers to nearby edge nodes. On the other hand, since edge nodes usually have limited computation resources, a proper task scheduling strategy is necessary to balance workload in different edge nodes and optimize communication among them. In

Chapter 3 and 4, we respectively propose two relevant strategies, which can additionally meet the privacy requirements to protect uploaded model parameters. Federated learning optimization strategies targeting system heterogeneity can also be considered here.

Differential privacy masks uploaded model parameters through adding random Gaussian noise. Therefore, there will be tradeoff between the privacy protection level and convergence performance [27], which can further influence communication overhead. The design of differential privacy policies still remains largely open in federated learning.

2.8 Summary

In this research project, we proposed a federated learning optimization algorithm from the aspect of model parameter update rule, i.e. *FedUR*. The algorithm design is based on a full investigation about the impact of momentum and adaptive learning rate on federated learning. Although using accumulated local gradients can cause the update rule bias when each communication round has multiple local iterations, such bias will not impact the overall convergence trend, and can be further alleviated by tweaking the above two terms. We came up with the convergence upper bound under *FedUR* after each communication round and a given number of communication rounds. Through comparison with those upper bounds in *FedAvg*, we theoretically analyzed how to tune those two terms to minimize communication overhead through reducing communication frequency and accelerating learning process. Extensive experiments validated our theoretical analysis about convergence performance. In general, compared with existing adaptive federated learning algorithms, our proposed strategy can increase final convergence accuracy by 5.5-17.4%, and accelerate the whole learning process by 21.9-47.2% even under less frequent communication.

Chapter 3: Pairwise Markov Chain: A Privacy-Preserving Task Scheduling Strategy in Mobile Edge Computing

This chapter discusses the second detailed research problem in Section 1.2. The extra communication burden brought by Secret Sharing (SS) and secure Multi-Party Computation (MPC) can be greatly relieved by the introduction of mobile edge computing. On the other hand, since edge servers are usually assumed to have limited resources [29, 30, 48, 68], we need to further consider computation latency optimization by balancing tasks among those servers. This research project is related to the publication [69].

3.1 Introduction

As has been mentioned in Section 2.7.2, in federated learning, sensitive information can still get leaked from those shared model parameters or gradients via inference attacks [18–21]. In this case, follow-up works consider putting extra privacy masks to locally-learned model parameters uploaded to public servers. Currently, MPC [22, 28] and Differential Privacy (DP) [67,70] are usually considered. Therein, the design of DP needs to balance the privacy protection level with convergence performance, which is out of the scope of this dissertation. When it comes to MPC, SS [22] and Homomorphic Encryption (HE) [28] are two widely considered strategies. Compared with HE, SS has lower computation complexity and higher robustness to user dropouts, which make it more suitable to scenarios with mobile local devices, such as smart phones and tablets.

SS has been considered for not a short time in many multimedia applications. In computer vision, image matching is based on image features extracted through algorithms with very high computational requirements, such as Scale-Invariant Feature Transform (SIFT) [71]. People have to rely on cloud computing when the number of images to be processed

is large. However, some images, such as profiles and medical images, have sensitive contents that are not supposed to go public. In this case, when images are uploaded to clouds for SIFT feature extraction, image owners do not want to reveal the image content to the cloud server. In recent years, many related works have been done for privacy-preserving SIFT [72–74]. These works usually realize SIFT feature extraction in the encrypted domain through homomorphic addition, multiplication and comparison. Later in [75, 76], images are randomly split into two portions and transmitted to two clouds for SIFT processing, which can actually be seen as a (2,2) SS. When it comes to federated learning, those locally-learned model parameters and model aggregation tasks respectively correspond to images and image matching tasks.

However, MPC usually suffers from high communication overhead. For a data processing job of aggregating M secrets, at least $O(M)$ of communication overhead will be induced between public servers and local devices. Besides, with the introduction of SS to combat user dropouts, an extra $O(ML)$ communication overhead will be caused among local devices if each secret is separated to L shares. This high communication overhead poses great challenges in communication latency and transmission power management. Recently, mobile edge computing [29, 68] is proposed to replace conventional cloud computing, which can effectively lower communication overhead by offloading some computation tasks to nearby edge nodes instead of remote cloud servers.

Unfortunately, the bridge between SS, MPC and mobile edge computing is still not built up. Most of the current MPC schemes are still considered in a cloud computing setting, in which public servers are assumed to have unlimited computation and storage capacity, and mobile local devices directly communicate with those cloud servers. In this situation, communication overhead only happens between public servers and mobile local devices, and is usually reduced through algorithm design or data compression. When it comes to the deployment in mobile edge computing, communication between the edge and mobile local devices can be omitted considering edge servers being nearby. On the other hand, edge nodes are distributed across the whole edge network, and each edge node is usually believed to

have limited computation and storage capacity. In this case, relevant computation offloading strategies among those edge nodes become critical in determining achieved communication and computation latency or transmission power improvement. Currently, tons of works [77–82] have already proposed such strategies from different aspects. However, none of them considered privacy constraints during task scheduling. In MPC based on SS, we should avoid a certain number of shares of the same secret being stored in the same edge server. Fig. 3.1 shows a toy example of deploying a (2,2) SS in mobile edge computing. In this example, an end user splits information (derived local model parameters in federated learning) into two secret shares, then uploads them to two edge nodes (Edge node 1 and 2) for further processing (model aggregation). If either of these two edge nodes decides to switch the received secret shares to other nodes because of computation or storage capacity, they should avoid these three situations denoted by red arrows in Fig. 3.1:

- Red arrow ①: Edge node 1 decides to switch secret share 1 to another edge node, which happens to be edge node 2.
- Red arrow ②: Edge node 2 decides to switch secret share 2 to another edge node, which happens to be edge node 1.
- Red arrow ③: Both edge node 1 and 2 decide to switch both two secret shares to another edge nodes, which happen to be the same edge node K.

Based on the above observations, we propose a privacy-preserving stochastic task scheduling strategy in this chapter, which is based on (2,2) SS. A Markov chain will be constructed to facilitate the stochastic task scheduling on the edge nodes. Once edge nodes receive secret shares, they will decide whether to process those shares themselves or assign shares to other nodes based on the constructed Markov chain. Considering the privacy constraint, we cannot consider each node separately. Instead, each two nodes will be considered in pairs, and this is why we call our Markov chain "pairwise". The Markov chain will be constructed carefully with possible pair transitions. We can construct our pairwise Markov chain in advance based on the property that the stationary state of a Markov chain is not affected

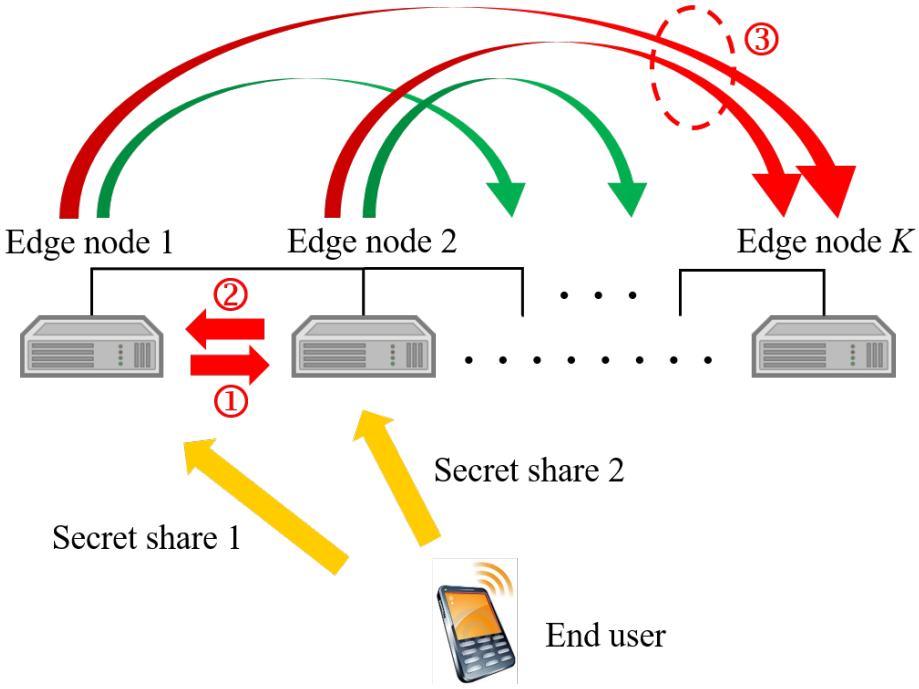


Figure 3.1: Toy example of (2,2) SS deployed on edge.

by its initial state. The optimized queuing and processing latency on edge nodes can then be computed and recorded in advance, which saves computational delay when deciding the scheduling. Simulation results validate that our proposed strategy can achieve an efficient task scheduling while ensure the privacy. The main contributions of this research project are as follows:

- We propose a mobile edge computing based deployment for privacy-preserving model parameter aggregation in federated learning. We formulate the task scheduling problem with the consideration of privacy constraint that no two shares of the same secret should be scheduled to the same edge node for processing. To the best of our knowledge, we are the first one to consider such a privacy constraint in task scheduling problems on edge.

- We propose a pairwise Markov chain to enforce the privacy constraint. The system states and transition probabilities are carefully designed. Achievability of the stationary state of the Markov chain is proved.

- We integrate the proposed pairwise Markov chain with the optimization model, construct a stochastic optimization problem for queuing plus processing latency minimization with the privacy constraint enforcement, and show that the problem can be solved through transforming it to a linear programming problem.

The rest of this chapter is organized as follows. Section 3.2 discusses related works. In Section 3.3, we give a full description of our privacy constrained task scheduling system. In Section 3.4, our constructed pairwise Markov chain for privacy constraint enforcement is introduced. Our optimization strategy through linear programming is presented in Section 3.5. Experimental results for validation and evaluation are given in Section 3.6. We discuss possible improvements and extensions in Section 3.7. Conclusions will be drawn in Section 3.8.

3.2 Related Works

Mobile edge computing helps reduce computation latency and communication overhead by processing information on the edge nodes near local users. On the other hand, since edge servers usually have limited computation and storage capacity, a well-designed task scheduling strategy is needed to decide which computing task to be assigned to which edge node. Up to now, several resource allocation and task scheduling works [77–82] have been done for mobile edge computing systems with different configurations. However, none of them considered privacy issues.

SIFT algorithm [71] has been widely used in image matching owing to its robustness. Since SIFT requires a large number of convolution computations, many cloud computing solutions have been proposed for it. With the concern of preserving image privacy against untrusted third party cloud servers, privacy-preserving SIFT schemes have been proposed. [72–74] propose to utilize homomorphic encryption to satisfy privacy requirements. Later, [75] claims that relying on just one server cannot well guarantee privacy, and [75,76] propose to randomly split images for encryption and transmit these two portions to two different

remote servers for privacy-preserving SIFT processing. When it comes to federated learning, those locally-learned model parameters and model aggregation tasks respectively correspond to images and image matching tasks. In this chapter, we propose an mobile edge computing based deployment for such a privacy-preserving application. During the whole process, it should be guaranteed that no edge nodes are assigned two shares of the same secret. Therefore, a task scheduling strategy with such a privacy constraint needs to be designed.

A Markov-chain-based task scheduling strategy is proposed in this chapter. The privacy constraint is guaranteed by paring the corresponding edge nodes. The achievability of our Markov chain is proved by irreducibility and positive recurrence [83]. Irreducibility is ensured by the structure of our Markov chain, and positive recurrence is claimed through Foster-Lyapunov theorem [84].

3.3 Model Construction and Problem Formulation

In this research project, we consider the privacy-preserving model parameter aggregation in federated learning as an application scenario. Each end device will derive model parameters based on its own dataset, which are combined together to generate a vector. This locally-learned model parameter vector is considered as a secret $\boldsymbol{I}$, which is split into two shares based on SS. Each element in the first share $\boldsymbol{I}_1$ is a number randomly selected within a certain range. The second share is generated through adding the original model parameter vector with the first share. In other words, we have $\boldsymbol{I}_2 = \boldsymbol{I} + \boldsymbol{I}_1$. Afterwards, two secret shares will be transmitted to two different edge nodes for privacy-preserving model aggregation. In the edge, each edge node will decide whether to process the received secret share by itself or switch the received secret share to another edge node. In this process, we need to ensure that no two shares of the same secret are assigned to the same edge node.

3.3.1 System Model

Let's consider an edge network, denoted as $G = (V, E)$, where V and E represent the set of nodes and links in the network. Each edge node is assumed to have both communication and

computing capabilities. All the links among edge nodes are considered full-duplex. In this chapter, we assume that communication among edge nodes are through high-speed wired links. For example, in a campus wireless network, the edge nodes could be access points (APs), which are connected by high speed Ethernet with transmission rate of 10 Gbps. Therefore, we assume that the transmission delay among the edge nodes is negligible compared with queuing delay in the edge node.

For an edge network with K edge nodes $V = \{v_1, v_2, \cdots, v_K\}$, when a secret share is uploaded by a user either through a wireless or wired link, we assume that it will always be transmitted to an edge node v_i, which is called the **reference node**. This reference node will decide whether it will process the secret share itself or switch the secret share to another node $v_j \neq v_i$. We assume that if a secret share gets switched to another node once, it will not be switched again. In other words, when the secret share arrives at the head-of-line of node v_j, it will be processed by v_j. Thus, the whole process can be seen as a *2-hop* behavior. In this chapter, we use P_i to denote the proportion of secret shares that are processed by node v_i itself when the edge network gets stable.

An example of the *2-hop* process with four edge nodes ($K = 4$) is illustrated in Fig. 3.2(a). From the perspective of reference node v_i, the *2-hop* process will generate K queues. The first queue records secret shares to be processed by node v_i itself, which is denoted as $\mathcal{Q}_{i,i}$. All the other queues $\mathcal{Q}_{i,j}$ record secret shares to be switched to node v_j. With a slight abuse of notation, we use $Q_{i,i}(t)$ and $Q_{i,j}(t)$ to denote their queue lengths in each time slot t, respectively. Besides, we use $D_{i,i}(t)$ and $D_{i,j}(t)$ to denote their head-of-line (HOL) delay [85], respectively. According to Little's Law [86], if the entire edge network can become stable, the queue length and HOL delay should have the following relationship in the stationary state.

$$d_{i,k} = \frac{1}{\beta_{i,k}} \lim_{M_q \to \infty} \sum_{m_q=0}^{M_q} m_q \cdot Pr\{q_{i,k} = m_q\} \tag{3.1}$$

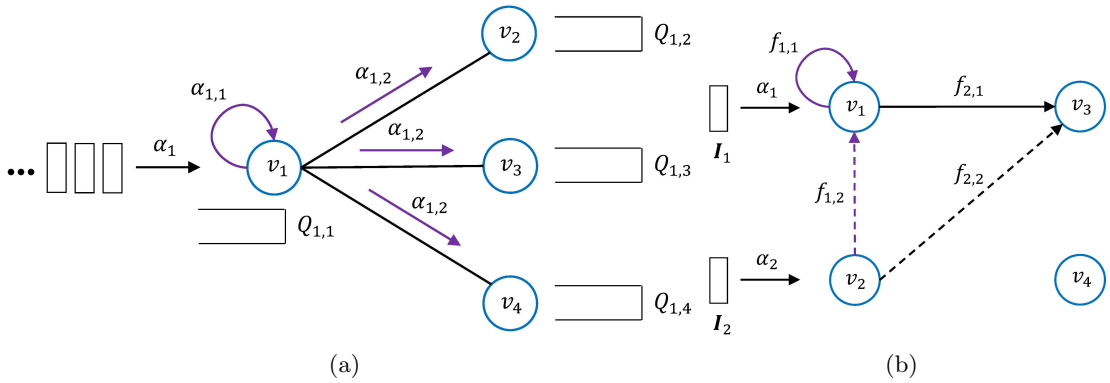


Figure 3.2: Example of our *2-hop* process and transmissions not permitted under privacy constraint. (a) *2-hop* process. (b) Transmissions not permitted.

Here, node v_k can be both v_i and v_j. $\beta_{i,k}$ represents the long-term average arrival rate for queue $\mathcal{Q}_{i,k}$. In this research project, we use the average arrival rate within a long enough time interval for approximation. Note that a unit arrival process is assumed in this chapter, which makes all the arrival rates between 0 and 1. $d_{i,k}$ and $q_{i,k}$ respectively represent the HOL delay and queue length for queue $\mathcal{Q}_{i,k}$ as the entire system becomes stable. m_q and M_q are the possible and maximum queue length individually. $Pr\{q_{i,k} = m_q\}$ reflects the probability where the stable queue length of $\mathcal{Q}_{i,k}$ is equal to m_q.

We can assume that the long-term average arrival rate of node v_i is α_i. This α_i can also be seen as the arrival rate in the first hop. Since once a secret share arrives at node v_i, it will be processed by node v_i itself with a proportion P_i, the arrival rate for $\mathcal{Q}_{i,i}$ can then be denoted as $\beta_{i,i} = P_i\alpha_i$. On the other hand, the proportion where an arrived secret share is scheduled to another node is $1 - P_i$ obviously. In this chapter, we assume that the scheduled node is randomly selected from the other $(K - 1)$ nodes, which makes the long-term average arrival rate of $\mathcal{Q}_{i,j}$ equal to $\beta_{i,j} = \frac{1}{K-1}(1 - P_i)\alpha_i$. From this equation, we can see that all $\beta_{i,j}$ are actually the same. For notation ease, we can respectively use $\alpha_{i,1}$ and $\alpha_{i,2}$ to represent $\beta_{i,i}$ and all the $\beta_{i,j}$.

In this chapter, we assume that all edge nodes have the same computation capacity. In

this case, service time is just related to tasks. Since federated learning will only aggregate model parameters based on the same machine learning model, we can consider service time for those secret shares the same, which is denoted as N_o, in the number of time slots. With the same arrival and service rate, we can claim that when the whole edge network becomes stable, all those queue lengths $Q_{i,j}(t)$ should converge to a single length q_2. Considering Little's Law discussed above, we also have the converged HOL delay d_2. Besides, we also use q_1 and d_1 to represent the converged queue length and HOL delay for $\mathcal{Q}_{i,i}$.

3.3.2 Optimization Problem Formulation

In this research project, we aim at minimizing the total latency under the system model introduced in the last section. According to the definition, the total latency is actually composed of queuing time and service time. However, as has been discussed in the last section, service time is fixed with the given servers and tasks. In this case, our optimization problem can actually be simplified to minimizing queuing time, which can be represented by HOL delay. In our defined system model, we have two kinds of HOL delay, d_1 and d_2. Obviously, we want them both to be minimized. In this case, we define the summation of these two kinds of HOL delay as our objective to be minimized, which is denoted as $\mathbb{T}$ shown below:

$$\mathbb{T} \triangleq d_1 + d_2 \tag{3.2}$$

From the perspective of each edge node v_i, there is a similar *2-hop* process. In this case, we can define a uniform optimization problem as below:

$$\begin{aligned} \min \quad & \mathbb{T} \\ \text{s.t.} \quad & \phi_p(\boldsymbol{I_1}, \mathcal{Q}_{i,k}) + \phi_p(\boldsymbol{I_2}, \mathcal{Q}_{i,k}) \neq 2 \quad i, k = 1, 2, \cdots, K \end{aligned} \tag{3.3}$$

The indicator function $\phi_p(\boldsymbol{I}, \mathcal{Q}_{i,k})$ is defined as follows:

$$\phi_p(\boldsymbol{I}, \mathcal{Q}_{i,k}) \triangleq \begin{cases} 1 & \boldsymbol{I} \in \mathcal{Q}_{i,k} \\ 0 & \boldsymbol{I} \notin \mathcal{Q}_{i,k} \end{cases} \tag{3.4}$$

The constraint in (3.4) actually corresponds to the privacy constraint in this research project, which requires that no edge nodes should get both two shares of the same secret. Fig. 3.2(b) shows an example with four edge nodes. In this example, two shares of the same secret individually arrive at node v_1 and v_2. We illustrate two pairs of transmissions that should not be permitted, $(f_{1,1}, f_{1,2})$ and $(f_{2,1}, f_{2,2})$. The privacy constraint is enforced through our pairwise Markov chain to be described in detail in Section 3.4. We will tune the transition probabilities in our Markov chain to optimize the delay performance.

3.4 Pairwise Markov Chain for Privacy Constraint Enforcement

In this research project, we consider an attack model like this: an attacker wants to recover the content of some secrets. In order for this, the attacker randomly selects and eavesdrops a node in our edge network. Therefore, our task scheduling policy should avoid transmitting two shares of the same secret to the eavesdropped node. Since the eavesdropped node can be any node in the edge network, our policy should be further extended to the case that two shares of the same secret should not be transmitted to any node in the edge network. When it comes to a traditional task scheduling problem, the whole process can actually be described by a Markov chain. Here, we carefully modify the Markov chain by considering each two edge nodes in pairs to reflect the privacy constraint.

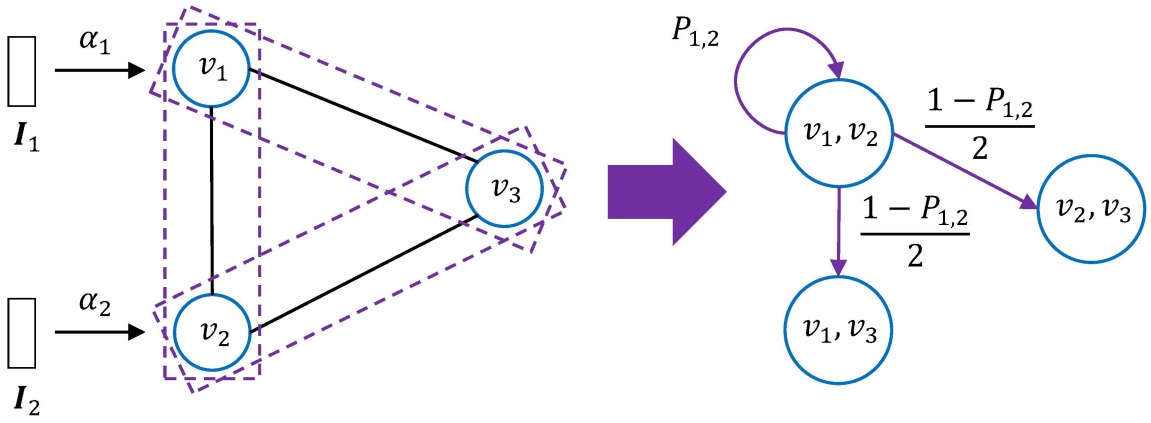


Figure 3.3: Example of an edge network and transformed pairwise architecture.

3.4.1 Optimization Problem Reconsideration and Pairwise Markov Chain Construction

Our constructed pairwise Markov chain is based on a pairwise architecture. We start from a simple example of an edge network with three edge nodes, shown in Fig. 3.3.

From this figure, we can see that each two edge nodes are considered in pairs to generate a new node. Based on our pairwise architecture, if two different edge nodes v_1 and v_2 receive two secrets of the same share, generally speaking, they will have two choices: processing the two shares themselves, or transmitting the two shares to another pair. If they choose to process the two shares themselves, obviously, there will be no violation of the privacy constraint. If they choose to schedule the shares to another pair, there will be another two choices: transmitting the two shares to v_1 and v_3, or v_2 and v_3. When secret shares are transmitted to v_1 and v_3, it can actually be seen as the case that the first secret share is still processed by v_1 itself, while the second secret share is transmitted from v_2 to v_3. A similar case applies to secret shares being transmitted to v_2 and v_3. The detailed assignments are shown in Fig. 3.4. It is easy to imagine when an edge network has $K(K > 3)$ nodes, the **reference node pair** (v_{i_1}, v_{i_2}) can have three kinds of choices (instead of just two in the above example) when two received secret shares are determined to be transmitted to another pair. Besides being transmitted to node pair (v_{i_1}, v_{j_2}) and (v_{j_1}, v_{i_2}), the secret

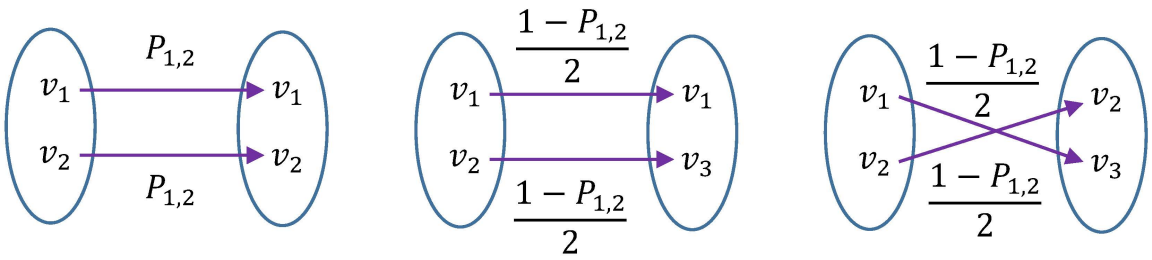


Figure 3.4: Detailed assignments of secret shares.

shares can also be transmitted to node pair (v_{j_1}, v_{j_2}). Here, $i_1, i_2, j_1, j_2 = 1, 2, \cdots, K$, and i_1, i_2, j_1 and j_2 are all different.

With the above pairwise architecture, two shares of the same secret have no chance to be transmitted to the same node. On the other hand, we need to reconsider the system model and optimization problem described in Section 3.3. Previously, we consider the whole system from the perspective of each edge node. With the introduction of privacy constrain, we should consider the system in node pairs. Fortunately, the system still follows Little's Law, but with a little modification for the arrival rate of each edge node. In Fig. 3.3, we give the proportion of secret shares processed by node pair (v_1, v_2) itself, denoted as $P_{1,2}$. Based on previous discussion, we have shown that even if two shares of the same secret will be transmitted to another node pair, the situation still exists that one of these two secret shares will still be processed by the original node. In other words, the proportion of a node P_{i_1} is not necessarily equal to the proportion of a node pair P_{i_1,i_2}. In the example shown in Fig. 3.3, we actually have $P_1 = P_2 = P_{1,2} + \frac{1-P_{1,2}}{2} = \frac{1+P_{1,2}}{2}$. When the case is extended to an edge network with K edge nodes, we should have the following relationship:

$$P_{i_1} = P_{i_2} = P_{i_1,i_2} + \frac{\binom{1}{K-1} - 1}{\binom{2}{K} - 1}(1 - P_{i_1,i_2}) = \frac{2 + (K-1)P_{i_1,i_2}}{K+1} \tag{3.5}$$

The arrival rates need to be modified accordingly when described by P_{i_1,i_2}.

From (3.5), we can see that P_{i_1} and P_{i_2} are the same. Actually, if we consider the

whole edge network from the perspective of node pair (v_{i_1}, v_{i_2}), the first-hop arrival rate of these two nodes α_{i_1} and α_{i_2} are also the same, denoted as λ_{i_1,i_2} for notation ease. This is because a node pair corresponds to two shares of the same secret. Obviously, these two secret shares always arrive at the same time. In this situation, the minimal HOL delay summation, described in (3.2), of these two nodes should be the same. In other words, we just need to minimize the HOL delay summation from the perspective of one of these two nodes, which makes the optimization objective in (3.3) unchanged. Besides, with our generated pairwise architecture, the privacy requirement can be met, which corresponds to the constraint in (3.3).

Similar to traditional scheduling policies, our pairwise scheduling process can also be described by a Markov chain, which is called as pairwise Markov chain in this research project. Each system state of our constructed pairwise Markov chain is denoted as $\boldsymbol{Z}(t) = (\boldsymbol{O}(t), \boldsymbol{Q}(t))$, where $\boldsymbol{O}(t)$ and $\boldsymbol{Q}(t)$ represent the working status and queue length vector, respectively. Each element $o_k(t)$, with $k = 1, 2, \cdots, K$, in $\boldsymbol{O}(t)$ reflects how many time slots are still needed for the k-th server to complete its current task. Recall that in Section 3.3.1, we assume that the total number of time slots for an edge node to complete a task is N_o. Therefore, each $o_k(t)$ has $N_o + 1$ possible values, with $o_k(t) = 0, 1, 2, \cdots, N_o$, where $o_k(t) = 0$ means that the server is idle in this time slot. Each element $q_k(t)$ in $\boldsymbol{Q}(t)$ represents the queue length with a structure similar to that of $o_k(t)$. For the edge network with K nodes, the state space $S \subseteq \{0, 1, \cdots, N_o\}^K \times \{0, 1, \cdots M_q\}^K$ represents all possible states of our Markov chain that can be reached from the initial state, which is defined as $\boldsymbol{Z}(0) \triangleq (\boldsymbol{0}_{K\times 1}, \boldsymbol{0}_{K\times 1})$. M_q depends on the storage limit of each server.

3.4.2 Pairwise Markov Chain Achievability

According to [83], an irreducible Markov chain has a positive stationary distribution if and only if all of its states are positive recurrent. In other words, we can prove that our Markov chain has a stationary distribution by showing that it is irreducible and positive recurrent.

Proposition 3.4.1. *The pairwise Markov chain proposed in this research project is irreducible.*

Proof. For irreducibility, we should show that any two states can be reached from each other in our Markov chain. Since by the definition of our Markov chain, the initial state can reach any other states, we just need to show that any states can reach the initial state. In our Markov chain, we assume that the initial state has a possible transition to itself. This transition is necessary and reasonable, since we should consider the case that no tasks are in the edge in a certain time slot. Such an assumption will just make our Markov chain aperiodic. This means that for a given set of transition probabilities, our Markov chain will just have one stationary distribution, which does not influence our proof.

With the above assumption, we can give a proof for irreducibility as follows. For any state $\boldsymbol{Z}(t)$, we can find the server with the longest queue length q_{max}. Recall that each task can be completed in N_o time slots. Then, we can assume an event that in $N_o(q_{max}+1)$ time slots, there are no tasks arriving at the edge. From Section 3.4.1, we know that the arrival rate of node pair (v_{i_1}, v_{i_2}) is λ_{i_1,i_2}. Then, the probability of this event is $(1-\lambda_{i_1,i_2})^{N_o(q_{max}+1)} > 0$, which means that this event can happen. If this event happens, the edge will complete all tasks that are both in service and queued in $N_o(q_{max}+1)$ time slots. In other words, the current state will transit to the initial state $\boldsymbol{Z}(0)$ in at most $N_o(q_{max}+1)$ time slots. Note that some states $\boldsymbol{Z}(t)$ may transit to the initial state in less than $N_o(q_{max}+1)$ time slots with our task switch operations. However, this does not matter since for the initial state, we have already assumed a possible transition to itself. Since any state in our Markov chain can have such an event, any state can reach back to the initial state, which proves the irreducibility. □

Now, we have shown that our Markov chain is irreducible. We still have to show that our Markov chain is positive recurrent. Here, we utilize Foster-Lyapunov theorem [84] to prove it.

Proposition 3.4.2. *The pairwise Markov chain proposed in this research project is positive*

recurrent.

Proof. Consider a Lyapunov function defined as follows:

$$V(\boldsymbol{Z}(t)) \triangleq \|\boldsymbol{O}(t)\| + \|\boldsymbol{Q}(t)\| = \sum_{k=1}^{K} o_k(t) + q_k(t) \tag{3.6}$$

Here, $\|\cdot\|$ is the L_1-norm. Then, according to Foster-Lyapunov theorem, it suffices to show that for any given state $\boldsymbol{Z}_c$, our Markov chain has:

$$E[V(\boldsymbol{Z}(t+1)) - V(\boldsymbol{Z}(t))|\boldsymbol{Z}(t) = \boldsymbol{Z}_c] \leq -\delta,\ \boldsymbol{Z}_c \in F \tag{3.7a}$$

$$E[V(\boldsymbol{Z}(t+1)) - V(\boldsymbol{Z}(t))|\boldsymbol{Z}(t) = \boldsymbol{Z}_c] < C,\ \boldsymbol{Z}_c \notin F \tag{3.7b}$$

Here, $E[\cdot]$ calculates the expected value. δ and C are two strict positives. F denotes some finite set. Next, we will show how to find δ, C and F.

In our Markov chain, we define F to include all states where every node in the server is busy with some task. Formally, $F = \{\boldsymbol{Z}_F = (\boldsymbol{O}_F, \boldsymbol{Q}_F)|o_{F,k} \neq 0 \text{ for } \forall o_{F,k} \in \boldsymbol{O}_F, k = 1, 2, \cdots, K\}$. In this case, for any $\boldsymbol{Z}_c \in F$ and $K \geq 3$, we have:

$$\begin{aligned}
&E[V(\boldsymbol{Z}(t+1)) - V(\boldsymbol{Z}(t))|\boldsymbol{Z}(t) = \boldsymbol{Z}_c] \\
=&E[\|\boldsymbol{O}(t+1)\| + \|\boldsymbol{Q}(t+1)\| - \|\boldsymbol{O}(t)\| - \|\boldsymbol{Q}(t)\||\boldsymbol{Z}(t) = \boldsymbol{Z}_c] \\
=&\sum_{k=1}^{K}[o_k(t+1) - o_k(t)] + \sum_{k=1}^{K}[q_k(t+1) - q_k(t)] \\
\leq& -K + 2 < 0
\end{aligned} \tag{3.8}$$

Since every edge node is busy with some task, after one time slot passes, each element in the working status vector can only decrease by 1. With totally K edge nodes, $\|\boldsymbol{O}(t+1)\|$ will decrease by K compared with $\|\boldsymbol{O}(t)\|$. Besides, with the unit arrival process assumption,

there are at most two new tasks (one secret split into two shares) arriving at node pair (v_{i_1}, v_{i_2}) in the current time slot. Then, no matter whether these two tasks are processed by the current nodes or transmitted to other nodes, $\|\boldsymbol{Q}(t+1)\|$ will not be further changed. Therefore, $\|\boldsymbol{Q}(t+1)\|$ can at most increase by 2 compared with $\|\boldsymbol{Q}(t)\|$. Then, the expected value is at most equal to (3.8). Furthermore, if we want to make the scheduling problem meaningful, K should be larger than 2. Therefore, (3.8) is less than 0. In this case, for any $\boldsymbol{Z}_c \in F$, the expected value is strictly less than 0. In other words, we can find a strictly positive δ, satisfying formula (3.7a).

On the other hand, for any $\boldsymbol{Z}_c \notin F$, some edge nodes may be idle, which may make the total decrease of $\|\boldsymbol{O}(t)\|$ less than K. Besides, similar to the case of $\boldsymbol{Z}_c \in F$, $\|\boldsymbol{Q}(t+1)\|$ can at most increase by 2 compared with $\|\boldsymbol{Q}(t)\|$. Then, we can see that the largest expected value happens when all edge nodes are idle. In other words, we have (3.9), which is not greater than 2. In this case, we can pick any $C > 2$. Then, we can have that for any $\boldsymbol{Z}_c \notin F$, formula (3.7b) is satisfied.

$$
\begin{aligned}
&E[V(\boldsymbol{Z}(t+1)) - V(\boldsymbol{Z}(t))|\boldsymbol{Z}(t) = \boldsymbol{Z}_c] \\
=&\sum_{k=1}^{K}[o_k(t+1) - o_k(t)] + \sum_{k=1}^{K}[q_k(t+1) - q_k(t)] \\
\leq& 0 + 2 = 2
\end{aligned} \tag{3.9}
$$

Based on the above analysis, we have proved that our Markov chain is irreducible and positive recurrent. In this case, we can claim that the chain has a stationary distribution. □

3.4.3 Privacy Constrained Stochastic Task Scheduling Modeling

Next, let's take a look at how to relate the privacy constraint to our constructed pairwise Markov chain. Previously, we have listed four situations that do not violate the privacy constraint. These four situations can be ensured through carefully designing transition

probabilities in our constructed pairwise Markov chain. For a network with totally K nodes, we will have $(K-1)K/2$ pairs, which is denoted as (v_{k_1}, v_{k_2}) in this section. From the perspective of a given reference node pair (v_{i_1}, v_{i_2}) and for each state $\boldsymbol{Z}(t)$, we will assign a probability for an assignment to node pair (v_{k_1}, v_{k_2}), denoted as $P^{k_1,k_2}_{\boldsymbol{Z}(t)}$. Note that as has been discussed before, $k_1 = i_1$ or $k_2 = i_2$ actually correspond to the cases that secret shares will be processed by the current nodes. We can construct relationships between these probabilities and transition probabilities to ensure the privacy constraint. Here, two cases are discussed in detail for relationship construction between those two probabilities. Similar discussions can be done from the perspective of any given node pair.

Case 1: In this case, we will discuss all the states in our constructed Markov chain whose elements in the working status vector are all not zero. This means that all edge nodes are in service. In other words, when our Markov chain arrives at the state $\boldsymbol{Z}(t) = ((o_k(t) \neq 0)_{K\times 1}, (q_k(t))_{K\times 1})$, it can only transit to the state $\boldsymbol{Z}(t+1) = ((o_k(t)-1)_{K\times 1}, (q_k(t) + \Delta q_k)_{K\times 1})$. Here, Δq_k can be 0 or 1. Value 0 means that there is no new task assigned to node v_k in the current time slot, while value 1 means that there is an assigned task to v_k. In this case, we should have:

$$Pr\{\boldsymbol{Z}(t+1)|\boldsymbol{Z}(t)\} = \begin{cases} \lambda_{i_1,i_2} P^{k_1,k_2}_{\boldsymbol{Z}(t)} & \Delta q_i = \Delta q_j = 1 \\ 1 - \lambda_{i_1,i_2} & \Delta q_k = 0, \forall k \end{cases} \tag{3.10}$$

It is easy to imagine that all $\Delta q_k = 0$ represents there are no newly arrived tasks.

Case 2: In this case, we will discuss all the states in our Markov chain who have elements with value 0 in the working status vector. This means that some edge nodes are idle, and can process their next queued tasks. Then, we should have $\boldsymbol{Z}(t+1) = ((o_k(t) + \Delta o_k)_{K\times 1}, (q_k(t) + \Delta q_k)_{K\times 1})$. Δo_k can be $-1, 0, N$, and Δq_k can be $-1, 0, 1$. Each combination of Δo_k and Δq_k corresponds to a subsequent system state. The value of Δo_k and Δq_k are determined by the arrival process, working status of each edge node and task

assignment strategy. When $o_k(t) \neq 0$, the corresponding edge node is in service, and Case 2 will get simplified to Case 1. If the edge node is idle ($o_k(t) = 0$), Δo_k can be 0 or N_o. It is 0 when the edge node has processed all the queued tasks and has no newly assigned task. In this situation, Δq_k can only be 0. Otherwise, Δo_k will be N_o. As for Δq_k, its value should be -1 if there is no newly arrived task, and 0 with a newly arrived task.

After the above discussion about possible combinations of Δw_k and Δq_k, it is the time to consider those transition probabilities. Before that, we further summarize those possible combinations into five categories. Each combination is denoted as c_i, with $i = 1, 2, \cdots, 5$. In detail, $c_1 = (\Delta q_k = 1, \Delta o_k = -1)$, $c_2 = (\Delta q_k = 0, \Delta o_k = N_o)$, $c_3 = (\Delta q_k = 0, \Delta o_k = 0)$, $c_4 = (\Delta q_k = 0, \Delta o_k = -1)$, and $c_5 = (\Delta q_k = -1, \Delta o_k = N_o)$. c_1 and c_2 correspond to the situations that there is a newly assigned task, and we use a set C_Y to include them. c_3, c_4 and c_5 correspond to the situations that there is no newly assigned task, which is included by the set C_N. Then, the transition probability can be described as follows:

$$Pr\{\boldsymbol{Z}(t+1)|\boldsymbol{Z}(t)\} = \begin{cases} \lambda_{i_1,i_2} P^{k_1,k_2}_{\boldsymbol{Z}(t)} & (\Delta q_i, \Delta o_i) \in C_Y, (\Delta q_j, \Delta o_j) \in C_Y \\ 1 - \lambda_{i_1,i_2} & (\Delta q_k, \Delta o_k) \in C_N, \forall k \end{cases} \tag{3.11}$$

Based on the above discussion, we can construct relationships between $P^{k_1,k_2}_{\boldsymbol{Z}(t)}$ and transition probabilities. The parameters to be tuned are actually $P^{k_1,k_2}_{\boldsymbol{Z}(t)}$.

3.4.4 Optimization Problem Modeling

Next, we will talk about how to complete constructing our optimization model in detail. Recall that in Section 3.3, we gave a general idea of the proportion P_{i_1} and P_{i_2} (P_i in Section 3.3), the privacy constraint, and parameters to be tuned for our optimization problem. In Section 3.4.1, we have shown that $P_{i_1} = P_{i_2}$, and we just need to consider the HOL delay summation from the perspective of one of those two nodes v_{i_1} and v_{i_2}. Besides, in the last section, we constructed our pairwise Markov chain for privacy constraint enforcement,

and defined parameters to be tuned as $P^{k_1,k_2}_{\boldsymbol{Z}(t)}$. In this case, our optimization problem can be rewritten as (3.12). For simplicity, $Pr\{\boldsymbol{Z}(t+1)|\boldsymbol{Z}(t)\}$ is represented by $Pr_{z_1,z}$. The stationary distribution is described as $Dist_z$.

$$\begin{aligned} \min_{P^{k_1,k_2}_{\boldsymbol{Z}(t)}} \quad & \mathbb{T} = d_1 + d_2 \\ \text{s.t.} \quad & \sum_{z\in S} Pr_{z_1,z} \cdot Dist_z = Dist_{z_1}, \forall z_1 \in S \\ & \sum_{z\in S} Dist_z = 1 \\ & P^{k_1,k_2}_{\boldsymbol{Z}(t)} \geq 0 \end{aligned} \tag{3.12}$$

Here, the first two constraints represent stationary state equations of our Markov chain. The relationship between $P^{k_1,k_2}_{\boldsymbol{Z}(t)}$ and $Pr_{z_1,z}$ is described in equation (3.10) and equation (3.11). With our Markov chain, $d_{i_1,k}$ can be calculated through the extension of equation (3.1) as follows ($d_{i_2,k}$ has the same form):

$$d_{i_1,k} = \frac{1}{\beta_{i_1,k}} \sum_{m_q=0}^{M_q} m_q \cdot Pr\{q_{i_1,k} = m_q\} = \frac{1}{\beta_{i_1,k}} \sum_{m_q=0}^{M_q} m_q \sum_{\substack{z\in S \\ q_{i_1,k}=m_q}} Dist_z \tag{3.13}$$

Here, M_q corresponds to the storage limit of each server. Recall that when $k = i_1$, $\beta_{i_1,i_1} = \alpha_{i_1,1}$, $q_{i_1,i_1} = q_1$ and $d_{i_1,i_1} = d_1$, while for all $k \neq i_1$, $\beta_{i_1,k} = \alpha_{i_1,2}$, $q_{i_1,k} = q_2$ and $d_{i_1,k} = d_2$.

If we can solve the optimization problem described in (3.12), the proportion P_{i_1} and P_{i_2} can be calculated with those $P^{k_1,k_2}_{\boldsymbol{Z}(t)}$. In detail, we can firstly calculate the node pair proportion P_{i_1,i_2} based on (3.14). Then, P_{i_1} and P_{i_2} can be derived based on the relationship

described in (3.5).

$$P_{i_1,i_2} = \frac{\sum_{z \in S} Dist_z \cdot \sum_{z_1 \in S_1} P_z^{k_1,k_2}}{\sum_{z \in S} Dist_z \cdot \sum_{z_1 \in S_2} P_z^{k_1,k_2}} \tag{3.14}$$

Here, $\boldsymbol{Z}(t)$ and $\boldsymbol{Z}(t+1)$ are respectively simplified to z and z_1. Given any system state $z \in S$, S_1 indicates a portion of subsequent system states of z, These subsequent system states correspond to the case that $k_1 = i_1$ and $k_2 = i_2$. In other words, the newly arrived tasks are decided to be processed by the current node pair. S_2 corresponds to the set of all possible subsequent system states of z.

3.5 Optimization Problem Solving

As has been discussed in the last section, we need to firstly solve the optimization problem described in (3.12) in order to derive the proportion P_{i_1} and P_{i_2}. The optimization model described in (3.12) can theoretically be solved. However, considering its nonlinearity, it is not computationally practical. In this section, we will give a full description of how to transform our optimization problem to a linear equivalent form, and solve it through linear programming.

Firstly, we let $X_z^{k_1,k_2} = Dist_z \cdot P_z^{k_1,k_2}$. Since $\sum_{k_1,k_2=1,k_1 \neq k_2}^{K} P_z^{k_1,k_2} = 1$, we should have $\sum_{k_1,k_2=1,k_1 \neq k_2}^{K} X_z^{k_1,k_2} = Dist_z$ correspondingly. After that, the original optimization model is considered together with equation (3.5), (3.11), (3.13) and (3.14), and can be transformed

to the following form:

$$\begin{aligned}
\min_{X_z^{k_1,k_2},P_{i_1,i_2}} \quad & \mathbb{T} = d_1 + d_2 \\
\text{s.t.} \quad & C_1(X_z^{k_1,k_2}, P_{i_1,i_2}) = 0 \\
& C_2(X_{z_1}^{k_1,k_2}) = 0, \forall z_1 \in S \\
& \sum_{z\in S} \sum_{k_1,k_2=1,k_1\neq k_2}^{K} X_z^{k_1,k_2} = 1 \\
& X_z^{k_1,k_2} \geq 0
\end{aligned} \tag{3.15}$$

Where d_1 and d_2 can be described by $X_z^{k_1,k_2}, P_{i_1,i_2}$ with the following two formulas:

$$\begin{aligned}
d_1 =& \frac{K+1}{\alpha_{i_1}[2+(K-1)P_{i_1,i_2}]} \sum_{m_q=0}^{M_q} m_q \sum_{\substack{z\in S \\ q_1=m_q}} \sum_{k_1,k_2=1,k_1\neq k_2}^{K} X_z^{k_1,k_2} \\
d_2 =& \frac{K+1}{\alpha_{i_1}(1-P_{i_1,i_2})} \sum_{m_q=0}^{M_q} m_q \sum_{\substack{z\in S \\ q_2=m_q}} \sum_{k_1,k_2=1,k_1\neq k_2}^{K} X_z^{k_1,k_2}
\end{aligned} \tag{3.16}$$

The first and second constraint respectively come from equation (3.14) and the first stationary state equation of our Markov chain. In detail, we have:

$$\begin{aligned}
C_1(X_z^{k_1,k_2}, P_{i_1,i_2}) =& \sum_{z\in S} \sum_{z_1\in S_1} X_z^{k_1,k_2} - P_{i_1,i_2} \sum_{z\in S} \sum_{z_1\in S_2} X_z^{k_1,k_2} \\
C_2(X_{z_1}^{k_1,k_2}) =& \sum_{z\in S} \sum_{z_1\in S_3} \alpha_{i_1} X_z^{k_1,k_2} + \\
& \sum_{z\in S} \sum_{z_1\in S_4} (1-\alpha_{i_1}) \sum_{k_1,k_2=1,k_1\neq k_2}^{K} X_z^{k_1,k_2} - \sum_{k_1,k_2=1,k_1\neq k_2}^{K} X_{z_1}^{k_1,k_2}
\end{aligned} \tag{3.17}$$

Here, S_3 and S_4 are respectively the set of subsequent states with and without newly arrived tasks. With (3.15), (3.16) and (3.17), the original optimization problem is transformed to an equivalent linear programming problem. We can use a one-dimensional search algorithm proposed in [80] to solve it. The optimal set of $X_z^{k_1,k_2}$, denoted as X_p, will be firstly obtained for each given $P_{i_1,i_2} \in [0,1]$. Then, we will conduct a horizontal comparison for all combinations of the optimal set X_p and given P_{i_1,i_2} to find the optimal P^* and the corresponding ultimate optimal set X^*. Finally, all transition probabilities can be calculated through $P_z^{k_1,k_2} = X_z^{k_1,k_2} \cdot (\sum_{k_1,k_2=1,k_1\neq k_2}^{K} X_z^{k_1,k_2})^{-1}$ and equation (3.11).

3.6 Simulation and Evaluation

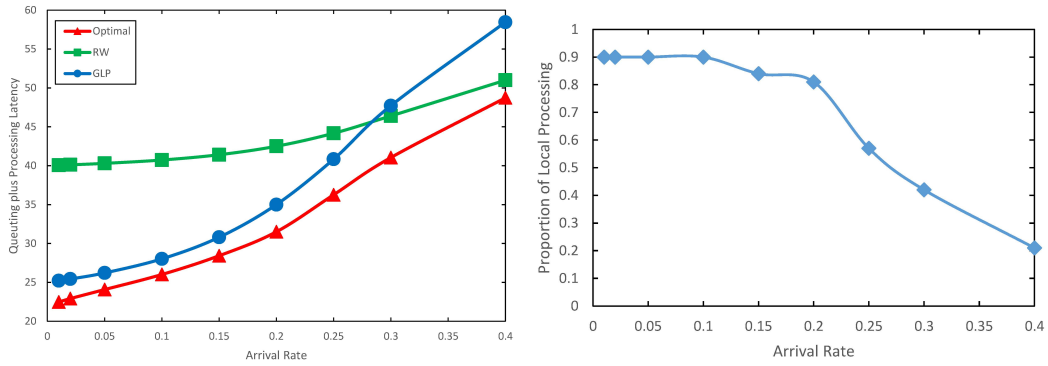
In this section, we will give a full analysis for the efficiency of our proposed privacy-preserving stochastic task scheduling strategy. Our scheme is compared with two baselines:

- Random Walk (RW): Each idle pair receiving two secrets of the same secret will randomly select another pair, which does not include themselves, and transmit those two portions to that pair for processing. This case actually corresponds to the proportion for local processing $P_{i_1} = P_{i_2} = 0$.

- Greedy Local Processing (GLP): Each idle pair receiving two shares of the same secret will choose to process those two shares themselves anytime. This case actually corresponds to the proportion for local processing $P_{i_1} = P_{i_2} = 1$.

In the simulation, we consider an edge with six nodes. The largest possible number of queued tasks M_q is set to be 5. As has been mentioned in Section 3.3.1, the processing latency N_o for all tasks in this research project can be considered the same. Here, we assume that $N_o = 20$ time slots. Besides, we respectively add a bias of 100 time slots to reflect possible node and link corruptions. Such biases are randomly applied to the whole network.

Fig. 3.5(a) shows the comparison among our task scheduling strategy, RW and GLP.



(a) (b)

Figure 3.5: Efficiency evaluation results. (a) Queuing plus processing latency vs. arrival rate. (b) Optimal proportion for local processing vs. arrival rate

From this figure, we can see that as the arrival rate grows, all of the three task scheduling strategies have longer latency, which is consistent to our intuition. Overall, our task scheduling strategy has the most optimal result. When the arrival rate is low, GLP almost has the same latency as our task scheduling strategy. This is because at this time, edge nodes usually do not have queued tasks. In this case, tasks are better to be served locally as soon as they reach the edge nodes. Transmission to other nodes can meet additional link corruptions besides node corruptions. When the arrival rate rises, GLP begins to achieve performance similar to our task scheduling strategy. This indicates that passing tasks to other nodes is proper for a higher arrival rate. Finally, Fig. 3.5(b) shows the optimal proportion of local processing P_{i_1} or P_{i_2} for different arrival rates. From this figure, we can notice that the proportion P_{i_1} or P_{i_2} are not exactly 1, 0.9 actually, when the arrival rate is low. The reason is that node corruptions will cause queued tasks. When tasks are queued in one node, it is better to distribute them to other nodes to fully utilize computation resources of the whole network.

3.7 Future Work

In this section, we want to discuss briefly about potential improvements and extensions of our privacy-preserving scheduling policy in this chapter.

- In this chapter. we assume that the attacker can only eavesdrop one edge node, which can be overcome through (2,2) SS. In the next chapter, we extend (2,2) SS to (R,L) SS to tackle the case with multiple edge nodes getting hacked. Even worse, all edge nodes have the possibility to be owned by the same entity and all get compromised. This case may not be solved properly just by privacy-preserving scheduling. In addition, schedulers may also be attacked. It is also interesting to look into the case that schedulers act maliciously.

- The privacy-preserving task scheduling strategy in this chapter is based on a comparatively simple mobile edge computing architecture, in which all the edge nodes are connected to each other and have the same computation and storage capacity. In the next chapter, the scheduling problem is considered in a more general mobile edge computing structure. In the future, we will also strive to encompass heterogeneous edge devices into the scheduling algorithm design.

- Communication latency and throughput constraints are not considered in this chapter, which restricts the scalability of the scheduling policy in this chapter. In the real world, communication between local devices and edge nodes are usually through wireless networks. Besides, communication among edge nodes can either be wired or wireless. A more granular and comprehensive task scheduling strategy needs to be developed, which considers communication factors, such as communication overhead, bandwidth, reliability, and energy.

3.8 Summary

In this chapter, we propose a privacy-preserving stochastic task scheduling strategy for locally-learned model parameter aggregation in federated learning based on (2,2) SS. Secrets are randomly split into two shares for encryption and transmitted to two edge nodes for processing. During the whole process, all edge nodes should not have the chance to get both two shares of the same secret. In order to guarantee such a privacy constraint, we construct a pairwise Markov chain to take care of it. Edge nodes are considered in pairs in all system states of our proposed Markov chain. The achievability can be proved through showing irreducibility and positive recurrence. Queuing plus processing latency can be minimized based on our constructed pairwise Markov chain. Simulation results validate that our task scheduling strategy can achieve minimal latency and guarantee the privacy constraint simultaneously.

Chapter 4: Communication-Aware Secret Share Placement in Hierarchical Edge Computing

This chapter can be seen as an extension of the last chapter. We consider (R,L) Secret Sharing (SS) instead of (2,2) SS. Besides computation latency, we also need to consider communication overhead among those edge servers, especially after each secret being separated to more than two shares. This research project is related to the publication [23].

4.1 Introduction

In this research project, we design a communication-aware secret share placement strategy in mobile edge computing to minimize **communication overhead** among different secrets and shares in terms of **weighted transmission hop counts**. The computation and storage capacity of each edge node is considered by applying a limitation with regard to queue length. The strategy is deployed in a hierarchical mobile edge computing architecture shown in Fig. 4.1. Hierarchical architectures [31–34] are widely considered recently in mobile edge computing, which provide flexibility in balancing traffic loads in the whole edge network, and bring improved quality of service (QoS).

Instead of directly involving mobile local devices in the computation process, data in each user will be separated into L shares and uploaded to L different nearby access points. Afterwards, all the computation will be conducted on edge, which will further reduce the possibility of packet loss due to user dropouts. Fig. 4.2 illustrates an example with $L = 2$. Since communication happens among shares of the same secret and different secrets, we should put all those uploaded secret shares as closely as possible. On the other hand, our scheduling policy should also guarantee that a certain number of shares of the same secret

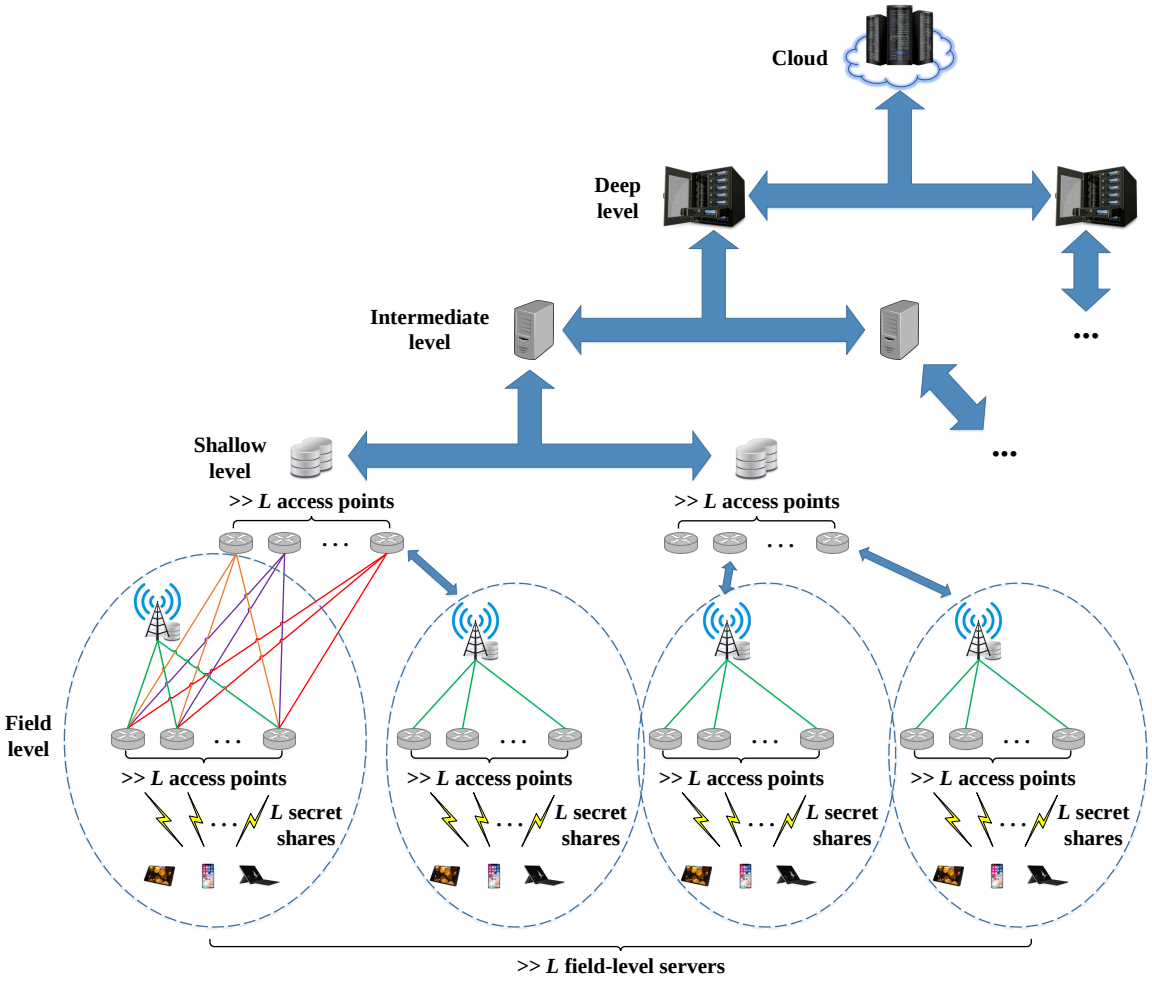


Figure 4.1: Hierarchical mobile edge computing network.

should never be assigned to the same edge server. Note that this number is not necessary R if an adversary can compromise multiple edge nodes at the same time.

The related optimization problem is a typical combinatorial search problem, which is NP-hard. In this case, sub-optimal solutions can be derived through heuristic algorithms. Classical heuristics include Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). In this chapter, based on characteristics of the optimization problem, we at first developed two basic heuristics, i.e. the top-down and bottom-up heuristic, which can outperform GA and PSO in a certain cases. Afterwards, we further integrated the above

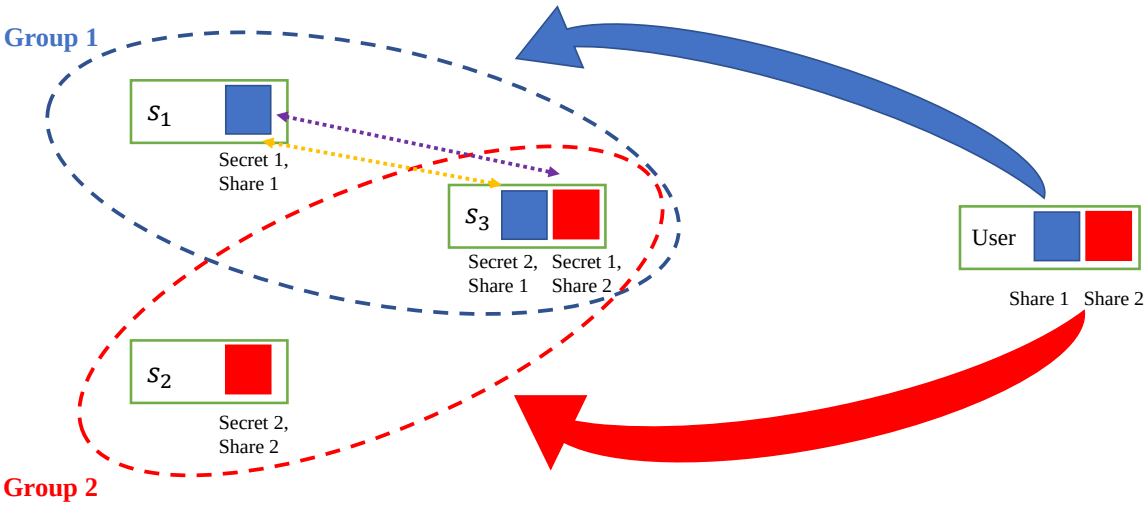


Figure 4.2: An example of privacy-preserving task scheduling based on MPC

two heuristics and proposed the Bottom-Up Top-Down ($BUTD$) heuristic, which can outperform all the above four heuristics when communication among different shares of the same secret is comparable to that among different secrets. The main contributions of this research project are as follows:

- Based on SS, MPC and the hierarchical edge computing network structure, we propose a communication-aware secret share placement strategy to minimize communication overhead in terms of weighted transmission hop counts among different secrets and shares. We show that the whole scheduling process can be modeled as an optimization problem with the computation and storage limitation and privacy constraint, which is actually an NP-hard problem. Heuristic algorithms can be developed to find the sub-optimal solution in this case.

- Based on the characteristics of the optimization problem, we develop two basic heuristic algorithms, i.e. the top-down and bottom-up heuristic, which respectively minimize weighted communication hop counts among different secrets with acceptable hop counts among different shares of the same secret, and vice versa. We systematically analyze that these two heuristics can outperform two classical heuristics, i.e. GA and PSO, in a

certain cases.

- We further integrate the top-down heuristic with bottom-up heuristic, and propose *BUTD*, which outperforms all the above four heuristics when communication among different shares of the same secret is comparable to that among different secrets. Therein, the bottom-up heuristic is applied together with consideration of remaining computation and storage space within a neighborhood at the initialization stage of the top-down heuristic.

The rest of this chapter is organized as follows. In Section 4.2, we discuss related works. In Section 4.3, we give a full description of our communication-aware privacy-preserving task scheduling system. Our two basic heuristics are presented in Section 4.4, followed by Section 4.5, which talks about our proposed *BUTD* heuristic. Strategies to simplify algorithm deployment are provided in Section 4.6. Experimental results and analysis are given in Section 4.7, and conclusions will be drawn in Section 4.9.

4.2 Related Works

MPC popularizes recently with the development of federated learning [3], in which a global machine learning model can be trained without the need of uploading raw data to public servers. However, contributors may be concerned that those uploaded gradients can still leak sensitive information to a certain extent [18–21]. Considering this, many recent works [22, 24, 25, 67] propose to further protect data to be uploaded via SS and MPC.

4.2.1 Secret Sharing and Secure Multi-Party Computation

MPC is usually believed to originate from Yao's garbled circuit protocol [87]. Later, SS is further introduced to improve the privacy-preserving level and combat package loss during transmission. A typical (R, L) SS scheme can tolerate an adversary controlling at most $R-1$ shares of a secret. Currently, the two most popular secret sharing schemes are Shamir [88] and additive secret sharing. Shamir secret sharing is secure against a passive adversary when $R < \frac{L}{2}$, while against an active adversary when $R < \frac{L}{3}$. Additive secret sharing can

tolerate an adversary controlling $L-1$ shares no matter whether the adversary is passive or active. Typical MPC schemes include [89–91]. Later, Damgård *et al.* [92] developed an MPC scheme, called *SPDZ*, which is unconditionally secure against active and adaptive corruption of up to $L-1$ of the L parties. This work is followed by two major improvements, i.e. *SPDZ2* [93] and *MASCOT* [94], which accelerate data preprocessing via advanced check of Message Authentication Code (MAC) and oblivious transfer, respectively.

Recently, federated learning gives SS and MPC new vitality considering high consilience between their deployment scenarios. Therein, Mohassel and Zhang [24] proposed a two-server model to have linear regression, logistic regression and neural network training, which was later extended to a three-server situation in [95]. Bonawitz *et al.* [22] systematically designed a secure aggregation protocol in federated learning based on SS and MPC. In [67], the authors further introduced differential privacy to parameter aggregation in federated learning, which can protect against collusion between public servers and local devices. Later, Li *et al.* [25] proposed a chained MPC structure to reduce communication overhead between public servers and local devices. In this structure, several local devices are grouped into a chain and only communicate with their neighbors. Only the very first and last local device will communicate with the public server. However, such a structure is vulnerable to packet loss or user dropout, and SS is hard to deploy here considering that those secret shares need to be broadcast to several other local devices.

4.2.2 Combinatorial Search

As has been discussed previously, SS and MPC usually suffer from high communication overhead. Recently, the proposal of edge computing can effectively lower it by offloading some computation tasks to nearby edge servers. However, how to deploy SS and MPC in edge computing still remains largely open.

Different from cloud computing, computation offloading strategies among edge nodes become critical in determining data processing latency and energy consumption, especially

considering that edge nodes have limited computation and storage capacity, and are distributed and hierarchically structured [31–34]. In [69], based on $(2, 2)$ SS and MPC, we designed a structure, called pairwise Markov chain, to realize proper computation offloading among edge nodes. The relevant optimization problem can be solved through linear programming.

In this research project, we generalize the $(2, 2)$ SS scheme to (R, L). We find that the relevant optimization problem can be considered as a combinatorial search problem, which is NP-hard. According to the taxonomy in [96], our problem is quite related to problems in *Group 9*, i.e. scheduling of single-user workflow jobs in cloud environments. Representative related works propose to use heuristic algorithms, such as genetic algorithm (GA) [97, 98], particle swarm optimization (PSO) [99, 100], partial critical paths (PCP) [101, 102], to find sub-optimal solutions to those problems. However, all the above works did not consider any privacy constraints. Besides, we will show in Section 4.5.1 that with a proper heuristic design, we can introduce some intrinsic restrictions to help us find better sub-optimal solutions to our optimization problem.

4.3 Model Construction and Problem Formulation

Fig. 4.2 shows a toy example of our targeted communication-aware secret share placement problem based on $(2, 2)$ SS. In this figure, a user wants to distribute two shares of its secret to a toy network with three public servers, which already has two secrets assigned. According to the share index, three servers are divided into two groups. We define the **intra-group** communication as all communication among shares of different secrets with the same index. An example is illustrated with a yellow double arrow dash line in Fig. 4.2. On the other hand, the **inter-group** communication includes all communication among different shares of the same secret. An example is illustrated with a purple double arrow dash line in Fig. 4.2. Our scheduling policy aims at helping the user distribute the two shares, and the final assignment result should have minimal intra-group and inter-group communication overhead.

4.3.1 System Model

Our system model design is based on the following three main assumptions: (1) The mobile edge computing has a hierarchical network structure illustrated in Fig. 4.1; (2) The number of field-level servers S is largely greater than L (represented by the symbol ">>" in Fig. 4.1), and secret shares will only be stored in those field-level servers; (3) Each level in the network has enough communication channels or access points (greatly larger than L) to meet requirements of the privacy constraint during secret share switching. Note that based on specific applications, the first two assumptions can be modified by just tuning the secret matrix $\boldsymbol{C}_a$ and share matrix $\boldsymbol{C}_b$ to be discussed below, which will not impact related algorithms in this research project. We are based on these assumptions for the simplicity of analysis. In addition, the third assumption can be realized through multipath routing protocols [103–105].

Let's then assume a job to schedule M secrets under an MPC scheme, which separates a secret to L shares. When the m-th secret is to be assigned, we use $\boldsymbol{a}_{m,l}$ to denote the index of the field-level server to be distributed to the l-th share of the secret. A secret share assignment vector $\boldsymbol{A}_m$ is further introduced to include all those $\boldsymbol{a}_{m,l}$, i.e. $\boldsymbol{A}_m = (\boldsymbol{a}_{m,1}, \boldsymbol{a}_{m,2}, \cdots, \boldsymbol{a}_{m,L})$. Besides, for all those $m-1$ assigned secrets, we introduce L sets of $B_{m,l}$ to include field-level server indices assigned to the l-th shares of those $m-1$ secrets, i.e. $B_{m,l} = \{\boldsymbol{a}_{1,l}, \boldsymbol{a}_{2,l}, \cdots, \boldsymbol{a}_{m-1,l}\}$. Considering the hierarchical network structure shown in Fig. 4.1, field-level servers will be indexed from the deep to field level. In detail, we have $\boldsymbol{a}_{m,l} = (d_{m,l}, i_{m,l}, s_{m,l}, f_{m,l})$, where $d_{m,l}$, $i_{m,l}$, $s_{m,l}$ and $f_{m,l}$ respectively represent the server index in the deep, intermediate, shallow and field level.

As has been mentioned before, our system model has two matrices to describe intra-group and inter-group communication overhead in terms of weighted communication hop counts, i.e. the secret and share matrix. In this chapter, these two matrices are respectively denoted as $\boldsymbol{C}_a$ and $\boldsymbol{C}_b$. Considering the hierarchical network structure, we assign different values or overhead to communication between two secret shares having different

server position relationships. We use $c_{\min}$ and $c_{\max}$ to represent the minimal and maximal communication overhead in terms of weighted hop counts in the hierarchical network structure. We assume that if two secret shares are distributed to the same field-level server, their communication has the minimal overhead. At the other extreme, if two secret shares are assigned to two different field-level servers allocated in different deep-level server clusters, their communication has the maximal overhead. In between, different communication overhead ranging from $c_{\min}$ to $c_{\max}$ will be assigned in different cases. In addition, based on (R, L) SS, if $R-1$ shares of a particular secret have been assigned to a particular field-level server, we will set the corresponding element in the diagonal of $\boldsymbol{C}_b$ to infinity, which enforces that any R shares of the same secret are not assigned to the same field-level server. Note that this setting is based on the assumption that an adversary can compromise only one edge node at a time, which needs to be tuned further if multiple edge nodes can be attacked at the same time. All the above settings will be realized through introducing two communication overhead coefficients, i.e. κ and ρ, described in Table 4.1, in which we have $1 < \kappa_1 < \kappa_2 < \kappa_3 < \kappa_{\max}$ and $1 < \rho_1 < \rho_2 < \rho_3 < \rho_{\max}$. In this research project, we will test a linear and exponential communication overhead increase with those two coefficients. In Section 4.3.2, we will provide the value of each element in $\boldsymbol{C}_a$ and $\boldsymbol{C}_b$ under those two kinds of communication overhead increase.

Table 4.1: Detailed Description of Value Assignments in $\boldsymbol{C}_a$ and $\boldsymbol{C}_b$

κ	ρ	Description
$\kappa_{\max}$	$\rho_{\max}$	Different deep-level server clusters, $d_{m_1,l_1} \neq d_{m_2,l_2}$.
κ_3	ρ_3	Different intermediate-level server clusters of the same deep-level server cluster, $d_{m_1,l_1} = d_{m_2,l_2}$ but $i_{m_1,l_1} \neq i_{m_2,l_2}$.
κ_2	ρ_2	Different shallow-level server clusters of the same intermediate-level server cluster, $i_{m_1,l_1} = i_{m_2,l_2}$ but $s_{m_1,l_1} \neq s_{m_2,l_2}$.
κ_1	ρ_1	Different field-level servers of the same shallow-level server cluster, $s_{m_1,l_1} = s_{m_2,l_2}$ but $f_{m_1,l_1} \neq f_{m_2,l_2}$.
1	1 or ∞	The same field-level server, $f_{m_1,l_1} = f_{m_2,l_2}$.

Note that the above value assignments are based on the assumptions discussed previously. We can actually further tune $\boldsymbol{C}_a$ and $\boldsymbol{C}_b$ to satisfy different network structures and enforce various security and privacy constraints. Heterogeneous communication links and conditions can also be encompassed. Moreover, we can take into account the data size factor during transmission, since the more data need to be carried during a transmission, the less hop counts we hope to have. In this research project, a weight w_s is introduced to represent the data size ratio between intra-group and inter-group communication.

4.3.2 Optimization Problem Formulation

With the above system model, we can formulate the following optimization problem to minimize communication overhead in terms of weighted hop counts

$$\arg\min_{\boldsymbol{A}_m} \sum_{l=1}^{L} D_a(\boldsymbol{a}_{m,l}, B_{m,l}) + w_s \cdot D_b(\boldsymbol{A}_m) \tag{4.1a}$$

subject to,

$$\forall s \in \{1, 2, \cdots S\}, \quad q_s < Q \tag{4.1b}$$

In (4.1a), the first and second term respectively correspond to intra-group and inter-group communication, which can be further decomposed to

$$D_a(\boldsymbol{a}_{m,l}, B_{m,l}) = \sum_{m'=1}^{m-1} \boldsymbol{C}_a(\boldsymbol{a}_{m,l}, \boldsymbol{a}_{m',l}) \tag{4.2}$$

and

$$D_b(\boldsymbol{A}_m) = \sum_{\substack{l_1<l_2 \\ l_1,l_2\in\{1,2,\cdots L\}}} \boldsymbol{C}_b(\boldsymbol{a}_{m,l_1}, \boldsymbol{a}_{m,l_2}) \tag{4.3}$$

(4.1b) defines the computation and storage limitation of each field-level server in terms of queue length. In this work, all field-level servers are assumed to have the same computation and storage limitation, which can buffer at most Q shares at the same time. Such an assumption can also be modified by changing the value of Q in each field-level server.

As has been mentioned in Section 4.3.1, we test two kinds of communication overhead increase in this research project. For linear increase, $\boldsymbol{C}_a$ and $\boldsymbol{C}_b$ are defined as follows

$$\boldsymbol{C}_a(\kappa) = \frac{c_{\max} - c_{\min}}{\kappa_{\max} - 1} \cdot \kappa + \frac{\kappa_{\max} \cdot c_{\min} - c_{\max}}{\kappa_{\max} - 1} \tag{4.4a}$$

$$\boldsymbol{C}_b(\rho) = \frac{c_{\max} - c_{\min}}{\rho_{\max} - 1} \cdot \rho + \frac{\rho_{\max} \cdot c_{\min} - c_{\max}}{\rho_{\max} - 1} \tag{4.4b}$$

The exponential increase has the following form

$$\boldsymbol{C}_a(\kappa) = \left(\frac{(c_{\min})^{\kappa_{\max}}}{c_{\max}}\right)^{\frac{1}{\alpha_{\max}-1}} \times e^{\frac{\kappa}{\kappa_{\max}-1} \cdot \ln \frac{c_{\max}}{c_{\min}}} \tag{4.5a}$$

$$\boldsymbol{C}_b(\rho) = \left(\frac{(c_{\min})^{\rho_{\max}}}{c_{\max}}\right)^{\frac{1}{\rho_{\max}-1}} \times e^{\frac{\rho}{\rho_{\max}-1} \cdot \ln \frac{c_{\max}}{c_{\min}}} \tag{4.5b}$$

Here, $\kappa_{\max}$ and $\rho_{\max}$ respectively represent the maximal value of κ and ρ. Note that here we directly use κ and ρ as parameters in $\boldsymbol{C}_a$ and $\boldsymbol{C}_b$ for simplicity. In fact, we further have an intrinsic mapping from $\boldsymbol{C}_a(\boldsymbol{a}_{m,l}, \boldsymbol{a}_{m',l})$ to $\boldsymbol{C}_a(\kappa)$, and from $\boldsymbol{C}_b(\boldsymbol{a}_{m,l_1}, \boldsymbol{a}_{m,l_2})$ to $\boldsymbol{C}_b(\rho)$, which have already been fully described in Table 4.1.

Unfortunately, the above optimization problem is a typical combinatorial search problem, which has long been proven an NP-hard problem. The search space for this problem is S^L. In other words, the search space will grow exponentially with the number of field-level servers, which can be further expanded by increasing the number of secret shares. In this case, efficient heuristic algorithms need to be developed to find sub-optimal solutions to this problem.

4.4 Basic Heuristic Algorithms

In this section, we will list all the four basic heuristic algorithms evaluated in this research project. The first two are classical heuristics, i.e. *GA* and *PSO*. Based on characteristics of the optimization problem, we additionally propose two heuristics, called the top-down and bottom-up heuristic.

4.4.1 Genetic Algorithm

Genetic algorithm (*GA*) was firstly proposed by Holland [106]. During initialization, a population with P individuals are randomly generated within a certain range. As to our optimization problem, each individual corresponds to one possible secret share assignment scheme $\boldsymbol{A}_m = \{x_{m,1}, x_{m,2}, \cdots, x_{m,L}\}$. Note that different from the representation in Section 4.3.1, the position of each server is indexed with a scalar $x_{m,l}$ here, which can bring convenience to secret assignment update to be discussed below. When assigning each secret based on *GA*, we will at first randomly generate P such $\boldsymbol{A}_m$. Each element or gene $x_{m,l}$ in $\boldsymbol{A}_m$ is an integer from 1 to S.

After initialization, *GA* will jump into the breeding process, which is composed of a given number of iterations or generations. In each generation, a children population with the same number of individuals P will be generated from the original population through parent selection, crossover and mutation. Here, we use tournament selection to select parents with the lowest fitness value (lowest communication overhead in our optimization problem). Afterwards, children are generated via uniform crossover and mutation based on a normal distribution $\mathcal{N}(0, \sigma^2)$ with a mean value of 0 and a pre-defined standard deviation σ.

GA will terminate after reaching a pre-defined maximal generations. Afterwards, all the best children in each generation will be further compared to obtain the overall best child, which is considered as the sub-optimal solution to our optimization problem.

4.4.2 Particle Swarm Optimization

Particle swarm optimization (*PSO*) is another well-known heuristic algorithm, which was firstly proposed by Kennedy and Eberhart [107]. *PSO* is modeled based on swarming or flocking behaviors in animals. Different from *GA*, which resamples the whole population in each iteration or generation, *PSO* maintains a static population or swarm throughout the whole iteration process, and tweaks each individual or particle in the swarm in response to new discoveries about the space. As to our optimization problem, each possible secret share assignment scheme $\boldsymbol{A}_m = (x_{m,1}, x_{m,2}, \cdots, x_{m,L})$ can be considered as a particle, and *PSO* can be applied here to find the sub-optimal scheduling scheme to our problem.

In detail, similar to *GA*, *PSO* will randomly generate a swarm $\boldsymbol{A}_m$ with P particles during initialization. Afterwards, *PSO* tweaks each particle in the swarm by introducing a velocity set $v_m = \{v_{m,1}, v_{m,2}, \cdots, v_{m,L}\}$, in which each element is impacted by four factors, i.e. $v^*_{m,l}$ – velocity of particle $x_{m,l}$ in the last iteration, $x^*_{m,l}$ – the best position of particle $x_{m,l}$ up to now, $x^+_{m,l}$ – the best position in the neighborhood of particle $x_{m,l}$ up to now, and $x^!_m$ – the best position of any particles up to now. Levels of influence to the velocity of those four factors are defined by four parameters ι, ι^*, ι^+ and $\iota^!$. In each iteration, values of those four parameters are randomly generated between 0 and pre-defined ι_0, ι^*_0, ι^+_0 and $\iota^!_0$, respectively. Then, the velocity in each iteration is updated as follows

$$v_{m,l} \leftarrow \iota v^*_{m,l} + \iota^*(x^*_{m,l} - x_{m,l}) + \iota^+(x^+_{m,l} - x_{m,l}) + \iota^!(x^!_m - x_{m,l}) \tag{4.6}$$

where $v^*_{m,l}$ will be given a random value between a pre-defined $-r$ and r inclusively during initialization. Afterwards, each particle will be updated by $x_{m,l} \leftarrow x_{m,l} + \epsilon v_{m,l}$, where ϵ represents the step size of each particle. Currently, $x^!_m$ is usually not considered by $\iota^!_0 \leftarrow 0$ to make the algorithm less likely to get stuck in local optima. Besides, ϵ is commonly set to 1. Considering that $x_{m,l}$ should be discrete, a rounding operation will be applied additionally.

Similar to *GA*, *PSO* will terminate after reaching the pre-defined maximal iterations.

It will also record the best particle in each iteration, and the overall best particle will be selected by comparing all those best particles, which is considered as the sub-optimal solution to our optimization problem.

4.4.3 Top-Down and Bottom-Up Heuristic Algorithm

Up to now, we have introduced two classical heuristic algorithms, which can be applied to our optimization problem. In this chapter, we propose another two basic heuristic algorithms based on the characteristics of our targeted optimization problem, which are respectively called the top-down and bottom-up heuristic algorithm.

The basic idea of these two heuristics is as follows. As is shown in (4.1a), our optimization problem has two components, i.e. the intra-group communication $\sum_{l=1}^{L} D_a(\boldsymbol{a}_{m,l}, B_{m,l})$ and inter-group communication $D_b(\boldsymbol{A}_m)$. In those two basic heuristics designed by us, we only consider either one of those two components and omit the other one. Then, the optimization problem will become fully decomposable to several sub-problems, and the search space will be greatly reduced from S^L to SL, which makes the brute-force method practical to obtain the optimal solution. Obviously, the so-called optimal solution is still sub-optimal when the omitted component is considered again. However, we will show later that these two heuristics can sometimes outperform *GA* and *PSO*.

Let's take a look at the top-down heuristic at first. In this research project, we consider the intra-group communication as the top-level communication, while the inter-group as the bottom-level. The top-down heuristic only considers the intra-group communication. As has been discussed in Section 4.3, the intra-group communication includes all communication among shares of different secrets with the same index, which are combined together to generate a group with that index. In this case, under the top-down heuristic, the optimization problem can be fully decomposed to L sub-problems for each secret to be assigned $\boldsymbol{A}_m = (x_{m,l}, x_{m,l}, \cdots, x_{m,L})$. In each sub-problem, we try to find the best server assignment $x_{m,l}$, which achieves the minimal sum of communication overhead to all the other members in Group l. With the brute-force method, we will search all those S field-level servers in the

network. Without the consideration of inter-group communication, those N sub-problems are independent of each other, and we can solve them one after another, which will only incur a linear search space of SL. This makes the brute-force method applicable. On the other hand, our optimization problem has a privacy constraint that a certain number of shares of the same secret should not be assigned to the same server. Therefore, if scheduling repetition passes the given privacy threshold, we will reschedule the current share to the second best server, and so on. Furthermore, the assignment of the first secret is generated randomly during initialization. Details of the top-down heuristic are summarized in Algorithm 3. Note that we use the scalar form to represent all the server positions here, i.e. $\boldsymbol{a}_{m,l} \mapsto x_{m,l}$. Such a mapping will be further discussed in Section 4.6.1.

Algorithm 3: *Top-down Heuristic.*

```
for l = 1 to L do
    Overhead ← 0;
    Best_Overhead ← ∞;
    for s = 1 to S do
        for m' = 1 to m − 1 do
            Overhead ← Overhead + C_a(s, x_{m',l});
        if Overhead < Best_Overhead then
            x_{m,l} ← s;
            Best_Overhead ← Overhead;
return A
```

Contrary to the top-down heuristic, the bottom-up heuristic only considers the inter-group or bottom-level communication. As has also been discussed in Section 4.3, the inter-group communication includes all communication among different shares of the same secret. Similar to the top-down heuristic, the bottom-up heuristic can also make our optimization problem fully decomposable to L independent sub-problems, which can be solved by searching an SL space with the brute-force method. Differently, the bottom-up heuristic will try to find the best server assignment $x_{m,l}$, which achieves the minimal sum of communication overhead to all the other shares in the same secret. Note that considering the distance is mutual, when assigning $x_{m,l}$, we just need to check its hop counts to $x_{m,1}$, $x_{m,2}$, $\cdots x_{m,l-1}$,

which have already been assigned. $x_{m,1}$ will be assigned randomly during initialization. Afterwards, the bottom-up heuristic will apply a sorting function towards $\boldsymbol{A}_m$ to further minimize intra-group communication overhead, which is realized with an $O(L^3)$ computation complexity in this work, and will be discussed more specifically in Section 4.6.2. Details of the bottom-up algorithm are described in Algorithm 4.

Algorithm 4: *Bottom-up Heuristic.*

```
1  x_{m,1} ← a random server between 1 and S;
2  for l = 2 to L do
3      Overhead ← 0;
4      Best_Overhead ← ∞;
5      for s = 1 to S do
6          for l' = 1 to l − 1 do
7              Overhead ← Overhead + C_b(x_{m,l'}, s);
8          if Overhead < Best_Overhead then
9              x_{m,l} ← s;
10             Best_Overhead ← Overhead;
11 return Sort(A_m)
```

Fig. 4.3 shows a comparison among *GA*, *PSO* and two basic heuristics proposed by us for a job of scheduling 10 secrets, each of which is separated into 10 shares. Linear overhead growth is utilized. The results are scaled by the number of secrets, i.e. 10 in Fig. 4.3, and sorted according to the overall communication overhead. From the figure, we can see that in both cases, *GA* and *PSO* achieve similar performance. Our top-down heuristic achieves the best performance when $w_s = 1$, while the bottom-up heuristic begins to outperform the other three heuristics when $w_s = 3$.

Actually, such trends aggravate when the weight keeps decreasing from 1 or increasing from 3. The reason is clear why the top-down heuristic can outperform bottom-up heuristic or vice versa, since intra-group or inter-group communication are assumed to carry more data, and those two heuristics respectively consider one of those two kinds of communication. Besides, the trends can be further changed if we consider different numbers of secrets and shares in a secret. For a job of scheduling M secrets, each of which is separated to L shares,

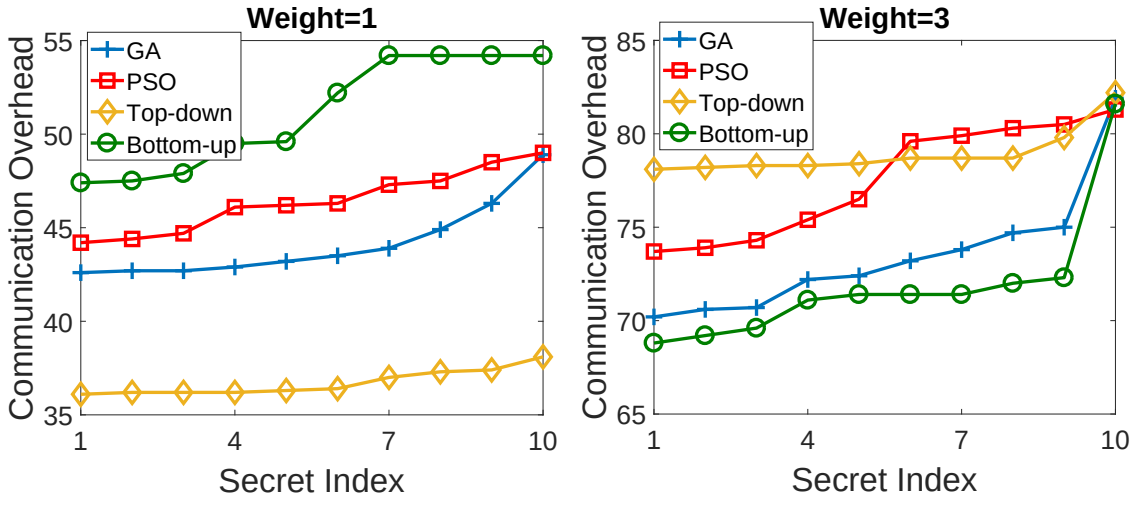


Figure 4.3: Comparison among GA, PSO, top-down and bottom-up method.

the number of intra-group and inter-group communications are respectively $(M-1)L$ and $\frac{L(L-1)}{2}$. Therefore, increasing the number of secrets will favor the top-down heuristic, since M is only related to the intra-group communication. The bottom-up heuristic will benefit from rising the number of shares, because the inter-group communication has a quadratic relationship with L, while the intra-group communication just keeps a linear increment towards L. On the other hand, it is also interesting to see that these two basic heuristics can outperform *GA* and *PSO* in some cases, which are two of most popular heuristic algorithms. In addition, running time can be greatly reduced with the search space shrunk from S^L to SL. In the example shown in Fig. 4.3, running time drops from the order of seconds to milliseconds. This further makes the two basic heuristics stand out.

4.5 Advanced Heuristic Algorithm

In this section, we will at first analyze why our two basic heuristics can sometimes outperform GA and PSO. Afterwards, we further propose an advanced heuristic algorithm, called the bottom-up top-down (BUTD) heuristic, which can outperform all those four heuristics

when intra-group communication is comparable to inter-group communication.

4.5.1 Performance Analysis

As has been discussed in Section 4.4.3, our two basic heuristics can outperform the other two classical heuristics, i.e. *GA* and *PSO*, in some situations of our optimization problem. Actually, even when communication among different shares of the same secret is comparable to that among different secrets, those two basic heuristics can still achieve performance similar to *GA* and *PSO* sometimes. This is caused by the intrinsic restrictions introduced to those two basic heuristics.

As to the top-down heuristic, let's at first consider the case where all servers have no computation and storage limitation or the queue length $Q = \infty$. In this situation, the shares of different secrets with the same index should be scheduled to exactly the same server, since we only considers the intra-group communication. When we look back to the inter-group communication, we will notice that all the following secrets have exactly the same inter-group communication distance pattern as that of the first secret. In other words, all the secrets will have exactly the same inter-group communication overhead sum. When the computation and storage limitation is introduced, though not all the shares of different secrets with the same index can be scheduled to exactly the same server, they will still be assigned to a neighborhood. Under this circumstance, all the following secrets should still have an inter-group communication overhead sum similar to that of the first secret. This is called the intrinsic restriction of the top-down heuristic. Such a restriction will make the inter-group communication overhead sum *acceptable*. In other words, although the top-down heuristic only considers the intra-group communication during optimization, there is an intrinsic traction force to ensure that all the shares of the same secret not drift away from each other too much. This will make our top-down heuristic outperform *GA* and *PSO*, which change the secret share assignment A_m completely free of control, especially in cases with a small w_s, as is shown in Fig. 4.3(a).

Similar to the top-down heuristic, the bottom-up heuristic also has such an intrinsic

restriction, which is realized via the sorting function mentioned in Section 4.4.3. For each secret, the sorting function will pair each assigned server with the closest group to make the intra-group communication overhead sum *acceptable*.

4.5.2 Bottom-up Top-down (*BUTD*) Heuristic

From the above discussion, we can see that the intrinsic restrictions in our two basic heuristics can help to find a better sub-optimal solution to our optimization problem in different situations. On the other hand, the performance of those two heuristics largely depends on the first secret or share assignment, which are generated randomly during initialization. Furthermore, we notice that the assignment of the first share in the first secret does not matter, since intra-group communication does not need to be considered at this time. In this case, if we use the bottom-up heuristic for the first secret assignment, and all the following secrets are scheduled based on the top-down heuristic, ideally, we can achieve a job scheduling scheme with a global optimal solution. This is the general idea of *BUTD*.

However, as has been discussed previously, since field-level servers in the network have computation and storage limitation, when the queue length of a particular server reaches the given threshold Q, we can no longer schedule secret shares to this server. Instead, they can only be assigned to another server within a particular neighborhood. In the extreme case, later secret shares have to be switched to a server in another deep-level server cluster, which is far away from all the other shares in the same group, when all the closer servers have been fully occupied. This will cause large intra-group communication overhead. Therefore, in order to further reduce the overall communication overhead, it is important to find a sparse neighborhood for each share when scheduling the first secret.

In *BUTD*, we additionally consider the remaining computation and storage space in the neighborhood, and generate the following optimization problem at the initialization stage

of the top-down heuristic.

$$\arg\min_{\boldsymbol{A}_1} \sum_{\substack{l_1<l_2 \\ l_1,l_2\in\{1,2,\cdots L\}}} \boldsymbol{C}_b(\boldsymbol{a}_{1,l_1}, \boldsymbol{a}_{1,l_2}) - w_q \cdot \sum_{l=1}^{L} \sum_{s\in Ne(\boldsymbol{a}_{1,l})} Q - q_s \tag{4.7}$$

Here, we use $\boldsymbol{A}_1$ to denote the first secret to be assigned in a job. w_q is a weight to balance the communication overhead and remaining computation and storage space factor. $Ne(\boldsymbol{a}_{1,l})$ represents the given neighborhood of the server scheduled to the l-th share. In this research project, we are based on the hierarchical network structure discussed previously, and consider that the neighborhood of a particular field-level server is composed of all the field-level servers in the same shallow-level server cluster, including that particular server itself. Note that we use a subtraction instead of an addition between communication overhead and queue length information in (4.7), since neighborhoods with larger remaining computation and storage space, a.k.a. sparser neighborhoods, are preferred. Besides, we do not need to worry about that $Q - q_s$ becomes negative as long as there is still spare space in the network. After assigning the first secret, *BUTD* will continue scheduling the following secrets based on the top-down heuristic. The optimization problem described by (4.7) is actually another combinatorial search problem. Heuristics, e.g. *GA* and *PSO*, can further be used here to find the sub-optimal solution.

4.6 Algorithm Deployment Strategies

In this section, we will discuss two strategies introduced in this research project to improve computational efficiency and simplify algorithm deployment.

4.6.1 Server Index Mapping

In Section 4.3.1, based on the hierarchical network structure, the server assigned with the l-th share of a secret is indexed by a vector $\boldsymbol{a}_{m,l} = (d_{m,l}, i_{m,l}, s_{m,l}, f_{m,l})$, where $d_{m,l}$, $i_{m,l}$,

$s_{m,l}$ and $f_{m,l}$ respectively represent the server index in the deep, intermediate, shallow and field level. However, when we apply those heuristic algorithms in this research project, we treat the index of each server as a scalar $x_{m,l}$ for simplicity of computation. A mapping needs to be defined in this case, which is denoted as follows.

$$x_{m,l} \leftarrow D_{m,l} + I_{m,l} + S_{m,l} + f_{m,l} \tag{4.8}$$

Here, $D_{m,l}$ describes the total number of field-level servers in those $d_{m,l} - 1$ deep-level server clusters. $I_{m,l}$ is the total number of field-level servers in those $i_{m,l} - 1$ intermediate-level server clusters of the $d_{m,l}$ deep-level server cluster. $S_{m,l-1}$ represents the total number of field-level servers in those $s_{m,l} - 1$ shallow-level server clusters of the $i_{m,l}$ intermediate-level server cluster. Finally, we add the field-level server index $f_{m,l}$ to map the vector $\boldsymbol{a}_{m,l}$ to the scalar $x_{m,l}$.

Such a mapping and its reverse can be stored in the system via two tables in advance. We can directly call them when needed during job scheduling, which can greatly improve computational efficiency. Besides, we also expand both the original $\boldsymbol{C}_a$ and $\boldsymbol{C}_b$ to two-dimensional matrices in accordance with the scalar form server index.

4.6.2 Dynamic Matrix

In Section 4.4.3, we discussed that the bottom-up heuristic needed a sorting function to further minimize intra-group communication overhead. We realize this function through a dynamic communication overhead matrix.

Let's denote the m-th secret assignment result before sorting as $\boldsymbol{A}'_m = (x'_{m,1}, x'_{m,2}, \cdots, x'_{m,L})$. For each $x'_{m,l}$, we calculate the intra-group communication overhead in terms of each $B_{m,l'}$, $l' = 1, 2, \cdots, L$. Then, we can construct an $L \times L$ matrix $\boldsymbol{C}_{L\times L}$ to include all calculated overhead. Afterwards, the bottom-up heuristic will scan this matrix for L times to find the current minimal element $c_{i,j}$ in it. The row and column index pair of this minimal element (i, j) corresponds to the field-level server $x'_{m,i}$ and its corresponding group $B_{m,j}$. Then, we

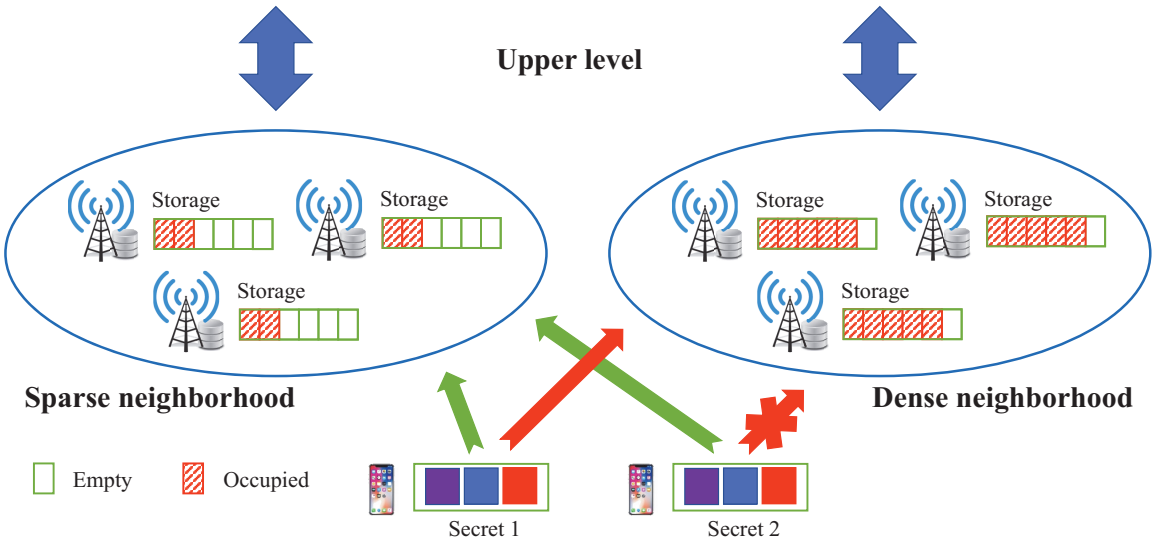


Figure 4.4: Dynamic communication overhead matrix.

will set $x_{m,j} = x'_{m,i}$. Each time the algorithm find the current minimal element $c_{i,j}$, it will set all the elements in the i-th row and j-th column of this matrix to infinity. In this case, it can avoid the paired field-level servers and groups being selected again. Since each scan will go through all the elements in the $L \times L$ matrix, and totally L scans are needed to pair all servers in $\boldsymbol{A}'_m$, the computation complexity of this sorting function is $O(L^3)$. Fig. 4.4 shows an example of dynamically changing the constructed communication overhead matrix. Note that such a sorting will not necessarily realize globally optimal pairing of servers and groups, but can make our bottom-up heuristic stay polynomial instead of going exponential.

Such an idea is also applied to our server computation and storage limitation enforcement. When the queue length in a server reaches the given threshold, we will set related elements in the secret matrix $\boldsymbol{C}_a$ and share matrix $\boldsymbol{C}_b$ to infinity. However, instead of setting all the elements in both the corresponding row and column to infinity, only elements in the corresponding row need to be set to infinity. For example, suppose the s-th server between 1 and S reaches the given queue length threshold. Then, elements in the s-th row of $\boldsymbol{C}_a$ and $\boldsymbol{C}_b$ will be set to infinity.

4.7 Simulation and Evaluation

In this section, we will evaluate all the heuristics discussed above with comprehensive simulations on an Intel(R) Core(TM) 3.4 GHz 4-core processor with 16 GB RAM. The simulation platform is MATLAB R2017a.

4.7.1 Simulation Setup

In this research project, we will consider two experimental settings, which are in general based on [31, 34]. The first one is a small-scale hierarchical edge computing network with a total number of 128 field-level servers, while a large-scale hierarchical edge computing network with 1,600 field-level servers is generated for the second setting. When it comes to the hierarchical network structure, in each experimental setting, we assume that all server clusters in the same level have the same capacity. For example, all shallow-level server clusters are assumed to have the same number of field-level servers. In the small-scale setting, we consider a job of scheduling 10 secrets, while the large-scale simulation considers a job of scheduling 600 secrets. The computation and storage limitation or queue length threshold Q of each field-level server is set to 5 for both situations. In the simulations, we assume that both these two networks have enough spare space for their corresponding job. In detail, during initialization, each field-level server in the small-scale environment is assumed to have a random number of tasks less than or equal to $Q-1$ which are already in the queue, while the randomly generated number is less than or equal to $Q-4$ in the large-scale environment.

When it comes to the privacy constraint, we consider a $(2, 10)$ SS scheme in the simulations, which can help to compare the performance of different heuristics in the most stringent environment. In such a scheme, all secrets in these two simulations are assumed to be separated into 10 shares, and all the diagonal elements in $\boldsymbol{C}_b$ need to be set to infinity at the very beginning to enforce that any two shares of the same secret should not be assigned to the same field-level server. The maximal and minimal communication overhead, i.e. $c_{\max}$ and $c_{\min}$, are set to 1 and 5 respectively in the experiments. Besides, we set

$\kappa_1 = \rho_1 = 2$, $\kappa_2 = \rho_2 = 3$, $\kappa_3 = \rho_3 = 4$ and $\kappa_{\max} = \rho_{\max} = 5$. Detailed description of the experimental settings are shown in Table 4.2. Note that heuristic algorithms can have performance oscillations when scheduling secret shares. Since we only need to evaluate the general trend, all the curves in the experiments are actually sorted overall communication overhead.

Table 4.2: Experimental Settings

Parameters	Small Scale	Large Scale
Number of Deep-Level Server Clusters	2	2
Number of Intermediate-Level Server Clusters	4	10
Number of Shallow-Level Server Clusters	4	4
Number of Field-Level Servers	4	20
Total Number of Servers	128	1,600
Number of Secrets	10	600
Number of Shares	10	10
Queue Length Limitation	5	5

4.7.2 Parameter Selection

As has been shown in previous sections, the heuristic algorithms introduced in this research project have several parameters to be tuned. A proper parameter setting can greatly improve the performance of those heuristics. Therefore, we will at first explore parameter selection. In this work, we only consider parameter selection in the small-scale environment, and will conduct a reasonable extension to parameters in the large-scale setting.

Parameters in *GA* and *PSO*

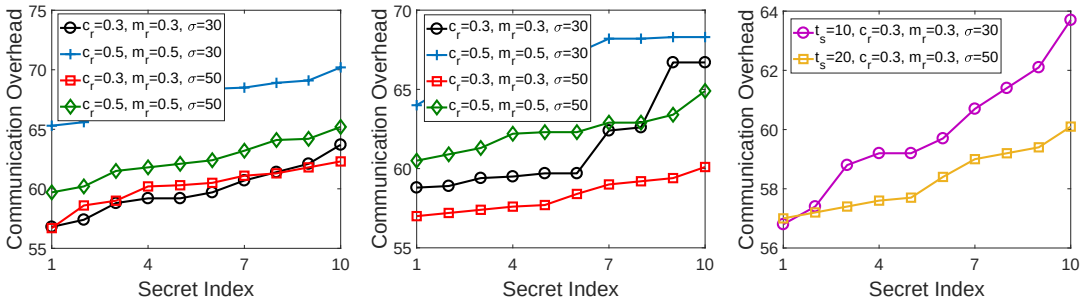
The performance of *GA* and *PSO* depend on several parameters. Therein, some of them have general guidance. In *GA*, such parameters are population size and number of generations, while in *PSO*, such parameters are swarm size and number of iterations. For these

parameters, we will use the recommended settings in [108]. Therein, population and swarm size are set to the minimum between 100 and $10 \times L$, which are 100 in this research project. The number of generations and iterations are set to $200 \times L$, which are 2,000 in this research project.

Except four parameters listed above, the other parameters depend on specific situations, which are comprehensively evaluated in this research project. For *GA*, we have tournament size t_s, crossover rate c_r, mutation rate m_r and standard deviation σ. For *PSO*, we evaluate the initial velocity v_0^* (with regard to the range between $-r$ and r inclusively), inertia ι, self adjustment weight ι^*, social adjustment weight ι^+ and neighbor size n_e. Large numbers of settings are tested in the simulations. In each setting, the performance is evaluated based on the average of 30 runs. Both linear and exponential communication overhead increase are considered, and w_s is set to 2. Some settings achieving comparatively good performance under linear communication overhead increase are illustrated in Fig. 4.5 and Fig. 4.6. Exponential counterparts have similar results, which are omitted here. Based on the above comparisons, generally, *GA* achieves the best performance when $t_s = 20$, $c_r = 0.3$, $m_r = 0.3$ and $\sigma = 50$, while *PSO* has the least communication overhead when $r = 5$, $\iota = 0.6$, $\iota^* = 1.5$ and $\iota^+ = 1.5$. These two parameter settings are used in the following simulations of the small-scale environment and as the basis of the parameter extension in the large-scale hierarchical network structure.

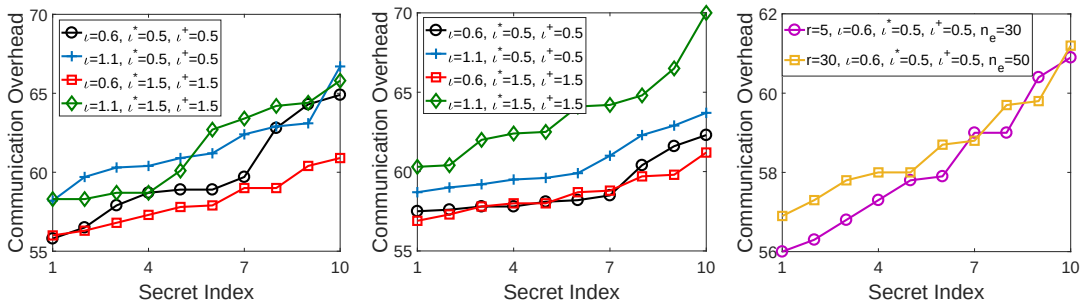
Weight Selection

In the previous sections of this chapter, we introduced two weights to describe the importance between two factors in the objective functions, i.e. w_s to represent the data size ratio between intra-group and inter-group communication, and w_q to balance the communication overhead and remaining computation and storage space. Therein, w_s is pre-determined by the job to be scheduled while w_q can be manually tuned. As has been discussed in Section 4.4.3, when w_s is small or large enough, intra-group communication will overwhelm inter-group communication or vice versa. In those situations, the top-down and bottom-up



(a) (b) (c)

Figure 4.5: *GA* performance comparison under linear communication overhead increase and different parameter settings. (a) $t_s = 10$; (b) $t_s = 20$; (c) Comparison of the best parameter setting in (a) and (b).

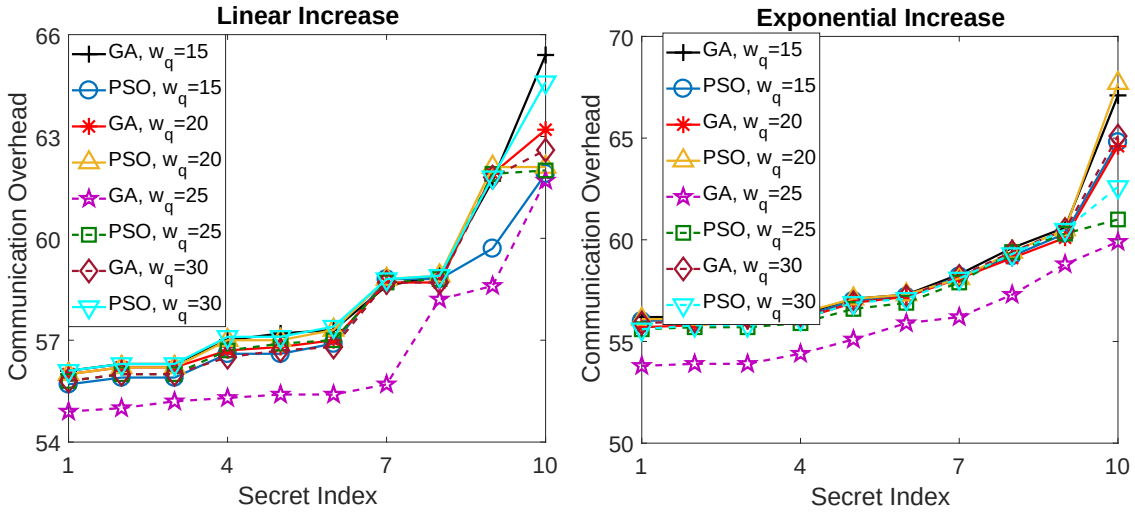


(a) (b) (c)

Figure 4.6: *PSO* performance comparison under linear communication overhead increase and different parameter settings. (a) $r = 5$ and $n_e = 30$; (b) $r = 30$ and $n_e = 50$; (c) Comparison of the best parameter setting in (a) and (b).

heuristic will perform the best respectively. On the other hand, when intra-group communication overhead has the similar weight to inter-group communication, the *BUTD* heuristic will outperform all the other 4 heuristics.

Based on Fig. 4.3, we can see that in the small-scale setting, when w_s is between 1 and 3, the top-down and bottom-up heuristic have similar performance. The reason is as follows. In the small-scale setting, we have the number of secrets $M = 10$ and the number of shares $L = 10$. In this case, the number of intra-group and inter-group communications are respectively $(M-1)L = 90$ and $\frac{L(L-1)}{2} = 45$. In other words, in order to make inter-group



(a) (b)

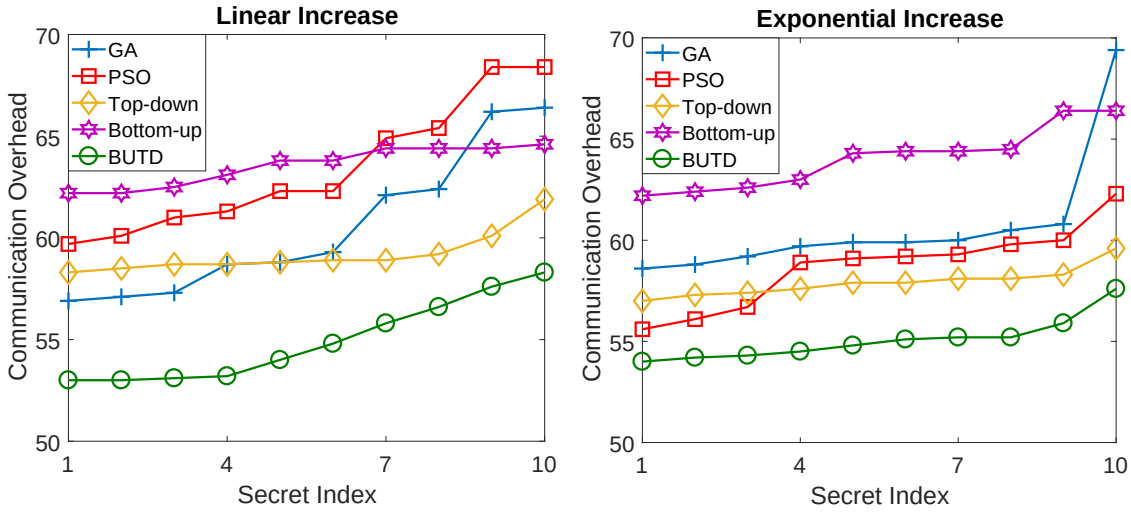
Figure 4.7: BUTD performance comparison under different initialization schemes and w_q.

communication have exactly the same weight as intra-group communication, we should have $w_s = 2$. When w_s is around 2 or between 1 and 3, those weights are similar to each other, and the top-down and bottom-up heuristic should perform comparably. In addition, as has been mentioned previously, the *BUTD* heuristic should perform the best in this range.

In this section, based on the case $w_s = 2$, we further explore the most appropriate value of w_q. Fig. 4.7 shows some candidate w_q in *BUTD*. Therein, both *GA* and *PSO* are considered at the initialization stage. The best parameter setting for *GA* and *PSO* derived in the last section are used. We also evaluate both the linear and exponential communication overhead increase. From these figures, we can see that in both two kinds of communication overhead increase, the best performance is achieved when *GA* is used during initialization, and w_q is set to 25.

4.7.3 Small-Scale Experimental Setting

In this section, we give an exhaustive performance comparison among all heuristic algorithms in this research project. The comparison is based on the best parameter settings discovered in the last section.



(a) (b)

Figure 4.8: Performance comparison in the small-scale environment.

Performance and Running Time

Fig. 4.8 compares the brought communication overhead under all the heuristic algorithms in this research project and the small-scale environment. In accordance with the parameter settings which we derived in the previous section, the weight w_s is set to 2. From the figure, we can see that in general, our proposed *BUTD* heuristic brings the least communication overhead in both the linear and exponential increase. Besides, compared with *PSO* and *GA*, the bottom-up, top-down and *BUTD* heuristic have a more steady performance. The performance of *GA* oscillates a lot when scheduling some secrets, which results in obvious increase at the tail of the corresponding curves. On the other hand, the performance of *PSO* can vary significantly between the linear and exponential increase. These results further validate the effect of intrinsic restrictions discussed in Section 4.5.1. Since *GA* and *PSO* are totally free to explore the whole space, their performance has much more randomness compared with the other three heuristics.

We further compare the running time of those five heuristic algorithms, whose results are shown in Table 4.3. From this table, we can see that the top-down, bottom-up and *BUTD*

heuristic take much less time than GA and PSO. This is because the search space of GA and PSO is S^L, while the other three heuristics only needs SL. Besides, GA is even more time-consuming than PSO due to the parent selection in each generation. Furthermore, utilizing GA during the initialization of $BUTD$ kind of increases its running time.

Table 4.3: Running Time Comparison

Heuristics	GA	PSO	Top-down	Bottom-up	$BUTD$
Time (s)	259.01	39.47	0.027	0.042	20.33

Secret Share Scheduling Distribution

In this section, we validate based on simulations that our $BUTD$ heuristic can effectively find comparatively sparse neighborhoods to schedule secrets. Our small-scale experimental setting has the following modifications in order for a better illustration. We assume that each field-level server can have at most $Q = 10$ secret shares in queue. Besides, we further assume that the first 10 field-level servers are totally empty at the beginning, while all the other servers have a queue length between 6 and $Q - 1$. Let's still consider the small-scale job of scheduling 10 secrets, each of which is further separated to 10 shares. Ideally, all the secret shares should be assigned to the first 10 empty servers, since those servers just have enough spare space, and both the minimal intra-group and inter-group communication overhead can be realized.

Fig. 4.9 shows a comparison of secret share scheduling distributions under all the five heuristics in this work. From this figure, we can see that our $BUTD$ heuristic can effectively find the sparse space, i.e. the first 10 field-level servers, and schedule most of secret shares to that region, while the other four heuristics are inclined to assign secret shares more uniformly, which can induce higher communication overhead.

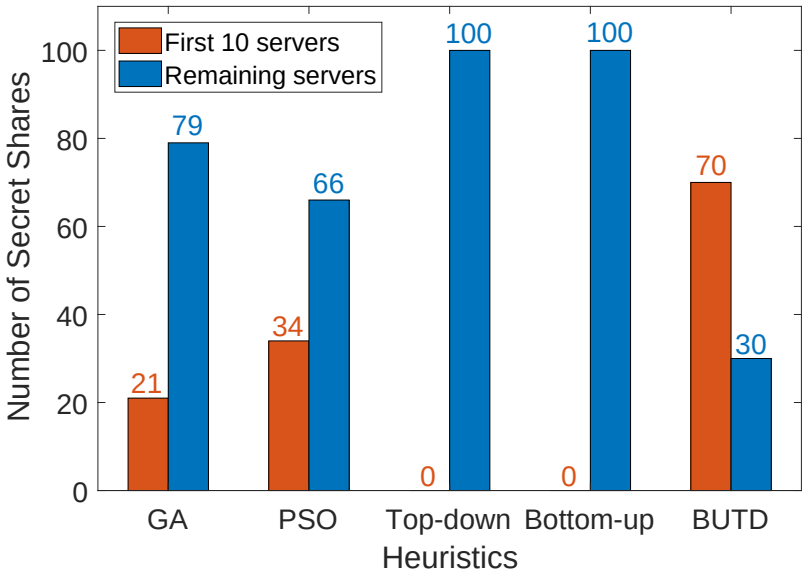


Figure 4.9: Secret share scheduling distribution under different heuristics.

Rising Trend

You may notice that in some situations listed above, the communication overhead curve shows a rising trend along with the secret index, especially when the optimal parameter setting is selected. Actually, such rising trends are inevitable, since neighborhoods of assigned secrets will be filled up anyway with new secrets continuing to come. On the other hand, these trends only happen when the whole network is almost full. In this situation, no good secret share assignments can be found and all heuristics will achieve similar performance, in which case we would recommend switching to another network.

The performance comparison shown in Fig. 4.10 further validates our thought. Here, we consider a hierarchical edge computing network with all field-level servers having $Q-1$ secret shares in queue to represent an almost full network. All the other small-scale experimental settings are still followed. From this figure, we can see that all the five communication overhead curves intertwine, which means that similar performance is achieved. Besides, you may also notice that the brought communication overhead of some secrets in the figure are missing, i.e. the 10th secret for *GA* and the 9th and 10th secret for *PSO*. This is because

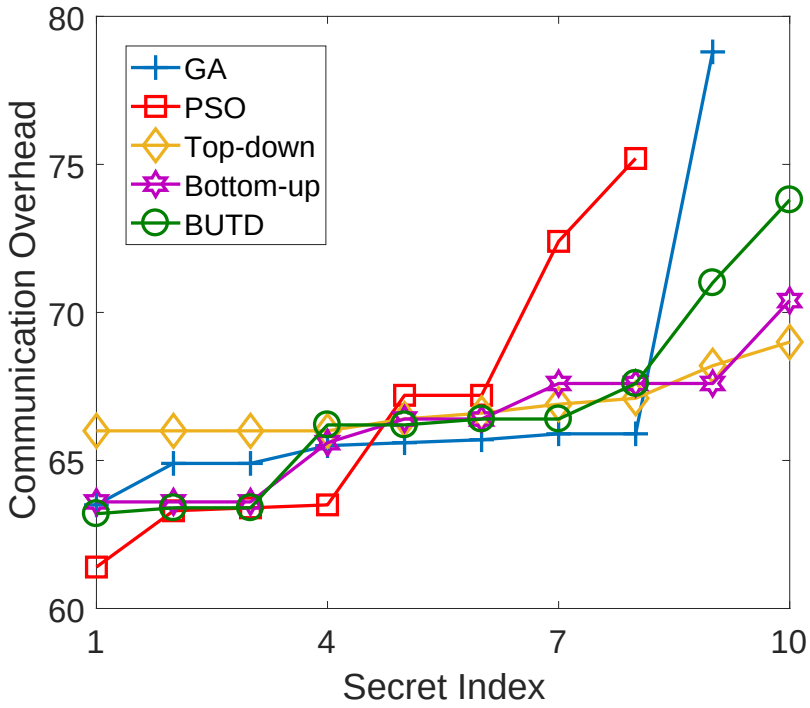


Figure 4.10: Scheduling results under different heuristics and an almost full network.

the corresponding heuristics cannot find a feasible scheduling scheme for those secrets within the pre-defined number of iterations.

4.7.4 Large-Scale Experimental Setting

We further conduct experiments in the large-scale experimental setting discussed in Section 4.7.1. For the parameter setting, we derive the following extension from that in the small-scale environment. In fact, most of the parameters do not need to be tuned here. We only need to care about parameters related to the number of public servers, since it is the only factor that can change the whole search space. Therfore, for *GA*, only the standard deviation σ needs to be considered. For *PSO*, they are the initial velocity v_0^* or range factor r and the neighbor size n_e. No parameters need to be tweaked for both the top-down and bottom-up heuristic, and only w_q needs to be changed for *BUTD*. We apply a linear increase along with the number of secrets and public servers for those parameters. In other words,

since the number of public servers is increased from 128 to 1600, i.e. 12.5 times, we will also multiply those parameters by 12.5. In this case, we have new $\sigma = 625$, $r = 63$ and $n_e = 375$.

The weight w_q in $BUTD$ is another parameter which needs to be tuned. Based on (4.7), we can see that w_q is related to the size of the given neighborhood. In this research project, we define the neighborhood as all the field-level servers in the same shallow-level server cluster, which increases from 4 for the small-scale experimental setting to 20 for the large-scale experimental setting, or by 5 times, according to Table 4.2. In this case, in order to rebalance the relationship between communication overhead and computation and storage space, we set w_q to 5 in the experiment shown in Fig. 4.11, which is 5 times less than its value in the small-scale experimental setting. Furthermore, we also consider rebalancing the importance between intra-group and inter-group communication here. The number of intra-group and inter-group communications are now respectively $(M-1)L = 5990$ and $\frac{L(L-1)}{2} = 45$. In other words, in order to make inter-group communication have exactly the same weight as intra-group communication, we need to set $w_s = 133$ in the large-scale environment.

Fig. 4.11 shows the performance comparison under all the five heuristics in this research project. From this figure, we can see that the $BUTD$ heuristic still performs the best in the large-scale environment. On the other hand, there are surges at the tail of the $BUTD$ curves, which correspond to the rising trend discussed in Section 4.7.3.

Note that we set w_s to 133 here just for validation. In practice, it is not quite possible to have such a big difference between intra-group and inter-group communication overhead. In this case, the top-down heuristic should already suffice when we have a large number of secrets to assign.

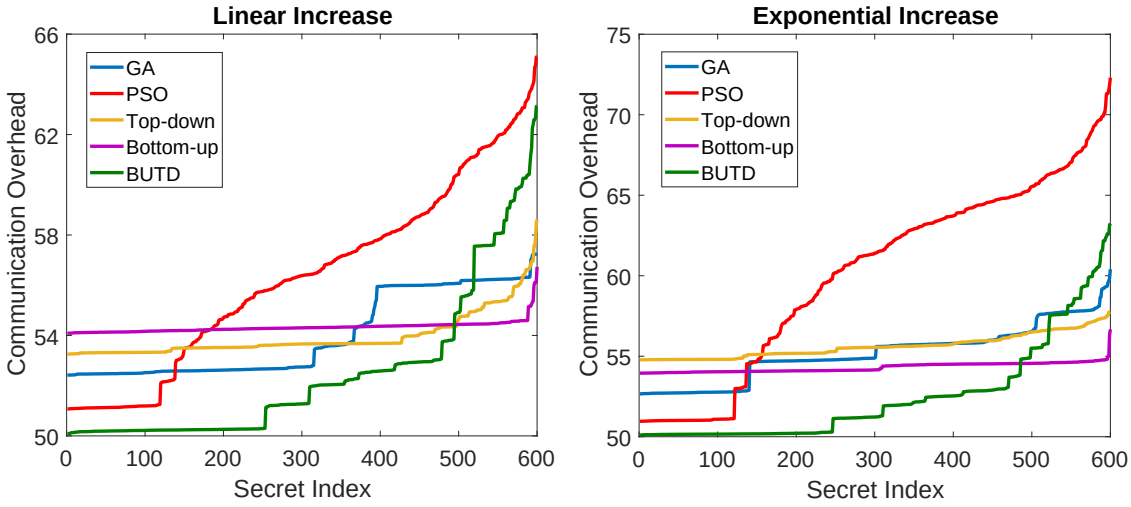


Figure 4.11: Performance comparison in the large-scale environment.

4.8 Future Work on Privacy-Preserving Task Scheduling with SS and MPC

In this chapter, we extend the discussion about secret share placement from the following three aspects: 1) We generalize (2,2) SS to (R,L) SS, and the situation is also considered that an adversary can simultaneously compromise multiple edge nodes; 2) The targeted mobile edge computing architecture is more comprehensive and practical. The hierarchical structure is currently widely investigated with its advantage in flexibility and scalibility; 3) Communication factors are encompassed, which significantly expands the applicability of related algorithms. In the future, the following problems are still worth exploring:

- Hop count is not a comprehensive metric to determine the optimal network path, which can also be affected by other communication factors, such as bandwidth and environment interference. Besides, the secret share placement strategy in this chapter is only based on a general network model. In the real applications, the secret matrix $\boldsymbol{C}_a$ and the share matrix $\boldsymbol{C}_b$ may need to be further modified according to different communication models and mobile edge computing structures. Even a dynamic $\boldsymbol{C}_a$

and $\boldsymbol{C}_b$ can be considered if communication conditions change.

- In federated learning, the wireless communication between local devices and central servers can still be a problem even under mobile edge computing. In this case, another communication matrix may need to be introduced if we want to further consider such a communication issue in the whole optimization problem.

- In general, intra-group communication is used for locally-learned model parameter aggregation in federated learning, while inter-group communication is responsible for synchronization issues. Different SS and MPC algorithms [22, 92] may need different information exchange for these two kinds of communication. A proper measure of the data size ratio w_s is needed for those specific SS and MPC algorithms. As has been discussed in this chapter, the top-down, bottom-up, and BUTD heuristic can respectively have the best performance when w_s has different values.

- Strictly speaking, Chapter 3 and 4 only discuss the optimization problem about secret share placement. During local model parameter aggregation, some granular arrangements need to be conducted for packet transmission. Existing works have talked about such arrangements in decentralized manners, such as ring all-reduce [109] and multitree all-reduce [110], which have potential integration with algorithms proposed in these two chapters.

- Scheduling strategies in this chapter do not consider computation latency optimization. Instead, computation and storage capacity is considered as a constraint. Alternatively, a tradeoff between communication and computation can be investigated, and comparison can be conducted with algorithms in this chapter. Besides communication heterogeneity discussed above, computation and energy level can also be different for different edge nodes, which may also impact secret share placement results.

4.9 Summary

In this research project, we proposed a communication-aware secret share placement strategy in hierarchical edge computing structures. We show that the related model is actually a combinatorial search problem, which is already proven to be NP-hard. In this situation, we introduce five heuristic algorithms to find sub-optimal solutions and compare their performance. Therein, *GA* and *PSO* are two classical heuristics which we can directly apply. In addition, based on the characteristics of our targeted problem, we further propose two basic heuristics, i.e. the top-down and bottom-up heuristic, which respectively only target the intra-group and inter-group communication. Furthermore, we introduce the *bottom-up* heuristic to the initialization stage of the top-down heuristic, and schedule secret shares together with the consideration of remaining computation and storage space within a particular neighborhood, i.e. all the field-level servers within the same shallow-level server cluster, which is called the *BUTD* heuristic. We show through both systematic analysis and comprehensive experiments that our three proposed heuristics can respectively achieve the best performance when intra-group communication overwhelms inter-group communication, vice versa, and those two kinds of communication are comparable.

Chapter 5: Federated Graph Neural Network for Fast Anomaly Detection in Controller Area Networks

In this chapter, we aim at reducing communication overhead in federated learning by solving the third detailed research problem in Section 1.2. Convergence of federated learning can be greatly slowed if adversaries compromise local devices and poison raw data or uploaded model parameters, which can further cause numerous unnecessary communication rounds between local devices and public servers. Intrusion Detection System (IDS) is usually one of the strategies to combat those attacks. However, considering the number of attack types and heterogeneity among end devices, conventional centralized machine learning can suffer a lot in developing IDSs in a timely manner. Recent works [111–115] have shown that federated learning can conversely be applied to combat those cybersecurity problems, which refers to self-learning IDSs. In this chapter, we validate the value and utility of federated learning based on a well-known research problem in cybersecurity, i.e. intrusion detection in Controller Area Network (CAN bus). Relevant intrusions can be seen as data poisoning attacks in federated learning. One paper related to this research project is currently under review [116].

5.1 Introduction

Modern vehicles rely on processing units, i.e. Electronic Control Units (ECUs) to accomplish different driving tasks, such as collision avoidance, anti-lock brakes and traction control. These ECUs are further interconnected by a bus or network for cooperations [117, 118]. A typical in-vehicle control network is depicted in Fig. 5.1. Currently, the most popular communication standard is the CAN bus, which was designed by Robert Bosch GmbH in 1983. The CAN bus is a broadcast medium, where messages sent by one ECU can be

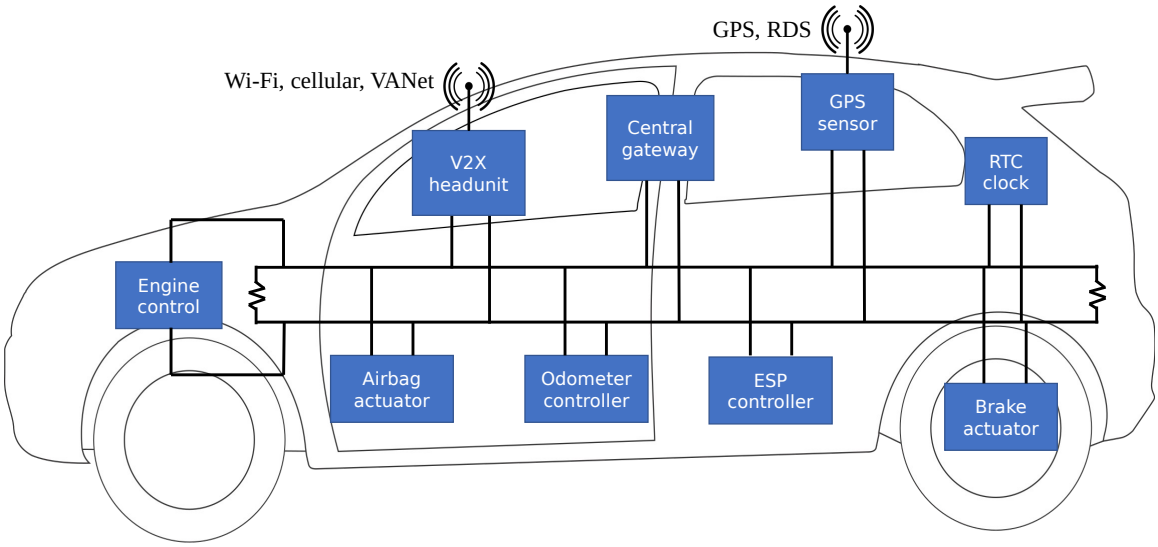


Figure 5.1: A typical in-vehicle control network

received by all the other ECUs in the same control network. The CAN bus uses a lossless bitwise arbitration method to resolve contentions during data transmissions. In detail, every CAN message is assigned a CAN ID based on its functionality and priority. A message with a lower ID will win the contention when two messages collide. Usually messages containing crucial information, such as those related to powertrain and vehicle safety, will be assigned lower CAN IDs, while infotainment and telematics messages will have higher CAN IDs [119].

The CAN bus protocol does not provide any authentication or encryption mechanisms. In this case, attackers have the chance to compromise the CAN bus in a vehicle through inserting forged messages. For a typical CAN ID, once forged messages overwhelm normal messages, attackers can take control of relevant operations in the targeted vehicle, which can cause severe consequences if that CAN ID is related to powertrain or vehicle safety. Furthermore, the recent development of Intelligent Transportation Systems (ITS) and IoT continuously expand the attack interfaces, which include but are not limited to sensors, Wi-Fi, and On-Board Diagnostics (OBD) [120]. In 2021, Upstream Security's research team provided a global automotive cybersecurity report based on 633 publicly reported incidents

in the last decade, which shows an exponential growth trend in cyberattacks on connected vehicles [121].

Considering this, several IDSs have been proposed to increase security of CAN bus. In general, if an adversary wants to make forged messages overwhelm normal ones, it can inject forged messages (DoS or fuzzy attack), suspend normal messages (suspension attack), or falsify data contents of normal messages (replay or spoofing attack). ECUs in the same CAN network usually transmit their messages with a comparatively fixed frequency, which makes the statistics of CAN message sequences comparatively stable. In this case, if the first two attack strategies happen, message frequencies or message sequences are likely to be changed, which are the motivations for the existing message frequency or sequence-based detection methods [122–125]. Besides, some existing works [126–128] consider message falsification attacks. They are based on the observation that message contents with the same CAN ID will neither vary too much.

The situation becomes complicated when considering message injection/suspension attacks together with message falsification attacks. Message falsification attacks will not change message frequencies or sequence patterns, which will evade IDSs only considering message injection/suspension. On the other hand, IDSs for message falsification cannot identify Denial-of-Service (DoS) attacks [129] or bus-off attacks [130], which do not need to change normal message contents. Recently, some related works tried to detect both of the above two kinds of attacks simultaneously based on Long Short-Term Memory (LSTM) autoencoder [126, 131, 132] and bloom filtering [125]. However, these schemes need to generate a separate model and analyze relevant messages for each CAN ID, which can induce very high computation complexity and intrusion detection delay. In order to detect all the attacks, we can also apply ensemble learning [133] to train multiple classifiers, but it will still lead to increased computation complexity and detection delay.

Based on the above observations, we propose a CAN bus IDS based on GNN in this chapter, which can efficiently detect message injection, suspension, and falsification attacks simultaneously. Directed attributed graphs are constructed to include both statistical CAN

message sequences and message contents. Therein, message sequences are described by nodes, edges and edge attributes in graphs. Data contents in messages with a typical CAN ID are preprocessed by the **READ** [134] method, and summarized as the corresponding node attributes. With these generated CAN message graphs, a two-stage GNN-based classifier cascade can be trained to build our IDS, which is illustrated in Fig. 5.2. The first stage has similar functionality as existing IDSs, i.e. anomaly detection. Considering that attacked data are usually hard to acquire in the real world, normal data usually dominate the training set, which will become highly imbalanced. In this case, we replace the traditional *softmax* layer in GNN with a one-class classification layer for the anomaly detection purpose. Once an attacked data sample is captured by the first-stage classifier, this data sample will further go to the second-stage classifier for attack classification. Inspired by [135], besides the *softmax* layer for multi-class classification, the second-stage classifier also has an *openmax* layer to tackle new anomalies from potentially unknown classes, which will be buffered for further investigation and open world recognition [136, 137].

Furthermore, in Section 5.6, we will show that different vehicle states can lead to variations in message sequences, message contents, and further message graphs. Therefore, the model trained on one vehicle will be constrained by its limited driving scenarios (e.g., a vehicle may mostly drive in local with low speed and a lot of stops) and vehicle states, so cannot be applied to other vehicles with different driving scenarios and vehicle states (e.g., a vehicle may mostly drive on highway with high speed and few stops). To take advantage of crowdsourcing while protecting user data privacy, we further adopt a federated learning framework to train a universal model that covers a wide range of driving scenarios and vehicle states. The main contributions of this research project are summarized as follows:

- We propose a CAN bus IDS which can efficiently detect CAN message injection/suspension and message falsification attacks at the same time. Instead of a simple combination of the above two traditional IDSs, we define a CAN message graph to integrate message contents with statistical message sequences in terms of CAN ID pairs.

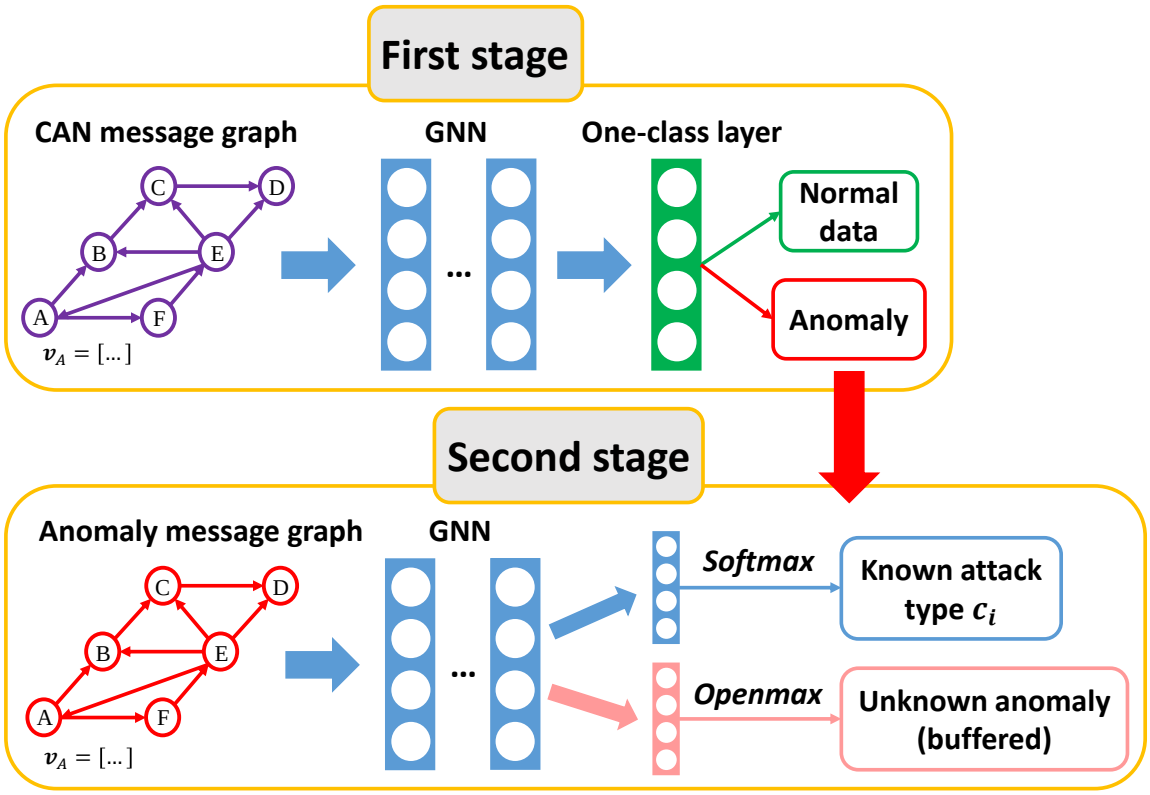


Figure 5.2: Architecture of the two-stage classifier cascade

- We develop a GNN which is fit for directed attributed graphs. Considering that attacked data are hard to acquire in the real world, which may cause highly imbalanced training sets, we develop a two-stage classifier cascade to tackle normal and attacked CAN data respectively. An *openmax* layer is introduced to cope with new anomalies from potentially unknown classes.

- Based on the observation that changes of vehicle states can cause variations in generated CAN message graphs and further negatively impact intrusion detection performance, we consider federated learning to cover different driving scenarios and vehicle states while protecting data privacy.

- We evaluate the proposed IDS through extensive experiments based on several real-world datasets. Through comparisons with three baselines, we validate that our proposed IDS

can achieve similar performance to both IDSs based on statistical CAN message sequences and message contents. Besides, federated learning can effectively combine models derived under different driving scenarios and vehicle states, and greatly improve intrusion detection performance.

The rest of this chapter is organized as follows. Related works will be summarized in Section 5.2. CAN message graphs will be proposed in Section 5.3. Section 5.4 and Section 5.5 respectively give the design of the first-stage and second-stage classifier. The impact of vehicle states on message contents and message sequences will be analyzed in Section 5.6, which motivates the adoption of federated learning to take advantage of crowdsourcing while protecting user data privacy. Experimental results for validation and evaluation will be provided in Section 5.7. Some open problems will be discussed when federated learning is deployed in CAN bus IDSs in Section 5.8. Finally, conclusions will be drawn in Section 5.9.

5.2 Related Work

5.2.1 Attack Types

According to [120], attacks to in-vehicle networks or CAN buses can be divided into message sniffing, message injection, message suspension and message falsification. Therein, the later three categories of attacks can further impact normal functionalities of vehicles.

Message injection attack is usually considered together with message suspension attack, since they are both related to message frequency or statistical message sequence in terms of CAN ID. In practice, message falsification attack is realized through a combination of message sniffing, message injection and message suspension. Based on [123, 129, 130, 138–140], the specific attack types considered in this research project are summarized as follows:

- **Denial-of-Service (DoS):** This attack can also be called bus-off attack [130], which aims at paralyzing CAN bus systems through continuously injecting legitimate CAN messages with a low CAN ID, e.g. 0x000. Since the CAN bus uses a lossless bitwise arbitration

method to tackle data transmission contention, some functionalities assigned with higher CAN IDs will never get the chance to be transmitted.

- **Fuzzy:** This attack will generate and transmit CAN messages with random CAN IDs and data contents. The CAN IDs will range from 0x000 to 0x7FF [132], and some of them originally may not be used in the compromised vehicles. Such a type of attack can interfere with some functionalities of victims if extra lower CAN IDs are introduced into the systems. Besides, the randomly generated data contents can mislead vehicles if those CAN IDs are initially used.

- **Suspension:** This is just the message suspension attack. The attacker will try to compromise some ECUs in the CAN bus system, and stop them from sending any CAN messages.

- **Replay:** This attack will try to compromise some ECUs in the CAN bus system, and store some valid CAN messages in a particular time period, which will be transmitted later. Such a type of attack can mislead compromised vehicles since those stored CAN messages are actually outdated. If the attacker can further suspend those sniffed ECUs, it becomes message falsification attack.

- **Spoofing:** Similar to replay attack, spoofing attack will at first try to sniff some ECUs in the CAN bus system. However, spoofing attack will try to impersonate those compromised ECUs by simulating their message transmission frequencies, while the related data contents are usually forged. Message falsification attack can also be realized here by further suspending those compromised ECUs.

5.2.2 Intrusion Detection Systems

IDSs can be divided into anomaly detection-based and signature-based methods, which respectively identify intrusions by comparing with normal data and known attacks. Most CAN bus IDSs are anomaly detection-based methods. Message injection or suspension attacks will change CAN message frequencies or statistical message sequences in terms

of CAN ID pairs, which is the basis of related works. Early works [124, 141] directly use message frequencies for intrusion detection. Later, some data analysis methods are proposed based on statistical message sequences. Such methods compare two message sequences through statistical metrics, such as cosine similarity, Pearson correlation or chi-squared test [122, 123]. If a significant changes in message frequencies or sequences (metric values larger than given thresholds) is detected, they predict that intrusions happen in the second message interval. Recently, some works [122, 142] also consider machine learning based methods, such as LSTM autoencoder, to detect message injection or suspension attacks through statistical CAN message sequence reconstruction. The authors in [143] further explore the possibility to convert CAN message streams to images, and train a generative adversarial network (GAN) for CAN bus intrusion detection.

All the above IDSs cannot properly tackle message falsification attacks. Existing works targeting message falsification attacks are based on the assumption that message contents or some bits therein do not have much variance. Early works consider Hamming distance between the bit representation of each two CAN messages [125] or message entropy [144,145] for intrusion detection. For machine learning methods [127, 128, 146], they generally adopt techniques originally used for identifying human actions [147–149].

Recently, the development of natural language processing inspires some CAN bus IDSs based on LSTM [126,131] to detect message injection/suspension attacks together with message falsification attacks. The authors in [125] and [133] also propose to consider both CAN message frequencies and message contents through bloom filtering and ensemble learning. However, these schemes need to either consider each CAN ID separately or train multiple machine learning models. Considering each CAN ID separately will induce high data collection delay, since sufficient numbers of CAN messages are required for each CAN ID. Our proposed IDS inherits the advantage of those IDSs based on CAN message sequences, which only needs an enough number of overall CAN messages. On the other hand, training multiple machine learning models will bring much higher computation complexity than training a single model. Furthermore, most of existing CAN bus IDSs, especially those

based on LSTM, detect attacks on the basis of a trustworthy reference (a normal CAN message sequence in the previous time slot), which usually cannot be guaranteed during intrusion detection.

Finally, open world recognition has also drawn attention in CAN bus intrusion detection. The authors in [137] proposed a transfer learning architecture, which can be trained for unknown attacks. [136] realizes a similar functionality based on a one-class classifier. These two works can be seen as successors of our work. After our IDS buffers enough previously unknown attacks or anomalies, the open world recognition strategies can be introduced to tackle them. A comprehensive comparison between our proposed IDS and existing schemes is summarized in Table 5.1 to show the advantages of our method. A comparison of detection functionality is also listed in the table. Existing IDSs for the CAN bus system can only tell whether an attack happens or not (binary), while our proposed IDS can further figure out the specific attack type, and tackle previously unknown anomalies (open multi-class).

Table 5.1: Comparison between Our Proposed IDS and Existing Schemes

Schemes	[122–124, 141, 142]	[143]	[144, 145]	[127, 128, 146]	[125, 126, 131]	[133]	Our proposed IDS
Message injection	✔	✔	✖	✖	✔	✔	✔
Message falsification	✖	✖	✔	✔	✔	✔	✔
Collection delay	Low	Low	Low	Low	High	–	Low
Computation complexity	–	–	–	–	–	High	Low
Reference needed	✔	✖	✔	✖	✔	✖	✖
Detection functionality	Binary	Binary	Binary	Binary	Binary	Binary	Open multi-class

5.3 Preliminaries and Basics

5.3.1 CAN Message Description

Fig. 5.3 illustrates a typical CAN bus message format. The main part is composed of the following fields:

- **Arbitration:** This field contains an identifier of a given CAN message, i.e. CAN ID.

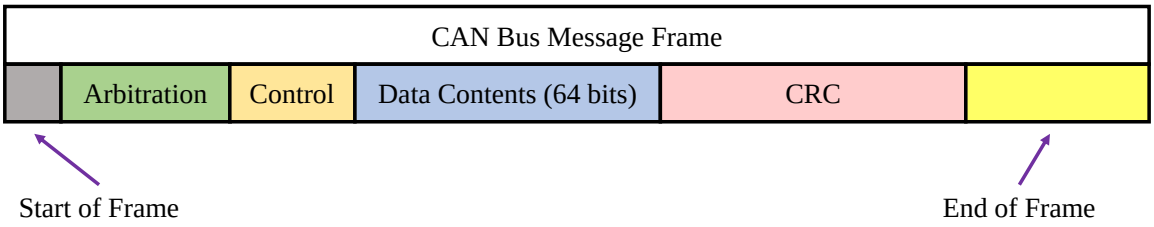


Figure 5.3: CAN bus message format

- **Control:** This field is mainly composed of two components. Remote transmission request (RTC) specifies the type of the current CAN message, i.e. a remote request frame or a real data frame. Data length code (DLC) gives the number of data bytes in the data field.
- **Data Contents:** This field includes data to be transmitted, the length of which is reflected in DLC.
- **Cyclic Redundancy Check (CRC)**: This filed provides the checksum for the current CAN message.

Our CAN bus intrusion detection system in this research project is based on CAN ID in the arbitration field and data contents of each CAN message. In the CAN standard, a CAN ID has either 11 or 29 bits, and the version with 11 bits is usually selected for CAN buses in vehicles. Data contents have a length between 0 and 8 bytes (or 64 bits). Recently, Bosch proposed an extended CAN standard, called CAN with Flexible Data-Rate (CAN-FD). Data contents in CAN-FD messages can have a length of up to 64 bytes. In this research project, we are still based on the CAN standard with up to 8 bytes of data.

CAN bus data can be streamed through using the *candump* tool in Linux CAN subsystem, a.k.a. SocketCAN. Fig. 5.4 shows an example with four streamed CAN bus messages. Therein, the red and blue box respectively include CAN IDs and data contents of those four messages, which are denoted with hexadecimal numerals. The other two components in streamed CAN messages are timestamp and CAN interface name.

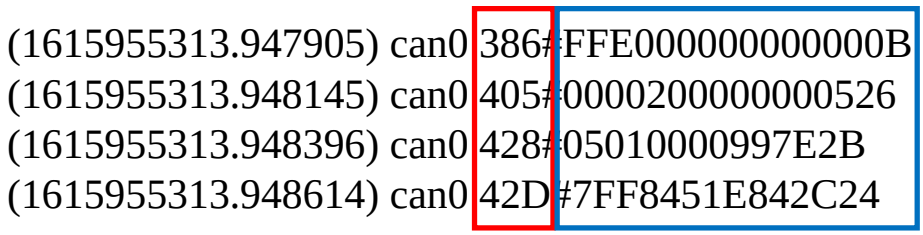


Figure 5.4: Streamed CAN bus messages

5.3.2 CAN Message Graph

As has been discussed in Section 5.1, CAN messages with each CAN ID are usually transmitted in a comparatively fixed frequency. Early works [124,141] directly use it for intrusion detection. Later, researchers notice that fixed frequencies can further infer stable statistical message sequences in terms of CAN ID pairs. In addition, several ECUs may need to collaborate with each other to accomplish a vehicle operation task, which can be realized through transmitting successive CAN messages. For example, after one CAN message reflecting gas increase is transmitted, we will probably see one message denoting an increase in revolutions per minute (RPM) of the vehicle engine. Then, another CAN message representing a vehicle acceleration will also be spotted [150]. Considering these two factors, the sequences should follow comparatively fixed patterns.

With this observation, data analysis metrics, such as cosine similarity and Pearson correlation, or machine learning models, such as LSTM autoencoder, can be applied based on statistical message sequences for intrusion detection [122, 142]. Note that in order for real-time analysis, CAN messages are considered in **intervals**, which usually vary from 100 to 200 messages. Streaming such numbers of CAN messages usually costs an order of milliseconds. If the message interval is too short, i.e. less than 100 messages, the message sequence will become unstable, since many messages with higher CAN IDs do not need to be transmitted very frequently, and these CAN IDs may not appear in some of message intervals.

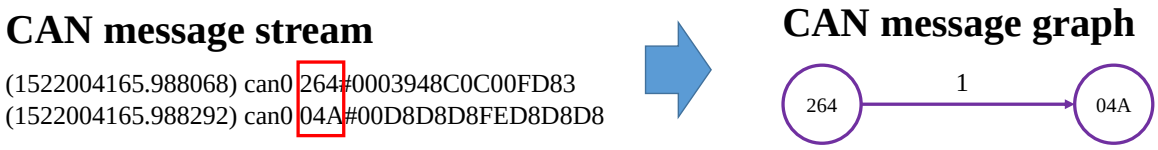


Figure 5.5: A toy example showing conversion from CAN message stream to CAN message graph

Besides statistical message sequence, message graph is actually another possible structure to describe CAN message streams. Compared with message sequences, message graphs can further embed message contents. Graph structures have both edge attributes and node attributes, and they provide the possibility to simultaneously detect all the three categories of attacks mentioned in Section 5.1.

Fig. 5.5 shows the conversion from a toy CAN message stream with two messages to a CAN message graph. Each node in the graph represents a CAN ID appearing in the given message interval. In this toy CAN message stream, a message with CAN ID `264` is followed by another CAN message `04A`. Therefore, we construct a directed edge from Node `264` to Node `04A` with an attribute or weight of 1. A normal CAN message graph is shown in Fig. 5.6. In each CAN message sequence, messages are considered in pairs similar to that shown in Fig. 5.5. Edge attributes or weights refer to the number of corresponding message pairs appearing in the given message interval. The structure of a graph is usually described by an adjacency matrix, denoted by $\boldsymbol{A}_{n\times n}$ in this research project. n here represents the number of nodes in a CAN message graph, which further infers the number of CAN IDs appearing in the corresponding message interval. In addition, since CAN message graphs have edge attributes, we further introduce an edge matrix $\boldsymbol{E}_{n\times n}$ to include this information.

5.3.3 CAN Message Content Preprocessing

The data content in each CAN message can be used for identifying message falsification attacks. As has been discussed in [132], signal semantics are usually described in blocks in message contents, and a proper data division can greatly improve intrusion detection

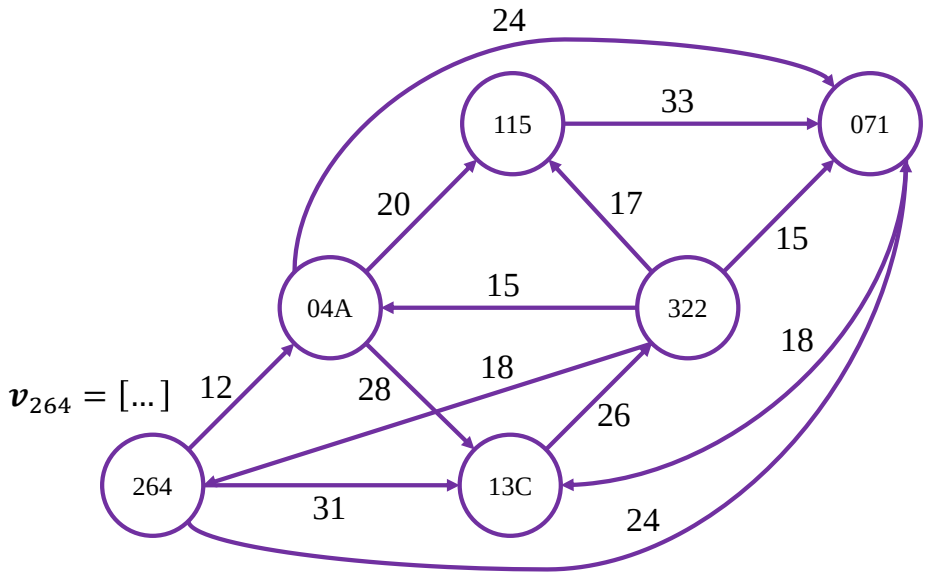


Figure 5.6: CAN message graph

accuracy. The authors in [134] proposed a well-behaved signal boundary extraction algorithm for CAN message contents, called READ, which divides message contents based on the bit-flip rate. The general idea behind this algorithm is that for each signal semantic, the most significant bit in the related data block will vary much slower than the least one. In this case, if a bit with a high bit-flip rate is followed by one with a low rate, these two bits probably belong to two different signal semantics or data blocks.

In order to calculate the bit-flip rate, a certain number of CAN messages need to be collected for each CAN ID. Note that the CAN message content division only needs to be conducted once based on the training set before our IDS model training. In this case, we do not need to worry about any data collection delay during intrusion detection. Besides, we neither need to tackle any CAN message content associated with a new CAN ID during intrusion detection, since it just infers a fuzzy attack according to Section 5.2.1. The details of bit-flip rate calculation for each CAN ID is summarized in Algorithm 5.

A bit-flip rate heatmap based on a dataset in [122] is illustrated in Fig. 5.7, which is composed of 44 CAN IDs and 23,963 normal CAN messages. Here, CAN IDs are indexed

Algorithm 5: Bit-Flip Rate Calculation

Input : *messageArray*: an array of CAN messages with the targeted CAN ID in the training set.

Output: $bitflip$: an array of bit-flip rates for message contents of the targeted CAN ID. *magnitude*: the corresponding magnitude array.

```
1  Initialize messageLen ← len(messageArray), b ← len(messageArray[0]),
     bitflip ← array(b);
2  previous ← messageArray[0];
3  for indexM = 1, 2, ··· , messageLen − 1 do
4      current ← messageArray[indexM];
5      for indexB = 0, 1, ··· , b − 1 do
6          if current[indexB] ≠ previous[indexB] then
7              bitflip[indexB] + +;
8      previous ← current;
9  for indexB = 0, 1, ··· , b − 1 do
10     bitflip[indexB] ← bitflip[indexB]/messageLen;
11     magnitude[indexB] ← ⌈log10(bitflip[indexB])⌉;
```

according to their values. From this heatmap, we can intuitively figure out several data blocks in different CAN IDs. A signal boundary will be applied where a light pixel is followed by a dark pixel. Besides, according to [134], we do not need to care about small variations in bit-flip rate. In this case, we further introduce a magnitude array on line 11 of Algorithm 5, and look for drops in the magnitude array instead. Finally, for convenience of GNN design in the next section, we need to describe all CAN message contents with a node matrix $\boldsymbol{V}_{n\times n'}$. After applying READ, message contents with different CAN IDs will be separated into different numbers of data blocks, each of which is further converted to a decimal number. n' in the node matrix denotes the maximal number of data blocks across all CAN IDs. We will further align the number of data blocks for all CAN IDs by complementing some 0s in the front.

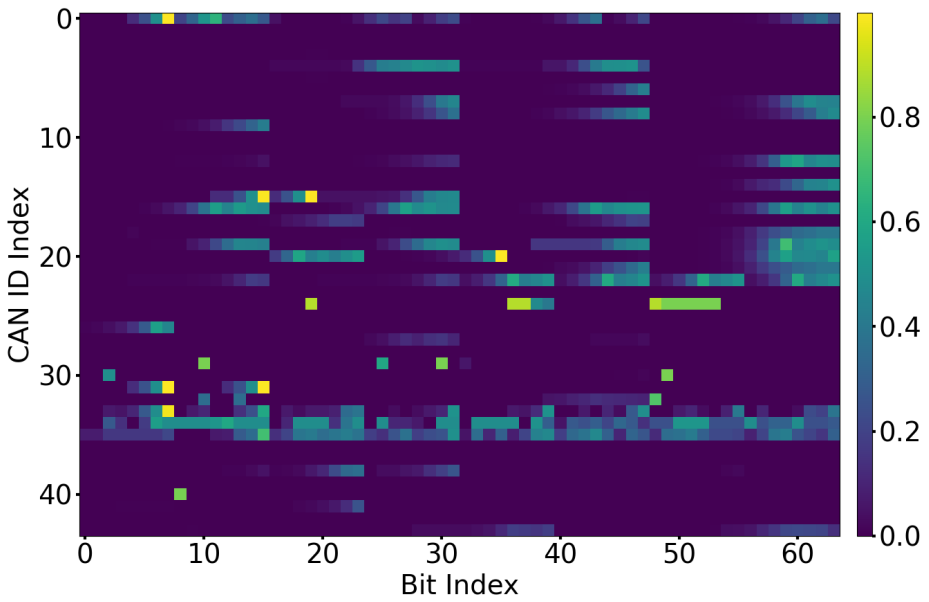


Figure 5.7: Bit-flip rate heatmap

5.4 First-Stage Classifier

5.4.1 Graph Neural Network

Graph learning has recently drawn increasing attention. All current graph learning frameworks can be divided into three levels, i.e. node level, edge level and graph level [151]. Therein, node level algorithms can be utilized to determine whether a typical CAN message is forged or not, which is similar to the idea in [125]. Our proposed CAN bus IDS is based on graph level algorithms. All three levels of models start with some graph convolution layers, and graph level schemes realize graph classification tasks via further introducing pooling and readout layers. In this research project, we are based on GNN proposed in [152] for CAN bus intrusion detection.

The convolution layer of GNN in this chapter takes the following form:

$$\boldsymbol{Z} = f(\tilde{\boldsymbol{D}}^{-1}\bar{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}) \tag{5.1}$$

The original GNN in [152] only targets undirected graphs without edge attributes and self-loops. In order to make it fit for CAN message graphs in this chapter, we at first concatenate the node matrix $\boldsymbol{V}$ with edge matrix $\boldsymbol{E}$ to generate a descriptor for CAN message graphs, denoted as $\boldsymbol{X}_{n\times(n+n')}$. Such an operation actually embeds edge attributes into node attributes. In other words, attributes of edges starting from a typical node are considered as part of attributes of that node. $\bar{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ represents the adjacency matrix with added self-loops, which are realized through an identity matrix $\boldsymbol{I}$. Since CAN message graphs considered in this chapter are directed and allow self-loops, we further tackle the adjacency matrix $\boldsymbol{A}$ to make all diagonal elements 0 before adding $\boldsymbol{I}$. $\tilde{\boldsymbol{D}}$ is a diagonal degree matrix with $\tilde{d}_{ii} = \sum_{j=1}^{n} \bar{a}_{ij}$, where $\tilde{d}_{ii}$ and $\bar{a}_{ij}$ are respectively elements in $\tilde{\boldsymbol{D}}$ and $\bar{\boldsymbol{A}}$ with the corresponding indices. $\boldsymbol{W} \in \mathbb{R}^{(n+n')\times c}$ denotes model parameters in the convolution layer, where c is the number of feature channels in the convolution layer. $f(\cdot)$ is a pointwise nonlinear activation function, such as Rectified Linear Unit (ReLU).

The whole graph convolution process can be explained as follows. Node and edge attributes are at first fit into the convolution layer through a linear feature transformation $\boldsymbol{XW}$. Afterwards, for each node, its "channel descriptor", denoted by $\boldsymbol{Y} = \boldsymbol{XW}$, will be propagated to its neighborhood including itself through $\bar{\boldsymbol{A}}\boldsymbol{Y}$. Here, we can see that why we need to include self-loops in the adjacency matrix $\boldsymbol{A}$. Then, we normalize the propagation results by multiplying them with the diagonal degree matrix $\tilde{\boldsymbol{D}}$, which aims at keeping a fixed feature scale after graph convolution. Finally, a pointwise nonlinear activation function $f(\cdot)$ is applied before outputting the graph convolution results.

Similar to convolution neural networks applied in image processing, in order to capture graph substructure features in different scales, we need to apply and stack multiple convolution layers. In this case, in the t-th convolution layer, we further have:

$$\boldsymbol{Z}_t = f(\tilde{\boldsymbol{D}}^{-1}\bar{\boldsymbol{A}}\boldsymbol{Z}_{t-1}\boldsymbol{W}_t) \tag{5.2}$$

Here, $\boldsymbol{Z}_{t-1}$ and $\boldsymbol{Z}_t$ respectively represent the input and output matrix of the c_l-th convolution layer, and we further have $\boldsymbol{Z}_0 = \boldsymbol{X}$. $\boldsymbol{W}_t \in \mathbb{R}^{c_{t-1} \times c_t}$ includes model parameters in the t-th convolution layer, with $c_0 = n + n'$. After acquiring all $\boldsymbol{Z}_t$ from those convolution layers, the outputs are stacked through concatenation, i.e. $\boldsymbol{Z}_{1:t} = [\boldsymbol{Z}_1, \boldsymbol{Z}_2, \cdots, \boldsymbol{Z}_t]$.

Up to now, the design of graph convolution layers have finished. For graph classification tasks, pooling and readout layers need to be further introduced. Besides pooling layers for downsampling, the authors in [152] further introduce a sorting operation to sort nodes in a graph according to their structural roles, which can make similar graphs fit into readout layers in a comparatively consistent node order. Such an operation is beneficial to tackling CAN message graphs in this research project. As will be discussed in Section 5.6.1, vehicle states can sometimes cause variations in CAN message graphs, since some ECUs in vehicles are not always activated in order to prolong battery life. Such a situation can further induce node indexing issues, which can be solved by this sorting operation. The readout layers are usually composed of some 1-dimensional (1-D) convolution layers and dense layers.

5.4.2 One-Class Classification

Traditionally, the last layer of a neural network is a *sigmoid* or *softmax* layer for classification tasks. However, to the best of our knowledge, nearly all the current attacked data are simulated for CAN bus IDS evaluation. Intrusions to CAN buses are hard to acquire in real vehicles, which will make the training set highly imbalanced. Considering this, the first-stage classifier will be designed as an anomaly detection-based IDS. The conventional *sigmoid* or *softmax* layer will be replaced with a one-class classification layer, and only normal CAN bus data will be used to train the model.

One-class classification is widely used for anomaly detection, which can train a classifier only by using normal data. One-class support vector machine (OC-SVM) [153] and support vector data description (SVDD) [154] are two currently two most popular related algorithms. Compared with OC-SVM, SVDD has better computation scalability and better performance in tackling curse of dimensionality [155]. Therefore, we introduce SVDD to our GNN for

intrusion detection.

Similar to OC-SVM, SVDD comes from the traditional support vector machine (SVM). However, instead of using a hyperplane to define the soft classification border, SVDD tries to find a hypersphere with a center $\boldsymbol{o} \in \mathbb{R}^{c_g}$ and a radius $r > 0$ as the border. Here, c_g represents the output dimensionality of the last dense layer in our GNN. In general, the related optimization problem of SVDD can be defined as follows:

$$\underset{r,\boldsymbol{o},\boldsymbol{\xi}}{\arg\min}\ r^2 + \frac{1}{\nu m}\sum_{k=1}^{m}\xi_k \tag{5.3a}$$

subject to,

$$\forall k,\ ||\phi_{c_g}(\boldsymbol{X}_k) - \boldsymbol{o}||^2 \leq r^2 + \xi_k,\ \xi_k \geq 0 \tag{5.3b}$$

In Eq. (5.3a), $\boldsymbol{\xi}$ is called a slack vector, in which all slack variables $\xi_k \geq 0$ altogether construct a soft classification border or hypersphere. The soft classification border allows some of data samples or support vectors originally from one class to cross the borderline and fall in another class, which helps to avoid model overfitting via reducing the radius of the hypersphere to a certain extent. On the other hand, the hypersphere cannot be shrunk without any limitation, since this can potentially induce model underfitting. In this case, a weight $\nu \in (0, 1]$ is introduced here to balance model overfitting and underfitting. In Eq. (5.3b), the function $\phi_{c_g}(\boldsymbol{x}_k)$ is the kernel function usually used in SVM for space mapping. Here, we can consider the whole architecture of GNN except the last one-class classification layer as the kernel function. $\boldsymbol{X}_k$ refers to the k-th CAN message graph discussed in Section 5.4.1 in the training set. $||\cdot||^2$ is the $\mathcal{L}^2$ norm.

With a small trick, we can embed the constraint of the above optimization problem into its objective function. At first, we can set:

$$\xi_k = \max\{0, ||\phi_{c_g}(\boldsymbol{X}_k) - \boldsymbol{o}||^2 - r^2\} \tag{5.4}$$

Then, we replace ξ_k in Eq. (5.3a) with Eq. (5.4). Besides, we also consider a model parameter regularization term to further dodge model overfitting. At last, we can get the following optimization problem without any constraint, which can be seen as the loss function of our whole GNN model.

$$\underset{r,\boldsymbol{W}_{all}}{\arg\min}\ r^2 + \frac{1}{\nu m}\sum_{k=1}^{m}\max\{0, ||\phi_{c_g}(\boldsymbol{X}_k) - \boldsymbol{o}||^2 - r^2\} + \frac{\lambda}{2}||\boldsymbol{W}_{all}||^2 \tag{5.5}$$

Here, λ is the weight decay to balance the penalty for large model parameters. $\boldsymbol{W}_{all}$ denotes a vector which includes all the parameters in our GNN model except the last one-class layer. We no longer consider the center of the hypersphere $\boldsymbol{o}$ as a coefficient to be tuned here, and follow the strategy in [155] to fix $\boldsymbol{o}$. Some data samples are at first chosen from the training set. Afterwards, we let these data samples pass through the initialized whole GNN except the last one-class classification layer, and record the outputs from the last dense layer. Finally, the center $\boldsymbol{o}$ is set to the mean of these outputs. Note that the above loss function needs to investigate all the CAN message graphs in the training set at the same time. If gradient descent is used for model parameter update, CAN message graphs for training need to be considered in batches. With the concern of computation complexity, the training process in this research project is based on mini-batch Stochastic Gradient Descent (SGD).

5.5 Second-Stage Classifier

After the first-stage classifier filters out all the anomalies, they will further be sent to the second-stage classifier to detect the specific attack type. The second-stage classifier can be seen as a signature-based IDS. In order to tackle the potential new anomalies from unknown classes, such as zero-day attacks, this classifier further introduces an *openmax* layer.

During the training process, the second-stage still utilizes the traditional *softmax* function in the output layer. The *openmax* layer is deployed during intrusion detection to replace

the *softmax* layer, which revises the output of the penultimate layer in the original classifier to reject data samples not close enough to any known class. In this research project, the output of the penultimate layer can be denoted as an activation vector $\mathcal{A}_{C\times 1}$, where C represents the number of known types of attacks. The design of the *openmax* layer is based on meta-recognition algorithms [156, 157], in which the distribution of the activation vector is analyzed through Extreme Value Theory (EVT), and the Weibull distribution is figured out. In detail, the EVT fitting in the *openmax* layer design adapts the concepts of Nearest Class Mean [158, 159] or Nearest Non-Outlier [160] per attack type to the activation vector. Each attack type c_i is represented with a mean activation vector (MAV), which is derived through averaging all the activation vectors of the training samples belong to this attack type, denoted as $\bar{\mathcal{A}}_{c_i}$. Then, within each attack type, we calculate the Euclidean distance between the activation vector of each training sample $\mathcal{A}_{c_i,j}$ and the MAV $\bar{\mathcal{A}}_{c_i}$. Afterwards, we conduct the Weibull fitting for each attack type based on EVT:

$$||\mathcal{A}_{c_i,j} - \bar{\mathcal{A}}_{c_i}||_2 \sim Weibull(\alpha_i, \beta_i, \boldsymbol{\gamma}_i) \tag{5.6}$$

where α_i, β_i and $\boldsymbol{\gamma}_i$ respectively denote the shape, scale and position parameter to describe a Weibull distribution. $\boldsymbol{\gamma}_i$ has the same dimensionality as $\bar{\mathcal{A}}_{c_i}$ for shifting the data. All the Weibull fitting is pre-computed at the end of the training process.

When a new anomaly CAN message graph comes to the second-stage classifier, the *openmax* layer will follow Algorithm 6 for attack type classification with potential unknown anomaly rejection. From Algorithm 6, we can see that the *openmax* layer in general adapts the *softmax* function to open world recognition, which is realized by introducing an extra class c_0 (line 8 of Algorithm 6). Such a class is used to include all the anomaly CAN message graphs that are not quite similar to any existing attack type, and further infer a potential unknown class. Line 4 of Algorithm 6 generates probabilities in which the new anomaly CAN message graph belongs to a selected number of top-ranked classes or attack types. Such probabilities are derived from the corresponding fitted Weibull distributions, and used

to revise the activation vector (line 5 of Algorithm 6).

Algorithm 6: *Openmax* Attack Type Classification with Potential Unknown Anomaly Rejection

Input : $\mathcal{A}(\boldsymbol{X}) = [a_1, a_2, \cdots, a_C]^T$: activation vector of the new anomaly CAN message graph $\boldsymbol{X}$. $\bar{\mathcal{A}}_{c_i}$: MAV representing the known attack type c_i. $\{\alpha_i, \beta_i, \boldsymbol{\gamma}_i\}$: parameter set describing the fitted Weibull distribution related to the known attack type c_i. κ: number of "top" classes to revise.

Output: c^*: attack type, unknown class if $c^* = c_0$ or probability $P(c = c^*|\boldsymbol{X}) < \epsilon$

1 Initialize $\boldsymbol{S} = [s_1, s_2, \cdots, s_C] \leftarrow \boldsymbol{0}$, $\boldsymbol{\omega} = [\omega_1, \omega_2, \cdots, \omega_C] \leftarrow \boldsymbol{1}$;

2 $\boldsymbol{S} \leftarrow argsort(\mathcal{A}(\boldsymbol{X}))$;

3 **for** $i = 1, 2, \cdots, \kappa$ **do**

4 $\quad \omega_{s_i} \leftarrow 1 - \frac{\kappa - i}{\kappa} e^{-(\frac{||\mathcal{A}(\boldsymbol{X}) - \bar{\mathcal{A}}_{s_i} - \boldsymbol{\gamma}_{s_i}||_2}{\beta_{s_i}})^{\alpha_{s_i}}}$;

5 $\mathcal{A}(\boldsymbol{X}) \leftarrow \mathcal{A}(\boldsymbol{X}) \circ \boldsymbol{\omega}$; `//Element-wise product.`

6 $a_0 \leftarrow \sum_{i=1}^{C} a_i(1 - \omega_i)$;

7 **for** $i = 0, 1, \cdots, C$ **do**

8 $\quad P(c = c_i|\boldsymbol{X}) = \frac{e^{a_i}}{\sum_{j=0}^{C} e^{a_j}}$;

9 $c^* = \arg\max_c P(c = c_i|\boldsymbol{X})$;

5.6 Federated Graph Neural Network Learning

5.6.1 Vehicle State and CAN Message Relationship

Currently, existing works are based on the assumption that CAN bus intrusions can change statistical message sequences and message contents. In fact, changes of vehicle states can also cause CAN message variations but in a *reasonable* way. Intuitively, CAN message contents will change to reflect different vehicle states. On the other hand, vehicle states can also affect CAN message sequences, since some ECUs may not get activated all the time to prolong battery life in vehicles. For example, tire pressure sensors will sleep most of the time and wake up only when vehicles start to travel at high speeds (over 40 km/h), or during diagnosis and the initial CAN ID binding phases [161].

We further validate the above claims based on the dataset in [150] without any attacks, which is collected when the vehicle has an acceleration process from 0 to 30 mph (about

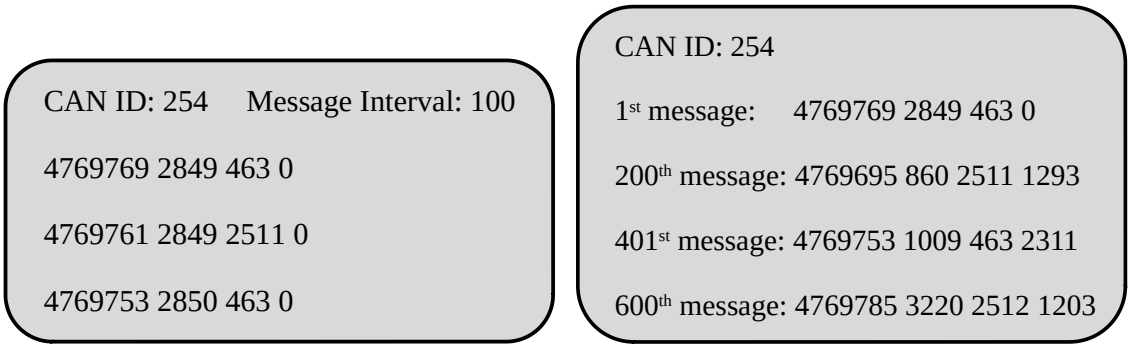


Figure 5.8: CAN message contents with CAN ID `254`. (a) In the same message interval; (b) Several message intervals apart.

48 km/h) followed by a deceleration process back to 0 mph. Therein, the targeted CAN ID is `254`, which is related to vehicle speed information, and the dataset has 707 messages with this CAN ID. Message contents are preprocessed based on READ. Fig. 5.8(a) shows all relevant CAN messages within a 100 message long interval, which includes the 1st CAN message, while Fig. 5.8(b) lists the data content of the 1st, 200th, 401st and 600th message. In a single message interval, we can consider that the vehicle is in the same state, while two CAN messages from two different intervals may correspond to two different vehicle states. From Fig. 5.8(b), we can observe obvious value changes in the second and fourth data block, which are not reflected in Fig. 5.8(a). Besides, the fourth data block seems to be related to the vehicle speed, which makes its value vary within a certain range considering the vehicle speed limit. In the other words, message content changes with CAN ID `254` should have the above two features, or be *reasonable*. Note that the value of the third data block actually jumps between around 463 (odd indices) and around 2511 (even indices), which is not considered as a change here.

Next, we show that vehicle states can also affect CAN message sequences. We split the whole dataset with 23,963 CAN messages to 239 message intervals with each having 100 messages. For each message interval, we count the number of appearing times for each possible CAN ID pair, and generate the corresponding statistical message sequence.

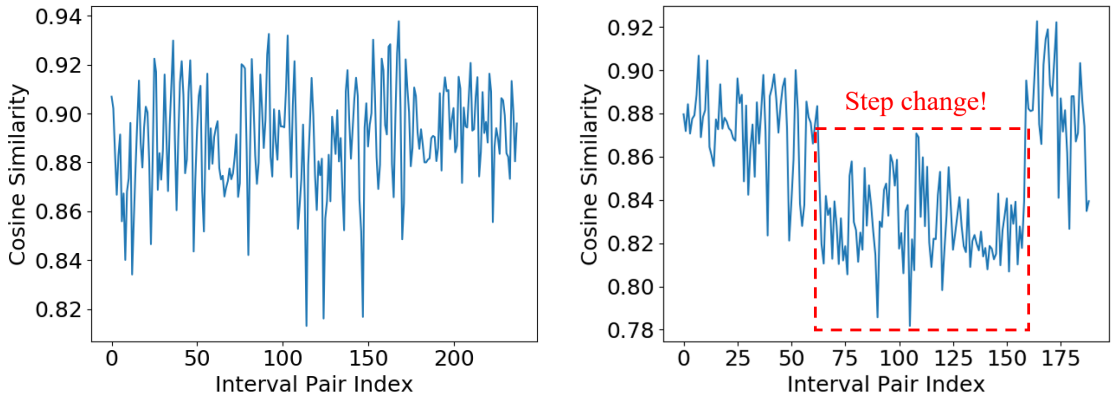


Figure 5.9: Cosine similarity comparison between 2 message sequences with (a) 1 message interval apart, and (b) 50 message intervals apart.

Note that in order to make a proper comparison, we at first go through all CAN message pairs in the dataset to record all possible CAN ID pairs. Afterwards, we compare two message sequences based on cosine similarity. Fig. 5.9 shows comparison results under two situations. In the first scenario, we compare any two consecutive CAN message sequences. In detail, cosine similarity between 1st and 2nd message sequence is indexed in 1st interval pair, cosine similarity between 2nd and 3rd message sequence is indexed in 2nd interval pair, and so on. In the second scenario, we compare any two CAN message sequences with 50 message intervals apart. In detail, cosine similarity between 1st and 51st message sequence is indexed in 1st interval pair, cosine similarity between 2nd and 52nd message sequence is indexed in 2nd interval pair, and so on.

Based on the above two comparisons, we can notice a step change when two message sequences are 50 intervals apart, which cannot be observed in the fist situation. In a short time, a.k.a when we compare two consecutive message sequences, we can consider that the vehicle state does not change. In this case, message sequences vary within a certain range, which is reflected in Fig. 5.9(a). On the other hand, two message sequences with 50 intervals apart will correspond to two different vehicle states. Since some ECUs, such as tire pressure sensors, will only get activated in some vehicle states, noticeable variations, such

as the step change in Fig. 5.9(b), can happen. In addition, message sequence changes may further have intrinsic connections with some message content variations, such as the above CAN ID 254 related to vehicle speed. In order to differentiate CAN message variations caused by vehicle state changes, we need to collect data from as many driving scenarios and vehicle states as possible, which may have to come from different vehicles considering the limitations discussed in Section 5.1. Federated learning can then be applied for model training while protecting data privacy.

5.6.2 Federated Learning

Federated learning can be seen as a type of distributed machine learning, in which a cloud server collaborates with several local users for model training. The whole training set in federated learning is actually composed of those local datasets owned by each local device, which are non-independent and identically distributed (non-IID). During model training, each local device will train a local model based on its own dataset, and only upload model parameters to the cloud server for aggregation, through which data privacy can get protected. In this research project, the non-IID property can be caused by different driving scenarios or vehicle states. In order to improve intrusion detection performance of our IDS, we consider a federated learning environment. Vehicles in different status can train our GNN model based on their own CAN bus data, and then upload model parameters to the cloud server for model aggregation.

In this research project, we evaluate two federated learning schemes, i.e. FedAvg [3] and FedProx [11], which are both based on mini-batch SGD to update local parameters. The whole training process can be divided into a certain number of communication rounds between the cloud server and vehicles. In each communication round, part of vehicles are selected to upload their model parameters to the cloud server for aggregation after local iterations. In the l-th selected vehicle, some generated CAN message graphs are selected at each local iteration. During the training of the first-stage classifier, for each selected message graph $\boldsymbol{X}_k$, we will derive the corresponding loss based on the loss function defined in Eq.

(5.5) and current local model parameters in the l-th vehicle, denoted as $f(r_l, \boldsymbol{W}_{all,l}, \boldsymbol{X}_k)$. Under mini-batch SGD, the mini-batch loss of the l-th selected vehicle can be calculated by averaging losses across all selected message graphs:

$$F(r_l, \boldsymbol{W}_{all,l}) = \frac{1}{m_l} \sum_{k \in P_l} f(r_l, \boldsymbol{W}_{all,l}, \boldsymbol{X}_k) \tag{5.7}$$

where P_l is the set of selected message graphs in the l-th vehicle at each local iteration, and m_l denotes the total number of elements in P_l.

When it comes to local model parameter update in the l-th vehicle, the mini-batch gradient $\boldsymbol{G}_\tau(r_l, \boldsymbol{W}_{all,l})$ at each local iteration τ can be derived by computing partial derivatives of $F(r_l, \boldsymbol{W}_{all,l})$ on each model parameter. Then, model parameters in the l-th vehicle will be updated as follows:

$$[r_l, \boldsymbol{W}_{all,l}]_{\tau+1} \leftarrow [r_l, \boldsymbol{W}_{all,l}]_\tau - \eta \boldsymbol{G}_\tau(r_l, \boldsymbol{W}_{all,l}) \tag{5.8}$$

Here, we use $[r_l, \boldsymbol{W}_{all,l}]_\tau$ to denote a vector which includes all parameters in our one-class GNN model. η represents the learning rate. After a given number of iterations, all the L selected vehicles will upload their learned model parameters to the cloud server for aggregation:

$$[r, \boldsymbol{W}_{all}] = \frac{1}{m} \sum_{l=1}^{L} m_l [r_l, \boldsymbol{W}_{all,l}] \tag{5.9}$$

The result is considered as the globally learned model parameters after each communication round. Finally, $[r, \boldsymbol{W}_{all}]$ will be broadcast to all vehicles for the following possible local iterations.

The above model parameter update is generally based on FedAvg, which will not necessarily provide convergence guarantee. FedProx improves the convergence performance by further introducing a proximal term. Such a term can ensure that updated local model

parameters will not drift away from the global model parameters derived in the last communication round. Mathematically, the mini-batch loss function of the i-th vehicle will be modified to:

$$F(r_l, \boldsymbol{W}_{all,l}) \leftarrow F(r_l, \boldsymbol{W}_{all,l}) + \frac{\mu}{2}||[r_l, \boldsymbol{W}_{all,l}] - [r, \boldsymbol{W}_{all}]_{pre}||^2 \tag{5.10}$$

Here, we use $[r, \boldsymbol{W}_{all}]_{pre}$ to represent the global model parameters derived in the previous communication round. μ is a hyperparameter to balance the effect of the proximal term.

The training of the second-stage classifier follows a similar process except that the traditional *softmax* and cross-entropy function are respectively used here as the output layer and loss function. In addition, the deployment of the *openmax* layer needs the Weibull fitting, which is conducted at the end of model training. In the federated learning scenario, the cloud server will at first request vehicles to upload the activation vectors of anomaly CAN message graphs and the corresponding labels in their training sets. Afterwards, the Weibull fitting will be conducted in the cloud server. Since only the activation vectors instead of raw data need to be uploaded, the privacy guarantee of federated learning is not violated. Besides, the Weibull fitting only needs one single communication round between the cloud server and vehicles, which will not induce much extra communication overhead.

5.7 Experimental Results

5.7.1 Datasets and Experiment Setup

In this section, we will conduct extensive experiments to evaluate our GNN-based IDS under CAN message injection attacks, message suspension attacks and message falsification attacks. In detail, for the first-stage classifier, we compare our IDS with three baselines on anomaly detection, which respectively represent IDSs considering statistical CAN message sequences and message contents. When it comes to the second-stage classifier, we evaluate its performance in classifying specific attack types and identifying new anomalies from unknown classes.

In the first baseline [122], the authors develop three kinds of strategies to detect CAN bus intrusions via message sequences, i.e. thresholds for cosine similarity, Pearson correlation and LSTM. Data for evaluation are collected on a Ford Transit 500 [150], which are separated into three sets. The first dataset is composed of 23,963 normal CAN messages, while message injection attacks are considered in the remaining two datasets. In the second and third dataset, CAN IDs related to vehicle speed and RPM (`254` and `115`) are respectively targeted, and compromised CAN messages are randomly injected into the these two datasets after given time spots. In total, the second and third dataset each have 88,492 and 30,308 CAN messages.

The second and third baseline [126, 132] both design IDSs based on message contents and LSTM. The evaluation dataset is *CAN Signal Extraction and Translation Dataset* provided by Hacking and Countermeasure Research Lab [162]. In order to make reasonable comparisons, we follow the strategies in [132, 163] to generate anomaly data with the five attack types discussed in Section 5.2.1, i.e. DoS, fuzzy, suspension, replay, and spoofing attack.

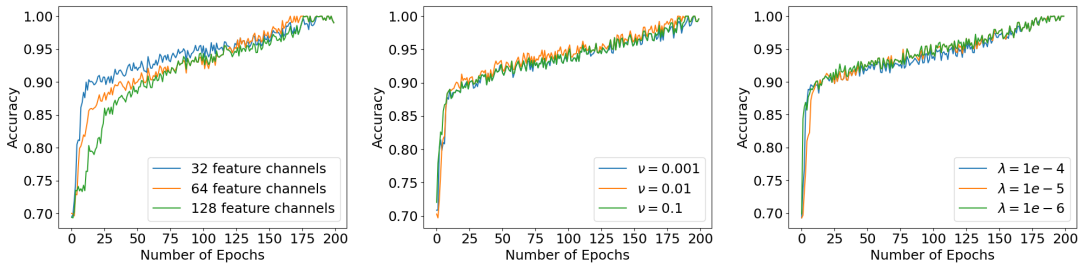
We generally follow the architecture of GNN model in [152]. Such GNN model at first has four graph convolution layers, with the last layer having only one feature channel for the convenience of node sorting, which is realized in a *SortPooling* layer. The *SortPooling* layer is followed by two 1-D convolution layers, one *MaxPooling* layer, one dense layer and one dropout layer. The original *softmax* layer is replaced with SVDD for the first-stage classifier, and the *openmax* layer is considered in the second-stage classifier. During the GNN model training, tons of hyperparamters can be tuned. In this research project, we focus on the number of feature channels in the first three graph convolution layers. We also test different settings of two hyperparameters in SVDD, i.e. ν and λ. In addition, the hyperparameter μ in FedProx will also be explored. We follow the settings in [152] for the remaining hyperparameters. In detail, the output of the *SortPooling* layer is set to a $k_s \times \sum_{t=1}^{4} c_t$ tensor, where k_s is set to a value which is not larger than the number of nodes in 60% of CAN message graphs in the training set. The first 1-D convolution layer

has 16 feature channels followed by a *MaxPooling* layer with a filter size 2 and step size 2. The second 1-D convolution layer has 32 feature channels, a filter size 5 and step size 1. The dense layer has 128 hidden neurons, followed by a dropout layer with a 0.5 drop rate. For activation functions, hyperbolic tangent function (tanh) is selected in graph convolution layers, and ReLU in all the other necessary layers. Mini-batch SGD is optimized with Adam [15].

5.7.2 Statistical CAN Message Sequences

In this subsection, we at first evaluate training performance under different GNN model settings, and select the best setting based on experimental results. In detail, we explore the number of feature channels in graph convolution layers with $\{32, 64, 128\}$, ν with $\{0.001, 0.01, 0.1\}$ and λ with $\{1e-4, 1e-5, 1e-6\}$. We use the first dataset in the first baseline as the training set, which only contains normal CAN messages. For message intervals, we test the case with a length of 100 CAN messages. The size of mini-batches is 10. Each setting is based on a 10-fold cross validation, which is further run for 10 times. The averaged accuracy curves under different hyperparameter settings are shown in Fig. 5.10. Based on those comparisons, we can see that the number of feature channels in graph convolution layers can affect convergence performance, while the other two hyperparameters do not have too much impact. In this experiment, we choose 32 feature channels, $\nu = 0.01$ and $\lambda = 1e-6$.

Afterwards, we conduct comparisons with the first baseline to show the effectiveness of our proposed IDS in detecting CAN message injection attacks. Note that in the original second and third dataset of the first baseline, data contents of injected CAN messages also get changed. In detail, all injected RPM and speed messages respectively end with "FFF" and "FFFF". In order to reflect the ability of our IDS in detecting intrusions only changing statistical message sequences, we randomly select one of normal messages with the same CAN ID in the same message interval, and use its data content in the relevant injected CAN messages. We make these modifications to simulate DoS attacks or bus-off attacks,



(a) (b) (c)

Figure 5.10: GNN model hyperparameter setting comparison related to the first baseline. (a) Feature channel; (b) ν; (c) λ.

which cannot be identified by IDSs considering CAN message contents. The comparison with the first baseline based on a 100 CAN message long interval is shown in Table 5.2, in which the considered metrics include accuracy, precision, recall and F1-score. In this table, RPM and speed respectively represent those two corresponding attacks. CS, PC, LSTM-CS and LSTM-PC respectively represent the three kinds of intrusion detection strategies in the first baseline, i.e. thresholds for Cosine Similarity, thresholds for Pearson Correlation and statistical CAN message sequence reconstruction based on LSTM for intrusion detection. GNN corresponds to our IDS. Based on the comparison, we can see that our IDS can achieve even better performance than the first baseline in detecting CAN message injection attacks.

Besides, in order to evaluate the scalability of our IDS, we further consider the situation that message intervals have different lengths during intrusion detection, i.e. 150 and 200. The results are shown in Table 5.3. From this table, we can see that our IDS can still achieve fairly high performance even when the length of message intervals during intrusion detection is different from the length during model training.

At last, we consider in this research project that message intervals longer than 200 will impact real-time performance of IDSs, since collecting 200 CAN messages will cost more than 100 ms. The average prediction time of our IDS for each CAN message graph is 3 ms.

Table 5.2: Comparison Results with the First Baseline (in %), Anomaly Threshold in the First Baseline: 0.87.

Attack	IDS	Accuracy	Prediction	Recall	F1
RPM	CS	96.65	91.40	96.59	93.92
	PC	97.32	93.48	97.73	95.56
	LSTM-CS	97.32	92.47	97.73	95.03
	LSTM-PC	96.80	91.49	97.73	94.51
	GNN	100	100	100	100
Speed	CS	89.20	68.29	88.89	77.24
	PC	90.57	71.60	92.06	80.55
	LSTM-CS	96.93	89.71	95.31	92.43
	LSTM-PC	96.93	88.57	96.88	92.54
	GNN	99.41	96.92	100	98.44

Table 5.3: Scalability Evaluation (in %)

Attack	Interval Length	Accuracy	Prediction	Recall	F1
RPM	150	97.65	93.48	97.73	95.56
	200	97.06	92.39	96.59	94.44
Speed	150	94.71	80.60	85.71	83.08
	200	96.43	87.14	96.83	91.73

5.7.3 CAN Message Contents

The related dataset used in this subsection is at first separated into a training set (the first 80%) and a test set (the last 20%). We assume here that the first 80% dataset should be able to cover most of vehicle states appearing in the last 20% of the whole data, which can be validated by the following experimental results. Similar to Section 5.7.2, the training set here also only contains normal CAN messages. The message interval length is set to 100 for both the training and test set. Although the second and third baseline consider CAN message contents for intrusion detection, they can both detect message injection (DoS and

fuzzy), message suspension, and message falsification (replay and spoofing) attack at the same time. Therefore, we also apply all the five attack types in the test set for comprehensive comparisons. Those attacks are randomly deployed in different message intervals of the test set. GNN model hyperparameters and the mini-batch size are the same as those selected in the last subsection. Table 5.4 shows related comparison results, where LSTM-P and CLAM respectively denote the second and third baseline.

Table 5.4: Comparison Results with the Second and Third Baseline (in %), Anomaly Threshold in the Second and Third Baseline: 0.83.

Attack	IDS	Accuracy	Prediction	Recall	F1
DoS	LSTM-P	97.26	94.20	91.60	92.90
	CLAM	98.75	95.70	97.10	96.40
	GNN	100	100	100	100
Fuzzy	LSTM-P	97.10	93.30	93.30	92.80
	CLAM	97.80	96.20	95.20	96.70
	GNN	98.14	97.56	97.01	97.28
Suspension	LSTM-P	97.38	91.90	93.80	92.90
	CLAM	97.90	93.90	94.90	94.40
	GNN	99.72	98.88	99.73	99.30
Replay	LSTM-P	97.25	92.90	95.80	94.30
	CLAM	97.75	94.90	95.80	95.30
	GNN	97.73	94.71	95.66	95.18
Spoofing	LSTM-P	95.75	91.80	90.80	91.30
	CLAM	97.00	94.80	92.90	93.80
	GNN	97.91	96.10	94.30	95.19

From the results, we can see that in general, our IDS can achieve comparable performance to the second and third baseline. After taking a deeper insight, we find that message falsification attacks, i.e. replay and spoofing attacks, are hard to detect if related data contents only have slight changes compared with normal CAN message contents, which explains why our IDS performs a little less effective towards message falsification attacks.

Considering the robustness of CAN bus, IDSs targeting statistical message sequences usually can achieve high performance, since injected CAN messages should overwhelm normal messages to take control of CAN buses, especially those DoS attacks, which will usually make significant changes to statistical message sequences. However, message falsification attacks can easily control CAN buses via slightly modifying normal CAN messages. For example, a speed reading change from 35 km/h to 40 km/h can trigger the activity of some ECUs, such as tire pressure sensors mentioned in Section 5.6.1. It is tricky to judge those slightly changed data contents, which sometimes can just be noise. How to balance the sensitivity of related IDSs is worth considering in practical applications.

Table 5.5: Real-Time Performance Comparison

IDS	NoO	DCT (ms)	DT (ms)
LSTM-P	100 per CAN ID	$\geq$ 991.20	7.6
CLAM	12 per CAN ID	$\geq$ 110.84	1.35
GNN	100 in total	48.87	3.20

Although LSTM-P and CLAM can detect all the three kinds of attack at the same time, they both need a certain number of observations for each CAN ID to detect intrusions based on LSTM, which can induce high data collection delay and impact real-time performance. Table 5.5 shows a real-time performance comparison with them. In this table, NoO, DCT, and DT respectively represent Number of Observations, Data Collection Time, and Detection Time for each intrusion detection. The NoO of LSTM-P and CLAM are the number for each CAN ID while our GNN corresponds to the total number of CAN messages. Based on our observations, even for CAN IDs with the highest message frequency, only 4 to 6 messages can be found in each 100 CAN messages. In this case, the total number of required CAN messages before each intrusion detection have to be much larger than 200 in LSTM-P and CLAM, or over 2 times larger than the necessary number in our proposed IDS. Such a number difference can further cause DCT difference, and seriously impact the real-time

performance, which is also reflected in Table 5.5. The DT of our proposed IDS is 3.20 ms in average. Although it is larger than the DT of CLAM, the gap is almost negligible compared with the difference of DCT. Last but not least, the memory consumption of our first-stage classifier for intrusion detection is 354.6 KB, which is also less than that of LSTM-P (13,417 KB) and CLAM (682 KB).

5.7.4 Attack Type Classification

In this subsection, we will evaluate the performance of our second-stage classifier. We still use the dataset in the last subsection, follow the strategies in [132,163] to generate anomaly data with the five attack types, and consider 100 CAN message long intervals for message graph generation. Different from the last subsection, we will apply those five attack types uniformly to the whole dataset before training (80%) and test (20%) separation. The model training of the second-stage classifier is quite similar to that of the first-stage classifier except that the traditional *softmax* and cross-entropy function are respectively used here as the output layer and loss function. GNN has the same structure and hyperpatameters as that in the first-stage classifier.

When it comes to specific attack type classification, the *openmax* layer will be used to replace the *softmax* layer. At first, let us not consider open world recognition, in which c_0 is not introduced and ϵ is set to 0. Fig. 5.11 shows a confusion matrix based on 4,740 attacked data samples, where each specific attack type owns 948 data samples. Based on the confusion matrix, we can see that our second-stage classifier works pretty well in identifying DoS and suspension attack, which can only cause difference in statistical CAN message sequences. On the other hand, replay and spoofing attack can be misclassified to each other because of CAN message content changes. It is sometimes very tricky to tell whether an attacked data sample belongs to replay or spoofing attack. Spoofing and replay attack respectively use randomly generated and previously seen CAN messages to replace normal ones. Chances are that randomly generated CAN messages can sometimes be similar to or even the same as those previously seen messages. The situation of fuzzy

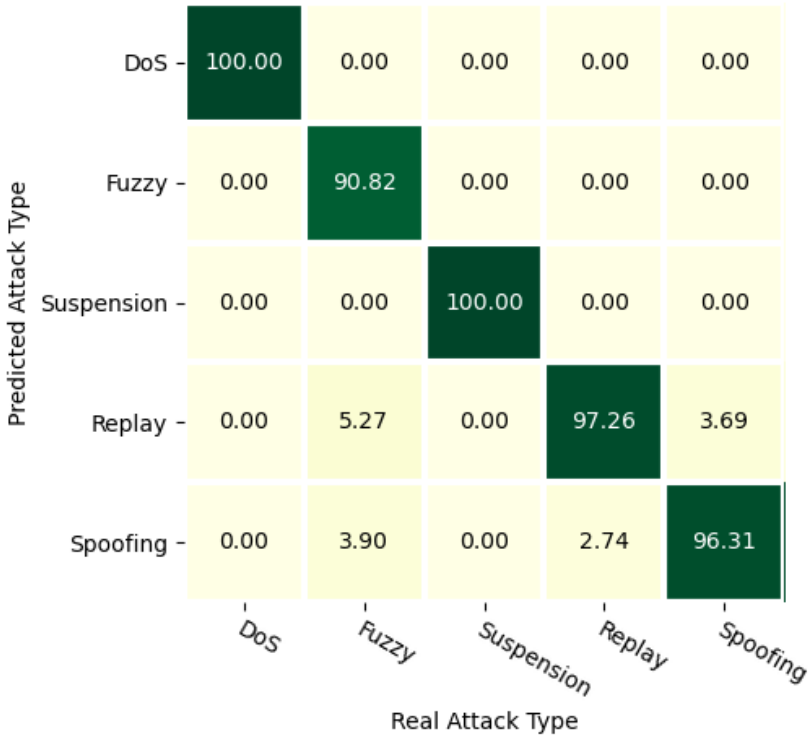


Figure 5.11: Specific attack type classification confusion matrix (in %)

attack is also complicated. Different from DoS and suspension attack, which usually focus on a single CAN ID, fuzzy attack randomly generates several CAN IDs. If a specific fuzzy attack does not cause an obvious change on the statistical CAN message sequence, it will be quite similar to a fuzzy or replay attack, which causes misclassifications shown in Fig. 5.11.

Next, let us evaluate the ability of our second-stage classifier in identifying new anomalies from unknown classes. Here, we respectively leave one attack type as the "unknown" class and consider the other four attack types during model training. For each targeted attack type, we determine the most proper probability threshold ϵ in Algorithm 6 based on the optimal cut-point in the corresponding receiver operating characteristic (ROC) curve. Table 5.6 shows the evaluation results. Note that here, we combine the four attack types used in model training to a "general" class, through which the open world recognition problem can

be seen as a two-class classification problem. In general, we can see that our second-stage classifier can effectively identify new anomalies not quite similar to any existing attack type, which usually infers unknown classes. Besides, fuzzy, replay and spoofing attack sometimes cannot be identified when they are considered as the "unknown" class. The reason behind this is actually quite similar to the above.

Table 5.6: Open World Recognition Performance Evaluation (in %)

Attack	Accuracy	Prediction	Recall	F1
DoS	100	100	100	100
Fuzzy	87.13	62.86	87.13	73.03
Suspension	100	100	100	100
Replay	95.45	84.01	95.36	89.33
Spoofing	93.38	77.90	93.35	84.93

5.7.5 Effect of Federated Learning

As has been discussed previously, vehicle states can affect statistical CAN message sequences and message contents. In this subsection, we evaluate the impact of federated learning on intrusion detection performance improvement based on the first-stage classifier. Related experiments are conducted based on datasets in the first baseline. We split the first dataset to 10 portions for local training to simulate an environment with 10 vehicles. The second and third dataset are still used as test sets. GNN model hyperparameters and the mini-batch batch size are the same as before in each simulated vehicle. The center of the hypersphere $\boldsymbol{o}$ in the one-class classification layer is set to the average of local centers derived from all simulated vehicle. We at first conduct experiments to explore the hyperparameter settings in FedProx, which are illustrated in Fig. 5.12. 10-fold cross validation is still utilized here. Considering that federated learning has model parameter aggregation after local training, accuracies are indexed by the number of communication rounds instead of epochs. Based

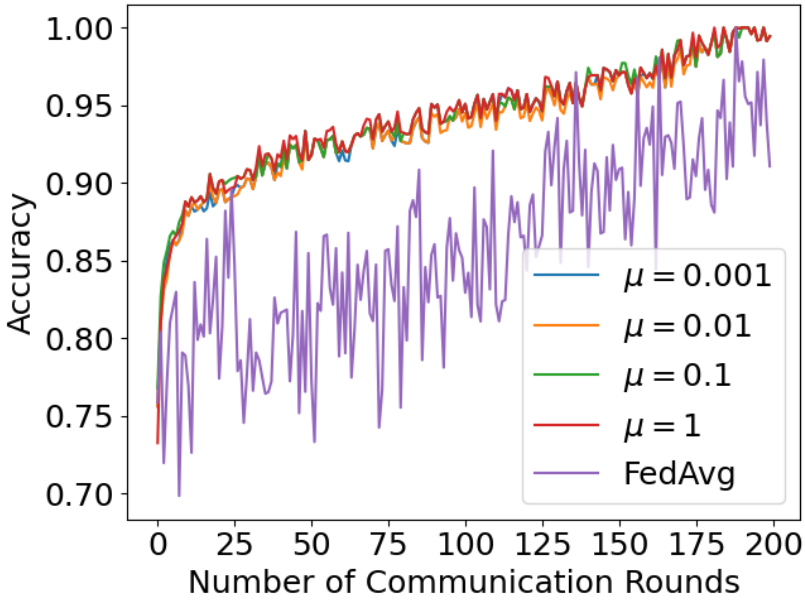


Figure 5.12: Hyperparameter setting comparison related to federated learning

on comparison results, we can see that μ does not have too much impact on convergence performance, and we select $\mu = 0.01$ in the following experiments. We also include the convergence performance of FedAvg to show the improvement of FedProx.

Table 5.7 shows the accuracy comparisons based on test sets and GNN model parameters after 120 communication rounds. For the situation without considering federated learning (Without FL), we only use the data in the first simulated vehicle for training. Based on the comparisons, we can see that intrusion detection performance can get seriously degraded if training sets only cover limited vehicle states. In the real-world, such situations can happen if we only track one vehicle having limited driving scenarios. However, this issue can be solved by collecting and integrating multiple local models derived from different vehicles. For more comprehensive comparison, we also include the traditional centralized learning scenario. We can see that the convergence performance of FedProx is comparable to that of centralized learning. We also did related evaluation for the second-stage classifier, and figured out similar trends.

Table 5.7: Accuracy Comparison of Federated Learning (in %)

		Without FL	FedAvg	FedProx	Centralized
RPM	100	32.01	80.03	99.04	100
	150	34.32	81.35	94.65	97.65
	200	36.30	88.61	97.65	97.06
Speed	100	33.33	78.82	99.41	99.41
	150	29.04	74.71	97.62	94.71
	200	37.29	84.98	98.82	96.43

5.8 Open Problems on CAN Bus IDSs based on Federated Learning

Several open problems still exist in further extending the CAN bus IDS proposed in this research project, especially after the introduction of federated learning.

- As is shown in Fig. 5.2, once anomalies are rejected as from unknown classes, they will be buffered for further investigation. If necessary, new anomalies can be further tackled by transfer learning [137] or extra one-class classifier training [136]. However, introducing new layers to existing learning models or directly considering extra classifiers can increase computation complexity, which may exceed the computation capacity of participating vehicles. In this case, a light-weighted learning model is quite necessary.

- In this research project, we assume that all vehicles participating in federated learning are from the same model. The lack of relevant public specifications can cause large divergence on CAN message streams from different vehicle models, which further brings challenges to the related machine learning model design. A potential strategy to tackle this challenge is to train an individual IDS for each vehicle model, and further combine them together through hierarchical structures.

- In this research project, we also assume that both vehicles and cloud servers are honest when developing the IDS. Since the IDS is constructed based on federated learning, poisoning attacks can also happen during model training. In this case, how to filter out these attacks can be a new challenge. Currently, Byzantine-tolerant distributed learning is widely considered as one potential solution. However, existing schemes either assume that the participants' training data are independent and identically distributed [164, 165], which violates the assumption of federated learning, or greatly degrade the convergence performance of federated learning [166].

5.9 Summary

In this research project, we propose a CAN bus IDS based on GNN, which can efficiently detect CAN message injection, suspension, and falsification attacks at the same time. A CAN message graph is developed here to integrate statistical message sequences with message contents. A GNN fit for directed attributed graphs is constructed and trained to predict intrusions. Considering that attack data are hard to acquire in the real world, which may cause highly imbalanced training sets, we develop a two-stage classifier cascade to tackle normal and attacked CAN data respectively. In the first-stage classifier, we replace the traditional *softmax* layer in GNN with a one-class classification layer for anomaly detection and train the GNN only with normal CAN messages. Once anomaly CAN data are spotted, they will be further passed to the second-stage classifier to determine the specific attack type, in which an *openmax* layer is introduced to tackle anomalies from potential unknown classes. We also notice that changes of vehicle states can affect CAN message graph patterns in a reasonable way. In this case, federated learning is considered to cover a wide range of driving scenarios and vehicle states while protecting user data privacy. We validate our IDS based on several real-world datasets and comparisons with three baselines. Experimental results show that our IDS can achieve similar performance to those IDSs only based on statistical CAN message sequences and message contents.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

In this dissertation, we provide comprehensive analysis about optimizing communication overhead in federated learning from the following three aspects. At first, federated learning needs several communication rounds between local devices and public servers for model aggregation, which are mostly through wireless networks. Besides, since uploaded model parameters in federated learning are still vulnerable to inference attacks, extra privacy masks are needed for them, such as Secret Sharing (SS) and secure Multi-Party Computation (MPC), which can further aggravate communication. In addition, the number of communication rounds can soar if adversaries try to slow convergence of federated learning by poisoning local data or uploaded model parameters.

In detail, we conduct the following four research projects. At first, We consider reducing communication overhead through convergence performance optimization in federated learning. We aim at achieving comparable final convergence accuracy under lower communication frequency and higher communication rate. The proposed optimization algorithm targets the model parameter update rule in federated learning, called *FedUR*. This algorithm introduces centralized machine learning-based adaptive learning strategies to federated learning. Convergence upper bounds under our optimization scheme are derived after each communication round with a certain number of local iterations, and after a given number of communication rounds. Through comparison with the bounds of original federated learning, we theoretically analyze how those strategies should be tuned to help federated learning effectively optimize convergence performance and reduce communication overhead.

Next, we focus on secret share placement after using SS and MPC to further protect uploaded model parameters. We find that the recently proposed mobile edge computing

relieves the concern about the communication between local devices and public servers, but raises new challenges in communication and computation management among edge servers. In this case, we propose a privacy-preserving task scheduling strategy based on (2,2) SS and mobile edge computing to reduce data processing latency, in which locally-learned model parameters are separated into two portions before uploaded to public edge servers for parameter aggregation based on MPC. We show that the related privacy constraint can be enforced through constructing a pairwise Markov chain, and carefully designing system states and transition probabilities. We further formulate the whole task scheduling problem as a stochastic latency minimization problem and solve it by converting it into a linear programming problem.

We further extend the (2,2) case to (R,L) case, and propose a communication-aware secret share placement strategy to optimize communication overhead by minimizing weighted transmission hop counts in a hierarchical edge computing architecture. The computation and storage capacity of each edge node are considered by applying a limitation with regard to queue length. We define a secret matrix $\boldsymbol{C}_a$ and a share matrix $\boldsymbol{C}_b$ to describe all communication among shares of different secrets with the same index and all communication among different shares of the same secret. We show that the constructed optimization problem is NP-hard, and efficient heuristic algorithms can be applied to find sub-optimal solutions. We respectively evaluate two traditional heuristics, i.e. Genetic Algorithm (GA) and Particle Swarm Optimization (PSO), introduce two basic heuristics, i.e. top-down and bottom-up heuristic, and further propose an advanced algorithm, called Bottom-Up Top-Down ($BUTD$) heuristic. Based on comparison, we find that our proposed $BUTD$ heuristic can outperform all the other four heuristics when communication among different shares of the same secret is comparable to that among different secrets.

Finally, we validate the value and utility of federated learning based on a well-known research problem in cybersecurity, i.e. CAN bus intrusion detection. Our proposed CAN bus intrusion detection system is designed based on Graph Neural Network (GNN). This work generates directed attributed graphs based on CAN message streams in given message

intervals. Afterwards, a GNN is trained based on generated CAN message graphs. Considering highly imbalanced training data, a two-stage classifier cascade is developed, which is composed of a one-class classifier for anomaly detection and a multi-class classifier for attack classification. An *openmax* layer is further introduced to the multi-class classifier to tackle new anomalies from unknown classes. We show that different driving scenarios and vehicle states will impact sequence patterns and data contents of CAN messages, which further cause changes in CAN message graphs. In this case, we develop a federated learning architecture to accelerate the learning process while preserving data privacy.

6.2 Future Work

In the future, we want to further explore the following research directions.

6.2.1 Future Work on Federated Learning Optimization

System and data statistical heterogeneity are widely considered when it comes to convergence performance optimization in federated learning. For system heterogeneity, it is usually about end device synchronization [5–7]. As to data statistical heterogeneity, existing works have considered battling unbalanced and non-IID data distributions across different end devices through end device selection (*FAVOR* [8] and *FOLB* [9]), batch size optimization (*CABS* [58], *BA-SGD* [59], and *Adaptive-B* [10]), and cross-client variance reduction (*FedProx* [11]), *VRL-SGD* [12], and *SCAFFOLD* [13]). Our proposed algorithm in Chapter 2 is orthogonal to all the above federated learning optimization strategies, which are developed based on the original federated learning or *FedAvg*. A proper integration with those strategies can further improve convergence performance.

If we further expand the scope to communication overhead optimization, uploaded model parameter compression has recently been considered [40,41,43]. Therein, a quantized version of local model parameters are uploaded to the central server for aggregation, which will bring new challenges to central servers on deriving global momentum and adaptive learning rate. In addition, communication management schemes, such as [48], can be integrated into our

algorithm to further optimize communication between end devices and central servers if resource constraints exist.

6.2.2 Future Work on Privacy and Security in Federated Learning

Recent works have shown that federated learning can get negatively affected by poisoning attacks [35–38] (security issues). In addition, uploaded model parameters in federated learning are vulnerable to inference attacks (privacy issues) [18–21]. For poisoning attacks, besides developing self-learning intrusion detection systems, Byzantine-tolerant distributed learning is another solution. However, the authors [36] have pointed out that the current relevant schemes are based on the assumption that local data owned by different local clients are Identical and Independently Distributed (IID). In other words, how to extend them to federated learning with non-IID data distributions still needs to be explored. As to inference attacks, an even higher privacy-preserving level can be reached by protecting uploaded model parameters based on secret sharing [22,23], secure multi-party computation [24,25], or differential privacy [26,67]. However, the first two strategies can further increase communication overhead if local model parameters need to be separated into several pieces and uploaded to different central servers. In this dissertation, we have proposed two secret share placement strategies to optimize computation latency and communication overhead in mobile edge computing. Several deployment issues still need to be tackled, which have been discussed in Section 4.8. In addition, differential privacy masks uploaded model parameters through adding random Gaussian noise. Therefore, there will be tradeoff between the privacy protection level and convergence performance [27], which can further influence communication overhead. The design of differential privacy policies still remains largely open in federated learning.

6.2.3 Future Work on Federated CAN Bus Intrusion Detection

New anomalies identified by the proposed CAN bus IDS in this dissertation can be further tackled by transfer learning [137] or extra one-class classifier training [136]. However, introducing new layers to existing learning models or directly considering extra classifiers can increase computation complexity, which may exceed the computation capacity of participating vehicles. In this case, a light-weighted learning model is quite necessary. Besides, in the proposed IDS, we assume that all vehicles participating in federated learning are from the same model. The lack of relevant public specifications can cause large divergence on CAN message streams from different vehicle models, which further brings challenges to the related machine learning model design. A potential strategy to tackle this challenge is to train an individual IDS for each vehicle model, and further combine them together through hierarchical structures. Furthermore, we also assume that both vehicles and cloud servers are honest. As has been mentioned above, poisoning attacks in federated learning have recently drawn increasing attention [35–38]. Since the IDS is constructed based on federated learning, poisoning attacks can also happen during model training. In this case, how to filter out these attacks can be a new challenge. Currently, Byzantine-tolerant distributed learning is widely considered as one potential solution. However, existing schemes either assume that the participants' training data are IID [164,165], which violates the assumption of federated learning, or greatly degrade the convergence performance of federated learning [166]. Further exploration is necessary.

Appendix A: Missing Proofs

A.1 Proof of Theorem 2.4.1

In this proof, we will use $\boldsymbol{m}(t)$ to approximate $\hat{\boldsymbol{m}}(t)$, since $\beta_1 < 1$, and based on Formula (2.7), we should have:

$$\lim_{t\to\infty} \hat{m}_i(t) = \lim_{t\to\infty} \frac{m_i(t)}{1-\beta_1^t} = m_i(t) \tag{A.1}$$

At the beginning of each interval $[j]$, we assume $\boldsymbol{w}_{[j],k}((j-1)\tau) = \boldsymbol{w}_{[j]}((j-1)\tau)$, and further $\boldsymbol{m}_{[j],k}((j-1)\tau) = \boldsymbol{m}_{[j]}((j-1)\tau)$. Then, based on the following two formulae:

$$m_i(t) \leftarrow \beta_1 m_i(t-1) + (1-\beta_1) G_i(t) \tag{A.2a}$$

$$\hat{m}_i(t) = \frac{1-\beta_1}{1-\beta_1^t} \sum_{c=1}^{t} \beta_1^{t-c} G_i(c) \tag{A.2b}$$

we will have:

$$\begin{aligned}
&\boldsymbol{m}_{[j],k}(t) - \boldsymbol{m}_{[j]}(t) \\
=&[\beta_1^{t-(j-1)\tau} \boldsymbol{m}_{[j],k}((j-1)\tau) + (1-\beta_1) \sum_{c=0}^{t-(j-1)\tau-1} \beta_1^c \boldsymbol{G}_{[j],k}(t-c)] - [\beta_1^{t-(j-1)\tau} \boldsymbol{m}_{[j]}((j-1)\tau) \\
&+ (1-\beta_1) \sum_{c=0}^{t-(j-1)\tau-1} \beta_1^c \boldsymbol{G}_{[j]}(t-c)] \\
=&(1-\beta_1) \sum_{c=0}^{t-(j-1)\tau-1} \beta_1^c (\boldsymbol{G}_{[j],k}(t-c) - \boldsymbol{G}_{[j]}(t-c))
\end{aligned} \tag{A.3}$$

Next, let's discuss the upper bound in Theorem 2.4.1. As has been discussed in Section

2.4.2, as long as we get the upper bound of $||\boldsymbol{w}^F_{[j]}(t)-\boldsymbol{w}_{[j]}(t)||$, we can derive the upper bound of $|F(\boldsymbol{w}^F_{[j]}(t))-F(\boldsymbol{w}_{[j]}(t))|$ through the *ρ-Lipschitz* assumption. For $||\boldsymbol{w}^F_{[j]}(t)-\boldsymbol{w}_{[j]}(t)||$, we have:

$$
\begin{aligned}
&||\boldsymbol{w}^F_{[j]}(t)-\boldsymbol{w}_{[j]}(t)|| \\
=&||\boldsymbol{w}^F_{[j]}(t-1)-\frac{\sum_{k=1}^K N_k\eta\boldsymbol{m}_{[j],k}(t-1)}{N}-\boldsymbol{w}_{[j]}(t-1)+\eta\boldsymbol{m}_{[j]}(t-1)|| \\
\leq&||\boldsymbol{w}^F_{[j]}(t-1)-\boldsymbol{w}_{[j]}(t-1)|| \\
&+\eta\frac{\sum_{k=1}^K N_k||\boldsymbol{m}_{[j],k}(t-1)-\boldsymbol{m}_{[j]}(t-1)||}{N} && \text{(Triangle inequity)} \\
=&||\boldsymbol{w}^F_{[j]}(t-1)-\boldsymbol{w}_{[j]}(t-1)||+\eta\beta(1-\beta_1) \\
&\frac{\sum_{k=1}^K N_k||\sum_{c=0}^{t-(j-1)\tau-1}\beta_1^c(\boldsymbol{w}_{[j],k}(t-c)-\boldsymbol{w}_{[j]}(t-c))||}{N} && \text{(Formula (A.3) and β-\textit{smooth})} \\
\leq&||\boldsymbol{w}^F_{[j]}(t-1)-\boldsymbol{w}_{[j]}(t-1)||+\eta\beta(1-\beta_1) \\
&\sum_{c=0}^{t-(j-1)\tau-1}\beta_1^c\frac{\sum_{k=1}^K N_k||(\boldsymbol{w}_{[j],k}(t-c)-\boldsymbol{w}_{[j]}(t-c))||}{N} && \text{(Triangle inequity)} \\
=&||\boldsymbol{w}^F_{[j]}(t-1)-\boldsymbol{w}_{[j]}(t-1)||+\eta\beta(1-\beta_1) \\
&\sum_{c=0}^{t-(j-1)\tau-1}\beta_1^c\frac{\sum_{k=1}^K N_k\delta_k((\eta\beta-1)^{t-c-1-(j-1)\tau}-1)}{N\beta} && \text{(Lemma 3 in [48])} \\
=&||\boldsymbol{w}^F_{[j]}(t-1)-\boldsymbol{w}_{[j]}(t-1)||+\eta\delta(1-\beta_1)\sum_{c=0}^{t-(j-1)\tau-1}\beta_1^c((\eta\beta-1)^{t-c-1-(j-1)\tau}-1) && \text{(A.4)}
\end{aligned}
$$

In other words, we have:

$$||\boldsymbol{w}_{[j]}^F(t) - \boldsymbol{w}_{[j]}(t)|| - ||\boldsymbol{w}_{[j]}^F(t-1) - \boldsymbol{w}_{[j]}(t-1)||$$

$$\leq \eta\delta(1-\beta_1) \sum_{c=0}^{t-(j-1)\tau-1} \beta_1^c((\eta\beta-1)^{t-c-1-(j-1)\tau}-1) \tag{A.5}$$

By induction, we can have the following derivation. Note that we assume $\boldsymbol{w}_{[j],k}((j-1)\tau) = \boldsymbol{w}_{[j]}((j-1)\tau)$.

$$||\boldsymbol{w}_{[j]}^F(t) - \boldsymbol{w}_{[j]}(t)||$$

$$= \sum_{y=(j-1)\tau+1}^{t} ||\boldsymbol{w}_{[j]}^F(y) - \boldsymbol{w}_{[j]}(y)|| - ||\boldsymbol{w}_{[j]}^F(t-1) - \boldsymbol{w}_{[j]}(t-1)||$$

$$\leq \eta\delta(1-\beta_1) \sum_{y=(j-1)\tau+1}^{t} \sum_{c=0}^{y-(j-1)\tau-1} \beta_1^c((\eta\beta-1)^{y-c-1-(j-1)\tau}-1)$$

$$= \eta\delta(1-\beta_1) \sum_{y=(j-1)\tau+1}^{t} \sum_{c=0}^{y-(j-1)\tau-1} \frac{(\frac{\beta_1}{\eta\beta+1})^c}{(\eta\beta+1)^{(j-1)\tau-y+1}} - \beta_1^c$$

$$= \eta\delta(1-\beta_1) \sum_{y=(j-1)\tau+1}^{t} \frac{1}{(\eta\beta+1)^{(j-1)\tau-y+1}} \cdot$$

$$\frac{(\eta\beta+1)^{y-(j-1)\tau} - \beta_1^{y-(j-1)\tau}}{(\eta\beta+1-\beta_1)(\eta\beta+1)^{y-(j-1)\tau-1}} - \frac{1-\beta_1^{y-(j-1)\tau}}{1-\beta_1} \qquad \text{(Geometric series)}$$

$$= \eta\delta(1-\beta_1) \sum_{y=(j-1)\tau+1}^{t} \frac{(\eta\beta+1)^{y-(j-1)\tau} - \beta_1^{y-(j-1)\tau}}{\eta\beta+1-\beta_1} - \frac{1-\beta_1^{y-(j-1)\tau}}{1-\beta_1}$$

$$= \eta\delta(1-\beta_1) \sum_{z=1}^{t-(j-1)\tau} \frac{(\eta\beta+1)^z - \beta_1^z}{(\eta\beta+1)-\beta_1} - \frac{1-\beta_1^z}{1-\beta_1} \tag{A.6}$$

Based on the following observation:

$$\frac{a^z - b^z}{a-b} = a^{z-1} + \frac{b(a^{z-1} - b^{z-1})}{a-b} \tag{A.7}$$

We can further have:

$$\begin{aligned}
&||\boldsymbol{w}^F_{[j]}(t) - \boldsymbol{w}_{[j]}(t)|| \\
=&\eta\delta(1-\beta_1)\sum_{z=1}^{t-(j-1)\tau}(\eta\beta+1)^{z-1} - 1 + \frac{\beta_1[(\eta\beta+1)^{z-1} - \beta_1^{z-1}]}{(\eta\beta+1)-\beta_1} - \frac{\beta_1(1-\beta_1^{z-1})}{1-\beta_1} \\
=&\eta\delta\sum_{z=1}^{t-(j-1)\tau}[(\eta\beta+1)^{z-1}-1] + \eta\delta\sum_{z=1}^{t-(j-1)\tau}\{-\beta_1(\eta\beta+1)^{z-1} + \beta_1 \\
&+ \frac{\beta_1(1-\beta_1)}{\eta\beta+1-\beta_1}[(\eta\beta+1)^{z-1} - \beta_1^{z-1}] - \beta_1 + \beta_1\beta_1^{z-1}\} \\
=&\eta\delta\sum_{z=1}^{t-(j-1)\tau}[(\eta\beta+1)^{z-1}-1] + \eta\delta(\frac{\beta_1(1-\beta_1)}{\eta\beta+1-\beta_1} - \beta_1)\sum_{z=1}^{t-(j-1)\tau}(\eta\beta+1)^{z-1} \\
&- \eta\delta(\frac{\beta_1(1-\beta_1)}{\eta\beta+1-\beta_1} - \beta_1)\sum_{z=1}^{t-(j-1)\tau}\beta_1^{z-1}
\end{aligned} \tag{A.8}$$

Set $x = t - (j-1)\tau$, we will have:

$$\begin{aligned}
&||\boldsymbol{w}^F_{[j]}(t) - \boldsymbol{w}_{[j]}(t)|| \\
=&\{\frac{\delta}{\beta}[(\eta\beta+1)^x - 1] - \eta\delta x\} \\
&- \{\frac{\eta\delta\beta_1[(\eta\beta+1)^x-1]}{\eta\beta+1-\beta_1} - \frac{\eta^2\delta\beta\beta_1(1-\beta_1^{x-1})}{(\eta\beta+1-\beta_1)(1-\beta_1)}\} \qquad \text{(Geometric series)} \\
=&h_1(x) - h_2(x)
\end{aligned} \tag{A.9}$$

A.2 Proof of Theorem 2.4.2

We at first define an equivalent learning rate $\bar{\eta}_{[j]}$, which meets the following relationship with I model parameters:

$$\begin{aligned}
&\bar{\eta}_{[j]} \sum_{k=1}^{K} ||\boldsymbol{G}_{[j],k}(t) - \boldsymbol{G}_{[j]}(t)|| \\
=&\sum_{k=1}^{K} \sqrt{\sum_{i=1}^{I} [\eta_{[j],i}(G_{[j],k,i}(t) - G_{[j],i}(t))]^2} \\
=&\sum_{k=1}^{K} ||\boldsymbol{\eta}_{[j]} \circ (\boldsymbol{G}_{[j],k}(t) - \boldsymbol{G}_{[j]}(t))|| \qquad \text{(A.10)}
\end{aligned}$$

Here, the symbol "$\circ$" represents the element-wise or Hadamard product of two vectors. Note that we will definitely have:

$$\min_{i \in \{1,2,\cdots,I\}} \eta_{[j],i} \leq \bar{\eta}_{[j]} \leq \max_{i \in \{1,2,\cdots,I\}} \eta_{[j],i} \qquad \text{(A.11)}$$

We still just need to look at $||\boldsymbol{w}^F_{[j]}(t) - \boldsymbol{w}_{[j]}(t)||$, considering the *$\rho$-Lipschitz* assumption. Besides, at the beginning of each interval $[j]$, we still assume $\boldsymbol{w}_{[j],k}((j-1)\tau) = \boldsymbol{w}_{[j]}((j-1)\tau)$, and further $\boldsymbol{G}_{[j],k}((j-1)\tau) = \boldsymbol{G}_{[j]}((j-1)\tau)$. Furthermore, we assume in this proof that in each interval $[j]$, $\hat{\boldsymbol{u}}_{[j]}$ can be used to approximate $\hat{\boldsymbol{u}}(t)$. In other words, we assume $\boldsymbol{\eta}_{[j]}(t) \approx \boldsymbol{\eta}_{[j]}$ for $\forall t \in [j]$.

$$\begin{aligned}
&||\boldsymbol{w}^F_{[j]}(t) - \boldsymbol{w}_{[j]}(t)|| \\
=&||\boldsymbol{w}^F_{[j]}(t-1) - \frac{\sum_{k=1}^{K} N_k \boldsymbol{\eta}_{[j]} \circ \boldsymbol{G}_{[j],k}(t-1)}{N} - \boldsymbol{w}_{[j]}(t-1) + \boldsymbol{\eta}(t) \circ \boldsymbol{G}_{[j]}(t-1)|| \\
\leq&||\boldsymbol{w}^F_{[j]}(t-1) - \boldsymbol{w}_{[j]}(t-1)||
\end{aligned}$$

$$+\frac{\sum_{k=1}^{K} N_k||\boldsymbol{\eta}_{[\boldsymbol{j}]} \circ (\boldsymbol{G}_{[j],k}(t-1) - \boldsymbol{G}_{[j]}(t-1))||}{N} \qquad \text{(Triangle inequity)}$$

$$=||\boldsymbol{w}^F_{[j]}(t-1) - \boldsymbol{w}(t-1)||$$

$$+\bar{\eta}_{[j]}\beta\frac{\sum_{k=1}^{K} N_k||\boldsymbol{w}_{[j],k}(t-1) - \boldsymbol{w}_{[j]}(t-1)||}{N} \qquad \text{(Formula (A.10) and } \beta\text{-}smooth\text{)}$$

$$=||\boldsymbol{w}^F_{[j]}(t-1) - \boldsymbol{w}_{[j]}(t-1)||$$

$$+\bar{\eta}_{[j]}\beta\frac{\sum_{k=1}^{K} N_k\delta_k((\bar{\eta}_{[j]}\beta-1)^{t-1-(k-1)\tau}-1)}{N\beta} \qquad \text{(Lemma 3 in [48])}$$

$$=||\boldsymbol{w}^F_{[j]}(t-1) - \boldsymbol{w}_{[j]}(t-1)|| + \bar{\eta}_{[j]}\delta((\bar{\eta}_{[j]}\beta-1)^{t-1-(j-1)\tau}-1) \qquad \text{(A.12)}$$

Then, following the induction process similar to that in Appendix A.1 and setting $x = t-(j-1)\tau$, we can derive:

$$||\boldsymbol{w}^F_{[j]}(t) - \boldsymbol{w}_{[j]}(t)|| \leq \frac{\delta}{\beta}[(\bar{\eta}_{[j]}\beta+1)^x - 1] - \bar{\eta}_{[j]}\delta x = h_3(x) \qquad \text{(A.13)}$$

Bibliography

Bibliography

[1] M. Anderson, "Technology device ownership: 2015," *Pew Research Center*, 2015.

[2] J. Poushter *et al.*, "Smartphone ownership and internet usage continues to climb in emerging economies," *Pew Research Center*, vol. 22, no. 1, pp. 1–44, 2016.

[3] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.

[4] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[5] J. Jiang, B. Cui, C. Zhang, and L. Yu, "Heterogeneity-aware distributed parameter servers," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 463–478.

[6] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *arXiv preprint arXiv:2007.07481*, 2020.

[7] J. Liu, H. Xu, L. Wang, Y. Xu, C. Qian, J. Huang, and H. Huang, "Adaptive asynchronous federated learning in resource-constrained edge computing," *IEEE Transactions on Mobile Computing (Early Access)*, 2021.

[8] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, 2020, pp. 1698–1707.

[9] H. T. Nguyen, V. Sehwag, S. Hosseinalipour, C. G. Brinton, M. Chiang, and H. V. Poor, "Fast-convergent federated learning," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 201–218, 2021.

[10] J. Zhang, S. Guo, Z. Qu, D. Zeng, Y. Zhan, Q. Liu, and R. A. Akerkar, "Adaptive federated learning on non-iid data with resource constraint," *IEEE Transactions on Computers (Early Access)*, 2021.

[11] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[12] X. Liang, S. Shen, J. Liu, Z. Pan, E. Chen, and Y. Cheng, "Variance reduced local SGD with lower communication complexity," *arXiv preprint arXiv:1912.12844*, 2019.

[13] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic controlled averaging for on-device federated learning," *arXiv preprint arXiv:1910.06378*, 2019.

[14] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[16] M. Zaheer, S. Reddi, D. Sachan, S. Kale, and S. Kumar, "Adaptive methods for nonconvex optimization," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[17] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.

[18] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 3–18.

[19] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 603–618.

[20] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 691–706.

[21] M. Song, Z. Wang, Z. Zhang, Y. Song, Q. Wang, J. Ren, and H. Qi, "Analyzing user-level privacy attack against federated learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2430–2444, 2020.

[22] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[23] H. Zhang and K. Zeng, "Communication-aware secret share placement in hierarchical edge computing," *IEEE Internet of Things Journal (Early Access)*, vol. 9, no. 5, pp. 3717–3728, 2021.

[24] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *2017 IEEE symposium on security and privacy (SP)*, 2017, pp. 19–38.

[25] Y. Li, Y. Zhou, A. Jolfaei, D. Yu, G. Xu, and X. Zheng, "Privacy-preserving federated learning framework based on chained secure multiparty computing," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6178–6186, 2020.

[26] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.

[27] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020.

[28] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018.

[29] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[30] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[31] A. Aldhalaan and D. A. Menascé, "Autonomic allocation of communicating virtual machines in hierarchical cloud data centers," in *2014 International Conference on Cloud and Autonomic Computing*. IEEE, 2014, pp. 161–171.

[32] B. S. Gu, L. Gao, X. Wang, Y. Qu, J. Jin, and S. Yu, "Privacy on the edge: Customizable privacy-preserving context sharing in hierarchical edge computing," *IEEE Transactions on Network Science and Engineering (Early Access)*, pp. 1–12, 2019.

[33] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, 2016, pp. 1–9.

[34] A. Kiani and N. Ansari, "Toward hierarchical mobile edge computing: An auction-based profit maximization approach," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2082–2091, 2017.

[35] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Model poisoning attacks in federated learning," in *Proceedings of the Workshop on Security in Machine Learning (SecML) 32nd Conference on Neural Information Processing System (NeurIPS)*, 2018, pp. 1–23.

[36] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2938–2948.

[37] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to {Byzantine-Robust} federated learning," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1605–1622.

[38] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *European Symposium on Research in Computer Security*. Springer, 2020, pp. 480–501.

[39] H. Zhang and K. Zeng, "FedUR: Federated learning optimization through adaptive centralized learning optimizers," *IEEE Transactions on Signal Processing (Under Review)*, 2022.

[40] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint arXiv:1712.01887*, 2017.

[41] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-i.i.d. data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3400–3413, 2020.

[42] N. Shlezinger, M. Chen, Y. C. Eldar, H. V. Poor, and S. Cui, "UVeQFed: Universal vector quantization for federated learning," *IEEE Transactions on Signal Processing*, vol. 69, pp. 500–514, 2020.

[43] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "FedPAQ: A communication-efficient federated learning method with periodic averaging and quantization," in *Proceedings of the PMLR International Conference on Artificial Intelligence and Statistics*, 2020, pp. 2021–2031.

[44] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 10, pp. 4229–4238, 2020.

[45] T. Chen, G. B. Giannakis, T. Sun, and W. Yin, "Lag: Lazily aggregated gradient for communication-efficient distributed learning," *arXiv preprint arXiv:1805.09965*, 2018.

[46] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7.

[47] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 1, pp. 269–283, 2021.

[48] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[49] X. Meng *et al.*, "MLlib: Machine learning in apache spark," *Journal of Machine Learning Research*, vol. 17, pp. 1–7, 2016.

[50] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on Spark," in *First International Workshop on Graph Data Management Experiences and Systems*, 2013, pp. 1–6.

[51] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012, pp. 15–28.

[52] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016, pp. 265–283.

[53] J. Dean *et al.*, "Large scale distributed deep networks," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1223–1231, 2012.

[54] F. Niu, B. Recht, C. Ré, and S. J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," *arXiv preprint arXiv:1106.5730*, 2011.

[55] W. Dai, J. Wei, X. Zheng, J. K. Kim, S. Lee, J. Yin, Q. Ho, and E. P. Xing, "Petuum: A framework for iterative-convergent distributed ML," *arXiv preprint arXiv:1312.7651*, 2013.

[56] X. Shi, B. Cui, Y. Shao, and Y. Tong, "Tornado: A system for real-time iterative analysis over evolving data," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 417–430.

[57] Z. Qu, S. Guo, H. Wang, B. Ye, Y. Wang, A. Zomaya, and B. Tang, "Partial synchronization to accelerate federated learning over relay-assisted edge networks," *IEEE Transactions on Mobile Computing (Early Access)*, 2021.

[58] L. Balles, J. Romero, and P. Hennig, "Coupling adaptive batch sizes with learning rates," *arXiv preprint arXiv:1612.05086*, 2016.

[59] P. Yin, P. Luo, and T. Nakamura, "Small batch or large batch?: Gaussian walk with rebound can teach," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1275–1284.

[60] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smithy, "FedDANE: A federated Newton-type method," in *Proceedings of the 53rd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2019, pp. 1227–1231.

[61] X. Zhang, M. Hong, S. Dhople, W. Yin, and Y. Liu, "FedPD: A federated learning framework with adaptivity to non-iid data," *IEEE Transactions on Signal Processing*, vol. 69, pp. 6055–6070, 2021.

[62] W. Liu, L. Chen, Y. Chen, and W. Zhang, "Accelerating federated learning via momentum gradient descent," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1754–1766, 2020.

[63] J. Mills, J. Hu, and G. Min, "Communication-efficient federated learning for wireless edge intelligence in IoT," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5986–5994, 2020.

[64] S. Bubeck, "Convex optimization: Algorithms and complexity," *arXiv preprint arXiv:1405.4980*, 2014.

[65] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[66] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009. [Online]. Available: http://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf

[67] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019, pp. 1–11.

[68] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[69] H. Zhang and K. Zeng, "Pairwise Markov chain: A task scheduling strategy for privacy-preserving sift on edge," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communication*, 2019, pp. 1432–1440.

[70] M. Hao, H. Li, G. Xu, S. Liu, and H. Yang, "Towards efficient and privacy-preserving federated deep learning," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[71] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[72] W. Lu, A. L. Varna, A. Swaminathan, and M. Wu, "Secure image retrieval through feature protection," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2009, pp. 1533–1536.

[73] C.-Y. Hsu, C.-S. Lu, and S.-C. Pei, "Image feature extraction in encrypted domain with privacy-preserving SIFT," *IEEE Transactions on Image Processing*, vol. 21, no. 11, pp. 4593–4607, 2012.

[74] S. Wang, M. Nassar, M. Atallah, and Q. Malluhi, "Secure and private outsourcing of shape-based feature extraction," in *International Conference on Information and Communications Security*. Springer, 2013, pp. 90–99.

[75] Z. Qin, J. Yan, K. Ren, C. W. Chen, and C. Wang, "Towards efficient privacy-preserving image feature extraction in cloud computing," in *Proceedings of the 22nd ACM International Conference on Multimedia*, 2014, pp. 497–506.

[76] S. Hu, Q. Wang, J. Wang, Z. Qin, and K. Ren, "Securing SIFT: Privacy-preserving outsourcing computation of feature extractions over encrypted image data," *IEEE Transactions on Image Processing*, vol. 25, no. 7, pp. 3411–3425, 2016.

[77] Z. Jiang and S. Mao, "Energy delay tradeoff in cloud offloading for multi-core mobile devices," *IEEE Access*, vol. 3, pp. 2306–2316, 2015.

[78] J. Kwak, Y. Kim, J. Lee, and S. Chong, "DREAM: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2510–2523, 2015.

[79] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, 2015.

[80] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *2016 IEEE International Symposium on Information Theory (ISIT)*, 2016, pp. 1451–1455.

[81] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2016.

[82] X. Wang, R. Jia, X. Tian, and X. Gan, "Dynamic task assignment in crowdsensing with location awareness and location diversity," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, 2018, pp. 2420–2428.

[83] R. Serfozo, *Basics of Applied Stochastic Processes*. Springer Science & Business Media, 2009.

[84] R. Srikant and L. Ying, *Communication Networks: An Optimization, Control and Stochastic Networks Perspective*. Cambridge University Press, 2013.

[85] A. Mekkittikul and N. McKeown, "A starvation-free algorithm for achieving 100% throughput in an input-queued switch," in *Proceedings of the IEEE International Conference on Communication Networks*, 1996.

[86] S. M. Ross, *Introduction to Probability Models*. Academic Press, 2014.

[87] A. C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science (SFCS 1986)*. IEEE, 1986, pp. 162–167.

[88] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[89] R. Cramer, I. Damgård, and U. Maurer, "General secure multi-party computation from any linear secret-sharing scheme," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000, pp. 316–334.

[90] U. Maurer, "Secure multi-party computation made simple," *Discrete Applied Mathematics*, vol. 154, no. 2, pp. 370–381, 2006.

[91] O. Catrina and S. de Hoogh, "Improved primitives for secure multiparty integer computation," in *International Conference on Security and Cryptography for Networks*. Springer, 2010, pp. 182–199.

[92] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Annual Cryptology Conference*. Springer, 2012, pp. 643–662.

[93] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure MPC for dishonest majority–Or: Breaking the SPDZ limits," in *European Symposium on Research in Computer Security*. Springer, 2013, pp. 1–18.

[94] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2011, pp. 169–188.

[95] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 35–52.

[96] R. V. Lopes and D. Menascé, "A taxonomy of job scheduling on distributed computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3412–3428, 2016.

[97] W. Guo, B. Lin, G. Chen, Y. Chen, and F. Liang, "Cost-driven scheduling for deadline-based workflow across multiple clouds," *IEEE Transactions on Network and Service Management,*, vol. 15, no. 4, pp. 1571–1585, 2018.

[98] R. O.Aburukba, M. AliKarrar, T. Landolsi, and K. El-Fakih, "Scheduling internet of things requests to minimize latency in hybrid fog–cloud computing," *Future Generation Computer Systems*, vol. 111, pp. 539–551, 2020.

[99] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, 2014.

[100] Z.-J. Wang, Z.-H. Zhan, W.-J. Yu, Y. Lin, J. Zhang, T.-L. Gu, and J. Zhang, "Dynamic group learning distributed particle swarm optimization for large-scale optimization and its application in cloud workflow scheduling," *IEEE Transactions on Cybernetics*, vol. 50, no. 6, pp. 2715–2729, 2020.

[101] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1400–1414, 2012.

[102] ——, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[103] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *NSDI*, 2011, pp. 99–112.

[104] S. Chen and M. Wu, "Anonymous multipath routing protocol based on secret sharing in mobile ad hoc networks," *Journal of Systems Engineering and Electronics*, vol. 22, no. 3, pp. 519–527, 2011.

[105] S. R. Pokhrel, Y. Qu, and L. Gao, "QoS-aware personalized privacy with multipath TCP for industrial IoT: Analysis and design," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4849–4861, 2020.

[106] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* MIT Press, 1992.

[107] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks.* IEEE, 1995, pp. 1942–1948.

[108] MathWorks. Particle swarm optimization. [Online]. Available: http://www.mathworks.com/help/gads/particleswarm.html.

[109] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *Journal of Parallel and Distributed Computing*, vol. 69, no. 2, pp. 117–124, 2009.

[110] J. Huang, P. Majumder, S. Kim, A. Muzahid, K. H. Yum, and E. J. Kim, "Communication algorithm-architecture co-design for distributed deep learning," in *Proceedings of the ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 181–194.

[111] B. Cetin, A. Lazar, J. Kim, A. Sim, and K. Wu, "Federated wireless network intrusion detection," in *Proceedings of the IEEE International Conference on Big Data (Big Data)*, 2019, pp. 6004–6006.

[112] T. V. Khoa, Y. M. Saputra, D. T. Hoang, N. L. Trung, D. Nguyen, N. V. Ha, and E. Dutkiewicz, "Collaborative learning model for cyberattack detection systems in IoT industry 4.0," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, 2020, pp. 1–6.

[113] R. Taheri, M. Shojafar, M. Alazab, and R. Tafazolli, "FED-IIoT: A robust federated malware detection architecture in industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 12, pp. 8442–8452, 2020.

[114] S. A. Rahman, H. Tout, C. Talhi, and A. Mourad, "Internet of things intrusion detection: Centralized, on-device, or federated learning?" *IEEE Network*, vol. 34, no. 6, pp. 310–317, 2020.

[115] J. Shu, L. Zhou, W. Zhang, X. Du, and M. Guizani, "Collaborative intrusion detection for VANETs: A deep learning-based distributed SDN approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4519–4530, 2020.

[116] H. Zhang and K. Zeng, "Federated graph neural network for fast anomaly detection in controller area networks," *IEEE Transactions on Information Forensics and Security (Under Review)*, 2022.

[117] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *The Ethics of Information Technologies.* Routledge, 2020, pp. 119–134.

[118] F. Martinelli, F. Mercaldo, V. Nardone, A. Orlando, and A. Santone, "Who's driving my car? A machine learning based approach to driver identification," in *ICISSP*, 2018, pp. 367–372.

[119] D. K. Nilsson, P. H. Phung, and U. E. Larson, "Vehicle ECU classification based on safety-security characteristics," in *IET Road Transport Information and Control-RTIC 2008 and ITS United Kingdom Members' Conference*, 2008, pp. 1–7.

[120] W. Wu, R. Li, G. Xie, J. An, Y. Bai, J. Zhou, and K. Li, "A survey of intrusion detection for in-vehicle networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 919–933, 2019.

[121] Upstream Auto. (2021) Upstream security's 2021 global automotive cybersecurity report. [Online]. Available: https://upstream.auto/2021report/.

[122] M. Jedh, L. ben Othmane, N. Ahmed, and B. Bhargava, "Detection of message injection attacks onto the CAN bus using similarity of successive messages-sequence graphs," *arXiv preprint arXiv:2104.03763*, 2021.

[123] R. Islam, R. U. D. Refat, S. M. Yerram, and H. Malik, "Graph-based intrusion detection system for controller area networks," *IEEE Transactions on Intelligent Transportation Systems (Early Access)*, pp. 1–10, 2020.

[124] H. M. Song, H. R. Kim, and H. K. Kim, "Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network," in *2016 International Conference on Information Networking (ICOIN)*. IEEE, 2016, pp. 63–68.

[125] B. Groza and P.-S. Murvay, "Efficient intrusion detection with bloom filtering in controller area networks," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 1037–1051, 2018.

[126] A. Taylor, S. Leblanc, and N. Japkowicz, "Anomaly detection in automobile control network data with long short-term memory networks," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2016, pp. 130–139.

[127] D. Kosmanos, A. Pappas, L. Maglaras, S. Moschoyiannis, F. J. Aparicio-Navarro, A. Argyriou, and H. Janicke, "A novel intrusion detection system against spoofing attacks in connected electric vehicles," *Array*, vol. 5, no. 100013, pp. 1–11, 2020.

[128] F. Amato, L. Coppolino, F. Mercaldo, F. Moscato, R. Nardone, and A. Santone, "CAN-bus attack detection with deep learning," *IEEE Transactions on Intelligent Transportation Systems (Early Access)*, pp. 1–10, 2021.

[129] C. Lin and A. Sangiovanni-Vincentelli, "Cyber-security for the controller area network (CAN) communication protocol," in *2012 International Conference on Cyber Security*. IEEE, 2012, pp. 1–7.

[130] K. Iehira, H. Inoue, and K. Ishida, "Spoofing attack using bus-off attacks against a specific ECU of the CAN bus," in *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2018, pp. 1–4.

[131] S. Longari, D. H. N. Valcarcel, M. Zago, M. Carminati, and S. Zanero, "CANnolo: An anomaly detection system based on LSTM autoencoders for controller area network," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1913–1924, 2020.

[132] H. Sun, M. Chen, J. Weng, Z. Liu, and G. Geng, "Anomaly detection for in-vehicle network using CNN-LSTM with attention mechanism," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 10, pp. 10 880–10 893, 2021.

[133] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, "Tree-based intelligent intrusion detection system in internet of vehicles," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[134] M. Marchetti and D. Stabili, "READ: Reverse engineering of automotive data frames," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 1083–1097, 2018.

[135] A. Bendale and T. E. Boult, "Towards open set deep networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1563–1572.

[136] X. Guo, A. Alipour-Fanid, L. Wu, H. Purohit, X. Chen, K. Zeng, and L. Zhao, "Multi-stage deep classifier cascades for open world recognition," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 179–188.

[137] S. Tariq, S. Lee, and S. S. Woo, "CANTransfer: transfer learning based intrusion detection on a controller area network using convolutional LSTM network," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 1048–1055.

[138] H. Lee, S. H. Jeong, and H. K. Kim, "OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2017, pp. 57–66.

[139] A. Tomlinson, J. Bryans, and S. A. Shaikh, "Towards viable intrusion detection methods for the automotive controller area network," in *2nd ACM Computer Science in Cars Symposium*, 2018, pp. 1–9.

[140] W. Choi, H. J. Jo, S. Woo, J. Y. Chun, J. Park, and D. H. Lee, "Identifying ECUs using inimitable characteristics of signals in controller area networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 6, pp. 4757–4770, 2018.

[141] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive CAN bus," in *2015 World Congress on Industrial Control Systems Security (WCICSS)*, 2015, pp. 45–49.

[142] M. Marchetti and D. Stabili, "Anomaly detection of CAN bus messages through analysis of ID sequences," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1577–1583.

[143] E. Seo, H. M. Song, and H. K. Kim, "GIDS: GAN based intrusion detection system for in-vehicle network," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, 2018, pp. 1–6.

[144] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011, pp. 1110–1115.

[145] M. Marchetti, D. Stabili, A. Guido, and M. Colajanni, "Evaluation of anomaly detection for in-vehicle networks through information theoretic algorithms," in *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, 2016, pp. 1–6.

[146] M.-J. Kang and J.-W. Kang, "A novel intrusion detection method using deep neural network for in-vehicle network security," in *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, 2016, pp. 1–5.

[147] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, "Sequential deep learning for human action recognition," in *International Workshop on Human Behavior Understanding*, 2011, pp. 29–39.

[148] L. Ding, W. Fang, H. Luo, P. E. Love, B. Zhong, and X. Ouyang, "A deep hybrid learning model to detect unsafe behavior: Integrating convolution neural networks and long short-term memory," *Automation in Construction*, vol. 86, pp. 118–124, 2018.

[149] H. Rahmani, A. Mian, and M. Shah, "Learning a deep model for human action recognition from novel viewpoints," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, pp. 667–681, 2018.

[150] L. B. Othmane, L. Dhulipala, N. Multari, and M. Govindarasu, "On the performance of detecting injection of fabricated messages into the CAN bus," *IEEE Transactions on Dependable and Secure Computing (Early Access)*, pp. 1–1, 2020.

[151] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2020.

[152] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018, pp. 1–8.

[153] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[154] D. M. Tax and R. P. Duin, "Support vector data description," *Machine Learning*, vol. 54, no. 1, pp. 45–66, 2004.

[155] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4393–4402.

[156] W. J. Scheirer, A. Rocha, R. J. Micheals, and T. E. Boult, "Meta-recognition: The theory and practice of recognition score analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 8, pp. 1689–1695, 2011.

[157] P. Zhang, J. Wang, A. Farhadi, M. Hebert, and D. Parikh, "Predicting failures of vision systems," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3566–3573.

[158] R. Tibshirani, T. Hastie, B. Narasimhan, and G. Chu, "Diagnosis of multiple cancer types by shrunken centroids of gene expression," *Proceedings of the National Academy of Sciences*, vol. 99, no. 10, pp. 6567–6572, 2002.

[159] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[160] A. Bendale and T. E. Boult, "Towards open world recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1893–1902.

[161] I. Rouf, R. D. Miller, H. A. Mustafa, T. Taylor, S. Oh, W. Xu, M. Gruteser, W. Trappe, and I. Seskar, "Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study," in *USENIX Security Symposium*, 2010, pp. 1–16.

[162] H. M. Song and H. K. Kim. (2020) CAN signal extraction and translation dataset. [Online]. Available: https://ocslab.hksecurity.net/Datasets/can-signal-extraction-and-translation-dataset.

[163] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer, "CANet: An unsupervised intrusion detection system for high dimensional CAN bus data," *IEEE Access*, vol. 8, pp. 58 194–58 205, 2020.

[164] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[165] G. Damaskinos, E.-M. El-Mhamdi, R. Guerraoui, A. Guirguis, and S. Rouault, "AGGREGATHOR: Byzantine machine learning via robust gradient aggregation," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 81–106, 2019.

[166] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5650–5659.

Curriculum Vitae

Hengrun Zhang received the B.S. degree in automation from East China University of Science and Technology, Shanghai, China, in 2012. He received the M.S. degree in control science and engineering from Shanghai Jiao Tong University, Shanghai, China, in 2015, and in computer science from George Mason University, Fairfax, VA, USA, in 2018. His research interests include task scheduling in networking, privacy and security in networking, federated learning applications in the Internet of Things (IoT) and performance optimization in federated learning.