Exploring Efficiency in Split Federated Learning under System Heterogeneity: A Study of System Synchronization and Resource Optimization

Haoran Gao

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Electrical and Computer Engineering) at

Concordia University

Montréal, Québec, Canada

August 2025

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Haoran Gao**

Entitled: **Exploring Efficiency in Split Federated Learning under System Heterogeneity: A Study of System Synchronization and Resource Optimization**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Wei-Ping Zhu

_____ Examiner
Dr. Dongyu Qiu

_____ Supervisor
Dr. Jun Cai

_____ Co-supervisor
Dr. Samuel D. Okegbile

Approved by _____
Wahab Hamou-Lhadj, Chair
Department of Electrical and Computer Engineering

_____ 2025

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Exploring Efficiency in Split Federated Learning under System Heterogeneity: A Study of System Synchronization and Resource Optimization

Haoran Gao

The rise of intelligent edge services has intensified the demand for scalable and privacy-preserving collaborative learning frameworks. Split Federated Learning (SFL), a hybrid paradigm that combines the layer-wise decoupling of Split Learning with the distributed aggregation of Federated Learning, offers an efficient training approach across heterogeneous devices. However, its scalability remains constrained by device heterogeneity and synchronization delays. This thesis proposes a novel Collaborative Split Federated Learning (CSFL) framework that facilitates real-time cooperative computation through direct device-to-device communication. At its core lies the Collaborative Relay Optimization Mechanism (CROM), in which efficient devices act as computational relays by executing intermediate model segments on behalf of bottleneck devices, thereby achieving balanced workload distribution. To minimize end-to-end training latency, we formulate a joint optimization problem over the model cut-layer and device pairing configuration. This problem is decomposed into two interdependent sub-tasks: cut-layer selection, addressed via a convergent alternating optimization strategy, and device pairing, modeled as a deferred-acceptance-based weighted matching game that aligns local utilities with global objectives. Extensive experiments on VGG-16 using the Tiny-ImageNet dataset under non-IID data partitions demonstrate that CSFL significantly reduces training latency and energy consumption while maintaining accuracy comparable to conventional SFL. Overall, this work advances collaborative edge learning by introducing a unified framework for system-level co-optimization, emphasizing the importance of algorithm-architecture codesign for future intelligent systems.

Acknowledgments

It is with profound gratitude that I acknowledge the individuals whose guidance and support have been pivotal to the completion of this thesis.

I am deeply indebted to Professor Jun Cai, who graciously welcomed me into his research group. His trust in my potential and the intellectual freedom he afforded me have laid the foundation for my academic development. His mentorship provided not only scholarly direction, but also an environment in which curiosity and rigor could coexist.

I would also like to express my sincere appreciation to Dr. Samuel Okegbile, whose mentorship during the formative stages of this research was both generous and transformative. His clarity of thought, depth of knowledge, and unwavering support have shaped the trajectory of this work.

My deepest personal thanks go to my parents, whose quiet strength and daily prayers were a constant source of solace and encouragement. In moments of uncertainty, their devotion reminded me of the resilience that lies in faith and love. Though the path was not without obstacles, her presence illuminated it with purpose.

I am also grateful to my fellow lab members, whose intellectual camaraderie and everyday kindness enriched this journey. Their insights, encouragement, and shared perseverance have made this endeavor not only more rigorous, but also more human.

I sincerely thank everyone who supported me, whether through conversation, encouragement, or thoughtful advice.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter presents the motivation for the research, explains the problem context, summarizes the main contributions, and outlines the structure of the thesis.

1.1 Background

In recent years, the growing demand for low-latency, privacy-preserving intelligent services has accelerated the shift from centralized cloud learning to collaborative learning at the network edge [1, 2]. Applications such as real-time medical diagnosis, smart surveillance, and autonomous mobility increasingly rely on data generated by distributed edge devices and require fast, efficient training mechanisms to adapt to rapidly changing environments [3, 4].

In response to this trend, Federated Learning (FL) was introduced as a distributed training paradigm that preserves data privacy by keeping raw data local [5]. In FL, each client independently trains a local copy of the full model using its private dataset and periodically uploads model updates (e.g., weights or gradients) to a central server. The server then performs aggregation, most commonly using Federated Averaging (FedAvg), to obtain a global model that is redistributed to all clients for the next training round. This iterative process enables collaborative model learning without centralizing data [6, 7]. Additionally, FL typically follows a synchronous protocol, where the server waits for all selected clients to complete their local updates before proceeding. While this facilitates orderly global optimization, it can become a performance bottleneck in practice, especially

when client capabilities vary. More critically, FL assumes that all devices are capable of full-model training, which imposes substantial computational and memory burdens on resource-constrained edge clients. As such, application of FL becomes limited for real-world system involving IoT sensors, wearables, or embedded platforms.

To mitigate local resource burden, Split Learning (SL) [8] was proposed as an alternative paradigm that reduces local computation by partitioning the deep model between the client and the server at a designated cut-layer. During training, the client computes only the shallow portion of the model up to the cut-layer and transmits the resulting intermediate activations to the server. The server completes the remaining forward and backward passes and returns the gradients corresponding to the cut-layer, enabling the client to update its own portion of the model. This setup dramatically reduces the computational and memory footprint on edge devices [9]. Unlike FL, there is no built-in aggregation mechanism across clients, which prevents the system from converging to a globally consistent model. Furthermore, SL introduces high per-sample communication overhead due to frequent interaction between the client and server during each training step [10]. This overhead makes SL highly sensitive to network latency and reliability.

Consequently, FL and SL offer complementary advantages: FL ensures global consistency and privacy through parameter aggregation, but imposes high local demands; while SL enables computation offloading to the server, but lacks inter-client coordination and incurs significant communication overhead. These observations have motivated the development of hybrid paradigms that seek to jointly exploit the benefits of both approaches, among which Split Federated Learning (SFL) [11] has been proposed as a hybrid framework. In SFL, each client splits its model at a predefined cut-layer, computes the shallow layers locally, and transmits intermediate activations to the server, similar to the SL paradigm. After completing the backward propagation, the server performs an additional aggregation step to update its model segment, which also helps promote convergence during collaborative training. Meanwhile, federated aggregation [6] is applied to the client-side model segments, ensuring consistency across participating clients. Specifically, the client-side aggregation often adopts a FedAvg-like mechanism to update the shallow model layers distributed across participants, while the server-side aggregation fuses representations or parameters corresponding to the deeper layers. This dual aggregation strategy enables a hierarchical coordination

of training, where computation is decoupled between edge and cloud, and model synchronization occurs on both ends. As a result, SFL balances computation offloading, privacy preservation, and model consistency, making it a promising solution for collaborative learning in resource-constrained environments [12].

1.2 Motivation

Despite its conceptual appeal, SFL still faces critical limitations in realistic deployments. The framework typically assumes that all clients operate synchronously and possess comparable computational capabilities, which is rarely true in heterogeneous edge environments [13]. When device heterogeneity is significant, fast devices are forced to wait for slower ones during synchronization, resulting in straggler effects, idle time, and degraded system efficiency.

Edge devices are inherently constrained by limited computational power, battery life, and memory capacity [14]. In addition, they often suffer from unstable network connectivity, and exhibit substantial heterogeneity in terms of hardware capabilities and local data availability [15]. These limitations pose serious challenges to the deployment of conventional learning frameworks, which typically assume stable connections, homogeneous participants, and sufficient local resources.

In practical edge scenarios, devices exhibit substantial heterogeneity in terms of computing power, communication bandwidth, and data availability. The standard SFL framework, which relies on synchronized processing across all clients, becomes inefficient in such settings [11]. Specifically, the server must wait for all devices to complete their computations before proceeding, leading to the straggler effect: high-performance devices remain idle while waiting for bottleneck devices to finish their tasks. This results in significant synchronization latency and poor overall system utilization. Moreover, current edge learning frameworks are often static and non-cooperative. They assign a fixed cut-layer to all devices regardless of device heterogeneity and overlook peer collaboration as a strategy for balancing workload and reducing delay.

These limitations underscore the need for more adaptive, flexible, and cooperative learning strategies that explicitly account for device heterogeneity and system dynamics. Enabling collaborative training among edge devices—where stronger devices assist weaker ones—offers a promising

path forward to improve scalability, latency, and overall learning efficiency in next-generation distributed systems. Such collaboration can be supported by emerging device-to-device (D2D) communication technologies [16, 17], which provide low-latency links between nearby edge nodes.

1.3 Objectives

This thesis aims to develop a collaborative and adaptive SFL framework that mitigates synchronization bottlenecks and enhances training efficiency under device heterogeneity. To achieve this goal, the following objectives are pursued:

- Architectural Design: Design a flexible cooperative learning architecture that enables high-performance edge devices to assist less capable participants through model-level computation sharing. This architecture aims to reduce synchronization delays and improve system-wide resource utilization, while strictly preserving data locality.

- Optimization Formulation: Formulate a principled and extensible joint optimization problem that captures the critical trade-offs in system configuration. In particular, the formulation seeks to determine both the optimal model partitioning strategy (i.e., cut-layer selection) and the cooperation structure among clients, with the overall objective of minimizing end-to-end training latency under heterogeneous resource and network conditions.

- Solution Methodology: Develop efficient and generalizable solution strategies to solve the above optimization problem under realistic constraints. The proposed methods aim to ensure convergence and scalability, while maintaining compatibility with practical deployment scenarios involving limited computation and communication resources.

- Empirical Validation: Evaluate the proposed framework through extensive experiments conducted under heterogeneous system conditions and non-IID data. Performance is benchmarked against representative baselines in terms of training latency, energy efficiency, and model accuracy, demonstrating the effectiveness and adaptability of the approach in real-world edge environments.

Collectively, these objectives lay the foundation for a scalable and adaptive learning framework that addresses the practical challenges of deploying SFL in heterogeneous edge environments.

1.4 Contributions

This thesis is motivated by the belief that cooperative computation among heterogeneous edge devices offers a promising approach to mitigating resource imbalance, reducing synchronization latency, and improving the scalability of distributed learning systems. To validate this hypothesis, we introduce a novel paradigm, Collaborative Split Federated Learning (CSFL), which integrates device-to-device cooperation into the conventional SFL architecture. At the heart of this framework is the Collaborative Relay Optimization Mechanism (CROM), in which efficient devices assist bottleneck users by completing additional segments of the model on their behalf. This relay-style collaboration is designed to exploit the model partitioning flexibility of SL while preserving the federated aggregation of SFL.

To support this framework, we further develop a joint optimization scheme that minimizes end-to-end training latency through coordinated decisions over two critical components: the cut-layer configuration and the device pairing strategy. Recognizing that these two decisions operate on different temporal and algorithmic scales, we decompose the problem into two interdependent subproblems. We then employ an alternating iterative approach for cut-layer selection and model the device pairing as a weighted matching game, aligning local utility with global system performance.

The main contributions of this thesis are summarized as follows.

- We propose a novel CSFL framework as a principled extension of traditional SFL to address the limitations of synchronization and resource heterogeneity in edge environments. The collaborative nature of CSFL is realized through the integration of the Collaborative Resource Optimization Mechanism (CROM), a dedicated mechanism that coordinates computation sharing, device pairing, and model partitioning across heterogeneous clients. We formally define the system architecture, analyze key design challenges, and demonstrate how CROM enables efficient and privacy-preserving collaboration under practical deployment constraints.

- Building upon the CSFL framework, we investigate the core problem of jointly optimizing

the cut-layer selection and device pairing strategies. We formulate this as a stochastic optimization problem that minimizes training latency while accounting for device capabilities, communication bandwidth, and model consistency constraints.

- To address this problem efficiently, we propose a scalable two-stage approach that decouples the joint optimization into alternating subproblems at different time scales. The proposed methods balance adaptability and convergence, enabling dynamic yet stable optimization of collaboration structures.

- We validate the proposed framework through extensive simulations under non-IID data settings and heterogeneous system conditions. Results confirm the superiority of the CSFL framework in terms of training latency, scalability, and accuracy, while also highlighting its potential for broader application in practical edge learning systems.

To the best of our knowledge, this work is the first to systematically investigate the joint optimization of cut-layer selection and device pairing within the CSFL framework. This thesis is built upon the following publications:

- H. Gao, J. Cai, S. D. Okegbile, et al., "Toward Efficient CSFL: Joint Optimization of Cooperative Pairing and Cut-Layer Allocation in Heterogeneous Environments," submitted to *IEEE Transactions on Mobile Computing*.

- H. Gao, S. D. Okegbile, J. Cai, "Advanced Relay-Based Collaborative Framework for Optimizing Synchronization in Split Federated Learning over Wireless Networks," submitted to *IEEE Wireless Communications*.

In parallel with the work presented in this thesis, I has actively contributed to a series of collaborative studies that extend the principles of machine learning and semantic communication to emerging intelligent network environments. These works contributed to the following publications:

- S. D. Okegbile, H. Gao, O. Talabi, et al., "Optimizing Federated Semantic Learning in Distributed AIGC-Enabled Human Digital Twins: A Multi-Criteria and Multi-Shard User Selection Framework," in *IEEE Transactions on Mobile Computing*, vol. 24, no. 7, pp. 5916-5933, July 2025.

- S. D. Okegbile, H. Gao, O. Talabi, et al., "FLeS: A Federated Learning-Enhanced Semantic Communication Framework for Mobile AIGC-Driven Human Digital Twins," in *IEEE Network*, 2025.

- S. D. Okegbile, H. Gao, J. Cai, "A Novel Secure Split Federated Semantic Learning Framework and its Optimization for Digital Twin Network Evolution," in *IEEE Transactions on Mobile Computing*, 2025.

1.5 Thesis Structure

The remainder of this thesis is organized as follows. Chapter 2 reviews the related work on federated learning, split learning, and split federated learning, with a particular focus on their adaptations to heterogeneous edge environments, as well as representative approaches to privacy preservation. Chapter 3 introduces the proposed CSFL framework, elaborates on its system architecture, and provides a comprehensive analysis of its overall training efficiency. Building upon this, Chapter 4 formulates a joint optimization problem that simultaneously addresses cut-layer selection and device pairing, and presents a two-stage solution framework that effectively decouples and solves these interdependent tasks. Chapter 5 details the experimental evaluation setup, including the model architecture, parameter settings, and performance benchmarks. Chapter 6 concludes the thesis by summarizing the key findings and outlining potential directions for future research. In addition, Appendix A provides a supplementary convergence analysis for one of the proposed methods.

Chapter 2

Literature Review

2.1 Heterogeneity-Aware Strategies in FL and SL

Device heterogeneity remains one of the core obstacles to efficient and scalable distributed training at the edge. In practical scenarios, edge clients often differ in computational capabilities, memory, and connectivity conditions [18]. This leads to the well-known straggler problem, where slower devices delay the progress of the entire training process. To address this, various strategies have been introduced within both FL and SL frameworks, particularly targeting synchronization latency caused by heterogeneous device behavior.

In FL, several mechanisms aim to reduce synchronization latency and improve robustness to heterogeneous devices. One prominent method is device clustering [19–21], where clients are grouped based on hardware profiles or training speeds. Within each group, synchronized updates are performed independently, thus reducing the impact of stragglers without fully discarding synchronization. However, the effectiveness of this strategy depends heavily on the initial clustering criteria. If the criteria are poorly chosen or become outdated, the grouping may become imbalanced or even counterproductive, undermining the intended synchronization efficiency.

Another widely adopted approach is semi-synchronous scheduling [22–24], which allows the server to proceed with model aggregation at specific synchronization points without requiring responses from all clients. Clients that miss a given round can continue training locally and contribute in later rounds, thereby improving the continuity of the training process. Despite its simplicity, this

method may lead to aggregation bias or slower convergence, especially when updates from slower clients are persistently delayed or excluded.

Additionally, relay-assisted federated learning [25] has been proposed to mitigate bandwidth bottlenecks and reduce communication latency. In this setup, edge relay nodes or auxiliary servers perform local aggregation or intermediate forwarding, enabling distributed aggregation and communication offloading. While effective in reducing direct server-client overhead, this architecture incurs additional deployment and maintenance costs, and introduces extra system complexity that may hinder practical scalability.

SL, by contrast, reconfigures the structure of the learning framework. However, SL does not fully eliminate the synchronization challenge. Due to the tight layer-wise dependency between clients and the server, most SL frameworks rely on synchronous coordination during both the forward and backward pass [26]. Attempts to incorporate asynchronous mechanisms [27, 28] into SL build upon the observation that each client's data can be handled independently and that SL does not involve any global model aggregation at the server. The server can adopt a request-response pattern and process activations as they arrive, without requiring synchronized coordination across devices. This design enables asynchronous execution and better accommodates heterogeneity in computation and connectivity, but it still presents certain limitations.

Device clustering techniques [29] are also applicable in SL. However, the grouping suffers from the same limitations discussed above—its effectiveness heavily depends on the choice of clustering criteria, which, if poorly defined, may lead to unbalanced groupings and limited performance gains.

While these strategies provide partial relief from synchronization delays, they are often designed as auxiliary scheduling or clustering mechanisms that operate independently from the model training process itself. As such, they lack tight integration with the underlying learning framework and do not leverage architectural properties such as model partitioning or inter-device computation delegation, which could fundamentally mitigate straggler effects in heterogeneous environments.

2.2 Split Federated Learning and Collaborative Extensions

SFL integrates the model-partitioning structure of SL with the aggregation paradigm of FL. In a typical SFL setup, clients compute the initial layers of a deep model and transmit intermediate activations to the server, which completes the forward and backward computations. After multiple rounds of such interactions, an aggregation step ensures model consistency. This hybrid structure aims to balance privacy preservation, computational offloading, and collaborative training.

To better adapt to heterogeneous edge environments, recent SFL variants have explored the decoupling of conventional synchronized updates. One such adaptation employs a sequential processing strategy, allowing clients to transmit activations asynchronously [11, 30]. The server processes these inputs in a streaming manner without requiring full synchronization. Due to the asynchronous scheduling, server-side aggregation is removed entirely, and only periodic local aggregation among clients is retained. While this design alleviates the central bottleneck, the sequential processing of client updates introduces the risk of temporal bias, as earlier updates may disproportionately influence the model's trajectory [31, 32]. This not only compromises fairness but can also lead to convergence instability, especially in non-IID settings where data distributions differ significantly across clients. Without corrective mechanisms such as update reordering or temporal weighting, the model may overfit to early clients or fail to generalize across the broader data distribution.

This approach, while alleviating synchronization burdens to some extent, fundamentally treats device heterogeneity as a constraint to be mitigated rather than a structural property to be exploited. They lack mechanisms to actively coordinate inter-device computation, adapt model partitioning to device-specific capabilities, or leverage localized collaboration opportunities [11, 30, 31]. One promising yet underexplored system-level mechanism in this context is task offloading, a well-established strategy in edge computing in which overloaded devices dynamically delegate tasks to nearby idle nodes [33, 34].

Task offloading typically involves deciding which tasks to offload, selecting suitable candidate nodes, and optimizing offloading decisions to minimize execution latency, energy consumption, or maximize throughput. However, most existing offloading frameworks primarily focus on relatively

coarse-grained workloads, such as entire applications or computation tasks, with limited consideration for structured dependencies within tasks [35]. As such, the integration of task offloading with structured computational tasks, such as deep neural network (DNN) training, remains rarely explored. Building upon this gap, CSFL leverages the inherent structural partitioning flexibility of split neural networks to introduce fine-grained, model-segment-level task offloading. By explicitly embedding collaborative inference and workload redistribution into the training process itself, CSFL realizes a deeper integration of offloading principles into collaborative learning, thus fundamentally addressing the challenges posed by heterogeneous device environments.

This collaboration is made possible by the model-partitioning flexibility inherent in SL [27], which allows deep networks to be sliced and dynamically reassigned across devices. Combined with the decentralized scheduling paradigm of task offloading, this design facilitates real-time coordination and efficient utilization of edge resources.

By integrating CROM into the SFL framework, CSFL transforms the conventional client-server topology into a cooperative, decentralized structure that improves responsiveness, reduces synchronization latency, and enhances system-wide scalability.

2.3 Game-Theoretic Foundations of CSFL

The CSFL framework inherently involves both cooperative and competitive interactions among participating devices. Game theory provides a rigorous mathematical foundation for modeling such strategic interactions among rational agents in environments characterized by limited resources or conflicting interests. In general, game-theoretic models can be categorized into non-cooperative and cooperative games. Non-cooperative games focus on independent decision-making, where each agent seeks to maximize its own utility without explicit communication or binding agreements [36]. In contrast, cooperative games allow agents to form coalitions or partnerships, coordinating their actions to achieve mutually beneficial outcomes [37].

In the context of CSFL, the cooperative computation mechanism can be naturally formulated as a device pairing problem, wherein participating devices must form stable, mutually beneficial partnerships. Matching games, a specialized class within cooperative game theory, are designed to

model situations where agents' utilities depend on pairing or matching relationships [38]. Matching theory provides a systematic approach to determining pairings between agents from distinct sets based on individual preferences. Accordingly, the device pairing process in CSFL can be effectively formulated and analyzed within the framework of matching games.

2.4 Privacy Preservation in Distributed Learning

The inter-device cooperation enabled by CROM introduces new privacy challenges. Unlike traditional FL or SL frameworks where data never leaves the originating device or is transmitted only to a central server, CROM involves direct exchange of intermediate activations and delegated computations between devices. Such interactions increase the attack surface and may lead to unintended information exposure if adversarial participants or compromised devices are involved [39].

To address these concerns, secure execution and privacy-preserving computation must be embedded within the CSFL framework. One promising line of defense is the use of Trusted Execution Environments (TEEs) [40], such as Intel SGX and ARM TrustZone [41,42], which provide isolated and tamper-resistant regions of memory for processing sensitive data. In the context of CROM, TEEs can ensure that delegated model segments are computed securely on assisting devices, preventing access or manipulation by the host operating system or other processes.

In parallel, Privacy-Preserving Deep Learning techniques [43] offer cryptographic and statistical guarantees for safeguarding training data and model parameters. Secure Multi-Party Computation [44] enables collaborative computation without revealing individual inputs, making it suitable for scenarios where multiple devices contribute to shared model updates. Homomorphic Encryption [45] supports direct operations on encrypted activations or gradients, though often with trade-offs in computational overhead. Differential Privacy [46,47], widely adopted in federated learning settings, injects noise into model updates or data statistics to limit the influence of any single user's data, thereby mitigating inference attacks.

Taken together, these techniques provide a foundation for enabling secure, privacy-aware collaboration among devices. In this work, we assume that such mechanisms—whether through hardware

isolation, cryptographic protocols, or statistical obfuscation—are available to support the deployment of CROM. This ensures that collaborative model computation can proceed without compromising the privacy and integrity of user data.

Chapter 3

The Collaborative Split Federated Learning Framework

This chapter introduces the general architecture of the proposed CSFL framework. We first present the core mechanism, the Collaborative Relay Optimization Mechanism (CROM), which enables device cooperation through model relaying. We then describe the system architecture and outline the operational procedures that govern the collaborative training workflow. Finally, we highlight several essential design elements, including layer splitting, device matching, inter-device communication, and aggregation, which collectively define the structure and logic of the proposed framework.

3.1 Collaborative Relay Optimization Mechanism (CROM)

As a foundational component of the proposed CSFL framework, CROM is introduced to facilitate low-latency, resource-efficient, and fairness-aware training in heterogeneous edge learning environments. CROM is designed to systematically orchestrate inter-device collaboration by exploiting the underutilized computational capacity of efficient devices to assist resource-constrained counterparts, called bottleneck devices, during critical phases of model training.

Through dynamic workload redistribution, this relay-based mechanism effectively mitigates synchronization delays, enhances system-wide throughput, and accelerates model convergence,

all while maintaining global model consistency. In addition, by leveraging the intrinsic privacy-preserving properties of split learning, where only intermediate activations, rather than raw data, are exchanged, CROM safeguards device privacy throughout the collaborative process. Collectively, these capabilities establish a principled and scalable foundation for secure, efficient and cooperative intelligence at the network edge.

3.2 System Architecture

The conceptual structure of the CSFL framework is illustrated in Fig.3.1, which resembles a three-layer U-shaped SL architecture [48, 49] from the perspective of bottleneck devices: the front-end layers are computed locally by the bottleneck devices, the intermediate layers are relayed and computed by matched efficient devices, and final layers are executed by the server.

The forward propagation procedure is divided into two stages, dynamically adapting to the heterogeneous configurations of the participating devices. During the first stage, all devices perform local forward computations using their own datasets. Due to their enhanced computational resources, efficient devices typically finish this stage earlier than bottleneck devices and transition to the second stage.

In the second stage, each bottleneck device terminates its computation after reaching a predefined cut-off layer and transmits both the intermediate activations and the associated layer index to its matched efficient device. Based on this information, the efficient device identifies the partition point and proceeds to compute the remaining layers of the bottleneck device's forward path. For clarity, we refer to the data transferred between devices at this point as intermediate smashed data, to distinguish it from the final smashed data ultimately sent to the server. Upon receiving these data, the efficient device completes the forward computation on behalf of its matched partner.

This collaboration architecture enables partial offloading of the bottleneck devices' forward pass to computationally capable partners. Leveraging both the device heterogeneity and model partitioning inherent to SL, CSFL ensures that the final smashed data from all devices can be delivered to the server in a synchronized and latency-aware manner. The key operational components of CSFL are described in the following.

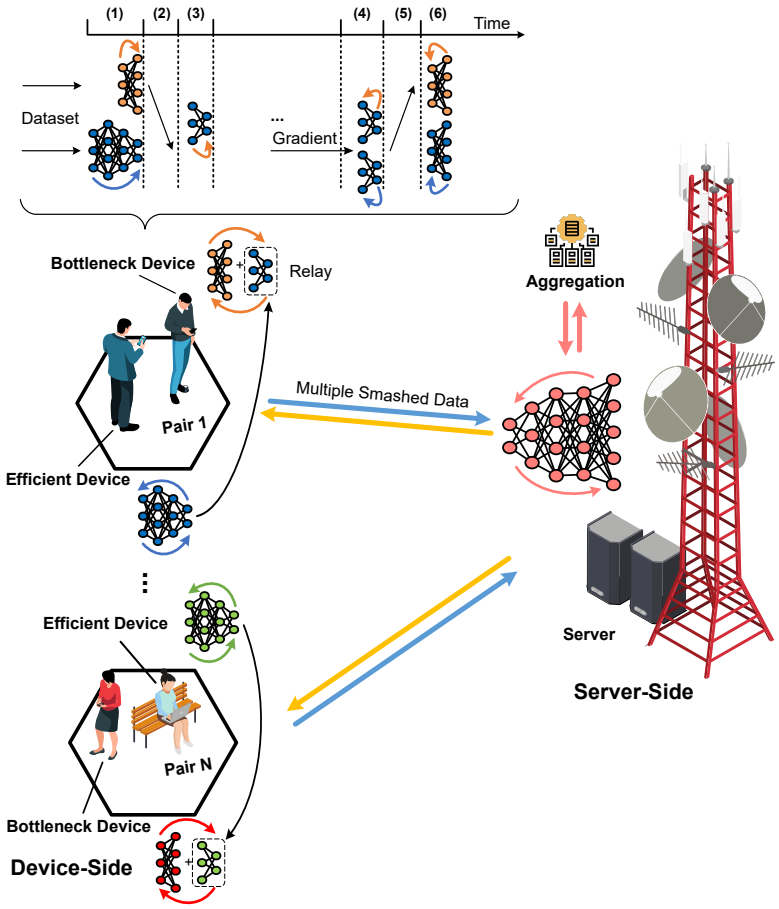

Figure 3.1: Collaborative Training with Device Pairing and Server Interaction.

- **Device Classification**: In the proposed framework, participating devices are categorized into two types according to their computational capabilities and communication conditions. Efficient devices possess relatively strong processing power and favorable channel conditions, enabling them to perform deeper local computations and act as relays to assist bottleneck devices in collaborative training. Bottleneck devices, by contrast, are constrained by limited computational resources or weaker communication links, and thus often become the primary source of synchronization delays. This classification forms the basis of the CSFL mechanism, whereby efficient devices not only contribute to their own training but also alleviate the burden of bottleneck devices through relay collaboration.

- **Layer Splitting Selection**: The selection of cut-layer defines the computational boundary between the device side and the server side, thereby determining the extent of local processing before the model is offloaded to the server. In conventional SFL, this boundary primarily influences communication cost and workload allocation. In contrast, CSFL extends its impact to inter-device collaboration: the choice of split layer directly determines whether an efficient device can effectively serve as a relay. A deeper cut layer increases the computational burden on the relay, whereas a shallower split increases the communication overhead of bottleneck devices. Hence, optimal layer splitting must jointly balance computational efficiency, communication cost, and collaborative feasibility to maximize system-wide performance.

- **Efficient-Bottleneck Matching**: In the CSFL framework, the most crucial aspect is matching efficient devices with bottleneck devices, which is key to resolving synchronization challenges and optimizing overall system performance. First, correct matching enables efficient devices and bottleneck devices to remain synchronized in forward propagation and subsequent computations. After matching, efficient devices will independently complete their own forward propagation tasks and then provide support as relays during the processing phase of bottleneck devices. If the matching is not optimal, efficient devices may not fully utilize their computing power, while bottleneck devices may keep hindering the overall system efficiency, leading to inefficient use of time and resources. Accurate matching can significantly enhance system performance, ensure optimal resource utilization in each part, and effectively address the synchronization challenges posed by device heterogeneity.

- **Inter-device Wireless Communication**: CROM operates within a wireless environment, where information is transmitted between devices—specifically, between efficient and bottleneck devices after they are matched. Inter-device wireless communication serves as the crucial bridge between these matched devices, facilitating real-time coordination and minimizing delays in the computation process. efficient devices rely on these communication links to receive data from bottleneck devices and subsequently provide computational support. Without efficient wireless communication, the system would face challenges in maintaining the continuous flow of information necessary for optimal collaboration. Thus, wireless

communication is a critical component, ensuring that the distributed nature of the framework functions smoothly and effectively.

- **Determination of Partition Points**: The selection of partition points is based on the device matching results. Unlike the previously discussed split layer selection, partitioning involves selecting the relay model between devices, aiming to mitigate potential system delays. In this sense, the partition point specifies the inter-device boundary. Specifically, efficient devices relay models typically exclude the full client model, as bottleneck devices handle the initial computations. Therefore, the partition point indicates the completed and unfinished layers, guiding efficient devices to preserve necessary layers and skip redundant computations. If the partition point is not optimally selected, the efficient device may retain unnecessary computational layers during the relaying process, resulting in excess computational load and increased data transmission. Thus, selecting the correct partition point is crucial to avoid resource waste and training delays.

- **Aggregation**:The aggregation process in the CSFL framework follows the SFL framework, aiming to ensure model consistency and synchronization by integrating client model weights. However, in CSFL, this process is more complex, as efficient devices share model parts to support bottleneck devices. Minor discrepancies in model parameters can significantly affect system performance. Therefore, successful aggregation is essential to align bottleneck clients' output with expectations and prevent deviations.

Chapter 4

Joint Optimization of Cut-Layer and Device Pairing

Building upon the structural foundations established in Chapter 3, this chapter formalizes the CSFL framework through a mathematical characterization. We identify two critical decision dimensions, namely device coordination and model partitioning, and analyze their impact on training latency and energy consumption, leading to a joint optimization problem that captures key performance trade-offs. To address the inherent coupling and timescale disparity between these dimensions, we propose a two-stage solution: an offline alternating strategy [50, 51] optimizes the cut-layer configuration, while an online matching game [52, 53] dynamically assigns collaboration pairs during training. This hierarchical design enables scalable coordination between long-term model adaptation and short-term cooperative scheduling across dynamic edge environments.

4.1 Mathematical Modeling of CSFL

The CSFL framework operates within a heterogeneous edge intelligence environment, comprising a set of distributed edge devices $\mathcal{K} = \{1, 2, \ldots, K\}$, connected to a central edge server s. Each device $k \in \mathcal{K}$ is assigned a local dataset $\mathcal{D}_k = \{x_i^{(k)}, y_i^{(k)}\}_{i=1}^{D_k}$, where $x_i^{(k)}$ and $y_i^{(k)}$ denote the input and label of the i-th sample, and D_k is the dataset size. Devices exhibit heterogeneous characteristics in terms of computational capacity, channel condition, and data volume. Following the

SL paradigm, the full model θ is split into two parts: the client-side model θ_c and the server-side model θ_s, where the split layer index is denoted by J_c. Each device holds an identical copy of the client-side model θ_c, and the server is responsible for updating θ_s.

To enable collaboration, the system classifies devices into two disjoint sets: efficient devices (EDs) and bottleneck devices (BDs), denoted as $\mathcal{U}$ and $\mathcal{V}$ respectively, with $\mathcal{U} \cup \mathcal{V} = \mathcal{K}$ and $\mathcal{U} \cap \mathcal{V} = \emptyset$. A pairing strategy is then employed to assign each BD a supporting ED, thereby forming collaboration pairs (u, v) with $u \in \mathcal{U}$, $v \in \mathcal{V}$.

The initialization phase begins by determining the optimal cut layer J_c, classifying devices into ED and BD sets, and generating the initial pairings. Model parameters $\theta_c^{(k)}$ and θ_s are initialized accordingly. Based on local resource profiles, each device is assigned its data partition and prepares for local training. These procedures collectively define the training state at $t = 0$, after which collaborative training rounds are launched.

4.1.1 Local Forward Computation

The forward propagation is initially performed independently on each device, followed by a cooperative stage in which efficient devices serve as relays to assist bottleneck devices. The detailed workflow for both device types is presented below.

(1) **ED-side computation**: Each ED first performs independent forward propagation using its local dataset $\mathcal{D}_u(t)$ on the client-side model $\theta_c^{(u)}(t)$. The computations proceed layer by layer, where for each layer j, the output is computed as

$$z_u^{(j)}(t) = W_c^{(j)}(t)\, e_u^{(j)}(t) + b_c^{(j)}(t), \tag{1}$$

$$e_u^{(j+1)}(t) = g\left(z_u^{(j)}(t)\right), \tag{2}$$

where $W_c^{(j)}(t)$ and $b_c^{(j)}(t)$ denote the weights and biases, and $g(\cdot)$ is the activation function. The output at the designated cut layer J_c is denoted as $e_u^{(J_c)}(t)$.

Once the ED completes its own local forward pass, it waits to receive the intermediate results from its paired BD. Specifically, the BD transmits its stopping point layer $J_v (< J_c)$ and the

corresponding intermediate feature $e_v^{(J_v)}(t)$. Upon reception, the ED resumes the forward computation on behalf of the BD from layer $J_v + 1$ up to J_c, thereby generating $e_v^{(J_c)}(t)$. Finally, the ED uploads both $e_u^{(J_c)}(t)$ and $e_v^{(J_c)}(t)$ to the edge server.

(2) **BD-side computation**: In parallel, the BD v executes forward propagation using its dataset $\mathcal{D}_v(t)$. Due to limited computational resources, the BD only proceeds up to a certain layer $J_v < J_c$, producing intermediate features $e_v^{(J_v)}(t)$. At this point, it pauses computation and transmits both J_v and $e_v^{(J_v)}(t)$ to the ED.

4.1.2 Server-Side Execution

The server-side execution in the CSFL framework strictly follows the standard SFL paradigm, operating independently and in parallel across all collaboration pairs [27].

(1) **Forward processing**: Upon receiving the intermediate outputs from both the efficient and bottleneck devices in each pair, the server concatenates them to form the input to the server-side model θ_s. It then completes the forward propagation through the remaining layers to generate the final predictions, serving as the basis for loss computation. The output for sample i on device k at time t is given by

$$\hat{y}_i^{(k)}(t) = g\left(z_u^{(J)}(t)\right), \tag{3}$$

where J denotes the index of the output layer of the model and $g(\cdot)$ is the activation function.

(2) **Model update**: Using the predicted outputs $\hat{y}_i^{(k)}(t)$ and ground truth $y_i^{(k)}$, the server computes the overall training loss using a standard sample-wise loss function $\ell(\cdot)$. The aggregated loss across all devices is defined as

$$\mathcal{L}_s^{(k)} = \frac{1}{\sum_{k\in\mathcal{K}} D_k} \sum_{k\in\mathcal{K}} \sum_{i=1}^{D_k} \ell(y_i^{(k)}, \hat{y}_i^{(k)}). \tag{4}$$

Based on this loss, it calculates gradients for the server-side model parameters, the gradients

are computed as

$$\nabla \mathcal{L}(t, \theta_s(t)) = \frac{1}{K} \sum_{k \in \mathcal{K}} \frac{\partial \mathcal{L}_s^{(k)}(t)}{\partial \theta_s(t)}, \tag{5}$$

and updates them using stochastic gradient descent (SGD) [54]:

$$\theta_s(t+1) \leftarrow \theta_s(t) - \eta_s \nabla \mathcal{L}(t, \theta_s(t)), \tag{6}$$

where η_s denotes the learning rate of the server-side model updates.

To enable backward propagation on the client side, the server transmits the corresponding upstream gradients, computed at the cut layer, back to the efficient devices. This design leverages the role of EDs as the sole uploaders of smashed data [8].

4.1.3 Local Backward Computation

The backward process in CSFL is jointly conducted by the EDs and the BDs following the completion of server-side backpropagation.

(1) *ED-side computation*: Each ED performs gradient computation not only for its own model segment but also for the unfinished portion of its paired BD's model, spanning from the cut layer J_c down to the BD's interruption point J_v. Let $\delta_u^{(j)}$ represent the backpropagated gradient at layer j. The error signal [55] is calculated recursively as

$$\delta_u^{(j)} \leftarrow \left(W_c^{(j+1)}(t)\right)^T \delta_u^{(j+1)}(t) \odot g'\left(z^{(j)}(t)\right) \tag{7}$$

Once the gradient at layer $J_v + 1$ is obtained, the ED transmits this gradient $\delta_v^{(J_v+1)}$ to the corresponding BD. It then continues its own backward pass independently. After reaching the input layer, the ED updates its model parameters using standard SGD:

$$\theta_c^{(u)}(t+1) \leftarrow \theta_c^{(u)}(t) - \eta_c \nabla \mathcal{L}(t, \theta_c^{(u)}(t)) \tag{8}$$

where η_c is the learning rate of client-side model updates.

(2) *BD-side computation*: After receiving the intermediate gradient from its paired ED, each

BD resumes the remaining portion of the backward pass. The local model is then updated accordingly to complete the training cycle on the bottleneck device.

(3) *Device-side aggregation*: As proposed in [6], upon completing their local updates, all devices transmit their client-side model parameters to a designated aggregation server (denoted by a), which aggregates the updates and redistributes the aggregated parameters to all participants. This procedure ensures global consistency across the distributed models.

A complete training round in CSFL thus spans from the initiation of forward propagation at the device level to the final client-side aggregation, forming a closed loop of collaborative model. A summary of important notations is given in Table 4.1.

Table 4.1: Summary of Notation

Notation	**Definition**
$\mathcal{K}$	Set of devices
$\mathcal{U}$	Set of efficient device cluster
$\mathcal{V}$	Set of bottleneck device cluster
$\mathcal{T}$	Set of training rounds
$\mathcal{D}_k$	Local dataset of device k
θ_s	Server-side model
$\theta_c^{(k)}$	Device-side model of device k
$z_k^{(j)}$	Linear transformation output at layer j of device k
$e_k^{(j)}$	Intermediate result at layer j of device k
η_c, η_s	Learning rate of device/server-side model update
$\delta_k^{(j)}$	Gradient/Error signal at layer j of device k
x_{uv}	Pairing decision
$m, \mathcal{M}$	Pairing configuration and the set of pairing configurations
$J_c, \mathcal{J}$	Cut layer and the set of cut layers
J_v	The layer at which an efficient device starts collaboration
$D_k^{(j)}$	Size of intermediate result at layer j of device k
$f_k/f_s/f_a$	Computing capability of device k/server in FLOP/s
κ	FLOPs required per parameter for gradient computation
$\hat{\eta}_{m_i}$	Average collaborative resource utilization for pairing n_i

4.2 Decision Variables and System Efficiency

Given the collaborative nature of the framework, the selection of the cut layer and the device pairing strategy play a critical role, directly affecting synchronization delay, computational workload distribution, and communication overhead. To systematically address these challenges, we introduce a set of formal variables and performance metrics to quantify system efficiency and analyze the interdependencies among them.

4.2.1 Pairing Strategies

The collaborative interaction among devices is governed by the adopted pairing strategy, which plays a pivotal role in improving system-wide efficiency. In this work, we consider a one-to-one pairing scheme, where each BD is matched with exactly one ED during each training round. The pairing decision is formulated as

$$x_{uv}^{(t)} \in \{0,1\}, \quad \forall u \in \mathcal{U},\, v \in \mathcal{V},\, t \in \mathcal{T} \tag{9}$$

$$\sum_{u=1}^{U} x_{uv}^{(t)} = 1, \quad \sum_{v=1}^{V} x_{uv}^{(t)} = 1, \tag{10}$$

where $x_{uv}^{(t)} = 1$ indicates that ED is paired with BD in training round t, and $\mathcal{T} = \{1, 2, \ldots, T\}$ denotes the set of all training rounds.

Let $\mathcal{M}$ represent the set of all feasible one-to-one pairing configurations in a given round, and let $M = |\mathcal{M}|$ be its cardinality. In the case where $U = V$, the total number of feasible pairing configurations simplifies to

$$M = \binom{V}{U} \times U! = \frac{V!}{(V-U)!} = V! \tag{11}$$

It is worth noting that in each pairing, the ED and BD inherently follow distinct optimization objectives. As the assisting agent, the ED must account for both the computational burden of processing the BD's remaining layers and the communication costs associated with uploading the intermediate features and gradients. In contrast, the BD is primarily concerned with the communication

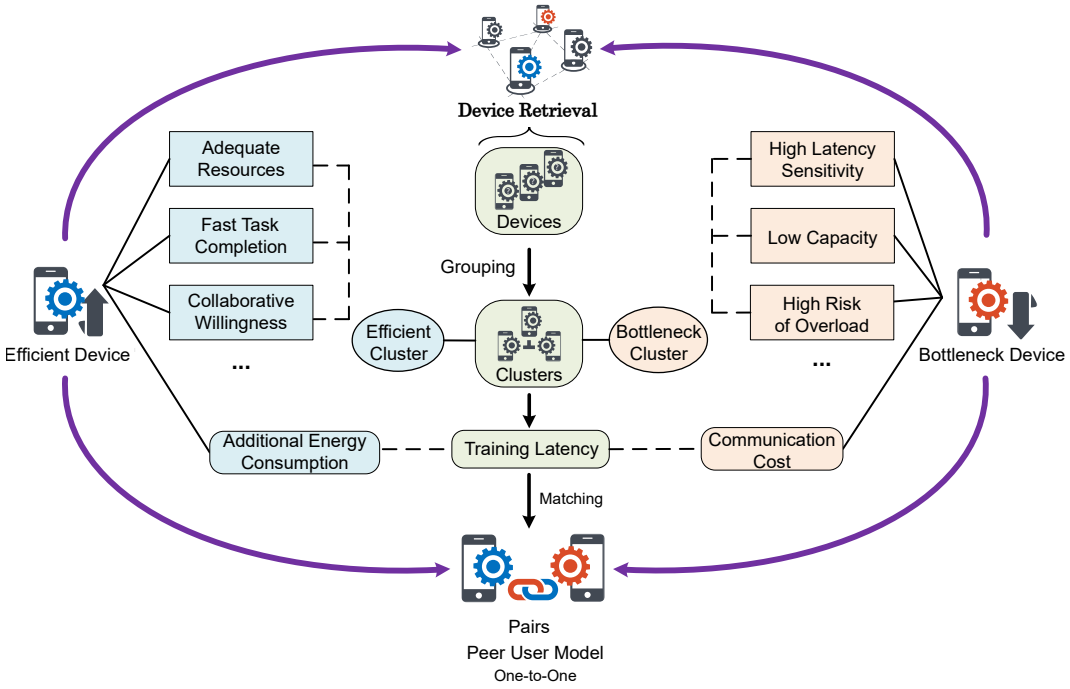

Figure 4.1: Pairing strategy aligned with shared goals and individual preferences.

overhead involved in transmitting its partial forward results. These differing concerns, illustrated in Fig.4.1, highlight the necessity of an efficient and balanced pairing strategy.

4.2.2 Cut Layer Selection

The choice of cut layer significantly affects both the computational burden distribution and the communication overhead throughout the training process. In the CSFL framework, cut layer selection not only partitions the model between edge devices and the server but also impacts the collaborative interaction between efficient and bottleneck devices [56].

Let θ denote the full model architecture, which consists of J layers. The feasible set of cut layers is then given by

$$J_c \in \mathcal{J} = \{2, 3, \ldots, J\} \tag{12}$$

The choice of J_c determines the position at which the model is divided between local and server.

Crucially, the size of the feature map at the cut layer directly determines the communication

cost incurred during forward propagation. In convolutional neural networks, feature maps typically shrink in spatial dimensions as the depth increases [57]. To quantify the communication overhead, we model the volume of transmitted intermediate features as

$$\begin{aligned} D_k^{(J_c)}(t) &= \|e_k^{(J_c)}(t)\|_p \times N_{\text{bit}} \\ &= \|e_k(t)\|_p \times \left(\alpha^{J_c} \prod_{i=1}^{d_m^k(t)} s_{J_c}(i)\right) \times N_{\text{bit}} \\ &= D_k(t) \times \prod_{j=1}^{J_c} \left(\alpha^j \prod_{i=1}^{d_m^k(t)} s_j(i)\right) \end{aligned} \tag{13}$$

Here, $\|e_k^{(J_c)}(t)\|_p$ denotes the norm of the intermediate activation vector at the cut layer for device k in round t, and N_{bit} represents the bit precision per activation (e.g., 32-bit float). The term α^j denotes the channel expansion factor at layer j, capturing how the number of output channels grows relative to input channels.[1] Moreover, $\prod s_j(i)$ captures the cumulative downsampling effect across spatial dimensions, such as that introduced by pooling or strided convolutions [59].

The resulting expression provides an estimate of the data volume that must be transmitted, allowing us to assess how different cut layer positions affect bandwidth usage and latency.

4.2.3 Training Latency

The end-to-end latency of a training round in CSFL consists of multiple components stemming from device-side computation, communication, server-side processing, and aggregation. These components are detailed as follows:

(1) *Device-Side Training Latency*: Let f_k represent the computing capacity (in FLOP[2]/s) of device k, and Φ_j denote the per-sample computation load at layer j [61, 62]. The latency experienced at the device level can be decomposed into three distinct segments:

First, the BD conducts local computation up to a partition point J_v, which is determined by ensuring that its execution time is not shorter than the ED's full forward computation. This

[1] In practice, especially within the shallow-to-mid regions of CNNs where cut layers are typically chosen, the channel dimension often remains constant or grows slowly [58]. Thus, for analytical tractability, we assume $\alpha^j \approx 1$ and omit it in subsequent derivations.

[2] Floating-Point Operations [60]

partition point satisfies:

$$J_v = \min \left\{ j^* \geq 1 \;\middle|\; \sum_{j=1}^{j^*} \frac{\Phi_j \cdot D_v}{f_v} \geq \sum_{j=1}^{J_c} \frac{\Phi_j \cdot D_u}{f_u} \right\} \tag{14}$$

Consequently, the BD's initial computation delay is given by:

$$T_c^w = \sum_{j=1}^{J_v} \frac{\Phi_j \cdot D_v}{f_v} \tag{15}$$

Second, the BD transmits its intermediate activation $e_v^{(J_v)}$ to the paired ED. The size of this intermediate feature is approximated by:

$$D_v^{(J_v)}(t) = D_v(t) \times \prod_{j=1}^{J_v(t)} \prod_{i=1}^{d_m^v(t)} s_j(i) \tag{16}$$

Accordingly, the associated transmission latency is:

$$T_c^{tr} = \frac{D_v^{(J_v)}}{R_{v,u}} \tag{17}$$

Third, the ED completes the remaining forward layers (from $J_v + 1$ to J_c) on behalf of the BD. This collaborative computation time is given by:

$$T_c^a = \sum_{j=J_v+1}^{J_c} \frac{\Phi_j \cdot D_v^{(J_v)}}{f_u} \tag{18}$$

By aggregating all three components, the total device-side latency becomes:

$$T_c = T_c^w + T_c^{tr} + T_c^a \tag{19}$$

(2) *Uplink Transmission Latency*: After completing local computations, the ED transmits both

its own and the BD's outputs at the cut layer J_c to the server. The corresponding latency is:

$$T_{tr} = \frac{D_u^{(J_c)} + D_v^{(J_c)}}{R_{u,s}} \tag{20}$$

where $R_{u,s}$ is the transmission rate from the ED to server.

(3) *Server-Side Computation Latency*: Let f_s denote the processing speed of the server, and $D_s^{(J_c)}$ represent the combined data received from all EDs. The server processes the upper layers from J_c to J with latency:

$$T_c^s = \sum_{j=J_c}^{J} \frac{\Phi_j \cdot D_s^{(J_c)}}{f_s} \tag{21}$$

(4) *Server-Side Updating Latency*: The server then performs backpropagation and model aggregation. Let κ represent the number of FLOPs required per model parameter during gradient computation and backpropagation, which reflects the computational complexity of each parameter update. The total update latency is:

$$T_b^s = \frac{\sum_{j=J_c}^{J} |\theta_s^j| \cdot \kappa + K \cdot |\theta_s|}{f_s} \tag{22}$$

(5) *Downlink Transmission Latency*: Upon completing the update, the server sends the upstream gradients for both devices to the ED:

$$T_{tr}^s = \frac{1}{R_{s,u}} \cdot \left(|\delta_u^{(J_c)}| + |\delta_v^{(J_c)}| \right) \tag{23}$$

where $R_{s,u}$ represents the transmission rate from the server to the related ED.

(6) *Device-Side Updating Latency*: Once gradients are received, the ED continues the backward propagation for both itself and the BD up to J_v:

$$T_b^a = \sum_{j=J_v+1}^{J_c} \frac{|\theta_c^{(u,j)}| \cdot \kappa}{f_u} \tag{24}$$

The ED then transmits the intermediate gradient to the BD:

$$T_b^{tr} = \frac{|\delta_v^{(J_v+1)}|}{R_{u,v}} \tag{25}$$

The BD finishes the remaining backward propagation and updates its model:

$$T_b^{w} = \sum_{j=1}^{J_v} \frac{|\theta_c^{(v,j)}| \cdot \kappa}{f_v} \tag{26}$$

Hence, the overall device-side update latency is:

$$T_b = T_b^{a} + T_b^{tr} + T_b^{w} \tag{27}$$

(7) *Device-Side Aggregation Latency*: Following model updates, devices upload their local parameters to the aggregation server a, and receive the aggregated results:

$$T_a = \frac{K \cdot |\theta_c^{(k)}|}{f_a} + \frac{|\theta_c^{(k)}|}{R_{k,a}} + \frac{|\theta_c^{(k)}|}{R_{a,k}} \tag{28}$$

where f_a denotes the server's processing capability, and $R_{k,a}$, $R_{a,k}$ denote the uplink and downlink communication rates between device k and server a, respectively.

All transmission rates defined in this section are derived from the Shannon–Hartley theorem [63]. Let tr and rr denote the transmitter and receiver, respectively. The transmission rate $R_{tr,rr}$ is determined by the bandwidth B_{tr}, transmit power p_{tr}, channel gain $h_{tr,rr}$, and noise power σ_{rr}^2 at the receiver [64], and is given by

$$R_{tr,rr} = B_{tr} \log \left(1 + \frac{p_{tr}|h_{tr,rr}|^2}{B_{tr}\sigma_{rr}^2}\right) \tag{29}$$

Given $\mathcal{P} \in \mathcal{M}$ indicates a pairing configuration consisting of U collaborative pairs. The total latency of one training round can therefore be summarized as:

$$T_{ove} = \max_{(u,v)\in\mathcal{P}} (T_c + T_{tr} + T_b) + \max_{k\in\mathcal{K}} T_c^s + T_b^s + \max_{u\in\mathcal{U}} T_{tr}^s + T_a \tag{30}$$

In parallel with latency modeling, we also evaluate the energy consumption of each collaborative pair. For the ED, the energy arises from both computation and transmission processes. The computation cost is modeled using the CMOS energy model [65]:

$$E_u = \sum_{j=J_v+1}^{J_c} \frac{\Phi_j \cdot D_v^{(J_v)}}{f_u} \cdot F_u^3 \cdot \rho + \frac{D_v^{(J_c)} + |\delta_v^{(J_v+1)}|}{R_u} \cdot p_u \tag{31}$$

where F_u is the CPU frequency and ρ the energy coefficient. The BD's energy consumption, dominated by transmission, is given by:

$$E_v = \frac{D_v^{(J_v)}}{R_v} \cdot p_v \tag{32}$$

These latency and energy expressions collectively serve as foundations for the multi-objective optimization problem introduced in the subsequent section. By quantifying the impact of device heterogeneity and collaborative strategies on system performance, the CSFL framework enables principled exploration of trade-offs between responsiveness, workload distribution, and energy efficiency.

4.3 Problem Formulation and Objective

Building upon the system efficiency analysis in Section 4.2, we now formulate a joint optimization problem that governs the selection of cut layers and pairing strategies. The goal is to minimize overall training latency while maintaining an acceptable level of energy consumption and model performance. This problem inherently involves discrete decisions and resource-dependent constraints, which reflect the collaborative and heterogeneous nature of CSFL.

4.3.1 Problem Formulation

To tackle the intertwined challenges of heterogeneity, workload imbalance, and synchronization delay, we formulate a long-term optimization problem over T collaborative training rounds. The objective is to minimize the average end-to-end latency across all rounds, while satisfying energy

and workload balancing constraints. The problem is formulated as follows:

$$\text{P1}: \quad \min_{J_c, \mathcal{P}^{(t)}} \ \frac{1}{T}\sum_{t=1}^{T} T_{ove}^{(t)} \tag{33}$$

$$\text{s.t.} \quad E_u^t \le E_u^{th}, \quad E_v^t \le E_v^{th}, \tag{33a}$$

$$J_v^{(t)} + 1 \le J_c, \tag{33b}$$

$$\hat{\eta}_{th} \le \hat{\eta}(t, J_c) \le 1, \tag{33c}$$

$$(9),\ (10), \text{and } (12).$$

Constraint (33a) ensures that the energy consumed by both EDs and BDs during computation and communication remains within acceptable bounds. Constraint (33b) guarantees the logical correctness of collaboration: the cut layer must be above the BD's stopping point so that EDs can complete the unfinished layers. Constraint (33c) promotes fairness in workload distribution by imposing a lower bound on the system-wide collaboration balance.

To quantify workload allocation, we define the collaboration ratio η_u for each ED as:

$$\eta_u = \frac{\alpha \cdot \sum_{j=J_v+1}^{J_c} \left(\Phi_j \cdot D_v^{(J_v)} + |\theta_c^{(u,j)}| \cdot \kappa \right)}{F_u \cdot (T_c^a + T_b^a)} \tag{34}$$

where α is a scaling factor that maps FLOPs to physical CPU cycles based on the ED's hardware capability. The numerator captures the assisted computational load from both forward and backward passes, while the denominator reflects the effective computational window available.

We then adopt the Jain's fairness index [66] to evaluate workload balance across all EDs:

$$\hat{\eta}(J_c) = \frac{\left(\sum_{u \in \mathcal{U}} \eta_u\right)^2}{U \cdot \sum_{u \in \mathcal{U}} \eta_u^2} \tag{35}$$

As illustrated in Fig. 4.2, the decision variables of cut layer and pairing configuration are mutually dependent: the choice of J_c affects the workload and feasibility of collaboration, while pairing directly influences the optimality of cut layer placement. This mutual dependency results in a highly non-trivial search space, rendering direct optimization computationally intractable. Thus, an efficient decomposition strategy is needed to jointly optimize these coupled decisions.

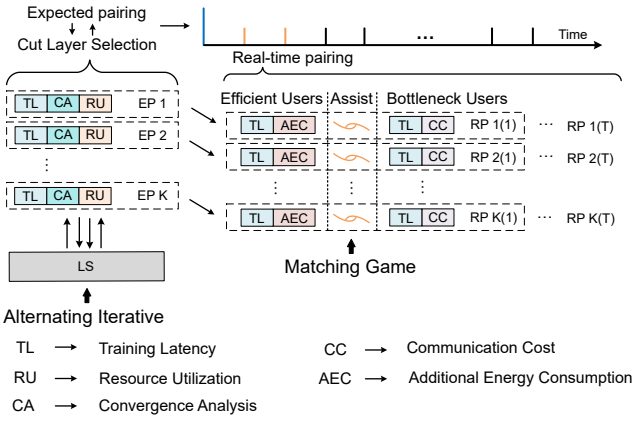

Figure 4.2: Relationship between two subproblem.

4.3.2 Problem Decomposition

In practical deployment, the cut layer is often fixed throughout the training period, whereas the pairing strategy must adapt dynamically across iterations. This asymmetry in decision frequency motivates a decoupled treatment of the two variables.

- *Cut layer optimization*: The cut layer J_c is considered a high-level decision variable, optimized offline under long-term system behavior, while the pairing strategy adapts accordingly. Given that both data volume $\mathcal{D}$ and channel conditions h vary over time, it is infeasible to predefine pairing strategies for every round. To enable tractable optimization, we introduce the expectation operator $\mathbb{E}[\cdot]$ to characterize the average-case latency, assuming that $\mathcal{D}$ and h follow known distributions. Based on the expected system state, a steady-state pairing strategy $\mathcal{P}_e$ is derived. The resulting problem is formulated as:

$$\text{P1.1}: \quad \min_{J_c, \mathcal{P}_e} \ \mathbb{E}_{\mathcal{D},h}\left[T_{ove}\right] \tag{36}$$

$$\text{s.t.} \quad \mathcal{P}_e \in \mathcal{M}, \tag{36a}$$

$$(9),\ (10),\ (12),\ (33b), \text{and } (33c).$$

Problem P1.1 is non-convex due to the discrete nature of J_c, the nonlinear relationship between computation/communication latency and J_c, and the use of max operations in latency evaluation, which introduces discontinuities in the objective function.

- *Pairing optimization*: Once the cut layer is determined, pairing configurations are refined at each training round t to minimize the round-specific training latency. The corresponding optimization problem is given by

$$\text{P1.2}: \quad \min_{\mathcal{P}^{(t)}} \; T_{ove}^{(t)} \tag{37}$$
$$\text{s.t.} \quad (9), (10), \text{and } (33a).$$

As the pairing configuration defines a set of discrete one-to-one mappings, problem P1.2 constitutes a combinatorial optimization problem, which is inherently NP-hard.

4.4 Cut Layer Optimization

To effectively address the joint optimization problem P1.1, which simultaneously considers the selection of the cut layer J_c and the expected pairing configuration $\mathcal{P}_e$, we first define a compact objective function as

$$f(\mathcal{P}_e, J_c) = \mathbb{E}_{\mathcal{D},h}\left[T_{ove}\right],$$

where the expectation is taken over the distributions of device data volumes and channel conditions. This reformulation captures the long-term average latency under stochastic environments.

Recognizing the interdependence of the decision variables and the disparity in their temporal scales, where pairing decisions are adjusted frequently while the cut layer remains relatively stable, we decouple P1.1 into two sub-problems that are solved iteratively. Furthermore, to promote fairness in resource utilization, we incorporate constraint (33c) directly into the objective as a regularization term, yielding the following two-stage optimization formulation.

In the first stage, we update the expected pairing configuration $\mathcal{P}_e^{(\tau+1)}$ based on the current cut

layer $J_c^{(\tau)}$ by solving:

$$\text{P1.1.1}: \quad \mathcal{P}_e^{(\tau+1)} = \arg\min_{\mathcal{P}_e} f(\mathcal{P}_e, J_c^{(\tau)}) + \lambda_\eta^{\mathcal{P}} \left(\hat{\eta}_{th} - \hat{\eta}(J_c^{(\tau)})\right) \tag{38}$$

$$\text{s.t.} \quad \hat{\eta}(J_c^{(\tau)}) \leq 1, \tag{38a}$$

$$(9), \text{and } (10).$$

The second stage fixes the newly obtained pairing configuration $\mathcal{P}_e^{(\tau+1)}$, and seeks the cut layer $J_c^{(\tau+1)}$ that minimizes the regularized latency objective:

$$\text{P1.1.2}: \quad J_c^{(\tau+1)} = \arg\min_{J_c} f(\mathcal{P}_e^{(\tau+1)}, J_c) + \lambda_\eta^{J} \left(\hat{\eta}_{th} - \hat{\eta}(J_c)\right) \tag{39}$$

$$\text{s.t.} \quad \hat{\eta}(J_c) \leq 1, \tag{39a}$$

$$(12), (33b), \text{and } (36a).$$

Here, τ denotes the iteration index and $\lambda_\eta^{\mathcal{P}}, \lambda_\eta^{J} \in [0, 1]$ are trade-off coefficients that balance latency against workload fairness. For P1.1.1, a matching theory-based algorithm may be employed to determine a near-optimal $\mathcal{P}_e$, as detailed in Section 4.5.

Given the discrete nature of J_c and the non-smooth latency landscape introduced by max operations, solving P1.1.2 directly is computationally challenging. To enable gradient-based search, we relax the domain of J_c to a continuous interval and approximate its optimal value by identifying stationary points of the smooth surrogate objective:

$$\text{P1.1.3}: \quad J_c^{(\tau+1)} = \arg\min_{J_c} f(\mathcal{P}_e^{(\tau+1)}, J_c) + \lambda_\eta^{J} \left(\hat{\eta}_{th} - \hat{\eta}(J_c)\right) \tag{40}$$

$$\text{s.t.} \quad J_c \in [2, J], \tag{40a}$$

$$(33b), (36a), \text{and } (39a).$$

To solve problem P1.1.3 within the continuous domain, we first introduce an auxiliary objective function that consolidates the expected training latency and the workload balance constraint into a unified formulation:

$$G(J_c) = f(\mathcal{P}_e^{(\tau+1)}, J_c) + \lambda_\eta^{J} \left(\hat{\eta}_{th} - \hat{\eta}(J_c)\right),$$

We then proceed to compute the partial derivative of the auxiliary objective function $G(J_c)$ with respect to the continuous relaxation of the cut layer J_c, thereby enabling gradient-based optimization. Given that $G(J_c)$ consists of two structurally distinct components, we begin by analyzing the derivative of the latency-related term $f(\mathcal{P}_e^{(\tau+1)}, J_c)$.

According to the system latency model defined in (30), the cut layer J_c directly influences several time-critical components, including the cumulative per-layer computational cost Φ_j, the auxiliary starting layer J_v, the intermediate feature sizes $D_k^{(j)}$, the gradient vector $\delta_v^{(j)}$, and the model parameters assigned to both server and device sides, i.e., $|\theta_s^{(j)}|$ and $|\theta_c^{(k,j)}|$.

The derivation of $\sum_{j=1}^{J_c} \Phi_j$ is simplified through the adoption of integral approximation [67]:

$$\sum_{j=1}^{J_c} \Phi_j \approx \int_1^{J_c} \Phi_j \, dj, \quad \frac{d}{dJ_c}\left(\int_1^{J_c} \Phi_j \, dj\right) = \Phi_{J_c} \tag{41}$$

Furthermore, as J_c changes, the value of J_v, which determines where the BD stops its local computation, also varies implicitly. Using the thresholding condition defined in (14), we obtain the sensitivity of J_v with respect to J_c as:

$$\frac{\partial J_v}{\partial J_c} = \frac{\Phi_{J_c} \cdot D_u \cdot f_v}{\Phi_{J_v} \cdot D_v \cdot f_u} \tag{42}$$

This derivative captures the coupling between ED and BD workload and is critical in determining how the collaborative boundary adjusts during cut layer updates.

The influence of J_c on communication cost is analyzed by examining the size of intermediate features. Let $S(J_c) = \prod_{j=1}^{J_c} \prod_{i=1}^{d_m^k} s_j(i)$, representing the cumulative spatial downsampling up to layer J_c. Its derivative unfolds as:

$$\frac{dS(J_c)}{dJ_c} = \left(\prod_{j=1}^{J_c-1} S_j\right) \cdot \frac{dS_{J_c}}{dJ_c} \tag{43}$$

where the inner term becomes

$$\frac{dS_{J_c}}{dJ_c} = \sum_{i=1}^{d_m^k} \left(\prod_{\substack{i'=1 \\ i' \neq i}}^{d_m^k} s_{J_c}(i') \cdot \frac{\partial s_{J_c}(i)}{\partial J_c} \right) \tag{44}$$

As a result, the gradient of the intermediate feature size at the cut layer is written as:

$$\frac{dD_k^{(J_c)}}{dJ_c} = D_k \cdot \prod_{j=1}^{J_c-1} S_j \cdot \sum_{i=1}^{d_m^k} \left(\prod_{\substack{i'=1 \\ i' \neq i}}^{d_m^k} s_{J_c}(i') \cdot \frac{\partial s_{J_c}(i)}{\partial J_c} \right) \tag{45}$$

To reduce complexity, following [68], we assume consistent behavior across spatial dimensions for each convolutional layer. This assumption implies that the downsampling ratio along different spatial axes evolves in a uniform manner as the network deepens. This simplification yields an analytical approximation of the feature size gradient with respect to the cut layer:

$$\frac{dD_k^{(J_c)}}{dJ_c} \approx D_k^{(J_c)} \cdot d_m^k \cdot \ln s_{J_c} \tag{46}$$

Analogously, the size of the intermediate output at J_v also varies with J_c, and its derivative can be written as:

$$\frac{dD_k^{(J_v)}}{dJ_c} \approx D_k^{(J_c)} \cdot d_m^k \cdot \ln s_{J_v} \cdot \frac{\partial J_v}{\partial J_c} \tag{47}$$

Given that the analytic form of the gradient vector $\delta_v^{(J_v+1)} = \frac{\partial \mathcal{L}_c}{\partial e_v^{(J_v+1)}}$ is typically intractable due to nonlinearity [69], we apply a finite difference approximation based on adjacent layers:

$$\frac{\partial}{\partial J_v} \frac{\partial J_v}{\partial J_c} \left| \frac{\partial \mathcal{L}_c}{\partial e_v^{(J_v+1)}} \right| \approx \frac{\partial J_v}{\partial J_c} \cdot \left(\left| \frac{\partial \mathcal{L}_c}{\partial e_v^{(J_v+2)}} \right| - \left| \frac{\partial \mathcal{L}_c}{\partial e_v^{(J_v+1)}} \right| \right) \tag{48}$$

On the server side, the dependency of latency on the parameter count motivates analysis of the gradient of model size:

$$\frac{d}{dJ_c} \left(\int_{J_c}^{J} |\theta_s^j| \, dj \right) = -|\theta_s^{J_c}| \tag{49}$$

For the ED-side model, the effective parameter size is jointly influenced by the collaborative

depth J_v and the upper boundary J_c. Thus, the derivative of its model complexity becomes:

$$\frac{d}{dJ_c}\left(\int_{J_v+1}^{J_c} |\theta_c^{u,j}|\, dj\right) = |\theta_c^{u,J_c}| - |\theta_c^{u,J_v+1}| \cdot \frac{\partial J_v}{\partial J_c} \tag{50}$$

Finally, for the bottleneck device, the local backward path ends at J_v, and its parameter sensitivity is given by:

$$\frac{d}{dJ_c}\left(\int_{1}^{J_v} |\theta_c^{v,j}|\, dj\right) = |\theta_c^{v,J_v}| \cdot \frac{\partial J_v}{\partial J_c} \tag{51}$$

By systematically combining all derivative components, we obtain the composite gradient of $f(\mathcal{P}_e^{(\tau+1)}, J_c)$ with respect to J_c, which serves as the core driver in approximating the optimal cut layer within the continuous domain.

In the second term, $\lambda_\eta^J\,(\hat{\eta}_{th} - \hat{\eta}(J_c))$, the fairness metric $\hat{\eta}(J_c)$ reflects the overall workload balance across collaborative pairs, and is inherently influenced by the choice of cut layer J_c. To quantify its sensitivity with respect to cut layer adjustment, we begin by analyzing the derivative of each individual ratio η_i, which captures the local workload contribution of the i-th ED. Mathematically, the derivative of η_i with respect to J_c follows a rational form:

$$\frac{d\eta_i}{dJ_c} = \frac{C_2 \cdot \frac{dC_1}{dJ_c} - C_1 \cdot \frac{dC_2}{dJ_c}}{C_2^2} \tag{52}$$

where C_1 and C_2 denote the numerator and denominator of η_i, respectively, as defined in (34). These components respectively characterize the cumulative computational load and the effective computing window available on the ED. The partial derivative of the numerator C_1 accounts for the dynamics in both forward and backward collaborative computation:

$$\begin{aligned}\frac{dC_1}{dJ_c} = \alpha\Bigg[&\left(\Phi_{J_c} - \Phi_{J_v} \cdot \frac{\partial J_v}{\partial J_c}\right) \cdot D_v^{(J_v)} + \sum_{j=J_v+1}^{J_c} \Phi_j \cdot \frac{\partial D_v^{(J_v)}}{\partial J_c} \\ &+ \kappa \cdot \left(|\theta_c^{(u,J_c)}| - |\theta_c^{(u,J_v+1)}| \cdot \frac{\partial J_v}{\partial J_c}\right)\Bigg]\end{aligned} \tag{53}$$

The corresponding derivative of the denominator C_2 is obtained by recognizing the structural

proportionality between the numerator and denominator:

$$\frac{dC_2}{dJ_c} = \frac{F_u}{f_u \cdot \alpha} \cdot \frac{dC_1}{dJ_c} \tag{54}$$

Having derived the sensitivity of each η_i to the cut layer, we next compute the total derivative of the global workload balance metric $\hat{\eta}(J_c)$ using the standard formulation of Jain's fairness index. This yields:

$$\frac{d\hat{\eta}(J_c)}{dJ_c} = \frac{2 \cdot \sum_{u\in\mathcal{U}} \eta_u}{U \cdot \left(\sum_{u\in\mathcal{U}} \eta_u^2\right)^2} \left[\left(\sum_{u\in\mathcal{U}} \eta_u^2 \right) \cdot \sum_{u\in\mathcal{U}} \frac{d\eta_u}{dJ_c} - \left(\sum_{u\in\mathcal{U}} \eta_u \right) \cdot \sum_{u\in\mathcal{U}} \eta_u \cdot \frac{d\eta_u}{dJ_c} \right] \tag{55}$$

Combining the gradient expressions for both the expected latency term $f(\mathcal{P}_e^{(\tau+1)}, J_c)$ and the workload balance term $\hat{\eta}(J_c)$, we obtain the complete derivative of the auxiliary objective function $G(J_c)$. To identify a candidate for the optimal cut layer, we solve the condition

$$\frac{dG(J_c)}{dJ_c} = 0,$$

which corresponds to locating a stationary point of the relaxed objective.

Once a solution is obtained, we perform feasibility verification by checking whether the value of J_c satisfies the structural constraints imposed by (33b) and (40a). If the result falls outside the permissible interval, we project it back onto the nearest feasible boundary [70]. Otherwise, the nearest smaller integer is selected as the optimal cut layer for the current iteration τ, ensuring compatibility with the discrete structure of the original problem. The detailed procedure for cut layer determination is summarized in Algorithm 1, and the convergence properties of this alternating scheme are formally proven in Appendix A.

Algorithm 1 Cut Layer Selection

Input: $\mathcal{U}$, $\mathcal{V}$, $\mathcal{J}$, iteration index τ and number of iteration $\mathcal{T}_i$.
Output: J_c^*

Initialize: $\lambda_\eta^{\mathcal{P}_e}$, λ_η^J and initial cut layer $J_c^{(0)}$
for each iteration $\tau = 0$ to $\mathcal{T}_i$ **do**

    /** Pairing Configuration **/
    Fixed cut layer J_c^τ
    Obtain all stable configurations Ψ^* using Alg. 2 and 3
    Select $\mathcal{P}_e^{(\tau+1)} = \arg\min_{\mathcal{P}_e^{(i)} \in \Psi^*} T_{ove}$

    /** Layer Selection **/
    Fixed pairing configuration $\mathcal{P}_e^{(\tau+1)}$
    Compute stationary point $b^{(\tau+1)}$
    if $b^{(\tau+1)} \leq 2$ **then**
        $b^{(\tau+1)} \leftarrow 2$
    else if $b^{(\tau+1)} \geq J$ **then**
        $b^{(\tau+1)} \leftarrow J$
    end if
    if $b^{(\tau+1)} \leq J_v + 1$ **then**
        $b^{(\tau+1)} \leftarrow J_v + 1$
    end if
    if $b^{(\tau+1)} \in \mathbb{Z} \cap [\max\{2, J_v + 1\}, J]$ **then**
        Set $J_c^{(\tau+1)} = b^{(\tau+1)}$
    else
        Set $J_c^{(\tau+1)} = \arg\min_{J_c \in \{\lfloor b^{(\tau+1)} \rfloor, \lceil b^{(\tau+1)} \rceil\}} G(J_c)$
    end if
end for
return $J_c^* = J_c^{(\tau)}$

4.5 Pairing Optimization

Solving the weighted stable matching problem in this context requires an algorithm that jointly respects local preference orders and system-wide latency objectives. We design a tailored pairing mechanism that extends the classical Gale–Shapley [71] deferred acceptance algorithm by embedding a delay-aware refinement process.

To this end, the collaborative device set is partitioned into two disjoint subsets: the efficient devices (EDs), denoted by $\mathcal{U}$, and the bottleneck devices (BDs), denoted by $\mathcal{V}$, with cardinalities satisfying $|\mathcal{U}| = |\mathcal{V}| = U$. This enables a one-to-one matching scenario, where each $u \in \mathcal{U}$ is uniquely paired with some $v \in \mathcal{V}$, forming a bijective mapping $\psi : \mathcal{U} \to \mathcal{V}$ such that $\psi(u) = v$ if

and only if $\psi^{-1}(v) = u$ [72].

The preference structures of devices are formulated based on individual energy or latency costs. For each ED $u \in \mathcal{U}$, its strict preference over BDs is determined by:

$$v_1 \succ_u v_2 \iff E_u(v_1) < E_u(v_2), \quad \forall v_1, v_2 \in \mathcal{V}, \tag{56}$$

and similarly, for each BD $v \in \mathcal{V}$, we define:

$$u_1 \succ_v u_2 \iff E_v(u_1) < E_v(u_2), \quad \forall u_1, u_2 \in \mathcal{U}. \tag{57}$$

Property 1 *The bilateral preference relations constructed over the two device sets form strict total orders. That is, for any pair of devices, exactly one of the following holds:* $u \succ v$, $v \succ u$, *or* $u \sim v$. *Moreover, the transitive closure of preferences is preserved, ensuring preference acyclicity.*

While these relations capture individual optimization tendencies, our framework additionally requires global alignment with respect to latency minimization. To integrate global considerations, we reformulate the problem as a weighted stable matching problem, where the matching outcome must satisfy both local stability and delay-optimality. Specifically, a matching ψ is said to be *globally stable* if no unmatched pair (u, v) exists such that:

$$\neg \left(v \succ_u \psi(u) \wedge u \succ_v \psi(v) \wedge G(\psi') < G(\psi) \right), \tag{58}$$
$$\text{where} \quad G(\psi) = \max_{(u,v)\in\psi} \left(T_c^{(u,v)} + T_{tr}^{(u,v)} + T_b^{(u,v)} \right).$$

The proposed solution proceeds in two stages. First, an initial stable matching ψ_0 is derived using the Gale–Shapley deferred acceptance algorithm, which guarantees the absence of blocking pairs based on individual preference profiles, as outlined in Algorithm 2.

Next, to further reduce the system-wide delay, we employ a *rotation-based refinement* [73, 74], which exhaustively explores the space of all stable matchings derived from ψ_0. This process is performed through:

Algorithm 2 Gale-Shapley for Stable Matching

Input: $\mathcal{U}$, $\mathcal{V}$, $\succ_u$ and $\succ_v$.
Output: A stable configuration ψ_0.

Initialize: $\psi_0 = \emptyset$
Mark each individual $u \in \mathcal{U}$ as 'unmatched'.
while HasUnmatched($\mathcal{U}$) **do**
    select an unmatched $u \in \mathcal{U}$ with untried candidates
    $v \leftarrow$ the top-ranked untried candidate of u
    if v is unmatched **then**
        $\psi_0 \leftarrow \psi_0 \cup \{(u, v)\}$
        mark u and v as matched
    else
        let (u', v) be the current match in ψ_0
        if $u \succ_v u'$ **then**
            $\psi_0 \leftarrow (\psi_0 \setminus \{(u', v)\}) \cup \{(u, v)\}$
            mark u' as unmatched
            mark u as matched
        else
            u remains unmatched
        end if
    end if
end while
return ψ_0

Step 1 — Rotation Identification: The finding process is conducted by constructing an improvement graph, where each participant is represented as a node, and directed edges indicate potential preference-improving switches. Specifically, a directed edge $(i \to \psi(j))$ is added if participant i prefers j's current partner over their own assignment, and j's current partner also prefers i over j. Each edge thus represents a mutually beneficial exchange that does not disrupt stability for individual participants. A rotation is identified as a directed cycle in this graph, indicating a sequence of participants who can cyclically exchange partners to achieve a mutually preferable outcome. Since the rotation is executed as a synchronized swap rather than a sequential change, no intermediate state introduces new blocking pairs, ensuring that stability is maintained throughout the transformation.

Step 2 — Rotation Elimination: Once a rotation is identified, all involved participants simultaneously switch to their next preferred partners within the cycle. This synchronized reassignment guarantees that no temporary blocking pair is formed during the transformation, preserving stability.

The complete set of stable matchings can be enumerated through iterative application of the

above steps, as detailed in Algorithm 3. Among these, we compute the latency associated with each configuration and select the one that minimizes the global delay $G(\psi)$ as the optimal outcome.

Algorithm 3 Rotation Enumeration for Stable Matching Sets

Input: $\mathcal{U}$, $\mathcal{V}$, $\succ_u$, $\succ_v$, and ψ_0.
Output: The set of all feasible stable configurations Ψ^*.

Initialize: $\Psi^* \leftarrow \emptyset$
Stack $\leftarrow [\psi_0]$ ▷ Initialize a stack with the initial stable matching
while Stack is not empty **do**
    $\psi \leftarrow$ Stack.pop()
    $\mathcal{R} \leftarrow$ FindAllRotations(ψ)
    if $\mathcal{R} = \emptyset$ **then**
        $\Psi^* \leftarrow \Psi^* \cup \{\psi\}$ ▷ No rotations, record ψ as a final stable matching
    else
        for each $r \in \mathcal{R}$ **do**
            $\psi' \leftarrow$ EliminateRotation(ψ, r)
            Stack.push(ψ')
        end for
    end if
end while
return Ψ^*

The computational complexity of the proposed mechanism is characterized as follows. The initial stable matching is obtained via the Gale–Shapley algorithm, which incurs a time complexity of $\mathcal{O}(U^2)$. Subsequently, the total number of rotations required to enumerate all stable matchings is upper bounded by $\mathcal{O}(U^2)$ in the worst case, with each rotation elimination step incurring an additional cost of $\mathcal{O}(U)$. Consequently, the overall computational complexity of the procedure is asymptotically bounded by $\mathcal{O}(U^3)$.

Chapter 5

Experimental Evaluation

This chapter presents the experimental evaluation of the proposed framework. The objective is to assess its performance in terms of training latency, energy efficiency, and model accuracy under heterogeneous edge environments. To this end, we implement a simulation platform using the VGG-16 architecture and the Tiny-ImageNet dataset, with non-IID data partitions across devices [75].

The effectiveness of the proposed joint optimization strategy, which consists of cut-layer selection and device pairing, is validated through comparison with several baseline schemes, including standard SFL without collaboration, random pairing, and greedy matching. Key evaluation metrics include end-to-end training latency per round, total energy consumption, and convergence behavior across rounds. All results are averaged over multiple runs to ensure consistency and robustness.

5.1 Experimental Setup

The evaluation of the proposed CSFL framework is conducted under a simulated heterogeneous edge environment comprising 20 devices—10 bottleneck devices (BDs) and 10 efficient devices (EDs). Devices are categorized into BDs and EDs on the basis of their computational capacities, reflecting processing power as the dominant factor of heterogeneity considered in the experiments. BDs are characterized by lower computational capabilities, with CPU frequencies ranging from 1.2 to 1.8 GHz, corresponding to a performance of 4.8–7.2 GFLOP/s under an assumed efficiency of 4 FLOPs per cycle [64, 76]. In contrast, EDs operate at 2.0 to 3.5 GHz, achieving 16–28 GFLOP/s at

8 FLOPs per cycle. Reflecting these disparities, BDs and EDs are allocated 10 MHz and 20 MHz of communication bandwidth, respectively.

The wireless communication model adheres to a Rayleigh fading channel [77], where the channel coefficients follow a circularly symmetric complex Gaussian distribution. A centralized server equipped with a 100 GFLOP/s processing unit and 200 MHz of bandwidth supports model aggregation, operating under a nominal signal-to-noise ratio (SNR) of 20 dB.

Training is performed using the *Tiny ImageNet* dataset[1], a lightweight subset of ImageNet containing 200 classes, each with 500 training and 50 validation images at a resolution of 64×64 pixels. This dataset is well-suited for constrained edge computing scenarios while maintaining sufficient complexity for meaningful performance evaluation. Each device receives a randomly sampled, class-imbalanced subset of the data, leading to a non-i.i.d. data distribution across participants.

The backbone model employed is VGG-16 [78], composed of 13 convolutional layers and 3 fully connected layers. To accommodate the dataset, the input layer is modified for 64×64×3 image resolution, and the output layer is adjusted to produce 200 class logits. Model architecture details, including per-layer computational costs and activation sizes, are summarized in Table 5.1.

Table 5.1: VGG-16 Model Structure for Tiny ImageNet.

Layer Name	**NN Units**	**Activation**
Conv1	$64 \times 64 \times 64$ (2× Conv)	ReLU
Pool1	$32 \times 32 \times 64$ (Max Pooling)	-
Conv2	$32 \times 32 \times 128$ (2× Conv)	ReLU
Pool2	$16 \times 16 \times 128$ (Max Pooling)	-
Conv3	$16 \times 16 \times 256$ (3× Conv)	ReLU
Pool3	$8 \times 8 \times 256$ (Max Pooling)	-
Conv4	$8 \times 8 \times 512$ (3× Conv)	ReLU
Pool4	$4 \times 4 \times 512$ (Max Pooling)	-
Conv5	$4 \times 4 \times 512$ (3× Conv)	ReLU
Pool5	$2 \times 2 \times 512$ (Max Pooling)	-
Flatten	1×2048	-
FC1	2048	ReLU
FC2	1024	ReLU
FC3	200 (Softmax)	Softmax

[1]Dataset available at: http://cs231n.stanford.edu/tiny-imagenet-200.zip

5.2 Baseline Comparison

A set of representative schemes is considered to benchmark the effectiveness of the proposed CSFL framework equipped with stable matching (denoted as CSFL-S). These baselines are chosen to reflect both non-collaborative and alternative collaborative learning strategies.

- **SFL**: This configuration follows the classical SFL protocol as described in [11]. Due to the absence of inter-device collaboration, SFL suffers from significant synchronization delays under device heterogeneity.
- **CSFL with random matching (CSFL-R)**: In this baseline, the proposed collaborative CSFL architecture is maintained, but the device pairing is performed randomly in each round. This variant serves to isolate the impact of the matching strategy.
- **CSFL with greedy matching (CSFL-G)**: This method employs a heuristic strategy aimed at minimizing latency in device pairing. In each round, BDs are sequentially selected in random order. Each selected BD greedily pairs with the currently available ED that results in the lowest estimated pairwise training delay.

These comparison baselines offer varying degrees of coordination and intelligence in pairing, and the main simulation parameters are summarized in Table 5.2.

Table 5.2: Simulation Parameters

Parameter	Value	Parameter	Value
B_v	10 MHz	f_v	[4.8, 7.2] GFLOP/s
B_u	20 MHz	f_u	[15, 28] GFLOP/s
B_s	200 MHz	f_s, f_a	100 GFLOP/s
D_u, D_v	[2000, 5000]	F_v	[1.2, 1.8] GHz
$\mathcal{T}$	450	F_u	[2, 3.5] GHz
ρ	10^{-27} F	U, V	10
η_c, η_s	0.001	$h_{tr,rr}$	$\mathcal{CN}(0,1)$
p_u	[17, 23] dBm	p_v	[23, 27] dBm
$\hat{\eta}_{\text{min}}$	0.2	N_{bit}	8 bit/P
κ	2.5 FLOPs/P	σ^2	-174 dBm/Hz

5.3 Performance Evaluation

The effectiveness of the proposed CSFL-S framework is evaluated from multiple perspectives, including model performance, training latency, energy efficiency, and collaboration fairness, benchmarked against both classical SFL and alternative pairing strategies. All results are reported over 450 training epochs.

Model Accuracy and Loss The model accuracy and cross-entropy loss [79] under CSFL-S and SFL are depicted in Figs. 5.1 and 5.2, respectively. Although CSFL-S exhibits slightly delayed convergence compared to SFL, both approaches achieve nearly identical final accuracy and loss values. This demonstrates that introducing collaboration via device pairing does not compromise the optimization trajectory of the global model. The modest increase in the number of training rounds required by CSFL-S is compensated by its reduced per-round latency.

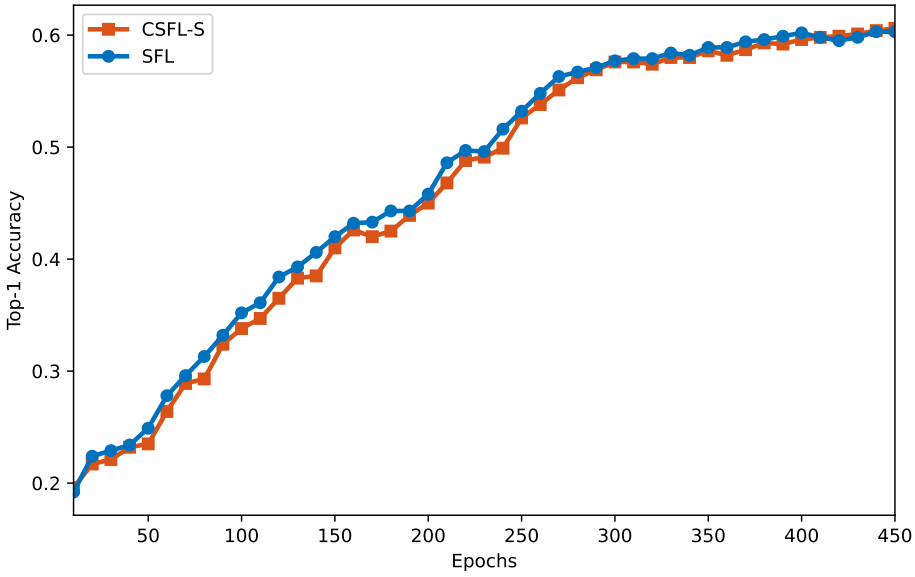

Figure 5.1: Performance in terms of accuracy.

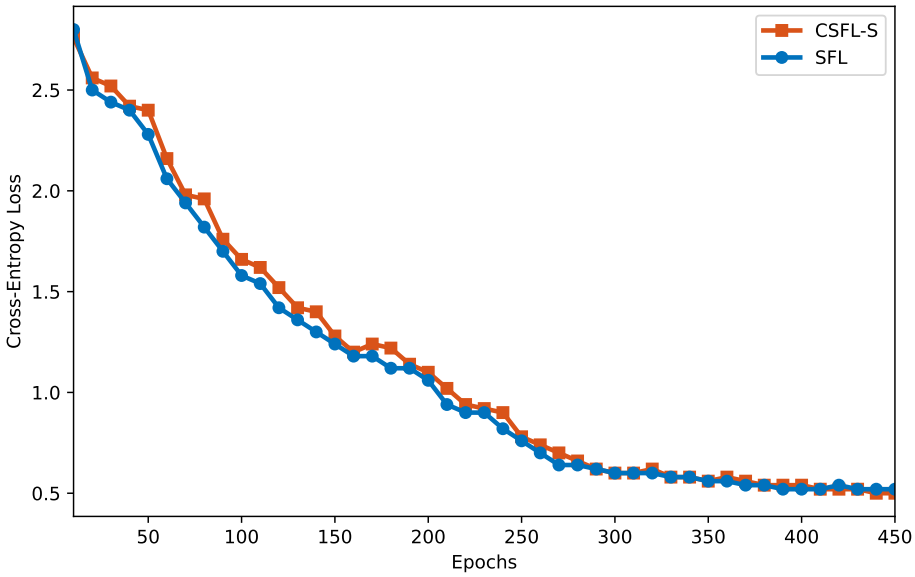

Figure 5.2: Performance in terms of loss.

Training Latency Dynamics Fig. 5.3 illustrates the training latency evolution across epochs under different pairing strategies. CSFL-S consistently maintains the lowest latency throughout the entire training process. This behavior is attributed to its preference-aware stable matching mechanism, which yields efficient and balanced collaboration patterns. In contrast, the greedy variant CSFL-G often leads to suboptimal global performance due to its sequential decision-making and locally optimal pairings. CSFL-R exhibits substantial fluctuations, reflecting its randomized nature. The baseline SFL, lacking collaborative capabilities, suffers from consistently high latency dominated by the slowest BD.

Energy Consumption Analysis Energy usage by EDs and BDs under different pairing configurations is presented in Figs. 5.4 and 5.5. CSFL-S yields the most energy-efficient performance across both device types. This improvement stems from its mutual preference matching, which inherently avoids high-energy communication and computation patterns. In comparison, CSFL-G often creates unbalanced pairings that lead to energy inefficiencies, while CSFL-R exhibits unstable energy profiles due to randomness in pairing.

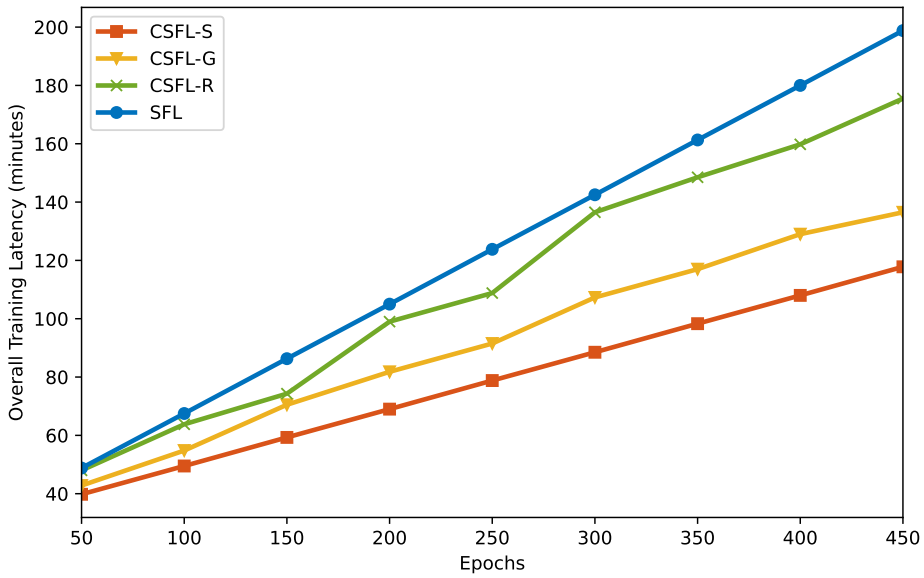

Figure 5.3: The training delay associated with epochs.

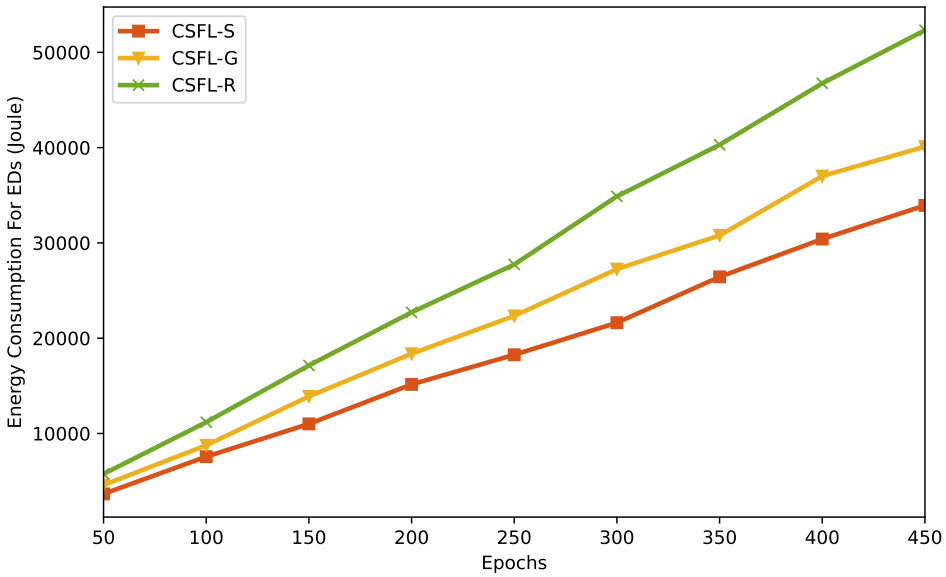

Figure 5.4: Assistance workload and transmission overhead for EDs.

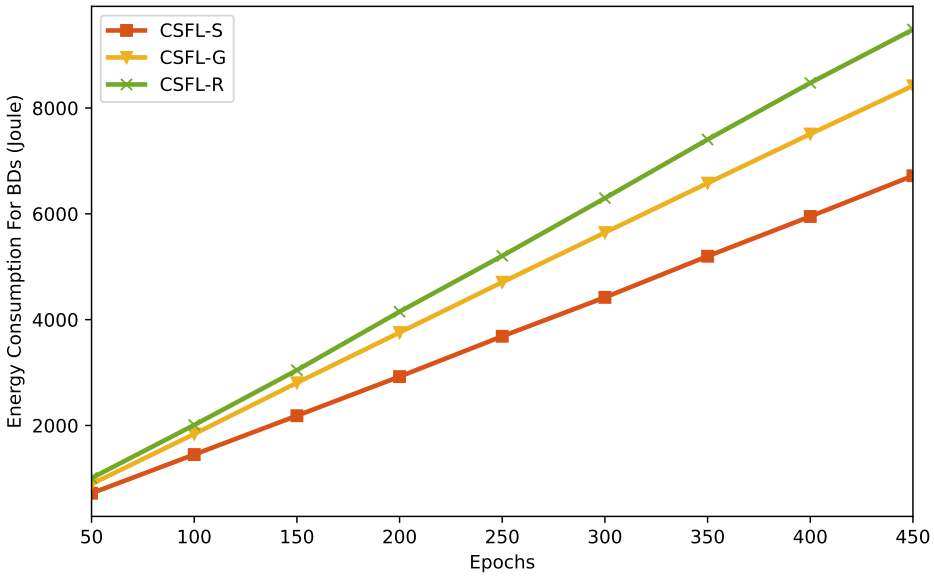

Figure 5.5: Transmission overhead for BDs.

Accuracy–Latency Trade-off Fig. 5.6 compares the relation between accuracy and cumulative training time across methods. As each paradigm exhibits a distinct per-round latency, the training time varies even when the number of epochs is fixed. CSFL-S achieves a favorable trade-off, attaining target accuracy at a substantially lower latency. Specifically, it reduces the total training time by approximately 40.7% relative to SFL, while preserving convergence accuracy. This confirms the efficiency benefits of incorporating intelligent collaboration into the learning process.

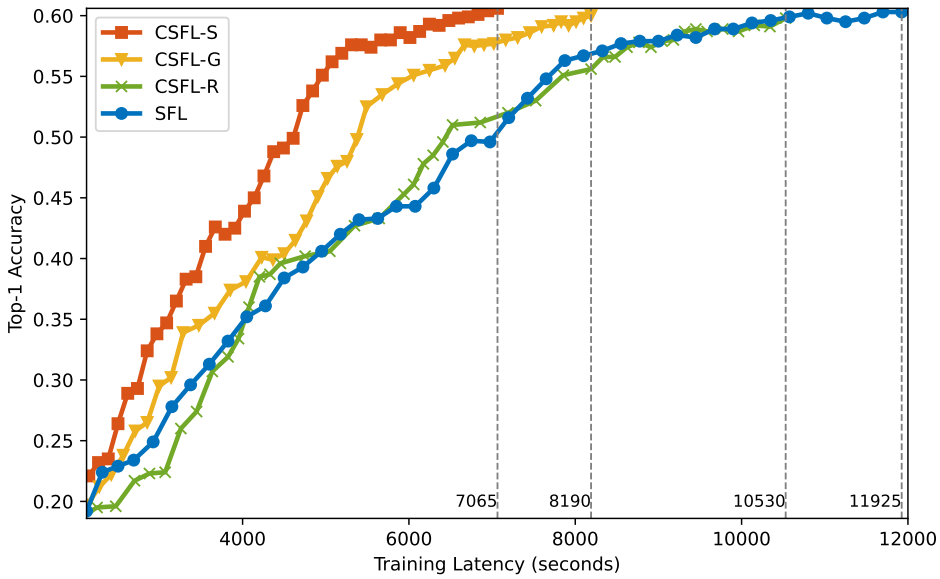

Figure 5.6: Temporal convergence of model accuracy during training.

Collaboration Fairness Fig. 5.7 reports the fairness index of workload distribution among EDs, where a higher value indicates more balanced participation. CSFL-S achieves a fairness index of 0.81, reflecting well-distributed assistance workloads and demonstrating the strength of stable matching in avoiding workload concentration. In contrast, CSFL-G attains only 0.542, as early-matching BDs monopolize stronger EDs, leaving later ones with suboptimal choices.

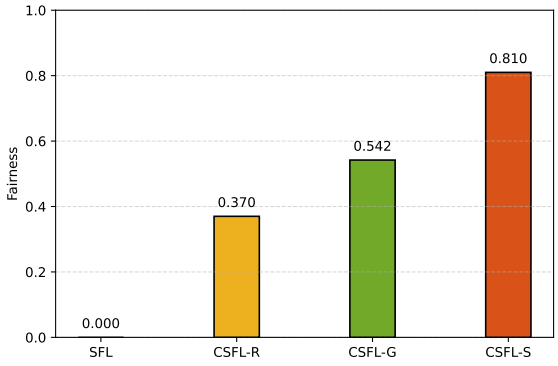

Figure 5.7: Fairness index of assistance workload distribution.

Scalability with Device Count The impact of device count on system performance is shown in Fig. 5.8. As the number of devices increases, both training latency and convergence rounds grow. Nevertheless, this expansion also improves model generalization. For instance, increasing the number of devices from 8 to 20 reduces training accuracy gaps by nearly 10%, while only marginally increasing the training time. These results validate the scalability of CSFL-S and its capability to maintain favorable accuracy–efficiency trade-offs in large-scale deployments.

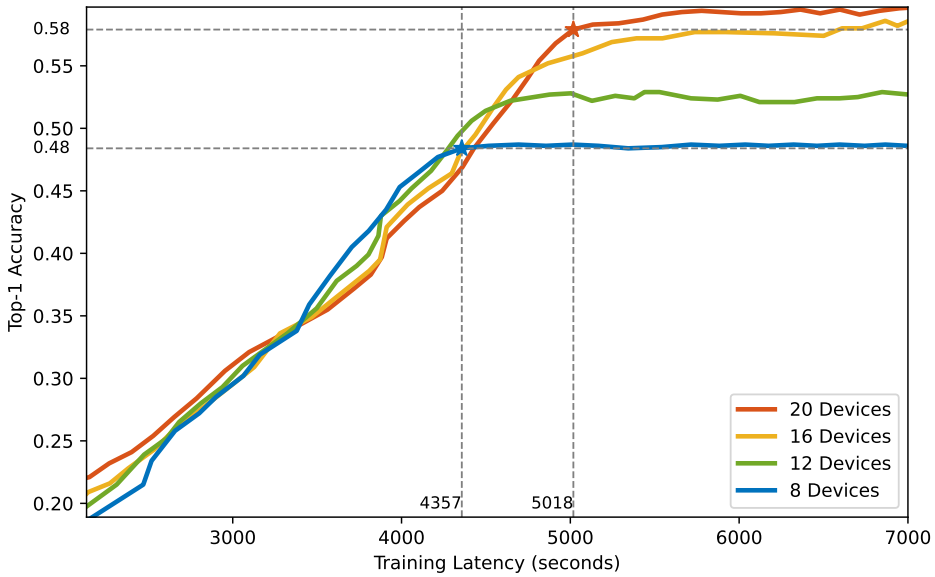

Figure 5.8: Impact of device quantity on training latency for model accuracy.

Chapter 6

Conclusion and Future Work

This thesis presents a *Collaborative Split Federated Learning* (CSFL) framework designed to enhance training efficiency in heterogeneous edge computing environments. At its core, CSFL jointly optimizes the cut-layer selection and the pairing strategy between devices, aiming to reduce synchronization delays caused by disparities in computing and communication capabilities. To address the challenges posed by the interdependence between these decisions, we propose a principled reformulation that decomposes the joint problem into tractable subproblems. These are solved via alternating optimization, leveraging both continuous relaxation and game-theoretic methods to ensure efficient coordination among devices. The proposed framework leads to a more balanced workload distribution, improves resource utilization, and enhances overall training efficiency.

Through comprehensive experiments, CSFL is shown to match the performance of traditional SFL frameworks while significantly improving training speed and collaboration quality. In particular, it consistently outperforms baseline methods, such as greedy and random pairing, by aligning local computations with system-wide objectives and achieving stable convergence across diverse system conditions.

Future work may extend this study along several practical and impactful directions. First, although this work assumes that the collaborative training process is protected by secure execution environments such as Trusted Execution Environments (TEEs), the actual deployment of such security measures in edge networks remains challenging. Many edge devices, especially low-cost or legacy hardware, may not support TEEs or may suffer from limited resources that make them

impractical to use. Even when available, using TEEs can lead to additional latency and memory consumption. Future studies can explore more lightweight and flexible privacy-preserving methods that better fit the constraints of real-world edge scenarios. In addition, emerging techniques such as semantic communication [80], may help reduce privacy risks by avoiding the transmission of redundant or sensitive intermediate representations. Further investigation is needed to understand how these privacy strategies impact the performance and training dynamics of CSFL systems.

Second, the current design limits collaboration to a one-to-one pairing between efficient devices (EDs) and bottleneck devices (BDs). While this simplifies system coordination and analysis, it may not fully utilize available resources in more complex scenarios. For instance, in some cases, a single ED may be capable of helping multiple BDs without significant overhead, while in other situations, a single BD might require assistance from more than one ED to meet latency constraints. Future work can therefore explore more general collaboration strategies, such as one-to-many or many-to-one structures, to enhance flexibility and improve resource utilization. These more complex pairing schemes may also require lightweight coordination protocols to avoid conflict or redundant assignments. Simulation-based evaluations can help determine under what conditions these generalized schemes outperform one-to-one pairing.

Third, while the current pairing strategy is solved via matching theory and enumerates stable outcomes through rotation elimination, this approach can still incur significant computational overhead, particularly when the number of participating devices grows. Although it ensures solution stability and fairness, its scalability may become a limiting factor in large-scale or real-time applications. Future work may explore more efficient approximations or learning-based heuristics that can offer near-optimal pairing decisions with substantially reduced complexity.

Finally, while this work includes comparative evaluations against several representative baselines, future studies may expand the benchmark scope to include more diverse deployment scenarios and learning models. For instance, integrating large language models, multimodal architectures, or real-time inference pipelines could expose new bottlenecks and performance trade-offs. Such extensions would further validate the generality and robustness of the proposed CSFL framework across a wider range of applications.

Appendix A

Proof of Convergence

The convergence of the alternating optimization algorithm employed to solve Problem P1.1 can be justified by analyzing the behavior of the decision variables $(\mathcal{P}_e^{(\tau)}, J_c^{(\tau)})$ over successive iterations τ. Let $\xi(\mathcal{P}_e, J_c)$ denote the composite objective function that integrates the expected training latency and a fairness-aware penalty term. The algorithm proceeds by iteratively minimizing ξ with respect to each variable while keeping the other fixed.

Let $\mathcal{M}$ represent the finite set of all feasible pairing configurations and $[2, J]$ be the bounded domain for the cut layer variable. Since $(\mathcal{P}_e, J_c)$ belongs to the Cartesian product $\mathcal{M} \times [2, J]$, the domain of the optimization is compact. In each iteration, the algorithm first selects $\mathcal{P}_e^{(\tau+1)}$ by minimizing ξ while fixing $J_c^{(\tau)}$, i.e.,

$$\xi(\mathcal{P}_e^{(\tau+1)}, J_c^{(\tau)}) \leq \xi(\mathcal{P}_e^{(\tau)}, J_c^{(\tau)}).$$

Subsequently, with the updated pairing $\mathcal{P}_e^{(\tau+1)}$ fixed, the algorithm updates the cut layer via

$$\xi(\mathcal{P}_e^{(\tau+1)}, J_c^{(\tau+1)}) \leq \xi(\mathcal{P}_e^{(\tau+1)}, J_c^{(\tau)}).$$

Together, these steps imply that the objective value does not increase across iterations:

$$\xi(\mathcal{P}_e^{(\tau+1)}, J_c^{(\tau+1)}) \leq \xi(\mathcal{P}_e^{(\tau)}, J_c^{(\tau)}).$$

The non-increasing property of the objective function is further supported by the fact that both components of ξ are bounded below. The expected latency term is inherently non-negative, and the penalty term involving $\hat{\eta}(J_c)$ is constrained by design, ensuring that ξ is lower bounded. That is, there exists a constant $\underline{F} \in \mathbb{R}$ such that

$$\xi(\mathcal{P}_e, J_c) \geq \underline{F}, \quad \forall\, (\mathcal{P}_e, J_c) \in \mathcal{M} \times [2, J].$$

Since the algorithm generates a monotonic and bounded sequence of objective values, convergence in function value is guaranteed. Moreover, because $\mathcal{M}$ is finite and the feasible set for J_c is compact, the sequence of variable updates admits accumulation points. Consequently, the algorithm converges to a stationary solution of Problem P1.1.

Cut Layer Optimization Convergence Fig. A.1 illustrates the iterative convergence behavior of the cut-layer optimization process. The algorithm reliably converges within seven iterations, ultimately selecting the seventh layer (*Pool3*) as the optimal partition point. This behavior confirms the effectiveness of the gradient-guided refinement strategy embedded in our alternating optimization.

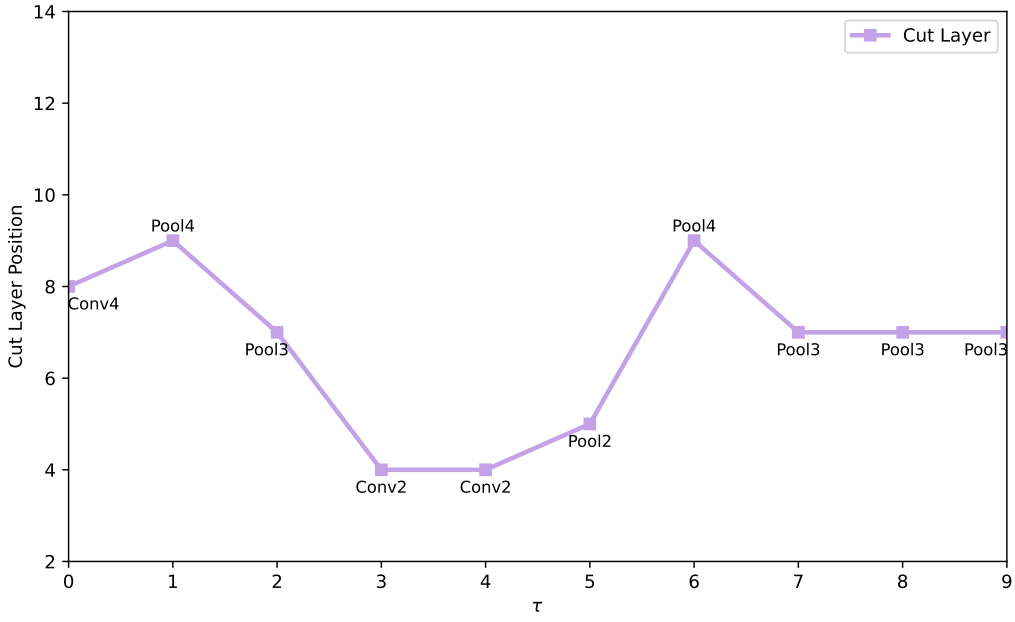

Figure A.1: Optimized Cut Layer Positions versus τ.

References

[1] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 63–71, 2018.

[2] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Toward an intelligent edge: Wireless communication meets machine learning," *IEEE Communications Magazine*, vol. 58, no. 1, pp. 19–25, 2020.

[3] H. Yang, K. Zheng, K. Zhang, J. Mei, and Y. Qian, "Ultra-reliable and low-latency communications for connected vehicles: Challenges and solutions," *IEEE Network*, vol. 34, no. 3, pp. 92–100, 2020.

[4] W. Chen, "Intelligent manufacturing production line data monitoring system for industrial internet of things," *Computer Communications*, vol. 151, pp. 31–41, 2020.

[5] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A survey on federated learning systems: Vision, hype and reality for data privacy and protection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3347–3366, 2021.

[6] 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*, vol. 54, pp. 1273–1282, 2017.

[7] N. H. Tran, W. Bao, A. Zomaya, M. N. Nguyen, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *IEEE INFOCOM 2019-IEEE conference on computer communications*, pp. 1387–1395, IEEE, 2019.

[8] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.

[9] Z. Lin, G. Qu, X. Chen, and K. Huang, "Split learning in 6g edge networks," *IEEE Wireless Communications*, vol. 31, no. 4, pp. 170–176, 2024.

[10] A. Singh, P. Vepakomma, O. Gupta, and R. Raskar, "Detailed comparison of communication efficiency of split learning and federated learning," *arXiv preprint arXiv:1909.09145*, 2019.

[11] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 36, pp. 8485–8493, 2022.

[12] Y. Gao, M. Kim, C. Thapa, A. Abuadbba, Z. Zhang, S. Camtepe, H. Kim, and S. Nepal, "Evaluation and optimization of distributed machine learning techniques for internet of things," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2538–2552, 2021.

[13] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et al.*, "Advances and open problems in federated learning," *Foundations and trends® in machine learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[14] 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.

[15] M. Ye, X. Fang, B. Du, P. C. Yuen, and D. Tao, "Heterogeneous federated learning: State-of-the-art and research challenges," *ACM Computing Surveys*, vol. 56, no. 3, pp. 1–44, 2023.

[16] X. Huang, P. Li, H. Du, J. Kang, D. Niyato, D. I. Kim, and Y. Wu, "Federated learning-empowered ai-generated content in wireless networks," *IEEE Network*, vol. 38, no. 5, pp. 304–313, 2024.

[17] H. Xing, O. Simeone, and S. Bi, "Decentralized federated learning via sgd over wireless d2d networks," in *2020 IEEE 21st international workshop on signal processing advances in wireless communications (SPAWC)*, pp. 1–5, 2020.

[18] Y. Shi, J. Nie, X. Li, and H. Li, "Heterogeneity-aware device selection for efficient federated edge learning," *International Journal of Intelligent Networks*, vol. 5, pp. 293–301, 2024.

[19] 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)*, pp. 1–7, 2019.

[20] S. D. Okegbile, H. Gao, O. Talabi, J. Cai, C. Yi, D. Niyato, and X. Shen, "Fles: A federated learning-enhanced semantic communication framework for mobile aigc-driven human digital twins," *IEEE Network*, pp. 1–1, 2025.

[21] O. Wehbi, S. Arisdakessian, O. A. Wahab, H. Otrok, S. Otoum, A. Mourad, and M. Guizani, "Fedmint: Intelligent bilateral client selection in federated learning with newcomer iot devices," *IEEE Internet of Things Journal*, vol. 10, no. 23, pp. 20884–20898, 2023.

[22] D. Stripelis and J. L. Ambite, "Semi-synchronous federated learning," *arXiv preprint arXiv:2102.02849*, 2021.

[23] C. You, D. Feng, K. Guo, H. H. Yang, C. Feng, and T. Q. Quek, "Semi-synchronous personalized federated learning over mobile edge networks," *IEEE Transactions on Wireless Communications*, vol. 22, no. 4, pp. 2262–2277, 2022.

[24] M. G. Herabad, "Communication-efficient semi-synchronous hierarchical federated learning with balanced training in heterogeneous iot edge environments," *Internet of things*, vol. 21, p. 100642, 2023.

[25] Z. Qu, S. Guo, H. Wang, B. Ye, Y. Wang, A. Y. Zomaya, and B. Tang, "Partial synchronization to accelerate federated learning over relay-assisted edge networks," *IEEE Transactions on Mobile Computing*, vol. 21, no. 12, pp. 4502–4516, 2022.

[26] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

[27] Q. Duan, S. Hu, R. Deng, and Z. Lu, "Combined federated and split learning in edge computing for ubiquitous intelligence in internet of things: State-of-the-art and future directions," *Sensors*, vol. 22, no. 16, p. 5983, 2022.

[28] C. Xu, Y. Qu, Y. Xiang, and L. Gao, "Asynchronous federated learning on heterogeneous devices: A survey," *Computer Science Review*, vol. 50, p. 100595, 2023.

[29] W. Wu, M. Li, K. Qu, C. Zhou, X. Shen, W. Zhuang, X. Li, and W. Shi, "Split learning over wireless networks: Parallel design and resource management," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 4, pp. 1051–1066, 2023.

[30] P. Joshi, C. Thapa, S. Camtepe, M. Hasanuzzamana, T. Scully, and H. Afli, "Splitfed learning without client-side synchronization: Analyzing client-side split network portion size to overall performance," *arXiv preprint arXiv:2109.09246*, 2021.

[31] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.

[32] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with layer-wise asynchronous model update and temporally weighted aggregation," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 10, pp. 4229–4238, 2019.

[33] P. Dong, J. Ge, X. Wang, and S. Guo, "Collaborative edge computing for social internet of things: Applications, solutions, and challenges," *IEEE Transactions on Computational Social Systems*, vol. 9, no. 1, pp. 291–301, 2021.

[34] R. Lin, T. Xie, S. Luo, X. Zhang, Y. Xiao, B. Moran, and M. Zukerman, "Energy-efficient computation offloading in collaborative edge computing," *IEEE Internet of Things Journal*, vol. 9, no. 21, pp. 21305–21322, 2022.

[35] Y. Hao, Y. Jiang, T. Chen, D. Cao, and M. Chen, "itaskoffloading: Intelligent task offloading for a cloud-edge collaborative system," *IEEE Network*, vol. 33, no. 5, pp. 82–88, 2019.

[36] J. Tang, J. Li, X. Chen, K. Xue, L. Zhang, Q. Sun, and J. Lu, "Cooperative caching in satellite-terrestrial integrated networks: A region features aware approach," *IEEE Transactions on Vehicular Technology*, vol. 73, no. 7, pp. 10602–10616, 2024.

[37] H. Fan, C. Sun, J. Long, L. Li, Y. Huo, and S. Wang, "Graph-driven resource allocation strategies in satellite iot: A cooperative game-theoretic approach," *IEEE Internet of Things Journal*, vol. 12, no. 4, pp. 3463–3481, 2025.

[38] Y. Gu, W. Saad, M. Bennis, M. Debbah, and Z. Han, "Matching theory for future wireless networks: Fundamentals and applications," *IEEE Communications Magazine*, vol. 53, no. 5, pp. 52–59, 2015.

[39] C. Park, D. Hong, and C. Seo, "An attack-based evaluation method for differentially private learning against model inversion attack," *IEEE Access*, vol. 7, pp. 124988–124999, 2019.

[40] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/Ispa*, vol. 1, pp. 57–64, 2015.

[41] M. A. Mukhtar, M. K. Bhatti, and G. Gogniat, "Architectures for security: A comparative analysis of hardware security features in intel sgx and arm trustzone," in *2019 2nd International Conference on Communication, computing and Digital systems (C-CODE)*, pp. 299–304, 2019.

[42] S. Pinto and N. Santos, "Demystifying arm trustzone: A comprehensive survey," *ACM computing surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[43] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis, "Ppfl: Privacy-preserving federated learning with trusted execution environments," in *Proceedings of the 19th annual international conference on mobile systems, applications, and services*, pp. 94–108, 2021.

[44] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4961–4973, 2021.

[45] Y. Pan, Z. Chao, W. He, Y. Jing, L. Hongjia, and W. Liming, "Fedshe: privacy preserving and efficient federated learning with adaptive segmented ckks homomorphic encryption," *Cybersecurity*, vol. 7, no. 1, p. 40, 2024.

[46] B. Jia, X. Zhang, J. Liu, Y. Zhang, K. Huang, and Y. Liang, "Blockchain-enabled federated learning data protection aggregation scheme with differential privacy and homomorphic encryption in iiot," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 6, pp. 4049–4058, 2021.

[47] S. D. Okegbile, J. Cai, H. Zheng, J. Chen, and C. Yi, "Differentially private federated multitask learning framework for enhancing human-to-virtual connectivity in human digital twin," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 11, pp. 3533–3547, 2023.

[48] X. Li, N. Wang, L. Zhu, S. Yuan, and Z. Guan, "Fuse: a federated learning and u-shape split learning-based electricity theft detection framework," *Science China Information Sciences*, vol. 67, no. 4, p. 149302, 2024.

[49] Z. Zhao, D. Liu, Y. Cao, T. Chen, S. Zhang, and H. Tang, "U-shaped split federated learning: An efficient cross-device learning framework with enhanced privacy-preserving," in *2023 9th International Conference on Computer and Communications (ICCC)*, pp. 2182–2186, 2023.

[50] C. Xu, J. Li, Y. Liu, Y. Ling, and M. Wen, "Accelerating split federated learning over wireless communication networks," *IEEE Transactions on Wireless Communications*, vol. 23, no. 6, pp. 5587–5599, 2023.

[51] J. C. Bezdek and R. J. Hathaway, "Convergence of alternating optimization," *Neural, Parallel & Scientific Computations*, vol. 11, no. 4, pp. 351–368, 2003.

[52] Z. Jia, M. Sheng, J. Li, D. Zhou, and Z. Han, "Joint hap access and leo satellite backhaul in 6g: Matching game-based approaches," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 4, pp. 1147–1159, 2020.

[53] D. Chen, C. S. Hong, L. Wang, Y. Zha, Y. Zhang, X. Liu, and Z. Han, "Matching-theory-based low-latency scheme for multitask federated learning in mec networks," *IEEE Internet of Things Journal*, vol. 8, no. 14, pp. 11415–11426, 2021.

[54] Y. Tian, Y. Zhang, and H. Zhang, "Recent advances in stochastic gradient descent in deep learning," *Mathematics*, vol. 11, no. 3, p. 682, 2023.

[55] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[56] L. Zhang and J. Xu, "Learning the optimal partition for collaborative dnn training with privacy requirements," *IEEE Internet of Things Journal*, vol. 9, no. 13, pp. 11168–11178, 2021.

[57] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[58] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[59] 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, 2002.

[60] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[61] Q. Zeng, Y. Du, K. Huang, and K. K. Leung, "Energy-efficient resource management for federated edge learning with cpu-gpu heterogeneous computing," *IEEE Transactions on Wireless Communications*, vol. 20, no. 12, pp. 7947–7962, 2021.

[62] Z. Lin, G. Zhu, Y. Deng, X. Chen, Y. Gao, K. Huang, and Y. Fang, "Efficient parallel split learning over resource-constrained wireless edge networks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 10, pp. 9224–9239, 2024.

[63] J. Park, S. Samarakoon, M. Bennis, and M. Debbah, "Wireless network intelligence at the edge," *Proceedings of the IEEE*, vol. 107, no. 11, pp. 2204–2239, 2019.

[64] S. D. Okegbile, H. Gao, O. Talabi, J. Cai, D. Niyato, and X. Shen, "Optimizing federated semantic learning in distributed aigc-enabled human digital twins: A multi-criteria and multi-shard user selection framework," *IEEE Transactions on Mobile Computing*, vol. 24, no. 7, pp. 5916–5933, 2025.

[65] C. Yi, J. Cai, K. Zhu, and R. Wang, "A queueing game based management framework for fog computing with strategic computing speed control," *IEEE Transactions on Mobile Computing*, vol. 21, no. 5, pp. 1537–1551, 2020.

[66] A. Yekanlou, J. Cai, and S. D. Okegbile, "Maximizing efficiency: Relocation and deduplication for result caching in distributed and collaborative edge computing networks," in *2024 IEEE 100th Vehicular Technology Conference (VTC2024-Fall)*, pp. 1–6, 2024.

[67] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[68] D. Waagen, K. Rainey, J. Gantert, D. Gray, M. King, M. S. Thompson, J. Barton, W. Waldron, S. Livingston, and D. Hulsey, "Characterizing inter-layer functional mappings of deep learning models," *arXiv preprint arXiv:1907.04223*, 2019.

[69] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013.

[70] S. Crisci, F. Porta, V. Ruggiero, and L. Zanni, "Hybrid limited memory gradient projection methods for box-constrained optimization problems," *Computational Optimization and Applications*, vol. 84, no. 1, pp. 151–189, 2023.

[71] C.-P. Teo, J. Sethuraman, and W.-P. Tan, "Gale-shapley stable marriage problem revisited: Strategic issues and applications," *Management Science*, vol. 47, no. 9, pp. 1252–1267, 2001.

[72] A. E. Roth and M. Sotomayor, "Two-sided matching," *Handbook of game theory with economic applications*, vol. 1, pp. 485–541, 1992.

[73] D. Gusfield and R. W. Irving, *The stable marriage problem: structure and algorithms*. MIT press, 1989.

[74] P. Eirinakis, D. Magos, I. Mourtos, and P. Miliotis, "Finding all stable pairs and solutions to the many-to-many stable matching problem," *INFORMS Journal on Computing*, vol. 24, no. 2, pp. 245–259, 2012.

[75] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[76] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[77] M. K. Simon and M.-S. Alouini, *Digital communication over fading channels*. John Wiley & Sons, 2004.

[78] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[79] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[80] H. Wei, W. Ni, W. Xu, F. Wang, D. Niyato, and P. Zhang, "Federated semantic learning driven by information bottleneck for task-oriented communications," *IEEE Communications Letters*, vol. 27, no. 10, pp. 2652–2656, 2023.