Split Fine-Tuning for Large Language Models in Wireless Networks

Songge Zhang, *Graduate Student Member, IEEE*, Guoliang Cheng, *Graduate Student Member, IEEE*, Wen Wu, *Senior Member, IEEE*, Xinyu Huang, *Graduate Student Member, IEEE*, Lingyang Song, *Fellow, IEEE*, and Xuemin Shen, *Fellow, IEEE*

***Abstract*—Fine-tuning is the process of adapting the pre-trained large language models (LLMs) for downstream tasks. Due to substantial model parameters, fine-tuning LLM on mobile devices demands considerable memory resources, and suffers from high communication overhead and fine-tuning delay. In this paper, we propose an efficient LLM fine-tuning scheme in wireless networks, named Split Fine-Tuning (SFT), which can accommodate LLM fine-tuning on mobile devices. Specifically, an LLM is split into a server-side part on the edge server and a device-side part on the mobile device to satisfy the device-side memory constraint. All devices share a server-side model and perform parallel fine-tuning to reduce fine-tuning delay. In addition, to reduce communication overhead incurred by data exchange between devices and the edge server, we propose an activation data compression scheme by jointly leveraging sparsification, stochastic quantization, and lossless encoding methods. Furthermore, we formulate a fine-tuning delay minimization problem under model accuracy and device-side memory constraints, taking device heterogeneity and channel dynamics into account. To solve the problem, the nonlinear mixed-integer problem is decoupled into two subproblems in different timescales. A two-timescale resource management algorithm is proposed to jointly optimize the compression rate and transformer block allocation in the large timescale using the augmented Lagrangian method, and determine spectrum resource allocation in the small timescale via sequential quadratic programming. Extensive simulation results demonstrate that the proposed scheme can reduce the fine-tuning delay by up to 66.4% and communication overhead by 93.6% compared to state-of-the-art benchmarks, while satisfying device-side memory and model accuracy constraints.**

***Index Terms*—Large language models, fine-tuning, split learning, resource management.**

Received 31 December 2024; revised 19 May 2025; accepted 4 June 2025. Date of publication 19 June 2025; date of current version 31 December 2025. This work was supported in part by the Peng Cheng Laboratory Major Key Project under Grant PCL2023AS1-5, Grant PCL2024A01, and Grant 2025QYA002, in part by the Natural Science Foundation of China under Grant 62201311, in part by the GuangDong Basic and Applied Basic Research Foundation under Grant 2023B0303000019, and in part by Shenzhen Science and Technology Program under Grant JCYJ20241202125910015. The guest editor coordinating the review of this article and approving it for publication was Prof. Zhu Han. *(Corresponding author: Wen Wu.)*

Songge Zhang is with the School of Electronic and Computer Engineering, Peking University Shenzhen Graduate School, Shenzhen 518055, China, and also with the Frontier Research Center, Pengcheng Laboratory, Shenzhen 518055, China (e-mail: zhangsongge@stu.pku.edu.cn).

Guoliang Cheng and Wen Wu are with Frontier Research Center, Pengcheng Laboratory, Shenzhen 518055, China (e-mail: chenggl@pcl.ac.cn; wuw02@pcl.ac.cn).

Xinyu Huang and Xuemin Shen are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo N2L 3G1, Canada (e-mail: x357huan@uwaterloo.ca; sshen@uwaterloo.ca).

Lingyang Song is with the State Key Laboratory of Photonics and Communications, School of Electronics, Peking University, Beijing 100871, China, also with the School of Electronic and Computer Engineering, Peking University Shenzhen Graduate School, Shenzhen 518055, China, and also with Hunan Institute of Advanced Sensing and Information Technology, Xiangtan University, Xiangtan 411105, China (e-mail: lingyang.song@pku.edu.cn).

Digital Object Identifier 10.1109/JSTSP.2025.3581484

I. Introduction

RECENT advancements in large language models (LLMs), such as ChatGPT, LLaMA, and Vision Transformer [1], [2], [3], have sparked a new wave in artificial intelligence (AI) by showcasing impressive capabilities such as improved generalization and inference. These advancements have facilitated extensive applications across various fields, such as chatbots, search engines, writing assistants, and multimodal systems [4], illuminating the vision of artificial general intelligence [5], [6]. Pushing LLMs from the remote cloud server to the nearby network edge enables timely task response for mobile devices.

LLM fine-tuning is a key technique for LLMs to support personalized services and downstream tasks [7], which adjusts LLM parameters using local data for specific tasks while retaining pre-trained performance. To reduce the cost of full fine-tuning, fine-tuning can be efficiently performed through parameter-efficient fine-tuning (PEFT). The PEFT technique updates only a small subset of parameters or additional modules while keeping the rest of the model frozen, thereby reducing graphics processing unit (GPU) memory usage during training [8]. LLMs are often fine-tuned using data collected from mobile devices, such as behavioral records, location information, and multimedia content, which are often privacy-sensitive. Due to data privacy leakage concerns, mobile device data should not be uploaded to the centralized cloud [9]. Hence, fine-tuning should be performed in a distributed manner via the collaboration among multiple devices without sharing raw local data [10].

In the literature, LLM fine-tuning explores a distributed paradigm with PEFT techniques such as low-rank adaptation (LoRA) [11]. LoRA can modify a small portion of the LLM parameters instead of the entire LLM in the fine-tuning process. Although distributed schemes with PEFT are highly parameter-efficient, they still demand substantial memory to deploy the entire LLM, exceeding the memory capacity of typical mobile devices [12], [13]. As shown in Table I, typical NVIDIA Jetson mobile devices cannot meet the memory requirements for fine-tuning LLMs like GPT-3 and LLaMA [2]. To address this issue,

TABLE I
COMPARISON OF MOBILE DEVICE RESOURCES AND TYPICAL LLM DEPLOYMENT REQUIREMENTS

Mobile Devices		Typical LLMs		
Device	Memory	Model	#Params	Memory
Raspberry Pi-4B	4 GB	LLaMA-7B	7B	28 GB
Jetson Nano	8 GB	LLaMA-65B	65B	260 GB
Jetson TX2	8 GB	GPT-3	175B	700 GB
Jetson Xavier NX	16 GB	PaLM	540B	2.15 TB

split learning (SL) is a potential solution, which can divide the LLM into server-side and device-side parts, offloading most of the LLM to the server, thereby satisfying the device's memory constraints [14].

However, designing an efficient SL scheme for LLMs fine-tuning presents several challenges. Firstly, in the vanilla SL, all devices interact with the server in a sequential manner to complete the fine-tuning process, which results in a long fine-tuning delay. Secondly, the SL scheme needs to periodically exchange intermediate activations and activation gradients between the server and devices, resulting in significant communication overhead. The amount of activation and its gradient data size are typically large in the LLM. Taking ViT-16 and a mobile device with 5,000 data samples and 196 tokens as an example, the total immediate activation data volume at the cut layer can be approximately 2.81 GB in each fine-tuning epoch [15]. Thirdly, heterogeneous devices' computational capabilities and dynamic channel conditions result in varying delay for each device, i.e., the straggler effect, which further increases the fine-tuning delay.

To address the above challenges, we first propose a Split Fine-Tuning (SFT) scheme that satisfies mobile devices' memory constraints by splitting the LLM into device and server parts. All devices collaboratively fine-tune LLM with the server using the devices' local data and parallelly update local device-side LoRAs, which are aggregated in each epoch. The fine-tuning process among devices is conducted in parallel, and the server executes the subsequent fine-tuning process with only one server-side model. Once all devices complete device-side fine-tuning, each device uploads the updated device-side LoRAs for LoRA aggregation. The proposed scheme can efficiently reduce device-side memory consumption and computation workload by splitting the LLM. In addition, we design a novel compression scheme, which includes Top-K sparsification, stochastic quantization, and lossless encoding. The data compression scheme reduces the data volume while preserving model accuracy. Moreover, we analyze the fine-tuning delay performance, communication, computation overhead, and the memory consumption of the proposed scheme.

Furthermore, we propose a resource management algorithm to reduce LLM fine-tuning delay in wireless networks. We formulate a fine-tuning delay minimization problem under constraints of device-side memory and model accuracy, which is a nonlinear mixed-integer programming problem. To solve the problem, the problem is decomposed into two subproblems by distinguishing variables in different timescales, i.e., a large-timescale subproblem for determining compression rates and LLM transformer block allocation, and a small-timescale subproblem for determining the bandwidth allocation among devices. Specifically, the augmented Lagrangian method is adopted to iteratively solve the former subproblem, addressing the non-convexity of the constraints within the subproblem. A sequential quadratic programming (SQP) method is employed to iteratively approximate the latter nonlinear subproblem by iteratively constructing and solving a series of quadratic programming (QP) subproblems. Extensive simulation results demonstrate that the proposed scheme can reduce the overall fine-tuning delay by up to 66.4% and communication overhead by up to 93.6% compared to the state-of-the-art scheme. The main contributions of this paper are summarized as follows:

1) We propose an SFT scheme for LLM fine-tuning in wireless networks, enabling collaborative and parallel fine-tuning across multiple devices;
2) We propose a novel compression scheme that significantly reduces communication overhead;
3) We analyze the performance of the SFT scheme, including fine-tuning delay, memory consumption, communication overhead, and computational workload;
4) We propose a resource management algorithm to reduce fine-tuning delay.

The remainder of this paper is organized as follows. Section II reviews the related works. Section III presents the system model. Section IV presents the proposed scheme, followed by the performance analysis in Section V. Sections VI and VII detail the problem formulation and the corresponding solution, respectively. Section VIII provides the simulation results, and finally Section IX concludes the paper.

II. RELATED WORK

A number of studies aim to improve LLM fine-tuning performance from different perspectives. Adapters achieve PEFT by introducing lightweight trainable modules between existing layers of the model while freezing the rest [16]. Prefix-tuning was proposed to optimize a small set of prefix vectors prepended to the input at each layer, leaving the model's original parameters unchanged [17]. The LoRA can reduce the parameter update size by injecting low-rank trainable matrices into the attention mechanism while keeping the majority of the model frozen [18]. To achieve collaborative training over wireless networks, a very recent work leverages FL to allow multiple users to collaboratively fine-tune LLMs without sharing devices' data [19]. For example, the entire model was deployed on local devices for training, with model parameters or gradients aggregated at each round [20]. Some studies explore LLM fine-tuning via leveraging FL and PEFT techniques. A PromptFL framework was proposed to enable federated participants to train shared prompts instead of the entire model [21]. Jiang et al. introduced a soft label-enhanced federated fine-tuning approach that incorporates LoRA to reduce computational and communication costs [22]. Cai et al. proposed FedAdapter, which progressively adjusts adapter configurations to identify the most efficient settings, accelerating model convergence [1]. Additionally, some studies enable LLM fine-tuning in FL using techniques like distillation, pruning, and quantization. Liu et al. introduced an adaptive quantization scheme with ensemble distillation, which compresses a large model into a smaller one and applies cluster partitioning

for heterogeneous model training [23]. Greidi et al. proposed a sparse training to reduce computation and communication costs in FL [24]. These studies enable collaborative LLM fine-tuning across multiple devices while reducing the computational workload.

In addition to FL, SL is another potential solution for enabling efficient model training. The SL scheme can reduce the computational workload and memory consumption on devices by partitioning the model [25]. To accelerate the training of AI models, Wu et al. introduced a parallel-then-sequential SL strategy to enhance training speed [26]. Xu et al. proposed a synchronized SL (SFL) scheme, allowing multiple server-side and device-side models to train simultaneously, thereby speeding up the SL process [27]. Liao et al. addressed system heterogeneity and accelerated model training by optimizing bandwidth allocation [28]. To more efficiently improve SL, joint optimization of model partitioning and resource management has also been explored. For example, in [29], the optimization of the partitioning layer and bandwidth allocation was proposed to mitigate the straggler effect in wireless SFL. Very recently, a few pioneering works have been devoted to enhancing LLM fine-tuning via SL. Wu et al. split the large model into different components, deployed them across multiple devices, and fine-tuned these components on the device side using a pipeline approach [30]. Chen et al. split the adapter from the large model's backbone and deployed it on devices for fine-tuning [31]. Different from the existing scheme, we design a parallelized split scheme to accelerate the delay of LLM fine-tuning. Additionally, we implement a data compression scheme to reduce the communication overhead between the edge server and mobile devices.

III. System Model

A. Considered Scenario

In this paper, we consider a typical wireless network scenario consisting of multiple mobile devices and a base station equipped with an edge server with powerful computational and memory resources. The set of devices is defined as $\mathcal{N} = \{1, 2, \ldots, N\}$. Each device possesses local data and operates a small portion of the LLM. The data from all devices contributes to fine-tuning tasks such as image classification, natural language processing, etc. The edge server is primarily responsible for the LLMs in a fine-tuning training task and also manages model aggregation at each training round.

In this paper, we consider a transformer-based LLM architecture consisting of L layers. These transformer blocks are split into the server and devices, with the device locally processing l blocks while the remaining $L - l$ blocks are possessed by the server. Forward and backward propagation between the edge server and devices is carried out through the transmission of intermediate activations and activation gradients. The parameter size of each block is related to the dimensions of the multi-head self-attention (MSA) and multi-layer perceptron (MLP).

B. Fine-Tuning Model

We focus on supervised fine-tuning tasks, utilizing the LoRA adapter to reduce the cost and parameter number of fine-tuning. The dataset of each device is denoted as $\mathcal{D}_n = \{\mathbf{x}_j, \mathbf{y}_j\}_{j=1}^{D_n}$, where D_n denotes the total number of samples for device $n \in \mathcal{N}$ and $\sum_{n\in\mathcal{N}} D_n = D$ where D represents the total number of samples across all devices. Here, $\mathbf{x}_j$ is the j-th input sample, and $\mathbf{y}_j$ is its corresponding label. Let $\mathbf{\Theta}$ represent the LLM parameters. The loss function for each sample, denoted as $f_j(\mathbf{\Theta})$, measures the prediction error for the j-th sample. The objective of the training process is to minimize the empirical loss $F(\mathbf{\Theta})$, which is defined as

$$F(\mathbf{\Theta}) = \frac{1}{D}\sum_{j=1}^{D} f(\mathbf{x}_j, \mathbf{y}_j; \mathbf{\Theta}) = \frac{1}{D}\sum_{j=1}^{D} f_j(\mathbf{\Theta}). \tag{1}$$

This process seeks to minimize the average prediction error across all samples and identify the optimal parameter $\mathbf{\Theta}^\star$.

To enhance the efficiency of local model transmission and aggregation, we integrate a small, low-rank, homogeneous adapter into the locally pre-trained model. As depicted in Fig. 1(b), the LoRA adapter is designed to retain the same input dimension d and output dimension h as pre-trained large models, and is applicable to linear, embedding, and convolutional layers. Particularly for linear layers, the adapter reduces the parameter number by decomposing the conventional parameter matrix $\mathbb{R}^{d\times h}$ into two smaller matrices $\mathbf{A} \in \mathbb{R}^{d\times r}$ and $\mathbf{B} \in \mathbb{R}^{r\times h}$, where the rank r is significantly lower than both d and h. During the initialization phase, matrix $\mathbf{A}$ is initialized with a Gaussian distribution with mean zero and variance σ^2, while matrix $\mathbf{B}$ is set to zero [32]. In fine-tuning training, we define $\mathbf{\Theta} \in \mathbb{R}^{d\times m}$ as a trainable weight matrix of the pre-trained model, where the corresponding model update is expressed as $\mathbf{\Theta} + \Delta\mathbf{\Theta} = \mathbf{\Theta} + \mathbf{AB}$. LoRA adapter configuration allows the parameter number to be reduced significantly compared to full-parameter fine-tuning, thereby enhancing computational and communication efficiency. Accordingly, we deploy the LoRA adapter within the considered scheme. Thus, the objective of fine-tuning can be rewritten as

$$F(w) = \frac{1}{D}\sum_{j=1}^{D} f_j(\mathbf{\Theta} + \Delta\mathbf{\Theta}), \tag{2}$$

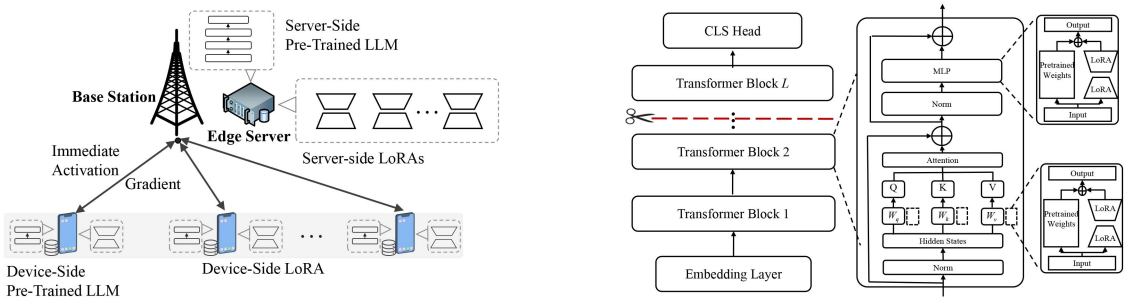
which is to minimize the average prediction error across all samples and identify optimal parameter $\Delta\mathbf{\Theta}^\star$.

IV. SFT Scheme

A. SFT Scheme

To enable collaborative fine-tuning on memory-constrained devices, we propose an innovative split fine-tuning architecture called SFT. Although this work focuses on wireless networks, the proposed SFT scheme is general that can be readily applied to other resource-constrained environments, such as the Internet of Things. The process of this scheme is detailed in Alg. 1 and primarily includes the following steps.

Initialization stage: This stage involves splitting the transformer blocks and distributing them to the devices. In this stage, the pre-trained model, consisting of L transformer layers, is split into two parts: $\mathbf{\Theta}_u$ for the device-side model and $\mathbf{\Theta}_s$ for the server-side model. The split pre-trained model is distributed from the edge server to the device only once



(a) Proposed SFT (b) Transformer Block

Fig. 1. (a) In the SFT frame, devices are trained parallelly with a shared server-side pre-trained model and multiple LoRAs; (b) All Transformer blocks are divided into the server-side and the device-side parts. Each transformer block consists of an MSA and an MLP, both of which are composed of pre-trained model weights and corresponding LoRA.

before fine-tuning, since it remains fixed and does not require updates during fine-tuning. There are a total of $L \times M$ LoRA adapters, where each of the L transformer layers is equipped with M adapters. Each adapter can be indexed by (l, m), where $l \in \{1, \ldots, L\}$ and $m \in \{1, \ldots, M\}$. The device-side model consists of the first l layers, denoted as $\mathcal{L}_u = \{1, \ldots, l\}$, while the server handles the remaining layers $\mathcal{L}_s = \{l+1, \ldots, L\}$. Accordingly, the device-side adapters $\Delta\boldsymbol{\Theta}_u$ comprise $\{\mathbf{A}^{l,m}, \mathbf{B}^{l,m}\}_{l\in\mathcal{L}_u, m\in\{1,\ldots,M\}}$, and the server-side adapters $\Delta\boldsymbol{\Theta}_s$ consist of $\{\mathbf{A}^{l,m}, \mathbf{B}^{l,m}\}_{l\in\mathcal{L}_s, m\in\{1,\ldots,M\}}$. These LoRA adapters are aggregated and updated during each training round and then distributed by the server to the devices at the beginning of the next round.

Forward propagation (FP) in device-side and server-side model: Assume the fine-tuning spans T rounds. In round t and k epoch, each device randomly selects a mini-batch from their local data, denoted by $\mathcal{B}_n(t,k) \subseteq \mathcal{D}_n$, to perform FP. Here, $B = |\mathcal{B}_n(t,k)|$ represents the size of each mini-batch, and D_n is the local dataset of device n. The FP involves passing $\mathcal{B}_n(t,k)$ through $\boldsymbol{\Theta}_u(t,k)$ and $\Delta\boldsymbol{\Theta}_u(t,k)$ to produce the output $\mathbf{y}_n^u(t,k)$. This process can be represented as

$$\mathbf{y}_n^u(t,k) = f(\boldsymbol{\Theta}_u(t,k), \Delta\boldsymbol{\Theta}_u(t,k), \mathcal{B}_n(t,k)). \tag{3}$$

All devices transmit their locally computed outputs to the server. The server then conducts FP based on the server-side $\boldsymbol{\Theta}_s(t,k)$ and n-th LoRA adapter $\Delta\boldsymbol{\Theta}_s(t,k)$. This process can be mathematically represented as

$$\mathbf{y}^s(t,k) = f(\boldsymbol{\Theta}_s(t,k), \Delta\boldsymbol{\Theta}_s(t,k), \mathbf{y}_n^u(t,k)). \tag{4}$$

Since we consider only one pre-trained model on the server side, the server will perform the LoRA FP sequentially according to the order in which the immediate activation is received.

Backward propagation (BP) in device-side and server-side model: BP requires updating the LoRA adapter parameters of the server and device. The purpose is to minimize the global loss function as defined in (2). The edge server calculates gradients for both the server-side and device-side LoRA adapter $\mathbf{g}_n^u(t,k)$ and $\mathbf{g}_n^s(t,k)$ based on the prediction outcomes $\mathbf{y}^s(t,k)$ and labels. Subsequently, the edge server updates server-side LoRA adapters parameters as follows:

$$\Delta\boldsymbol{\Theta}_s^n(t,k) \leftarrow \Delta\boldsymbol{\Theta}_s^n(t-1,k) - \epsilon_s \mathbf{g}_n^s(t,k), \forall n \in \mathcal{N}, \tag{5}$$

where ϵ_s denotes the learning rate for the LoRA adapter on the edge server. For simplicity, assume each transformer block is linked sequentially, with the LoRA adapter updating from the last to the cut $(l+1)$-th layer. When the gradient is updated to the cut layer, gradients for layers 1 through l of LoRA adapters are transmitted back to the corresponding device. The device then update the device-side LoRA adapters as follows:

$$\Delta\boldsymbol{\Theta}_u^n(t) \leftarrow \Delta\boldsymbol{\Theta}_u^n(t-1,k) - \epsilon_u \mathbf{g}_n^u(t,k), \forall n \in \mathcal{N}. \tag{6}$$

LoRA aggregation (LA) in each round: Each device will iteratively perform the preceding stages for K epochs. All devices then upload their updated LoRA adapters to the edge server for aggregation. Additionally, the server maintains multiple separate server-side LoRA adapters for each device, which are independently updated during local fine-tuning. These server-side adapters are also aggregated to obtain a global server-side LoRA. Both the server-side and device-side LoRA adapters are aggregated using the FedAvg method. The aggregation process can be mathematically represented as

$$\Delta\boldsymbol{\Theta}_s(t+1,1) = \sum_{n=1}^{N} \frac{D_n}{D} \Delta\boldsymbol{\Theta}_s^n(t,K), \tag{7}$$

$$\Delta\boldsymbol{\Theta}_u(t+1,1) = \sum_{n=1}^{N} \frac{D_n}{D} \Delta\boldsymbol{\Theta}_u^n(t,K). \tag{8}$$

This aggregated model will be used in the next fine-tuning round until the target accuracy is reached.

B. Convergence Analysis

The convergence behavior of the proposed SFT scheme is analyzed. To facilitate the analysis, we adopt the following commonly used assumptions [33], [34]:

- *Assumption 1:* There exists a global optimal parameter set $\boldsymbol{\Theta}^\star$ such that:

$$F(\boldsymbol{\Theta}^\star) \leq F(\boldsymbol{\Theta}), \quad \forall \boldsymbol{\Theta}. \tag{9}$$

- *Assumption 2:* Each local loss function $f_n(\cdot)$ is non-convex and L-smooth:

$$\|\nabla f_n(x) - \nabla f_n(x')\|_2 \leq L\|x - x'\|_2. \tag{10}$$

Algorithm 1: Split Fine-Tuning (SFT) Scheme.

Initialize the cut layer and LoRA adapter parameter;
Base station broadcasts 1 to l transformer block;
for *training round* $t = 1, 2, \ldots, T$ **do**
  Base station broadcasts the latest device-side LoRA adapter;
  for *device* $n \in \mathcal{N}$ **do**
    for *local epoch* $k = 1, 2, \ldots, K$ **do**
      # Device executes
      Randomly selects a mini-batch sample from the local data;
      Execute the FP of the device-side pre-trained model and LoRA adapter, then obtain an immediate result;
      Transmit immediate result to the server based on the compression scheme;
      # Server executes
      Receive the immediate result from the device;
      Execute the FP of server-side pre-trained model and n-th LoRA adapter;
      Update the server-side LoRA adapter;
      Transmit the device-side LoRA adapter's gradient;
    end
    Update the device-side LoRA adapter to the edge server;
  end
  All devices send the latest device-side LoRA adapter to the edge server;
  Edge server aggregates server-side LoRA adapters and device-side LoRA adapters into a new LoRA.
end

- *Assumption 3:* The stochastic gradient $g_n(t)$ is an unbiased estimator of the true gradient:

$$\mathbb{E}[g_n(t)] = \nabla f_n(\mathbf{\Theta}(t)). \tag{11}$$

- *Assumption 4:* The variance of the stochastic gradient is bounded by

$$\mathbb{E}[\|g_n(t) - \nabla f_n(\mathbf{\Theta}(t))\|^2] \leq \phi^2. \tag{12}$$

- *Assumption 5:* The global loss $F(\cdot)$ satisfies the Polyak–Łojasiewicz condition:

$$\frac{1}{2}\|\nabla F(\mathbf{\Theta})\|^2 \geq \tau(F(\mathbf{\Theta}) - F(\mathbf{\Theta}^\star)). \tag{13}$$

The analysis is built upon the framework in [33], which studies federated fine-tuning convergence under non-convex objectives with LoRA adapters. In our scheme, the model is split into device-side and server-side parts, each optimized independently over disjoint LoRA adapter subsets. Despite the decoupled updates, global convergence is still ensured via coordinated back-propagation and periodic parameter aggregation. Let T denote the number of communication rounds, and assume a learning rate η such that $\zeta = \eta - \frac{L\eta^2}{2} > 0$. Then, we can establish the following result.

Theorem 1: The optimality gap after T rounds satisfies

$$\mathbb{E}[F(\mathbf{\Theta}(T)) - F(\mathbf{\Theta}^\star)] \leq (1 - 2\zeta\tau)^T (F(\mathbf{\Theta}(0)) - F(\mathbf{\Theta}^\star)) + \sum_{t=0}^{T-1} (1 - 2\zeta\tau)^t \cdot \frac{L\eta^2\phi^2}{2N(T-t)}. \tag{14}$$

In Theorem 1, $\zeta = \eta - \frac{L\eta^2}{2}$ characterizes the effective descent rate, which depends on the learning rate η and the smoothness constant L; τ is the Polyak–Łojasiewicz constant that quantifies the gradient dominance property of the global loss; ϕ^2 is the upper bound on the variance of stochastic gradients. The detailed proof can be found in Appendix A.

Discussion: When $\zeta > 0$ and $\tau > 0$, the expression shows that the first term converges exponentially toward the optimum and the second term remains bounded as communication rounds increase, thus establishing the convergence of the proposed SFT scheme.

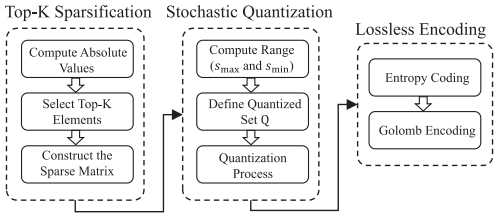


Fig. 2. Transmission compression scheme.

C. Intermediate Activation Compression Scheme

In the LLMs, the transmission of intermediate parameters between device-side transformer blocks and server-side transformer blocks can become a communication bottleneck. To address the high communication cost caused by transmitting the full activation matrix between transformer blocks, sparsification, quantization, and encoding techniques can be applied to reduce the amount of data transferred while retaining the most critical information [35]. Let $\mathbf{s}_l \in \mathbb{R}^{N\times D}$ represent the activations that are transmitted from the l-th transformer block to the $(l+1)$-th transformer block, where N is the number of patches (or tokens), and D is the embedding dimension. Directly transmitting the full matrix $\mathbf{s}_l$ between blocks incurs a high communication cost, especially when N and D are large. The transmission compression scheme can be represented as shown in Fig. 2.

Top-K sparsification: In Top-K sparsification, we selectively retain only the K largest-magnitude elements in the matrix $\mathbf{s}_l$, while setting the remaining elements to zero. This approach effectively reduces the communication cost by focusing on the most significant activations. The steps for applying Top-K sparsification to $\mathbf{s}_l$ are as follows:

1) Compute absolute values: For each element $\mathbf{s}_{l,i,j}$ in the matrix $\mathbf{s}_l$, compute the absolute value $|\mathbf{s}_{l,i,j}|$ to capture the magnitude of each activation.
2) Select Top-K elements: Identify the indices of the K largest values based on their magnitudes. Let $\mathcal{I}_K$ be the set of indices corresponding to these top K elements.
3) Construct the sparse matrix: Create a sparse version of $\mathbf{s}_l$, denoted as $\hat{\mathbf{s}}$, where only the elements in $\mathcal{I}_K$ retain their original values, and all other elements are set to zero. This sparse matrix can be expressed as

$$\hat{\mathbf{s}}_{l,i,j} = \begin{cases} \mathbf{s}_{l,i,j}, & \text{if } (i,j) \in \mathcal{I}_K; \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

A sparse matrix $\hat{\mathbf{s}}_l$ is derived through this process to reduce inter-block communication while preserving salient activations. The sparsification rate ρ denotes the ratio of retained elements

K to the total in $\mathbf{s}_l$, formulated as

$$\rho = \frac{K}{\dim(\mathbf{s}_l)}, \tag{16}$$

where $\dim(\mathbf{s}_l)$ denotes the cardinality of $\mathbf{s}_l$, and $\rho \in \mathbb{R}^+$ ensures a valid sparsification ratio.

Stochastic quantization: The quantization is to utilize fewer bits for representing the non-zero parameter $\hat{s}_l$ in the quantized vector. Initially, we compute the absolute values of $\hat{s}_l$, denoted as $|\hat{s}_l|$. We then identify the maximum and minimum non-zero values of $|\hat{s}_l|$ within the set, expressed as $s_{\max} = \max\{|s|, s \in \hat{s}_l\}$ and $s_{\min} = \min\{|s|, s \in \hat{s}_l\}$, respectively. For a quantization level $E \in \mathbb{Z}^+$, we define the set of quantized values $Q = \{Q_1, Q_2, \ldots, Q_E\}$, where each quantization point Q_e is calculated as

$$Q_e = \frac{e(s_{\max} - s_{\min})}{E} + s_{\min}. \tag{17}$$

The quantization objective is to map each non-zero scalar $s \in \hat{s}_l$ into this discrete quantized set Q. To achieve this, we select an index e such that $Q_e \leq |s| \leq Q_{e+1}$, thereby determining the quantization interval for s as $[Q_e, Q_{e+1}]$. The quantization procedure for any non-zero scalar $s \in \hat{s}_l$ is then defined as

$$\text{quant}(s, Q) = \begin{cases} \text{sgn}(s) \cdot Q_e, & \text{with probability } \frac{Q_{e+1}-|s|}{Q_{e+1}-Q_e}; \\ \text{sgn}(s) \cdot Q_{e+1}, & \text{otherwise,} \end{cases} \tag{18}$$

where $\text{sgn}(s) \in \{-1, +1\}$ determines the sign of s. Then, this approach yields the sparse and quantized model update $\text{quant}(\hat{s}_l, Q)$.

Lossless encoding: Given the distribution characteristics of activation, where smaller indices tend to appear more frequently, entropy coding can be utilized to reduce the data size. Additionally, the sparse binary matrix can be efficiently compressed using Golomb encoding [36], [37].

In our scheme, the compression rate β is treated as a derived metric determined by the sparsification rate ρ and the quantization level q. Therefore, optimizing the compression strategy is equivalent to jointly optimizing the values of ρ and q. To analyze the computational overhead introduced by the proposed compression scheme, we examine the complexity of its individual components. Given an activation matrix $s_l \in \mathbb{R}^{N \times D}$, the Top-K sparsification step involves computing absolute values and selecting the top K elements, which together require $\mathcal{O}(ND)$ time. The subsequent stochastic quantization and lossless encoding steps operate only on the K retained elements, each with a time complexity of $\mathcal{O}(K)$. Therefore, the total computational complexity of the compression process is $\mathcal{O}(ND + K)$, which simplifies to $\mathcal{O}(ND)$ under the typical condition that $K \ll ND$.

V. Performance Analysis

A. Fine-Tuning Delay Analysis

In this subsection, we analyze the delay in each fine-tuning round, which includes several phases.

1) *Model distribution (MD) latency:* At the beginning of fine-tuning, the edge server divides the transformer block into a server part and a device part. In the first round, the edge server broadcasts the complete device-side transformer block to all devices. In the subsequent rounds, the server distributes the aggregated device-side LoRA adapters. Let the initial data size of the allocated transformer block be denoted as $\Psi(l)$ and the data size of its corresponding LoRA as $\Psi^L(l)$. Following Shannon's theorem, the rate of transformer block distribution is given by $r = C \log_2(1 + \text{SNR}_s)$ where C and SNR_s represent the total bandwidth and the signal-to-noise ratio, respectively. The corresponding transmission latency can be expressed as

$$\tau_{MD}^t(l) = \begin{cases} \frac{\Psi(l)}{r}, & \text{if } t = 1 \\ \frac{\Psi^L(l)}{r}, & \text{if } t > 1. \end{cases} \tag{19}$$

2) *Device-side computation (DC) delay:* The device's local computation delay involves the transformer block's FP delay, which includes both the pre-trained model and LoRA. Let $\Phi_c^F(l)$ be the FLOPs required to be computed on device n, which will be discussed in the following subsection. The local computation delay for device n can be given by

$$\tau_{DC}^t(l) = \frac{\Phi_c^F(l)}{f_n C_n^u D_n^u}, \forall n \in \mathcal{N}, \tag{20}$$

where f_n is the GPU frequency of device n, C_n^U is the number of cores of the GPU at device n, and D_n^U represents the number of FLOPs executed in a single core cycle of the GPU.

3) *Immediate activation transmission (IAT) latency:* Each device transmits the immediate activation from the l-th transformer block to the server. Let Ψ^A denote the size of the output, which matches the size of the input token. The transmission rate for device n is given by $r_n^{UL}(b_n) = b_n \log_2(1 + \text{SNR}_n), \forall n \in \mathcal{N}$, where b_n represent the allocated bandwidth to device n and SNR_n represents its uplink signal-to-noise ratio. With the sparsification ratio β, the IAT latency is defined as

$$\tau_{\text{IAT}}^t(\beta, b_n) = \frac{\beta \Psi^A}{r_n^{UL}(b_n)}, \forall n \in \mathcal{N}. \tag{21}$$

4) *Server-side computation (SC) delay:* The server's computation delay involves considering the forward and BP delay of the server-side transformer block. FP uses the pre-trained model along with the corresponding LoRA, while BP only updates the LoRA parameters, without updating the pre-trained model's parameters. Let $\Phi_s^F(l)$ be the total FLOPs needed to be computed in the FP of server and $\Phi_s^B(l)$ represent the computational load during LoRA's BP. The training delay on server's training can be expressed as

$$\tau_{SC}^t(l) = \frac{(\Phi_s^F(l) + \Phi_s^B(l))}{f_n^s C_s D_s}, \forall n \in \mathcal{N}. \tag{22}$$

5) *Gradient transmission (GT) latency:* The server transmits the gradient of immediate activation. We assume the transmission rate for the server is $r_n^{DL}(b_n) = b_n \log_2(1 + \text{SNR}_s), \forall n \in \mathcal{N}$. The gradient transmission delay from the server to the device can be given by

$$\tau_{\text{GT}}^t(\beta, b_n) = \frac{\beta \Psi^G}{r_n^{DL}(b_n)}, \forall n \in \mathcal{N}. \tag{23}$$

6) *Device-side local updating (DU) delay :* Let γ_c represent the computational load during LoRA's BP. The BP time of the server

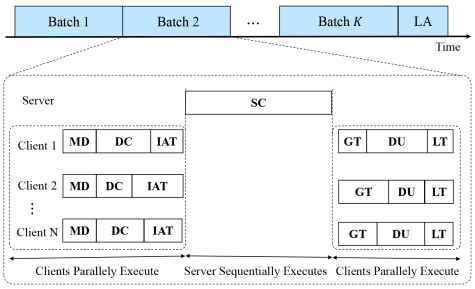


Fig. 3. Fine-tuning delay in each round.

is given by

$$\tau_{\mathrm{DU}}^{t}(l)=\frac{\Phi_{c}^{B}(l)}{f_{n}C_{n}^{u}D_{n}^{u}},\ \forall n\in\mathcal{N}. \tag{24}$$

7) *LoRA transmission (LT) latency:* Let Ψ^{L} denote the size of device-side LoRA. The latency of uploading LoRA from device to the server can be represented as

$$\tau_{\mathrm{LT}}^{t}(b_{n})=\frac{\Psi^{L}(l)}{r_{n}^{UL}(b_{n})},\ \forall n\in\mathcal{N}. \tag{25}$$

Each round of the total training delay consists of multiple phases, including transformer block distribution delay, device-side local computation delay, immediate result transmission delay, server-side computation delay, gradient transmission delay, and device-side local updating delay. As shown in Fig. 3, the overall delay for each fine-tuning round is

$$\begin{aligned}\tau_{n}^{t}(\beta,l,b_{n})&=\tau_{MD}^{t}(l)+\tau_{DC}^{t}(l)+\tau_{\mathrm{IAT}}^{t}(\beta,b_{n})+\tau_{SC}^{t}(l)\\&\quad+\tau_{\mathrm{GT}}^{t}(\beta,b_{n})+\tau_{\mathrm{DU}}^{t}(l)+\tau_{\mathrm{LT}}^{t}(b_{n}).\end{aligned} \tag{26}$$

We assume that the set of bandwidths allocated to all devices is denoted as $\mathbf{b}$. The delay for each training round can be expressed as

$$\tau_{t}(\beta,l,\mathbf{b})=\max\{\tau_{n}^{t}(\beta,l,b_{n})\},\ n\in\mathcal{N}, \tag{27}$$

and the total training delay after T rounds is given by

$$\tau(\beta,l,\mathbf{b})=\sum_{t=1}^{T}\tau_{t}(\beta,l,\mathbf{b}). \tag{28}$$

B. Memory Consumption Analysis

The memory consumption in each transformer block can be divided into four main components: model parameters, optimizer state, gradients, and activations. Each component's memory consumption depends on the number of parameters and the data type precision, which is represented by a variable α (e.g., $\alpha=4$ for FP32 precision).

1) Model Memory: The memory required to store model parameters is determined by the parameter number and the precision. The total memory consumption can be expressed as

$$M_{m}=\alpha(12D^{2}+18Dr). \tag{29}$$

2) Optimizer State: The optimizer states require additional memory to store updating-related variables such as momentum and variance terms for each parameter, as well as optional parameter copies depending on the optimizer type. For example: In the adaptive moment estimation (Adam) optimizer, the memory includes momentum and variance, requiring $\hat{\alpha}=2\alpha$ bytes. If mixed-precision training is used, an additional parameter copy is needed, increasing the memory to $\hat{\alpha}=3\alpha$. In the stochastic gradient descent (SGD) optimizer, the memory only includes momentum, requiring $\hat{\alpha}=\alpha$. The general memory requirement for optimizer states is therefore expressed as

$$M_{o}=\hat{\alpha}(12D^{2}+18Dr). \tag{30}$$

3) Gradient Memory: The memory required for gradient storage is proportional to the parameter number and depends on the precision of the gradients. This can be expressed as

$$M_{g}=\alpha(12D^{2}+18Dr). \tag{31}$$

4) Activation Memory: Activation memory depends on factors such as model size, recomputation strategies, and parallelization methods. activation memory can be approximated as

$$M_{a}=\alpha_{1}BND+\alpha_{2}BN^{2}A, \tag{32}$$

where α_1 and α_2 are coefficients that depend on the model architecture and training setup when using the estimation formula from the Megatron scheme $\alpha_1=34$ and $\alpha_2=5$ [38].

The total memory consumption in each transformer block can be expressed as

$$\begin{aligned}M_{t}&=M_{m}+M_{o}+M_{g}+M_{a}\\&=(2\alpha+\hat{\alpha})(12D^{2}+18Dr)+\alpha_{1}BND+\alpha_{2}BN^{2}A.\end{aligned} \tag{33}$$

The embedding layer's memory consumption for model weights and gradients is $4ND$, the output activations occupy $4B(N+1)D$, and there is additional memory consumption for local data of $4P^{2}CD$. The out layer's memory consumption is $4BND$. The memory consumption on the device side is expressed as

$$M^{c}(l)=16D^{2}+BND+lM_{t}. \tag{34}$$

C. Model Parameter

We first analyze the parameter size of the SFT scheme. The number of parameters in each transformer block mainly comes from three parts: the multi-head self-attention (MSA), feed-forward network (FFN), and layer normalization (LayerNorm). In the MSA, weight matrices are independently created for the query, key, and value, and with the additional LoRA matrix parameters, the total number of parameters is $4(Dr+rD)+4D^{2}$. The FFN consists of two fully connected layers, where the first expands the embedding dimension D to D_{mlp} and the second reduces it back to D. With LoRA, the total parameter number is $2(Dr+rD_{\mathrm{mlp}})+2DD_{\mathrm{mlp}}$, and given that $D_{\mathrm{mlp}}=4D$, the parameter number simplifies to $8D^{2}+10Dr$. Thus, the total number of parameters in each transformer block is approximately $12D^{2}+18Dr$. Additionally, the number of parameters in the embedding layer is $(P^{2}C+N+3)D$, which is related to the number and size of the patches. In the considered scheme, patches pass through multiple transformer blocks and are ultimately classified by a classification output layer, which is typically a fully connected layer mapping an input of dimension D to K classes, resulting in $DK+K$ parameters.

D. Computation Workload

The computational FLOPs of the device and server are related to the deployed model and model parameters. In the proposed frame, an embedding layer and l transformer blocks are deployed on the device side, while the remaining $(L-l)$ transformer blocks and the classification layer are deployed on the server side. Each parameter typically requires 2 to 4 FLOPs in floating-point operations (one multiplication and one addition), and for simplicity, we approximate this as 2 FLOPs per parameter. Then, the FLOPs required for the device's FP are given by $\Phi_c^F(l) = l(24BND^2 + 4BN^2D) + 2BNDK$ and the FLOPs required for BP are $\Phi_c^B(l) = l(48BND^2 + 8BN^2D) + 4BNDK$. The FLOPs required for the server's local FP and BP computations are $\Phi_s^F(l) = (L-l)(24BND^2 + 4BN^2D)$ and $\Phi_s^B(l) = (L-l)(48BND^2 + 8BN^2D) + 4BNDK$, respectively.

E. Communication Overhead

We further analyze the data transmission requirements for the MD, SC, GT, and LT stages, which will be discussed in the next subsection. During the MD stage, the pre-trained model and the initial LoRA model weights need to be distributed. Assuming each parameter occupies b bytes, the total data size to be transmitted in the ED stage is $\Psi(l) = bl(18Dr + 12D^2 + (P^2C + N + 3)D)$. In the SC stage, the intermediate patch sequence needs to be transmitted between the transformer blocks on the device and server sides, with a data size of BND. In the GT stage, the size of the gradients to be transmitted is equal to the size of the intermediate activations. During the LT stage, the size of the LoRA parameters to be transmitted is given by $\Psi^L(l) = 2lBDr$.

VI. Problem Formulation and Decomposition

A. Accuracy Model

The relationship between the objective accuracy function and the optimization parameters (sparsity rate and quantization level) is challenging to define theoretically due to the complex nature of model compression methods, such as sparsification, quantization, and encoding. These methods modify the model's parameters and structure, leading to performance changes that are highly nonlinear, with intricate dependencies across network layers and the inherent non-linearity of activation functions. To address this challenge, we approach the problem from a data-driven perspective. Specifically, we investigate how the accuracy is influenced by the sparsity rate and quantization level by fitting the observed data to a third-order function [39]. The fitted function is expressed as

$$A(\rho, q) = P_1 + P_2\rho + P_3 q + P_4\rho^2 + P_5 q^2 + P_6\rho q + P_7\rho^3 + P_8 q^3 + P_9\rho^2 q + P_{10}\rho q^2, \tag{35}$$

where ρ denotes the sparsity rate, and $q \in \mathbb{Z}^+$ represents the quantization level. P_1 to P_{10} are regression coefficients fitted from simulation data. Through this approach, we derived a model that captures the relationship between accuracy and the optimization parameters. The fitting results in Fig. 4 demonstrate a strong alignment between the predicted and actual data, achieving a mean squared error of less than 0.26%.

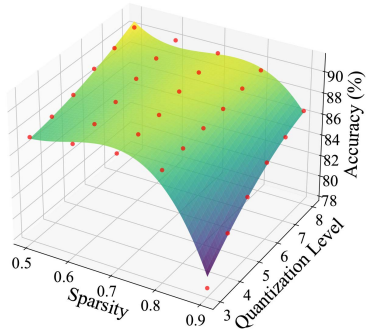


Fig. 4. Fitting results for SFT data processing accuracy.

B. Problem Formulation

In the proposed optimization problem, the objective is to minimize the fine-tuning delay while ensuring that the accuracy and memory constraints on devices are satisfied. To achieve this, we focus on jointly optimizing LLM fine-tuning parameters, including the compression ratio and the number of transformer blocks executed locally, as well as efficiently allocating spectrum bandwidth resources among devices. The final compression ratio β is jointly determined by sparsification, quantization, and encoding. In this optimization problem, we consider the tunable parameters ρ and E as the optimization variables. The joint optimization problem can be expressed as

$$\mathbb{P}: \quad \min_{\rho,q,l,\mathbf{b}} \quad \tau(\rho, q, l, \mathbf{b}) \tag{36a}$$

$$\text{s.t.} \quad \mathrm{A}(\rho, q) \geq \mathrm{A}_{\mathrm{th}}, \tag{36b}$$

$$\mathrm{M}^c(l) < M^c_{\max}, \quad \forall l \in \mathcal{L}, \tag{36c}$$

$$\rho_{\min} \leq \rho \leq \rho_{\max}, \tag{36d}$$

$$q_{\min} \leq q \leq q_{\max}, \tag{36e}$$

$$b_n \leq b_n^{\max}, \quad \forall n \in \mathcal{N}, \tag{36f}$$

$$\sum_{n\in\mathcal{N}} b_n = b_s^{\max}. \tag{36g}$$

Constraint (36b) ensures that the system's accuracy meets the minimum requirement, where A_{th} represents the maximum accuracy achievable by the LLM within an allowable accuracy degradation. Constraint (36c) enforces the storage resource usage on devices to remain within allowable limits. Constraints (36d) and (36e) bound the sparsity rate ρ and quantization level E within their respective ranges. Finally, constraints (36f) and (36g) ensure that both individual bandwidth allocations and the total bandwidth allocation comply with system-wide limitations, maintaining resource balance and operational feasibility.

C. Problem Decomposition

To efficiently solve the joint optimization problem presented in (36), we decompose it into two subproblems by leveraging the hierarchical structure of the decision variables. The first

subproblem focuses on optimizing the fine-tuning configuration, including the cut layer l, sparsification rate ρ, and quantization level q, to minimize the overall fine-tuning delay while satisfying accuracy and resource constraints. The second subproblem focuses on optimizing the network bandwidth allocations to minimize the fine-tuning delay for a given fine-tuning configuration.

1) Subproblem 1 - Joint Optimization of Sparsification Rate, Quantization Level, and Allocated Block: The first subproblem addresses the optimization of the cut layer l, sparsification rate ρ, and quantization level q, which directly impact the training process. The goal is to minimize the overall training delay while ensuring the system accuracy meets the specified requirements and resource constraints are satisfied. The subproblem can be formulated as follows:

$$\mathbb{P}_1: \quad \min_{\rho,q,l} \quad \tau(\rho,q,l) \tag{37a}$$

$$\text{s.t.} \quad \mathrm{A}(\rho,q) \geq \mathrm{A}_{\mathrm{th}}, \tag{37b}$$

$$\mathrm{M}_n^c(l) < M_{\max}^c, \quad \forall n \in \mathcal{N}, \tag{37c}$$

$$\rho_{\min} \leq \rho \leq \rho_{\max}, \tag{37d}$$

$$q_{\min} \leq q \leq q_{\max}, \tag{37e}$$

$$0 < l < L. \tag{37f}$$

2) Subproblem 2 - Optimization of Spectrum Bandwidth Allocation: The second subproblem focuses on the dynamic bandwidth allocation $\mathbf{b}$ of network resources. These parameters are optimized based on the training configuration ($l^\star$, $\rho^\star$, and $q^\star$) obtained from Subproblem $\mathbb{P}_1$. The objective is to minimize the training delay by effectively allocating network resources in response to the dynamic environment. The subproblem can be formulated as follows:

$$\mathbb{P}_2: \quad \min_{\mathbf{b}} \max_{n\in\mathcal{N}} \quad \tau_t(\mathbf{b_n}) \tag{38a}$$

$$\text{s.t.} \quad b_n \leq b_n^{\max}, \quad \forall n \in \mathcal{N}, \tag{38b}$$

$$\sum_{n\in\mathcal{N}} b_n = b_s^{\max}. \tag{38c}$$

Discussion: The two subproblems are decomposed based on large and small timescales and solved sequentially. First, in the large timescale, $\mathbb{P}_1$ is solved to determine the optimal training configuration, including $l^\star$, $\rho^\star$, and $q^\star$, as the model structure remains consistent across training rounds. These results are then used as fixed inputs in the small timescale to solve $\mathbb{P}_2$, where network resource allocation is optimized based on device-specific heterogeneous resources and dynamic channel conditions. This decomposition and sequential optimization effectively reduce computational complexity by clearly separating global and real-time optimization tasks, enabling efficient algorithm design tailored to system accuracy, resource utilization, and dynamic performance requirements.

VII. Proposed Solution

A. Fine-Tuning Configuration Algorithm

To simplify the subproblem $\mathbb{P}_1$, we introduce Lagrangian multipliers λ to relax some of the constraints, forming the Lagrangian function $\mathcal{L}(\rho,q,l,\lambda)$ as follows:

$$\mathcal{L}(\rho,q,l,\boldsymbol{\lambda}) = \tau(\rho,q,l) + \lambda_1(\mathrm{A}(\rho,q) - \mathrm{A}_{\mathrm{th}}) + \lambda_2(\mathrm{M}_{\max}^c - \mathrm{M}^c(l)), \tag{39}$$

where $\boldsymbol{\lambda} = [\lambda_1, \lambda_2]$ are Lagrangian multipliers that penalize the violation of accuracy and resource constraints, with $\lambda_1, \lambda_2 \geq 0$. By relaxing these constraints, we convert the original problem into an unconstrained optimization problem and aim to maximize the relaxed Lagrangian function.

Since l is a discrete variable with a finite range, we can decompose the problem into subproblems for different values of l. For each possible value of l, we optimize the continuous variables ρ and q, and select the combination that maximizes the objective function. This process is expressed as:

1) Solving the Discrete Variable Via Exhaustive Search: For each $l \in \{1, 2, \ldots, L\}$, we can use methods like exhaustive search to evaluate the optimal solution for each discrete value of l. The solution process begins by fixing l, which involves selecting a specific value for l. With l fixed, the next step is to optimize the continuous variables ρ and q by maximizing the Lagrangian function $\mathcal{L}(\rho,q,l,\lambda)$. After determining the optimal ρ and q for the fixed l, we compute and record the corresponding values of l, ρ, and q along with the Lagrangian function value. Thus, we can identify the optimal ρ, q, and the maximum objective function value corresponding to each possible value of l.

2) Optimizing the Continuous Variables: For a fixed value of l, optimizing the continuous variables ρ and q reduces to a single-variable optimization problem for each. This can be solved using several methods. If the Lagrangian function $\mathcal{L}(\rho,q,l,\lambda)$ is differentiable with respect to ρ and q, we can apply gradient ascent to iteratively update ρ and q in the direction that increases

Algorithm 2: Lagrange-Based Fine-Tuning Configuration Algorithm.

Input: Discrete variable range $l \in \mathcal{L}$, initial Lagrange multipliers λ_0, step size μ_k, tolerance ϵ
Output: Optimal solution $(\rho^\star, q^\star, l^\star)$

Initialize Lagrange multipliers $\lambda = \lambda_0$;
Set iteration counter $k = 0$;
repeat
  for *each* $l \in \mathcal{L}$ **do**
    Fix l and optimize the continuous variables ρ and q by maximizing $\mathcal{L}(\rho,q,l,\lambda)$;
    Obtain the optimal $(\rho_l^\star, q_l^\star)$ for the fixed l;
    Compute $\mathcal{L}(\rho_l^\star, q_l^\star, l, \lambda)$ and store $\{\rho_l^\star, q_l^\star, \mathcal{L}(\rho_l^\star, q_l^\star, l, \lambda)\}$;
  end
  Select $l^\star = \arg\max_l \mathcal{L}(\rho_l^\star, q_l^\star, l, \lambda)$ and corresponding $(\rho^\star, q^\star) = (\rho_{l^\star}^\star, q_{l^\star}^\star)$;
  Update the Lagrange multipliers:

$$\lambda_i^{k+1} = \lambda_i^k + \mu_k\, g_i(\rho^\star, q^\star, l^\star)$$

  where $g_i(\rho^\star, q^\star, l^\star)$ represents the degree of constraint violation;
  if $|\mathcal{L}(\rho^\star, q^\star, l^\star, \lambda) - \mathcal{L}_{prev}| < \epsilon$ ***and*** *all constraints are satisfied* **then**
    Convergence achieved;
    break;
  end
  Update $k = k + 1$;
until *convergence*;

Algorithm 3: SQP-Based Spectrum Bandwidth Allocation Algorithm.

Input: Initial feasible solution $(\mathbf{b}^0, \tau^{\star,0})$, maximum iterations $K_{\max}$, tolerance ϵ;
Output: Optimized solution $(\mathbf{b}^\star, \tau^\star)$;
Initialize $k = 0$;
repeat
  Compute objective value $\tau_n^k = \tau(b_n^k)\ \forall n \in \mathcal{N}$;
  Compute gradients $\nabla_{\mathbf{b}} \tau_n^k$ at the current iteration;
  Formulate the QP subproblem by linearizing the constraints as in Eq. (44);
  Solve the QP subproblem to obtain $\Delta\mathbf{b}$ and $\Delta\tau^\star$;
  Update variables via

$$\mathbf{b}^{k+1} = \mathbf{b}^k + \Delta\mathbf{b}, \tau^{\star,k+1} = \tau^{\star,k} + \Delta\tau^\star$$

  Check convergence criteria: **if** $|\tau^{\star,k+1} - \tau^{\star,k}| \le \epsilon$ ***or*** $\|\Delta\mathbf{b}\|_2 \le \epsilon$ **then**
    Convergence achieved;
    break;
  end
  Update $k = k + 1$;
  if $k \ge K_{max}$ **then**
    break;
  end
until *convergence*;

$\mathcal{L}(\rho, q, l, \lambda)$, thereby maximizing the function. Another method is to use convex optimization techniques, which are particularly efficient if $\mathcal{L}(\rho, q, l, \lambda)$ is a convex function in ρ and q. By using convex optimization, we can quickly and reliably solve for the optimal ρ and q when convexity is present. Additionally, grid search is applicable when differentiability or convexity is not guaranteed.

3) Updating the Lagrange Multipliers: To ensure that the relaxed solution satisfies the original problem's constraints, we use the subgradient method to iteratively update the Lagrange multipliers. We denote all the inequality constraints by $g_p(\rho, q) = \mathrm{A}(\rho, q) - \mathrm{A}_{\mathrm{th}} \ge 0$ for $p = 1$, $g_p(l) = M_{\max}^c - \mathrm{M}_n^c(l) \ge 0$ for $p = 2, \ldots, N+1$, $g_p(\rho) = \rho - \rho_{\min} \ge 0$ for $p = N+2$, $g_p(\rho) = \rho_{\max} - \rho \ge 0$ for $p = N+3$, $g_p(q) = q - q_{\min} \ge 0$ for $p = N+4$, $g_p(q) = q_{\max} - q \ge 0$ for $p = N+5$, $g_p(l) = L - l \ge 0$ for $p = N+6$, and $g_p(l) = l \ge 0$ for $p = N+7$. The update rule is given by

$$\lambda_i^{k+1} = \lambda_i^k + \mu_k\, g_i(\rho, q, l), \tag{40}$$

where μ_k is the step size. During each iteration, the Lagrange multipliers are adjusted based on the extent of violation of each constraint, ensuring that the solution progressively satisfies the constraints as the process converges.

Computational Complexity Analysis: The overall computational complexity of Algorithm 2 is $\mathcal{O}(KLC)$, where K denotes the number of Lagrange multiplier update iterations, L is the number of candidate values for the discrete variable l, and C represents the computational cost of optimizing the continuous variables ρ and q for a fixed l. In practice, both L and K are moderate (e.g., $L = 12$ for a 12-layer Transformer, and $K \le 30$ for subgradient convergence), making the total overhead tractable. Furthermore, this optimization is executed only once in the offline phase before the fine-tuning task starts. As a result, the algorithm introduces negligible runtime overhead compared to the fine-tuning duration.

B. Spectrum Resource Allocation Algorithm

Problem P_2 optimizes the fine-tuning delay by dynamically allocating bandwidth $\mathbf{b}$. The objective function $\tau(\mathbf{b})$ in P_2 is nonlinear, and the constraints include both equality and inequality conditions. Due to the inherent nonlinearity of the objective function and the complexity of the constraints, we adopt the SQP method to solve this problem. The process involves reformulating the problem to eliminate the nested max operator, followed by constructing and solving Quadratic Programming (QP) subproblems at each iteration.

1) Problem Reformulation: The original objective function $\min \max \tau(b_n)$ involves a nested $\max$ operator that complicates direct optimization. An auxiliary variable t is introduced and replace the $\max$ term with an inequality constraint:

$$\tau^\star \ge \tau(b_n), \quad \forall n \in \mathcal{N}. \tag{41}$$

The optimization problem is reformulated as

$$\mathbb{P}_3: \quad \min_{\mathbf{b}} \ \tau^\star \tag{42a}$$

$$\text{s.t.} \quad b_n \le b_n^{\max}, \quad \forall n \in \mathcal{N}, \tag{42b}$$

$$\sum_{n\in\mathcal{N}} b_n = b_s^{\max}, \tag{42c}$$

$$\tau^\star \ge \tau(b_n), \quad \forall n \in \mathcal{N}. \tag{42d}$$

The reformulated problem $\mathbb{P}_3$ minimizes the auxiliary variable $\tau^\star$ subject to linear and nonlinear constraints. The inequality constraint $\tau^\star \ge \tau(b_n)$ is nonlinear and must be linearized at each iteration to construct the QP subproblem.

2) Linearization of the Nonlinear Constraint: To construct the QP subproblem, the nonlinear inequality constraint $\tau^\star \ge \tau(b_n)$ is linearized around the current iterate $\mathbf{b}^k$. Using a first-order Taylor expansion, the constraint can be approximated as

$$\tau^\star \ge \tau(b_n^k) + \nabla_{\mathbf{b}} \tau_n^k (\mathbf{b} - \mathbf{b}^k), \quad \forall n \in \mathcal{N}, \tag{43}$$

where $\nabla_{\mathbf{b}} \tau_n^k$ is the gradient of $\tau(b_n)$ with respect to $\mathbf{b}$ evaluated at $\mathbf{b}^k$.

3) QP Subproblem Construction: At each iteration k, the QP subproblem is formulated by approximating the problem $\mathbb{P}_3$ with a quadratic objective function and linear constraints:

$$\mathbb{P}_4: \quad \min_{\Delta\mathbf{b}, \Delta\tau^\star} \ \tau^\star \tag{44a}$$

$$\text{s.t.} \quad 0 \le b_n^k + \Delta b_n \le b_n^{\max}, \quad \forall n \in \mathcal{N}, \tag{44b}$$

$$\sum_{n\in\mathcal{N}} (b_n^k + \Delta b_n) = b_s^{\max}, \tag{44c}$$

$$\tau^{\star,k} + \Delta\tau^\star \ge \tau_n^k + \nabla_{\mathbf{b}} \tau_n^k \Delta\mathbf{b}, \quad \forall n \in \mathcal{N}, \tag{44d}$$

where $\Delta\mathbf{b} = \mathbf{b} - \mathbf{b}^k$, and $\Delta\tau^\star = \tau^\star - \tau^{\star,k}$.

4) Solving the QP Subproblem: The QP subproblem defined in (44) is a convex optimization problem with linear constraints and a linear objective function, which can be efficiently solved using standard QP solvers like the interior-point method or active-set method. The steps to solve the QP subproblem are:

- Gradient computation: At the current iteration k, compute the gradient $\nabla_{\mathbf{b}}\tau_n^k$ for each device n, which is necessary for constructing the linearized constraints.
- Formulating of the QP problem: Use the computed gradient and the current iterate $(\mathbf{b}^k, \tau^{\star,k})$ to formulate the QP subproblem as described in (44). This involves substituting the linearized constraints into the objective function and ensuring all resource allocation constraints are satisfied.
- Solve the QP problem: Employ a standard QP solver (e.g., an interior-point method or active-set method) to solve the convex optimization problem. The solver provides the optimal increments $\Delta\mathbf{b}$ and $\Delta\tau^\star$.
- Update variables: Update the current variables using the obtained increments: $\mathbf{b}^{k+1} = \mathbf{b}^k + \Delta\mathbf{b}$, and $\tau^{\star,k+1} = \tau^{\star,k} + \Delta\tau^\star$.
- Check convergence: Evaluate the convergence criteria. If the change in the objective function $|\tau^{\star,k+1} - \tau^{\star,k}|$ is less than a predefined tolerance ϵ, or if the norm of the step sizes $\|\Delta\mathbf{b}\|_2 + |\Delta\tau^\star|$ is sufficiently small, the algorithm terminates. Otherwise, proceed to the next iteration.
- Repeat: Iterate the process from Step 1 until the convergence criteria are met or the maximum number of iterations $K_{\max}$ is reached.

Computational Complexity Analysis: At each iteration of the SQP-based resource allocation algorithm, the main computational cost arises from solving the QP subproblem. Given $N = |\mathcal{N}|$ devices, the bandwidth allocation vector $\mathbf{b}$ is N-dimensional. Including the auxiliary variable $\tau^\star$, the total number of optimization variables is $n = N + 1$. The QP subproblem formulated in (44) involves $2N + 1$ linear inequality constraints and one linear equality constraint. Assuming the QP subproblem is solved using an approximate method with per-iteration cost $\mathcal{O}(n^2)$, the complexity simplifies to $\mathcal{O}(N^2)$ per iteration. Therefore, the overall computational complexity of the SQP algorithm is $\mathcal{O}(KN^2)$, where K is the number of iterations before convergence.

VIII. Simulation Results

A. Simulation Setting

For the experimental evaluations, we focus on the image classification task using the CIFAR100 dataset [40], which contains 50,000 fine-tuning samples and 10,000 test samples across 100 classes, and the Tiny-ImageNet dataset [41], which includes 100,000 fine-tuning samples and 10,000 validation samples spanning 200 classes. The details of the parameter and hyper-parameter settings for model fine-tuning are presented in Table II. For the independent and identically distributed (IID) setting, we randomly partition the dataset into multiple shards and uniformly assign them to each device. For the non-IID setting, we use the Dirichlet distribution with a concentration parameter of 0.5 for the dataset partition.

In the default experimental setting, the simulated SFT architecture consists of 1 edge server and 8 devices. We consider a system bandwidth of 5 MHz, with an SNR of 17 dB between the devices and the server. The computation frequency of the devices is randomly generated between 0.5 GHz and 1.5 GHz, while the server's computation frequency is set to 40 GHz. For the large model fine-tuning setup, the batch size is set to 64, the hidden layer dimension is configured to 3072, and the image size is set to 224×224 with each patch size of 16×16. We adopt the mobile devices are NVIDIA Jetson Nano equipped with a 256-core GPU and a floating-point operation capacity of 4. The server is equipped with advanced GPUs featuring 2048 cores and a floating-point operation capacity of 4.

TABLE II
Simulation Parameters

Parameter	Value	Parameter	Value
B	30 MHz	# devices	8
D_n^u	4	LLM	ViT [15]
C_n^u	2×10^9	Cut layer point	5-th ViT block
C_s	1.5 GHz	Model size	86 M
D_s	2,048	Optimizer	SGD
f^s	3×10^9	Momentum	0.9
Learning rate	1e-4	Batch size	64
Decay	0.998	LoRA rank	16

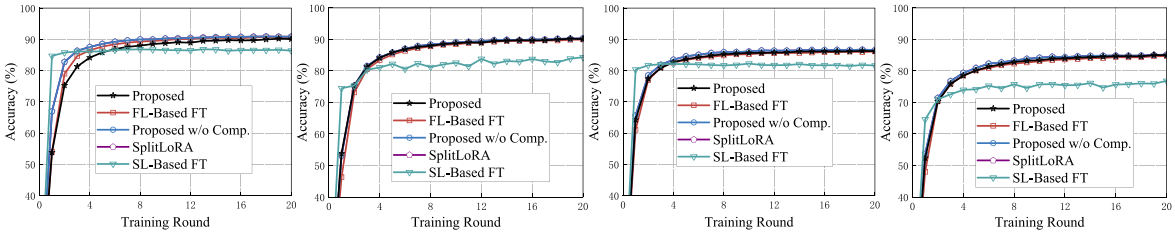
We compare the proposed SFT with the distributed learning schemes as follows. To ensure fairness, we follow the same hyper-parameter setting for various baseline schemes.

- *FL-Based Fine-Tuning (FT) [42]:* By incorporating LoRA tuning into FL, each device transmits only low-rank matrices to the server, significantly reducing communication overhead.
- *SL-Based FT [43]:* SL divides a neural network into two parts, typically split between devices and a central server, thereby reducing training overhead by offloading computations and minimizing data transfer and local processing demands on the device side.
- *SplitLoRA [44]:* SplitLoRA performs fine-tuning based on LoRA. Each device maintains a user-side pre-trained model and a LoRA adapter, while the server holds multiple server-side pre-trained models and corresponding LoRA adapters. Devices conduct training in parallel to mitigate the long delay caused by SL.
- *SFT w/o Compression:* The SFT operates without the joint compression scheme.

B. Performance Evaluation

1) Fine-Tuning Accuracy: Fig. 5 shows the simulation results of the training process, highlighting several key observations. First, the proposed SFT scheme demonstrates robust convergence under both IID (in Fig. 5(a) and (c)) and non-IID conditions (in Fig. 5(b) and (d)). Second, the accuracy achieved by our scheme is comparable to that of state-of-the-art benchmarks. Referring to the observation in the dropout layer, the distorted intermediate activations not only enhance the robustness of model training, but also lead to slight improvements in accuracy. Similarly, compressing the intermediate activations is expected to achieve a comparable accuracy. The results confirm that the efficiency improvements achieved through compression are not at the expense of training performance.

2) Memory Consumption: Table III presents the peak memory usage on the device side under different schemes. The results show that the proposed SFT and SL schemes reduce memory



(a) CIFAR100 IID (b) CIFAR100 non-IID (c) Tiny-ImageNet IID (d) Tiny-ImageNet non-IID

Fig. 5. Fine-tuning performance comparison among different schemes.

TABLE III
MEMORY COMPARISON ON DEVICE SIDE AMONG DIFFERENT SCHEMES

Memory (MB)	FL-Based FT	FL-LoRA	Proposed/SplitLoRA/ SL-Based FT
Input data	36.75	36.75	36.75
Activation	9171.11	9513.82	3941.23
Model+LoRA	327.88	337.03	141.19
Optimizer (SGD)	329.76	9.09	3.75
Total	9865.5 (2.39×)	9896.69 (2.4×)	4122.92 (1×)

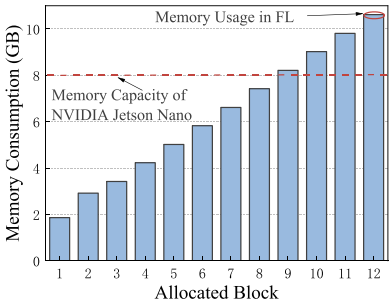


Fig. 6. Memory consumption on a device with respect to allocated ViT block.

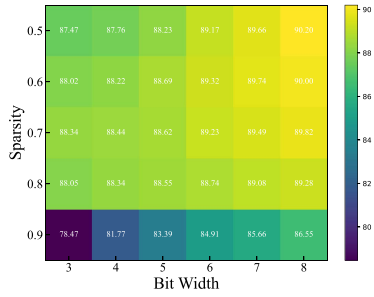
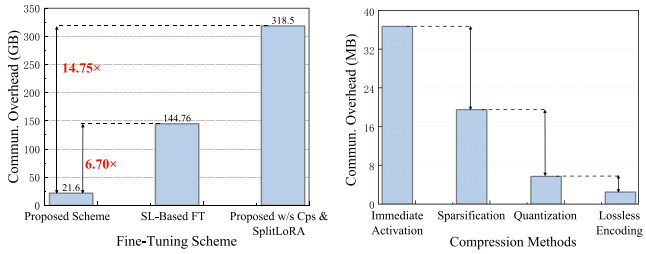


Fig. 7. Impact of sparsity and bit width on accuracy.



(a) Overall communication overhead. (b) Per-round communication overhead.

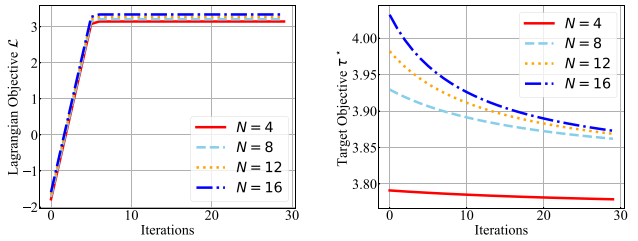
Fig. 8. Communication overhead in different fine-tuning schemes and different compression methods.

consumption by 58.2% compared to the FL scheme, where 5 ViT blocks were allocated on the device side. Additionally, we evaluate the performance of LoRA applied to the FL scheme and find that LoRA does not effectively reduce memory consumption. This is because activations account for a significant portion of memory consumption, while the optimizer, which LoRA reduces, contributes only a small fraction to the total memory consumption. Therefore, when fine-tuning LLM on the device, it is essential to split the LLM to address memory constraints on devices. Fig. 6 illustrates the relationship between the required memory size on the device and the number of allocated ViT blocks, showing a proportional increase. The red line in the figure represents the maximum memory capacity of the NVIDIA Jetson Orin Nano. The FL scheme exceeds this memory limit, making it unsuitable for deployment on resource-constrained devices.

3) Communication Overhead: The performance of the proposed compression scheme is shown in Fig. 7. The results demonstrate that the scheme can compress the transmission volume of intermediate activations within an allowable accuracy degradation. However, the compression cannot be arbitrarily small, as we observe that accuracy starts to drop sharply when the sparsity reaches 90%. To limit the accuracy loss within 2%, the sparsity and random quantization methods can achieve a maximum compression ratio of 12x with 80% sparsity and 3-bit quantization, which can be further increased to 20x when combined with lossless encoding.

The impact of different compression schemes on communication overhead, as well as the communication overhead of various fine-tuning methods, is illustrated in Fig. 8. As shown in Fig. 8(a), the SL-based FT scheme requires transmitting 144.76 GB of data to complete the fine-tuning process, which is 6.7 times that of our proposed scheme. Meanwhile, without compression, our scheme would require 14.75 times more data transmission, demonstrating the importance and effectiveness of the proposed compression scheme. In Fig. 8(b), we present the gains achieved by different compression methods on transmission, reducing the final transmitted data to 6.8% of the original activations, thereby effectively reducing communication overhead.

4) Optimization Algorithm Performance: Fig. 9 shows the convergence results of the proposed two-timescale algorithms. It can be observed that both the Lagrangian function and the target objective progressively converge, demonstrating the overall convergence of the proposed algorithm. Moreover, convergence



(a) Convergence of the Lagrangian-based optimization. (b) Convergence of the resource allocation optimization.

Fig. 9. Convergence of the two-timescale algorithm for different user numbers.

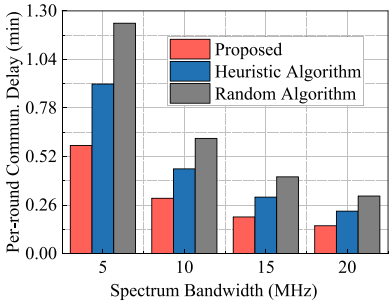


Fig. 10. Fine-tuning delay performance comparing among different optimization schemes.

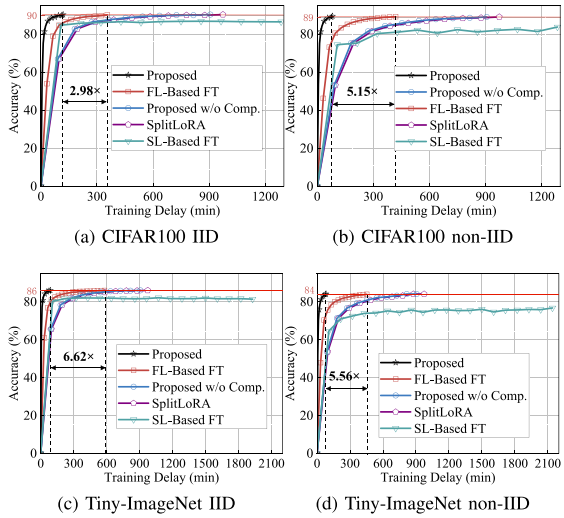


Fig. 11. Training delay performance comparison among different fine-tuning schemes.

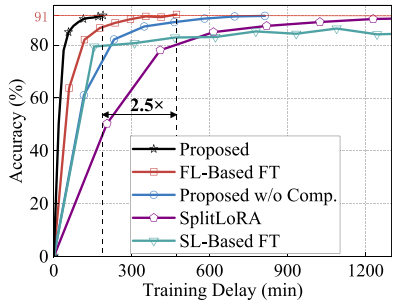


Fig. 12. Delay performance under the ViT-Large model.

is consistently achieved under different user numbers. Fig. 10 illustrates the delay performance of our optimization algorithm compared to other resource allocation methods. We evaluated two baseline algorithms: one where bandwidth resources are evenly distributed among devices and another where bandwidth resources are randomly allocated. Experimental results demonstrate that under various system bandwidth conditions, the proposed two-stage optimization algorithm consistently outperforms the baselines. The proposed algorithm maintains its performance gains even in resource-constrained networks. For instance, even under a constrained 5 MHz bandwidth, the proposed algorithm reduces communication delay by up to 53.1% in each fine-tuning round. This effectiveness is attributed to our communication compression scheme, which minimizes the communication volume to a very small scale during the fine-tuning process.

5) Fine-Tuning Delay: Fig. 11 presents the delay performance results, evaluated using the time required to reach the highest accuracy of the FT scheme. The results demonstrate that the proposed scheme achieves the fastest training speed across various datasets and under different data distribution conditions. For example, assuming a 5MHz wireless bandwidth and under the non-IID condition of CIFAR-100, the proposed algorithm requires only 81.4 minutes to reach the maximum accuracy. This is 5.15× faster than FL-based fine-tuning, which takes 419.6 minutes; 11.14× faster than the non-compressed version of our proposed scheme, which requires 906.9 minutes; and 11.96× faster than SplitLoRA. Additionally, the proposed scheme is 13 times faster than the SL scheme, which reaches 88% accuracy in 1071.1 minutes. The gain of the proposed scheme is attributed to the combination of sequential fine-tuning mechanism and model aggregation, which ensures high accuracy, and the compression scheme significantly reduces the overall fine-tuning delay.

6) Robustness: Fig. 12 presents the fine-tuning delay performance of all schemes on the ViT-Large model with the CIFAR-100 dataset. The results show that the proposed scheme consistently achieves the fastest fine-tuning performance, offering at least a 2.5× speedup compared to the benchmarks. The substantial increase in model parameters and dimensionality in ViT-Large leads to a significant volume of intermediate activations and gradients, which exacerbates the communication bottleneck. Hence, the schemes without compression or with parallel synchronization suffer from high communication latency. In contrast, FL-based FT is less affected since it only transmits a small amount of LoRA parameters, which narrows the performance gap between it and our proposed scheme.

Fig. 13 shows the fine-tuning delay on CIFAR-100 using ViT-Base with 20 devices. The proposed scheme achieves the fastest convergence, reaching the target accuracy with a 3.98× speedup over FL-based fine-tuning. As the number of devices increases, the competition for uplink bandwidth becomes more intense, leading to limited transmission resources per device. As a result, the performance of all benchmark schemes degrades significantly under such contention. In contrast, our scheme maintains low delay by employing communication-efficient compression and the sequential execution of the fine-tuning process on the

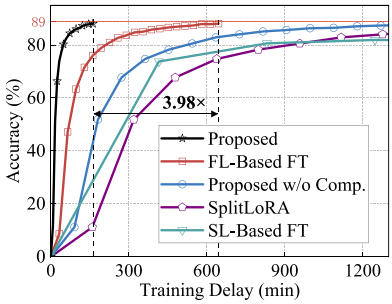


Fig. 13. Delay performance with 20 devices.

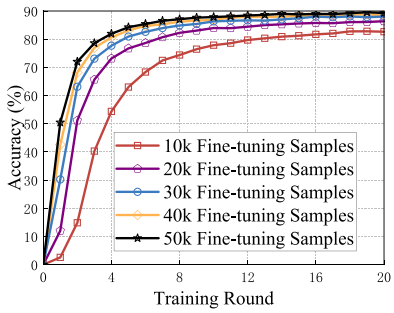


Fig. 14. Fine-tuning accuracy performance under varying amounts of training data.

server, which allows the full bandwidth to be utilized for gradient transmission. This design mitigates the impact of bandwidth contention and ensures stable and efficient convergence.

Fig. 14 shows the convergence performance of the proposed scheme under different data volumes ranging from 10,000 to 50,000 samples. As the number of samples increases, the model achieves higher accuracy within fewer training rounds. For instance, with 50,000 samples, it exceeds 85% accuracy within six rounds, while only reaching approximately 82% with 10,000 samples. Although the final accuracy shows a slight decrease under smaller data volumes, this behavior is expected due to the limited training information available on each device. Moreover, all benchmark schemes will also exhibit a similar decline in accuracy under the same conditions, indicating that the performance drop is primarily attributed to the reduced data size rather than the proposed scheme itself. Importantly, the proposed scheme consistently achieves rapid convergence across all data volume settings, reaching a stable accuracy within 20 training rounds.

These comprehensive results across varying model sizes, device numbers, and data volumes collectively demonstrate the proposed fine-tuning scheme's robustness and generalizability.

IX. Conclusion

In this paper, we have proposed a novel SFT scheme for facilitating efficient LLM fine-tuning in wireless networks. The SFT satisfies devices' memory constraints by splitting the LLM, accelerates fine-tuning with a parallelized SL scheme, and reduces communication overhead via a joint compression scheme. Furthermore, we have developed a two-timescale resource management algorithm to minimize the overall fine-tuning delay. Simulation results demonstrate that the proposed scheme reduces fine-tuning delay and communication overhead while satisfying device-side memory and accuracy constraints. This scheme is well-suited for collaborative LLM fine-tuning across multiple memory-constrained devices in resource-limited wireless networks, which can be extended to Internet of Things systems. In future work, we will investigate the performance of the proposed scheme in large-scale wireless networks.

Appendix A
Proof of Theorem 1

Firstly, the global objective function is defined as

$$F(\mathbf{\Theta}) = \frac{1}{N}\sum_{n=1}^{N} f_n(\mathbf{\Theta}), \tag{A.1}$$

where $f_n(\mathbf{\Theta})$ denotes the local loss function on device n. The global gradient and its squared norm satisfy

$$\nabla F(\mathbf{\Theta}) = \frac{1}{N}\sum_{n=1}^{N} \nabla f_n(\mathbf{\Theta}), \tag{A.2}$$

$$\|\nabla F(\mathbf{\Theta})\|^2 \le \frac{1}{N}\sum_{n=1}^{N} \|\nabla f_n(\mathbf{\Theta})\|^2. \tag{A.3}$$

Secondly, it is assumed that all N devices participate in training at round t. The adapter is updated using stochastic gradient descent as follows:

$$\bar{g}^a(t) = \frac{1}{N}\sum_{n\in\mathcal{N}} \nabla f_n(\mathbf{\Theta}(t)), \tag{A.4}$$

$$\mathbf{\Theta}(t+1) = \mathbf{\Theta}(t) - \eta \bar{g}^a(t). \tag{A.5}$$

Thirdly, for analytical simplicity, a virtual update is defined under the assumption that all devices participate:

$$v^a(t+1) = \mathbf{\Theta}(t) - \frac{\eta}{K}\sum_{k=1}^{K} \nabla f_n(\mathbf{\Theta}(t)). \tag{A.6}$$

Fourthly, based on the L-smoothness of $F(\cdot)$, the objective at $\mathbf{\Theta}(t+1)$ can be expanded as

$$F(\mathbf{\Theta}(t+1)) \le F(\mathbf{\Theta}(t)) - \eta\langle \nabla F(\mathbf{\Theta}(t)), \bar{g}^a(t)\rangle + \frac{L\eta^2}{2}\|\bar{g}^a(t)\|^2. \tag{A.7}$$

Then, taking the expectation over the stochastic gradient and using Assumptions 3 and 4, the following is obtained:

$$\mathbb{E}[F(\mathbf{\Theta}(t+1))] \le F(\mathbf{\Theta}(t)) - \eta\|\nabla F(\mathbf{\Theta}(t))\|^2 + \frac{L\eta^2}{2}\left(\|\nabla F(\mathbf{\Theta}(t))\|^2 + \frac{\phi^2}{N(t)}\right). \tag{A.8}$$

By letting $\zeta = \eta - \frac{L\eta^2}{2} > 0$, the inequality becomes

$$\mathbb{E}[F(\mathbf{\Theta}(t+1))] \leq F(\mathbf{\Theta}(t)) - \zeta\|\nabla F(\mathbf{\Theta}(t))\|^2 + \frac{L\eta^2\phi^2}{2N(t)}. \tag{A.9}$$

Next, the Polyak–Łojasiewicz condition (Assumption 5) is applied, leading to

$$\mathbb{E}[F(\mathbf{\Theta}(t+1)) - F(\mathbf{\Theta}^\star)] \leq (1-2\zeta\tau)(F(\mathbf{\Theta}(t)) - F(\mathbf{\Theta}^\star)) + \frac{L\eta^2\phi^2}{2N(t)}. \tag{A.10}$$

Finally, by unrolling the recursion over T communication rounds, the following convergence bound is derived:

$$\mathbb{E}[F(\mathbf{\Theta}(T)) - F(\mathbf{\Theta}^\star)] \leq (1-2\zeta\tau)^T(F(\mathbf{\Theta}(0)) - F(\mathbf{\Theta}^\star)) + \sum_{t=0}^{T-1}(1-2\zeta\tau)^t \cdot \frac{L\eta^2\phi^2}{2N(T-t)}. \tag{A.11}$$

Thus, Theorem 1 is proved.

ACKNOWLEDGMENT

The authors would like to thank Zuguang Li and Xiaopei Chen for his helpful feedback and suggestions throughout the work.

REFERENCES

[1] D. Cai, Y. Wu, S. Wang, F. X. Lin, and M. Xu, "Efficient federated learning for modern NLP," in *Proc. 29th Annu. Int. Conf. Mobile Comput. Netw.*, Oct. 2023, pp. 1–16.
[2] G. Xu, Z. Hao, Y. Luo, H. Hu, J. An, and S. Mao, "DeViT: Decomposing vision transformers for collaborative inference in edge devices," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 5917–5932, May 2024.
[3] T. Yao, Y. Li, Y. Pan, Y. Wang, X. Zhang, and T. Mei, "Dual vision transformer," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 9, pp. 10870–10882, Sep. 2023.
[4] W. Kuang et al., "Federatedscope-LLM: A comprehensive package for fine-tuning large language models in federated learning," in *Proc. ACM SIGKDD*, Aug. 2024, pp. 5260–5271.
[5] X. Shen, J. Gao, W. Wu, M. Li, C. Zhou, and W. Zhuang, "Holistic network virtualization and pervasive network intelligence for 6G," *IEEE Commun. Surveys. Tuts.*, vol. 24, no. 1, pp. 1–30, Firstquarter 2022.
[6] J. Shao et al., "WirelessLLM: Empowering large language models towards wireless intelligence," *IEEE J. Commun. Netw.*, vol. 9, no. 2, pp. 99–112, Jun. 2024.
[7] O. A. M. Adekanye, "LLM-powered synthetic environments for self-driving scenarios," in *Proc. AAAI*, Mar. 2024, pp. 23721–23723.
[8] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," in *Proc. 59th Annu. Meeting Assoc. Comput. Linguistics 11th Int, Joint Conf. Natural Lang. Process.*, Jan. 2021, pp. 4582–4597.
[9] X. Shen et al., "AI-assisted network-slicing based next-generation wireless networks," *IEEE Open J. Veh. Technol*, vol. 1, pp. 45–66, 2020.
[10] A. Li et al., "Efficient and privacy-preserving feature importance-based vertical federated learning," *IEEE Trans. Mobile Comput.*, vol. 23, no. 6, pp. 7238–7255, Jun. 2024.
[11] S. Ping, Y. Mao, Y. Liu, X. Zhang, and W. Ding, "FL-TAC: Enhanced fine-tuning in federated learning via low-rank, task-specific adapter clustering," in *Proc. ACM ICLR*, Apr. 2024, pp. 1–6.
[12] W. Wu et al., "AI-native network slicing for 6G networks," *IEEE Wireless Commun.*, vol. 29, no. 1, pp. 96–103, Feb. 2022.
[13] H. Wang and W.-Q. Zhang, "Unstructured pruning and low rank factorisation of self-supervised pre-trained speech models," *IEEE J. Sel. Topics Signal Process.*, vol. 18, no. 6, pp. 1046–1058, Sep. 2024, doi: 10.1109/JSTSP.2024.3433616.
[14] X. Huang, W. Wu, S. Hu, M. Li, C. Zhou, and X. Shen, "Digital twin based user-centric resource management for multicast short video streaming," *IEEE J. Sel. Topics Signal Process.*, vol. 18, no. 1, pp. 50–65, Jan. 2024.
[15] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. ACM ICLR*, Apr. 2021, pp. 1–21.
[16] N. Houlsby et al., "Parameter-efficient transfer learning for NLP," in *Proc. 36th Int. Conf. Mach. Learn.*, May 2019, pp. 2790–2799.
[17] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," in *Proc. 2021 Conf. Empirical Methods Natural Lang. Process.*, Apr. 2021, pp. 3045–3059.
[18] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig, "Towards a unified view of parameter-efficient transfer learning," in *Proc. ACM ICLR*, Oct. 2022, pp. 1–15.
[19] D. Han, M. Choi, J. Park, and J. Moon, "FedMes: Speeding up federated learning with multiple edge servers," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3870–3885, Dec. 2021.
[20] D. Yang et al., "DetFed: Dynamic resource scheduling for deterministic federated learning over time-sensitive networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 5162–5178, May 2024.
[21] T. Guo, S. Guo, J. Wang, X. Tang, and W. Xu, "PromptFL: Let federated participants cooperatively learn prompts instead of models–federated learning in age of foundation model," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 5179–5194, May 2024.
[22] J. Jiang, H. Jiang, Y. Ma, X. Liu, and C. Fan, "Low-parameter federated learning with large language models," in *Proc. Springer WISA*, Aug. 2024, pp. 319–330.
[23] Y.-J. Liu et al., "Ensemble distillation based adaptive quantization for supporting federated learning in wireless networks," *IEEE Trans. Wireless Commun.*, vol. 22, no. 6, pp. 4013–4027, Jun. 2023.
[24] R. Greidi and K. Cohen, "Sparse training for federated learning with regularized error correction," *IEEE J. Sel. Topics Signal Process.*, pp. 1–16, Sep. 2024.
[25] S. Zhang, W. Wu, P. Hu, S. Li, and N. Zhang, "Split federated learning: Speed up model training in resource-limited wireless networks," in *Proc. IEEE 43rd Int. Conf. Distrib. Comput. Syst.*, Jul. 2023, pp. 985–986.
[26] W. Wu et al., "Split learning over wireless networks: Parallel design and resource management," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 4, pp. 1051–1066, Apr. 2023.
[27] C. Xu, J. Li, Y. Liu, Y. Ling, and M. Wen, "Accelerating split federated learning over wireless communication networks," *IEEE Trans. Wireless Commun.*, vol. 23, no. 6, pp. 5587–5599, Jun. 2024.
[28] Y. Liao, Y. Xu, H. Xu, Z. Yao, L. Wang, and C. Qiao, "Accelerating federated learning with data and model parallelism in edge computing," *IEEE/ACM Trans. Netw.*, vol. 32, no. 1, pp. 904–918, Feb. 2024.
[29] Y. Liao, Y. Xu, H. Xu, Z. Yao, L. Huang, and C. Qiao, "ParallelSFL: A novel split federated learning framework tackling heterogeneity issues," in *Proc. ACM MobiCom*, Dec. 2024, pp. 845–860.
[30] H. Wu, X. Chen, and K. Huang, "Device-edge cooperative fine-tuning of foundation models as a 6G service," *IEEE Wireless Commun.*, vol. 31, no. 3, pp. 60–67, Jun. 2024.
[31] B. Ouyang, S. Ye, L. Zeng, T. Qian, J. Li, and X. Chen, "Pluto and charon: A time and memory efficient collaborative edge AI framework for personal LLMs fine-tuning," in *Proc. ACM ICPP*, Aug. 2024, pp. 762–771.
[32] Y. Sun, Z. Li, Y. Li, and B. Ding, "Improving LoRA in privacy-preserving federated learning," Mar. 2024, *arXiv:2403.12313*.
[33] Y. Sun, Z. Li, Y. Li, and B. Ding, "Improving LoRA in privacy-preserving federated learning," in *Proc. ACM ICLR*, pp. 1–17, Apr. 2024.
[34] S. Zhang, W. Wu, L. Song, and X. Shen, "Efficient model training in edge networks with hierarchical split learning," *IEEE Trans. Mobile Comput.*, early access, May 12, 2025, doi: 10.1109/TMC.2025.3569407.
[35] S. Zhang, W. Wu, L. Song, and X. Shen, "Efficient model training in edge networks with hierarchical split learning," in *Proc. IEEE Trans. Mobile Comput.*, May 2025. DOI: 10.1109/TMC.2025.3569407.
[36] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-i.i.d. data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3400–3413, Sep. 2020.
[37] S. Golomb, "Run-length encodings (corresp.)," *IEEE Trans. Inf. Theory*, vol. 12, no. 3, pp. 399–401, Jul. 1966.
[38] C. Liu and J. Zhao, "Resource allocation for stable LLM training in mobile edge computing," in *Proc. ACM MobiHoc*, Oct. 2024, pp. 81–90.
[39] X. Huang, X. Qin, M. Li, C. Huang, and X. Shen, "Adaptive digital twin-assisted 3C management for QoE-driven MSVS: A GAI-based DRL approach," *IEEE Trans. Cogn. Commun. Netw.*, vol. 11, no. 2, pp. 858–872, Apr. 2025, doi: 10.1109/TCCN.2024.3516046.

[40] A. Krizhevsky, G. Hinton, and H. Geoffrey, "Learning multiple layers of features from tiny images," 2009.
[41] A. Krizhevsky, "*Learning Multiple Layers of Features from Tiny Images*," Tech. Rep., University of Toronto, 2009.
[42] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. AAAI*, Apr. 2017, pp. 1273–1282.
[43] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Net. Comp. Appl.*, vol. 116, pp. 1–8, Aug. 2018.
[44] Z. Lin et al., "Splitlora: A split parameter-efficient fine-tuning framework for large language models," 2024, *arXiv:2407.00952*.

Songge Zhang (Graduate Student Member, IEEE) received the B.E. degree in electronic information engineering from Zhengzhou University, Zhengzhou, China, in 2019, and the M.S. degree in traffic information engineering and control from Beihang University, Beijing, China, in 2022. He is currently working toward the Ph.D. degree in communication and information systems with Peking University, Shenzhen, China. His research interests include edge intelligence, network for AI, and network resource management.

Guoliang Cheng (Graduate Student Member, IEEE) received the M.S. degree from the Guangdong University of Technology, Guangdong, China, in 2023. He is currently working toward the Ph.D. degree with the Southern University of Science and Technology, Shenzhen, China, and also with Peng Cheng Laboratory, Shenzhen, China. His research interests mainly focus on deep learning, edge computing and signal detection.

Wen Wu (Senior Member, IEEE) received the B.E. degree in information engineering from the South China University of Technology, Guangzhou, China, and the M.E. degree in electrical engineering from the University of Science and Technology of China, Hefei, China, in 2012 and 2015, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2019. He worked as a Postdoctoral Fellow with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. He is currently an Associate Researcher with the Frontier Research Center, Pengcheng Laboratory, Shenzhen, China. He has authored or coauthored more than 100 refereed IEEE journal and conference papers and six books/book chapters. His research interests include 6 G networks, network intelligence, and network virtualization. He was a Track Co-Chairs for IEEE VTC, and Workshop Co-Chairs for IEEE INFOCOM, IEEE Globecom, and IEEE ICCC. He was also on the Editorial Board for IEEE NETWORKING LETTER, Hindawi WCMC, China Communications, and Springer PPNA. He was the recipient of the World Top 2% Scientist Award 2023-2024, USENIX Security Distinguished Paper Award, IEEE HITC Award for Excellence (Early Career Researcher), and IEEE CIC/ICCC Best Paper Award.

Xinyu Huang (Graduate Student Member, IEEE) received the B.E. degree from the Qian Xuesen Experimental Class with Xidian University, Xi'an, China, in 2018, and the M.S. degree in information and communications engineering from Xi'an Jiaotong University, Xi'an, China, in 2021. He is currently working toward the Ph.D. degree in electrical and computer engineering with the University of Waterloo, Waterloo, ON, Canada. His research interests include digital agents/twins, multimedia communications, and generative AI.

Lingyang Song (Fellow, IEEE) received the Ph.D. degree from the University of York, U.K., in 2007. He worked as a Research Fellow with the University of Oslo, Norway, until rejoining Philips Research, U.K., in 2008. In 2009, he joined the School of Electronics Engineering and Computer Science, Peking University, Beijing, China, and is currently a Boya Distinguished Professor. His main research interests include wireless communications, mobile computing, and machine learning. Dr. Song is the coauthor of many awards, including the IEEE Leonard G. Abraham Prize in 2016, IEEE ICC 2014, IEEE ICC 2015, IEEE Globecom 2014, and the best demo award in ACM MobiHoc 2015. He was the recipient of the National Science Fund for Distinguished Young Scholars in 2017, and the First Prize in the Nature Science Award of the Ministry of Education of China in 2017. He was an IEEE ComSoc Distinguished Lecturer (2015–2018), an Area Editor of IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY (2019–), and is currently the Director of the IEEE Communications Society Asia Pacific Board (2024–2025). He is a Clarivate Analytics Highly Cited Researcher.

Xuemin (Sherman) Shen (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990. He is currently a University Professor with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. He is a registered Professional Engineer of Ontario, ON, Canada, an Engineering Institute of Canada Fellow, a Canadian Academy of Engineering Fellow, a Royal Society of Canada Fellow, a Chinese Academy of Engineering Foreign Member, and an Engineering Academy of Japan International Fellow. His research focuses on network resource management, wireless network security, Internet of Things, 5 G and beyond, and vehicular networks. He was the recipient of the "West Lake Friendship Award" from Zhejiang Province in 2023, the Canadian Award for Telecommunications Research from the Canadian Society of Information Theory (CSIT) in 2021, the R.A. Fessenden Award in 2019 from IEEE, Canada, Award of Merit from the Federation of Chinese Canadian Professionals (Ontario) in 2019, James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, Joseph LoCicero Award in 2015 and Education Award in 2017 from the IEEE Communications Society (ComSoc), and Technical Recognition Award from Wireless Communications Technical Committee (2019), AHSN Technical Committee (2013), the Excellent Graduate Supervision Award in 2006 from the University of Waterloo and the Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada. He was the General Chair for the 6 G Global Conference'23, and ACM Mobihoc'15, Technical Program Committee Chair/Co-Chair for IEEE Globecom'24, '16 and '07, IEEE Infocom'14, and IEEE VTC'10 Fall. He is the Past President of the IEEE ComSoc, the Vice President for Technical & Educational Activities, Vice President for Publications, Member-at-Large on the Board of Governors, Chair of the Distinguished Lecturer Selection Committee, and Member of IEEE Fellow Selection Committee of the ComSoc. He was also the Editor-in-Chief of the IEEE INTERNET OF THINGS JOURNAL, *IEEE Network*, and *IET Communications*.