Split Learning based Cloud-edge-end Collaborative Model Training in Heterogeneous Networks

Jian Wang, Gang Feng, *Senior Member*, *IEEE*, Yi-Jing Liu, Xinyi Xu, Lei Cheng, Wei Jiang, and Li Ping Qian, *Senior Member*, *IEEE*

***Abstract*—Large language models (LLMs) demonstrate significant potential for enabling intelligent endogenous networks owing to their amazing intelligence level. LLMs are currently deployed on cloud servers as their vast parameter scales introduce a substantial computational burden to pre-training and inference processes. This deployment paradigm faces severe challenges, including insufficient personalization, high inference latency, and privacy concerns, as an increasing number of mobile users enjoy the LLM services. To enhance the personalization of pre-trained LLMs, exploiting massive local datasets distributed across edge devices to fine-tune LLMs is essential. In this paper, we propose a split learning-based cloud-edge-end collaborative training framework (SCCT) to harness the abundant computational resources of cloud and edge servers while exploiting massive distributed datasets on edge devices for LLM fine-tuning. In SCCT, an LLM with a parameter-efficient fine-tuning module is deployed on the cloud server, while a small-scale language model (SLM) is deployed across each edge device participating in SCCT and its associated edge server in the manner of split learning. SLM and LLM collaborate in a serial manner for training and inference. To optimize the collaboration efficiency, we formulate a mixed integer nonlinear programming problem to minimize the training latency of SCCT, considering the resource heterogeneity of edge devices and network dynamics. To solve this problem, a two-timescale optimization algorithm is proposed to determine the optimal split points on the large timescale and the allocation of communication and computational resources of the edge server on the small timescale. Numerical results demonstrate that the proposed two-timescale optimization algorithm assisted SCCT outperforms the state-of-the-art LLMs deployment and training paradigm in terms of training efficiency and inference performance.**

***Index Terms*—Collaborative fine-tuning, heterogeneous networks, large language models, split learning.**

I. Introduction

RECENTLY, large language models (LLMs) have made significant breakthroughs, demonstrating impressive intelligence in tasks such as sentiment analysis, multi-turn dialogue generation, programming, and solving mathematical problems [1]–[5]. This amazing intelligence level makes LLMs become a key technology to realize intelligent endogenous networks [4]–[13]. LLMs' intelligence stems from their unprecedented parameter scale, with mainstream models like GPT-3, LLaMA-3.1, and DeepSeek-V3 routinely reaching hundreds of billions of parameters [14], [15]. The massive parameter scale introduces extremely high computational complexity and memory overhead in the training and inference processes, and thus LLMs' pre-training and inference processes can only be conducted on high-performance cloud servers deployed in data centers. Pre-trained LLMs, achieved through centralized training on public datasets, exhibit excellent generalization capabilities. However, pre-trained LLMs often perform poorly on some specific tasks since they lack domain-specific personalized knowledge [16].

Fortunately, the large amount of data generated at edge devices can be used to fine-tune pre-trained LLMs for specific tasks, so as to enhance their personalization capabilities [17], [18]. However, LLMs typically have a tremendous number of parameters, and thus, directly deploying LLMs on an individual edge device with limited memory and computational resources for centralized fine-tuning is infeasible. Federated learning, the most classic distributed learning framework, is also inapplicable to fine-tuning LLMs in resource-heterogeneous networks. Under the federated learning paradigm, clients (i.e., resource-limited edge devices) need to fine-tune LLMs locally and then transmit the model parameters to the server (i.e., cloud server) for parameter aggregation. The local training and parameter transmission overheads in this paradigm are unacceptable for edge devices. To efficiently coordinate the abundant computational resources of cloud servers with massive distributed datasets on edge devices for LLM fine-tuning, it is imperative to design an effective collaborative training framework that enables large-scale edge devices to effectively participate in the distributed fine-tuning of pre-trained LLMs.

Parameter-efficient fine-tuning (PEFT) is the most commonly used fine-tuning approach to personalize pre-trained LLMs [16], [19]–[21]. The core idea of PEFT lies in updating only a small subset of parameters while freezing the majority during the training process, so as to significantly reduce computational overhead. Existing mainstream PEFT paradigms include additive fine-tuning (e.g., Adapter [22]), selective fine-tuning (e.g., BitFit [23]), and reparameterized fine-tuning (e.g., LoRA [24]). PEFT is essentially a centralized training paradigm, i.e., the process of PEFT is executed on a single computing device. In other words, this computing device needs to save all the model parameters before performing PEFT.

This work was supported in part by the National Natural Science Foundation of China under Grant 62401115, in part by the fellowship of China National Postdoctoral Program for Innovative Talents under Grant BX20230057 and in part by the National Natural Science Foundation of China under Grant 62302450. *(Corresponding authors: Gang Feng.)*

Jian Wang, Gang Feng, Yi-Jing Liu, Xinyi Xu, Lei Cheng are with the National Key Laboratory of Wireless Communications, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: jianwang@std.uestc.edu.cn; fenggang@uestc.edu.cn; liuyijing@uestc.edu.cn; xinyixu@std.uestc.edu.cn; leicheng@std.uestc.edu.cn).

Wei Jiang and Li Ping Qian are with the Institute of Cyberspace Security, Zhejiang University of Technology, Hangzhou 310023, China (e-mail: weijiang@zjut.edu.cn; lpqian@zjut.edu.cn).

Split learning (SL) is an efficient decentralized distributed learning paradigm that allows multiple computing devices with limited and heterogeneous resources to train a large-scale neural network (NN) collaboratively [25]–[30]. The core idea of SL lies in splitting a large-scale NN model at specific split points into smaller NN subsets. These subsets are then deployed across a group of computing devices involved in the split learning process. In SL, these computing devices collaboratively execute the training and inference processes in a serial manner. The set of split points is determined on the basis of the available resources of the involved computing devices. Therefore, SL is inherently appropriate for large-scale NN model training in resource-limited heterogeneous networks [31].

PEFT is an efficient approach to enhance the personalization capability of LLMs while hardly affecting their generalization capability. However, directly performing existing PEFT paradigms for LLMs on resource-limited edge devices is still infeasible because of LLMs' tremendous parameter scales. SL allows multiple devices with limited and heterogeneous resources to train a large-scale NN model collaboratively. However, fine-tuning a pre-trained LLM only by using the SL paradigm is not advisable. Under this paradigm, a pre-trained LLM is divided into several sub-models, which are then deployed across corresponding edge devices for full-parameter fine-tuning. As a consequence, this approach may cause overfitting and significantly degrade the generalization ability of the LLM. In addition, due to the *autoregressive decoder* architecture of mainstream LLMs (e.g., ChatGPT, LLaMA, DeepSeek, etc.), directly applying SL for LLM fine-tuning can lead to unbearable communication overhead. Therefore, we need to aptly combine PEFT with SL to exploit the distributed datasets on edge devices for LLM fine-tuning.

In this paper, we propose an SL-based cloud-edge-end collaborative training framework (SCCT) for fine-tuning LLMs. In SCCT, a pre-trained LLM with PEFT module [16], [32] is deployed on the cloud server, while a relatively small-scale language model (SLM) is deployed across each edge device participating in SCCT and its associated edge server by the SL paradigm. For an edge device and its associated edge server, the SLM is split into a device-side sub-model and a server-side sub-model at a specific split point. The split point is determined according to the available resources of this edge device. Then, these two sub-models are deployed on the edge device and edge server, respectively. In SCCT, the SLM can be regarded as a plug-in of the LLM, and the SLM collaborates with the LLM in a serial manner. Through SCCT, a vast number of edge devices can collaboratively fine-tune LLMs for a specific task by exploiting their local datasets. To improve the training efficiency (i.e., average per-iteration latency) of SCCT while accounting for the heterogeneity of edge devices, we propose a two-timescale optimization algorithm for training resource scheduling. The main contributions of this paper are summarized as follows:

- We propose the SCCT to efficiently exploit private data and computational resources on massive edge devices to enhance the personalization of LLMs while preserving their generalization. Furthermore, SCCT allows edge devices to offload their partial computational load to the edge server.
- We propose a two-timescale optimization algorithm to improve the training efficiency of SCCT. In this algorithm, we optimize the split point decision on a large timescale while jointly allocating the computational resources on the edge server and the communication resources between the edge server and edge devices on a small timescale.
- We demonstrate that the two-timescale optimization algorithm-assisted SCCT outperforms existing cloud-only LLMs deployment paradigms in terms of both training efficiency and inference performance via simulations. Furthermore, ablation experiments confirm the effectiveness of our proposed two-timescale optimization algorithm in enhancing the training efficiency of SCCT.

The remainder of the paper is structured as follows. Section II provides a review of related work. In Section III, we present the system model and problem formulation. To optimize the training efficiency of SCCT, we propose a two-timescale optimization algorithm in Section IV. In Section V, we conduct extensive simulation experiments and present numerical results. Finally, we draw conclusions in Section VI.

II. Related Work

Recent research on the deployment of LLMs in wireless networks has been focused mainly on the efficient deployment paradigm [2], [3], [6], [36], [39]–[43], as well as fine-tuning schemes for LLMs [16], [19], [22]–[24], [32], [44]–[56]. As a distributed learning framework with significant potential to enable the distributed deployment of LLMs, split learning [31], [35], [37], [38], [57]–[64] has also gained considerable research attention in recent years.

A. Deployment Paradigms of LLMs in Wireless Networks

As LLMs have tremendous parameter scales, their training and inference processes require enormous computational and memory resources. Therefore, deploying LLMs on high-performance cloud servers is the prevailing deployment paradigm [33]. This deployment paradigm may inevitably lead to insufficient personalization capabilities, high inference latency, and potential privacy leakage [6]. Therefore, pushing LLMs from the cloud to the edge is imperative [6], [43]. However, due to the significantly limited computational and storage resources of edge servers and edge devices compared to cloud servers, it is often difficult to directly deploy LLMs at the edge. Currently, mainstream approaches to pushing LLMs from the cloud to the edge mainly fall into two categories. One involves compressing LLMs through distillation, pruning, and/or quantization to obtain lightweight models that can be deployed on edge servers and devices [21], [34], [42], [43], [65]–[70]. The other employs distributed deployment strategies, such as split learning, to enable LLM functionality across edge environments [35], [37], [59]–[64]. We summarize existing representative LLM deployment paradigms in Table I.

TABLE I
SUMMARY OF DEPLOYMENT PARADIGMS OF LLMS IN NETWORKS

	Deployment Location			Research Focus		Key Technologies			Others
	Cloud server	Edge server	Edge device	Training/Fine-tuning	Inference	Split Learning	PEFT	Model Compression	Resources Heterogeneity
[33]	✔				✔				
[34]		✔	✔	✔	✔		✔	✔	
[35]	✔	✔	✔	✔	✔	✔	✔		
[36]			✔		✔	✔			✔
[37]	✔	✔	✔		✔	✔			
[38]	✔	✔		✔			✔		
Ours	✔	✔	✔	✔	✔	✔	✔		✔

In [65], the authors stated that by applying model compression techniques to lighten LLMs, the compressed models can be deployed on edge devices. Model compression can reduce the number of parameters in LLMs or decrease the computational precision of parameters (e.g., from *float32* to *int8*), thereby significantly decreasing the amount of computational resources required for fine-tuning and inference. The degree of compression can be flexibly adjusted based on the available resources of the edge devices. In [34], the authors proposed deploying LLMs at the network edge through adaptive quantization. However, model compression inevitably leads to performance degradation, and the extent of this degradation increases with the degree of compression.

In [38], the authors proposed a synergy architecture to achieve cloud-edge collaborative inference. In this synergy architecture, a small-scale neural network is deployed on an edge server, while an LLM is deployed on a cloud server. The edge-side neural network and the cloud-side LLM collaborate in a serial manner to offer inference services to clients. This architecture effectively enhances the personalization capabilities of LLM inference, as the neural network deployed on the edge server can be updated in real-time based on the data distributed within its coverage area. However, this architecture does not yet address the privacy concerns because edge devices still need to upload their private data to the edge server for inference.

In [6], the authors proposed a mobile edge computing architecture based on split learning for LLMs in 6G. The authors proposed splitting the LLM into three sub-models and deploying them across edge devices, edge servers, and cloud servers. To be specific, the initial layers (i.e., device-side sub-model), including the input layer, are deployed on the edge devices, the intermediate layers (i.e., edge-side sub-model) are deployed on the edge server, and the remaining layers (i.e., cloud-side sub-model), including the output layer, are deployed on the cloud server. This architecture effectively protects data privacy as the initial inference process is performed on edge devices. However, in this distributed deployment architecture, the cloud server must transmit the edge-side and end-side sub-models to numerous edge servers and edge devices, potentially incurring substantial communication overhead.

In summary, both the currently adopted and proposed LLM deployment paradigms have yet to leverage the computational and data resources of massive edge devices. These data resources have significant value for the personalized fine-tuning of LLMs.

B. Fine-Tuning Methods of LLMs

Centralized pretraining on public datasets allows LLMs to obtain excellent generalization capabilities, while their personalization ability for specific tasks may be relatively weak. Therefore, fine-tuning LLMs for specific tasks based on corresponding datasets is essential [16].

A full parameter fine-tuning scheme was proposed in [44]. This fine-tuning method requires adjusting all parameters of LLMs in the training process, resulting in significant computational and memory overhead. Parameter-efficient fine-tuning (PEFT) is currently the most popular fine-tuning method for LLMs, as it significantly reduces computational overhead. Recent studies on PEFT mainly fall into three categories [16]: additive fine-tuning [22], [45]–[48], selective fine-tuning [23], [49]–[51], [56], and reparameterized fine-tuning [24], [52]–[55]. Among additive fine-tuning approaches, the Adapter method [22] stands out by integrating lightweight modules, i.e., adapters, into specific LLM layers. This architecture strategically restricts parameter adjustments to the inserted adapters during the training process while preserving the original parameters. In contrast, selective fine-tuning techniques like BitFit [23] operate through parameter differentiation. While maintaining most LLM parameters in a fixed state, the method selectively updates only bias terms, achieving remarkable parameter efficiency. As the most representative reparameterized fine-tuning paradigm, LoRA (Low-Rank Adaptation) [24] introduces an innovative fine-tuning approach through matrix factorization. By decomposing weight matrices into low-rank components, LoRA dramatically reduces trainable parameters while maintaining representational capacity, demonstrating exceptional computational efficiency.

However, existing PEFT paradigms are still essentially centralized fine-tuning frameworks, implying that the complete LLM needs to be stored locally before fine-tuning. Thus, the memory and computational overheads of executing existing PEFT paradigms for LLMs locally are too difficult for edge servers and edge devices.

C. Split Learning

Split learning is a classic distributed learning framework that allows multiple resource-constrained devices to collaboratively train models with large-scale parameters. Current research on split learning fall into two categories: one focuses on optimizing the efficiency of split learning in wireless networks [57], [58], [71], while the other explores applying split learning to the distributed deployment of LLMs [35]–[37], [59]–[64].

In [58], the authors improved the efficiency of split learning in wireless networks by jointly optimizing the split points, clustering strategy, and spectrum resource allocation. The authors of [71] proposed combining split learning with federated learning to improve the model's convergence speed. In [57], the authors proposed an adaptive split learning framework that jointly optimizes the split point and resource allocation using Lyapunov optimization methods.

In [36], the authors proposed deploying LLMs across multiple resource-heterogeneous edge devices in the manner of split learning, and they effectively orchestrate the distributed deployment of the LLM layers across the edge devices through matching theory. The authors of [37] introduced a reinforcement learning-based adaptive layer splitting algorithm to improve the performance of wireless LLM inference. In [35], the authors proposed splitting LLMs integrated with LoRA modules into three sub-models, which are deployed on the cloud server, edge servers, and edge devices, respectively. However, the authors of [35] did not take into account the resource heterogeneity of edge devices.

Existing deployment paradigms for LLMs based on split learning directly split the LLMs, which may significantly decrease the LLMs' generalization capability. Additionally, due to the massive parameter scale of LLMs, transmitting the split parts incurs excessive communication overhead.

TABLE II
SUMMARY OF KEY NOTATIONS

Notation	Description
$\mathcal{U}$	Set of the edge devices
N	Number of the edge devices
C_e	Total computational resources (in FLOPS) of edge server
M_e	Total memory resources of edge server
B_e	Total bandwidth between edge server and edge devices
K	Number of subcarriers
B_s	Bandwidth of one subcarrier
C_n	Computational resources (in FLOPS) of edge device u_n
M_n	Memory resources of edge device u_n
L	Number of layers of SLM
s_n	Split point of edge device u_n
w_l	Parameters of l-th layer of SLM
d_{hid}	Hidden dimension of the SLM layer
d_{in}	Length of input data sample
z	Parameter size of an SLM layer in bits
h	Computational complexity (in FLOPs) of an SLM layer
α_n	Proportion of C_e occupied by u_n
β_n	Number of subcarriers occupied by u_n
R	Number of training rounds
T	Number of epochs in one training round
J	Number of iterations in one training epoch

III. System Model and Problem Formulation

In this paper, we consider a three-layer cloud-edge-end network architecture as illustrated in Fig.1. At the top layer, a high-performance cloud server resides in a data center and is connected to an edge server in the middle layer through an optical fiber link. At the bottom layer, a set of edge devices connect to the edge server via wireless links. The set of edge devices is denoted as $\mathcal{U} = \{u_1, u_2, \cdots, u_N\}$, where $N = |\mathcal{U}|$ is the number of edge devices. These edge devices have heterogeneous and limited available resources, including computational, communication, and memory resources.

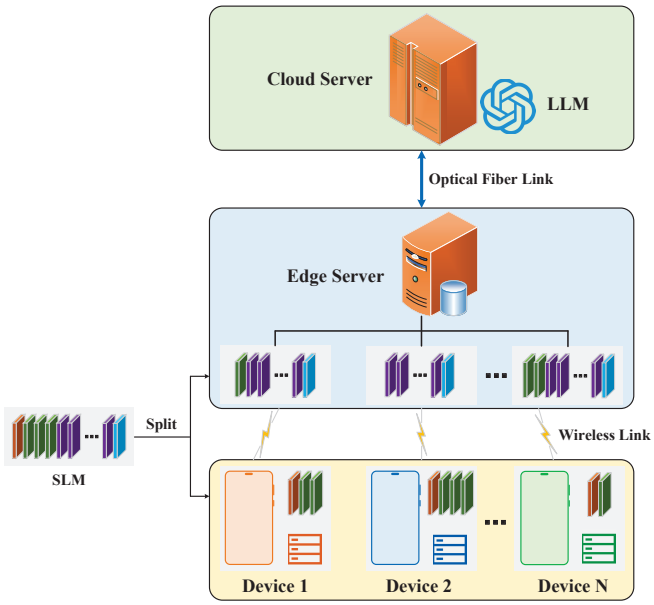

Fig. 1. Cloud-edge-end collaborative training framework.

As shown in Fig.1, an LLM with PEFT module (e.g., Adapter or LoRA) is deployed on the cloud server, while an SLM is deployed across each edge device in $\mathcal{U}$ and edge server in a split learning manner. Specifically, to deploy the SLM in a split learning manner on a device pair comprising an edge device and its associated edge server, the SLM must be split at a specific split point into a device-side sub-model and a server-side sub-model. Then, these two sub-models are deployed on this edge device and the edge server, respectively. The split point may vary across different edge devices, as it is determined based on each device's available resources. This means that any two edge devices may have different locally deployed device-side sub-models, and the corresponding server-side sub-models on the edge server may also be different. Therefore, each edge device reserves a customized device-side sub-model, while the edge server maintains N customized server-side sub-models accordingly.

We assume that the cloud server has sufficient computational and memory resources for fine-tuning pre-trained LLMs, as PEFT requires significantly fewer resources than pretraining. The amount of available computational and memory resources on the edge server is denoted as C_e and M_e, respectively. For $\forall\, u_n \in \mathcal{U}$, its available computational and memory resources are denoted as C_n and M_n, respectively. The communication links between the edge server and all the edge devices share a bandwidth of B_e. We assume that the bandwidth is multiplexed through using Frequency Division Multiple Access (FDMA), which means that the bandwidth B_e is evenly divided into K subcarriers, each with a bandwidth of $B_s = B_e/K$. The edge server dynamically schedules the allocation of subcarriers.

We assume that SLM is based on the Transformer encoder architecture [72], and the SLM consists of L Transformer encoder layers with the same structure and parameter scale. This assumption is reasonable, as most of the current mainstream

SLMs are based on the Transformer encoder architecture. Meanwhile, due to the autoregressive nature of the Transformer decoder architecture, decoder-based language models are not well-suited for integration under the split learning paradigm. Each layer comprises a multi-head attention (MHA) module and a feed-forward neural network (FFN). The hidden dimension of the transformer encoder layer is denoted as d_{hid}. For SLM, all the L layers are separable, and the set of feasible split points is denoted as $\mathcal{L} = \{1, 2, \cdots, L\}$. For any $l \in \mathcal{L}$, the parameter set of the l-th SLM layer is denoted as w_l. From the perspective of model saving and transmission, we denote the parameter size of a single transformer encoder layer by z (in bits). The number of bits for saving or transmitting one parameter is denoted as ξ, it depends on the parameter precision (e.g., *float16*, *float32*). We denote the computational complexity of performing forward propagation on one transformer encoder layer with inputting one data sample by h (in FLOPs). The length of the data sample is denoted as d_{in}. Through rigorous derivation, we obtain the exact expressions for z and h, presented as follows.

$$z = \xi \left(12 d_{hid}^2 + 4 d_{hid}\right), \tag{1}$$

$$h = 24 d_{in} d_{hid}^2 + 4 d_{in}^2 d_{hid}. \tag{2}$$

Proof: See Appendix A.

For $\forall u_n \in \mathcal{U}$, the split point decision variable is denoted as s_n, where $s_n \in \mathcal{L}$. This means splitting the SLM at layer s_n into two sub-models, with the first s_n layers (i.e., device-side sub-model) deployed on edge device u_n and the other $L - s_n$ layers (i.e., server-side sub-model) deployed on the edge server. The parameters saved on u_n and edge server are denoted as $\boldsymbol{w}_n^d = \{w_1, w_2, \cdots, w_{s_n}\}$ and $\boldsymbol{w}_n^e = \{w_{s_n+1}, w_{s_n+2}, \cdots, w_L\}$, respectively. The number of bits required to save or transmit $\boldsymbol{w}_n^d$ and $\boldsymbol{w}_n^e$ are denoted as $\boldsymbol{z}_{s_n}^d = s_n z$ and $\boldsymbol{z}_{s_n}^e = (L - s_n)\, z$, respectively. The computation complexity of forward propagation on $\boldsymbol{w}_n^d$ and $\boldsymbol{w}_n^e$ are denoted as $h_{s_n}^d = s_n h$ and $h_{s_n}^e = (L - s_n)\, h$, respectively.

For $\forall\, u_n \in \mathcal{U}$, u_n needs to occupy a proportion of memory and computational resources of the edge server for saving and training $\boldsymbol{w}_n^e$. The edge server allocates computational resources to u_n in proportion to α_n, where $\alpha_n \in [0, 1]$ and $\sum_{n=1}^{N} \alpha_n \leq 1$. Meanwhile, to transmit smashed data and gradient data between u_n and edge server, we assume that edge server allocates β_n subcarriers to u_n, where $\beta_n \in \{0, 1, 2, \cdots, K\}$ and $\sum_{n=1}^{N} \beta_n \leq K$.

A. Cloud-edge-end Collaborative Training Mechanism

In the aforementioned three-layer network, an LLM with a PEFT module is deployed on the cloud server. For a device pair comprising an edge device in $\mathcal{U}$ and its associated edge server, the SLM is split into a device-side sub-model and a server-side sub-model at a specific split point, which are then deployed on the corresponding sides. Based on this deployment framework, we design an SL-based cloud-edge-end collaborative training (SCCT) mechanism. In SCCT,

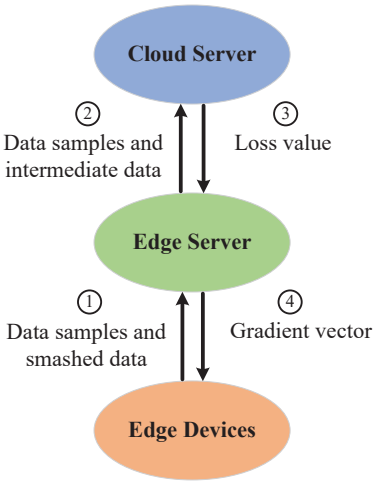

Fig. 2. Simplified workflow of cloud-edge-end collaborative training mechanism.

LLM and SLM perform the training and inference process collaboratively in a serial manner. As shown in Fig.2, in the forward propagation, the SLM executes the initial inference (i.e., feature extraction/data processing), and the LLM performs the remaining inference procedure. The computation order of the backward propagation is reversed, meaning the LLM calculates the gradients first, followed by the SLM.

In the collaborative training process, the LLM performs parameter-efficient fine-tuning by inserting serial adapter layers (i.e., Adapter module). It is important to note that the cloud server maintains a PEFT module for a specific task to enhance the LLM's personalization capability without affecting the generalization of the pre-trained LLM. When an edge server requests LLM's inference service for a specific task, the cloud server inserts the corresponding PEFT module into the LLM and then provides the inference service. Unlike the PEFT used during the training of LLM deployed on the cloud server, the SLMs deployed across the edge server and edge devices adopt a full parameter fine-tuning mode. Because SLM's parameter scale is much smaller than LLM's, its training overhead is affordable for the edge server and edge devices.

We assume the training process consists of R rounds, where each round consists of T epochs. During each epoch, J training iterations are performed, and each iteration includes one forward propagation and one backward propagation. In terms of timescale, a round is relatively large-scale, while an epoch is relatively small-scale. We assume that the computational resources on edge devices, as well as the wireless channel conditions between edge devices and the edge server, vary at the beginning of each epoch but remain unchanged during one epoch. The memory and data resources on edge devices are assumed to remain unchanged throughout the training process. The energy resources of edge devices are assumed to be sufficient for the entire duration of the training. We also assume that the memory and computational resources of the edge server remain unchanged during the training process. Furthermore, we assume that the computational and memory resources of the cloud server are sufficient and remain

unchanged throughout the whole training process. Because the communication link between the edge server and the cloud server is a wired optical fiber link, we assumed that the channel condition and transmission rate are static during the training process.

Considering the aforementioned dynamics of the network, the split point and resource allocation decisions need to be updated periodically to improve training efficiency. To reduce communication overhead, the split point decision is updated at the beginning of each round, while the resource allocation decision is updated at the beginning of each epoch. At the beginning of each round, the edge server needs to determine the split point set $\boldsymbol{s} = \{s_1, s_2, \cdots, s_N\}$. Furthermore, after the split point set is updated, the edge server or edge devices adopt an incremental transmission mode to transmit the parameter set. Taking Fig.3 as an example, the split point of u_n in the r-th round is $s_n^r = 3$, if $s_n^{r+1} = 2$, u_n needs to upload the parameter set w_3 to the edge server, and if $s_n^{r+1} = 4$ the edge server needs to send the parameter set w_4 to u_n. At the beginning of each epoch, the edge server needs to update the joint computational and communication resources allocation, i.e., $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \cdots, \alpha_N\}$ and $\boldsymbol{\beta} = \{\beta_1, \beta_2, \cdots, \beta_N\}$.

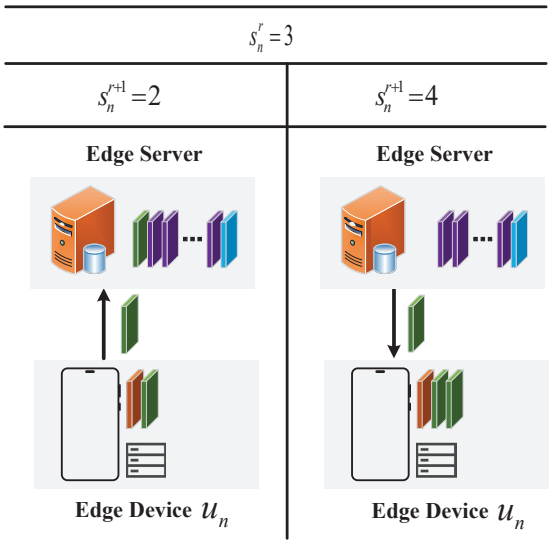

Fig. 3. Example of parameters transmission method when the split point is updated.

To thoroughly present the proposed cloud-edge-end collaborative training mechanism, we further introduce the detailed workflow of the training process in one iteration, as shown in Fig.4. We take the j-th iteration of the t-th epoch in the r-th training round as an example. The training process of the j-th iteration consists of two phases: forward propagation (FP) and backward propagation (BP), which are detailed below.

1) Forward Propagation: For $\forall u_n \in \mathcal{U}$, u_n first samples a batch of training data from its local dataset $\mathcal{D}_n$, where the batch size is denoted as b. The sampled training data and corresponding labels are denoted as $\boldsymbol{x}_n \in \mathbb{R}^{b\times d_{in}\times d_{hid}}$ and $\boldsymbol{y}_n \in \mathbb{R}^{b\times d_l\times d_{hid}}$, respectively. Note that d_l is the length of one label vector. Then u_n inputs the training data into the device-side sub-model with s_n^r layers, obtaining the smashed data $\boldsymbol{v}_n \in \mathbb{R}^{b\times d_{in}\times d_{hid}}$ by

$$\boldsymbol{v}_n = f(\boldsymbol{x}_n; \boldsymbol{w}_n^d(r,t,j)), \tag{3}$$

where $f(\boldsymbol{x}; \boldsymbol{w})$ represents the mapping function between input $\boldsymbol{x}$ and output given parameter set $\boldsymbol{w}$.

Then, u_n uploads the smashed data $\boldsymbol{v}_n$, corresponding labels $\boldsymbol{y}_n$ and training data $\boldsymbol{x}_n$ to the edge server via the allocated β_n subcarriers. The edge server inputs the received smashed data into the server-side sub-model with $L - s_n^r$ layers, obtaining intermediate data $\boldsymbol{m}_n$ by

$$\boldsymbol{m}_n = f(\boldsymbol{v}_n; \boldsymbol{w}_n^e(r,t,j)). \tag{4}$$

Next, the edge server uploads the all intermediate data $\{\boldsymbol{m}_1, \boldsymbol{m}_2, \cdots, \boldsymbol{m}_N\}$, corresponding labels $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_N\}$ and training data $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\}$ to the cloud server.

Finally, for $\forall u_n \in \mathcal{U}$ the cloud server inputs the intermediate data $\boldsymbol{m}_n$ into the LLM, obtaining the predicted results $\hat{\boldsymbol{y}}_n \in \mathbb{R}^{b\times d_l\times d_{hid}}$ by

$$\hat{\boldsymbol{y}}_n = f(\boldsymbol{m}_n; \boldsymbol{w}_{LLM}(r,t,j)), \tag{5}$$

where $\boldsymbol{w}_{LLM}(r,t,j)$ is the parameter set of LLM with PEFT module in the j-th iteration of t-th epoch of r-th training round. Then, for $\forall u_n \in \mathcal{U}$, the cloud server inputs the training data $\boldsymbol{x}_n$ into the LLM, obtaining the predicted results $\widetilde{\boldsymbol{y}}_n \in \mathbb{R}^{b\times d_l\times d_{hid}}$ by

$$\widetilde{\boldsymbol{y}}_n = f(\boldsymbol{x}_n; \boldsymbol{w}_{LLM}(r,t,j)). \tag{6}$$

2) Back Propagation: The cloud server calculates the loss value $\mathbb{L}_n^L$ of the LLM based on the predicted results and the true labels by

$$\mathbb{L}_n^L = f_{loss}(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n), \tag{7}$$

where f_{loss} is the loss function.

Then the cloud server calculates the loss value $\mathbb{L}_n^S$ of the SLM by

$$\mathbb{L}_n^S = \mathbb{L}_n^L - f_{loss}(\boldsymbol{y}_n, \widetilde{\boldsymbol{y}}_n). \tag{8}$$

As shown in Fig. 4, the cloud server performs backward propagation on the LLM, updating the parameters of the PEFT module by

$$\boldsymbol{w}_e^c(r,t,j+1) = \boldsymbol{w}_e^c(r,t,j) - \eta_c \nabla \mathbb{L}(\boldsymbol{w}_e^c(r,t,j)), \tag{9}$$

where $\mathbb{L} = \sum_N^{n=1} \mathbb{L}_n^L$, η_c is the learning rate of PEFT on LLM, and $\boldsymbol{w}_e^c$ is parameter set of PEFT module.

Then, the cloud server transmits the loss value set $\left\{\mathbb{L}_1^S, \mathbb{L}_2^S, \cdots, \mathbb{L}_N^S\right\}$ of SLM to the edge server. Subsequently, the edge server performs backward propagation on its locally deployed $L - s_n^r$ layers server-side sub-model by

$$\boldsymbol{w}_n^e(r,t,j+1) = \boldsymbol{w}_n^e(r,t,j) - \eta_s \nabla \mathbb{L}_n^S(\boldsymbol{w}_n^e(r,t,j)), \tag{10}$$

where η_s is the learning rate of the SLM.

The edge server transmits the gradient vector of the s_n^r-th layer to u_n, and then the backward propagation is conducted on u_n's local s_n^r layers device-side sub-model.

$$\boldsymbol{w}_n^d(r,t,j+1) = \boldsymbol{w}_n^d(r,t,j) - \eta_s \nabla \mathbb{L}_n^S(\boldsymbol{w}_n^d(r,t,j)). \tag{11}$$

After the backward propagation process, the cloud server updates the parameters of the PEFT module embedded in

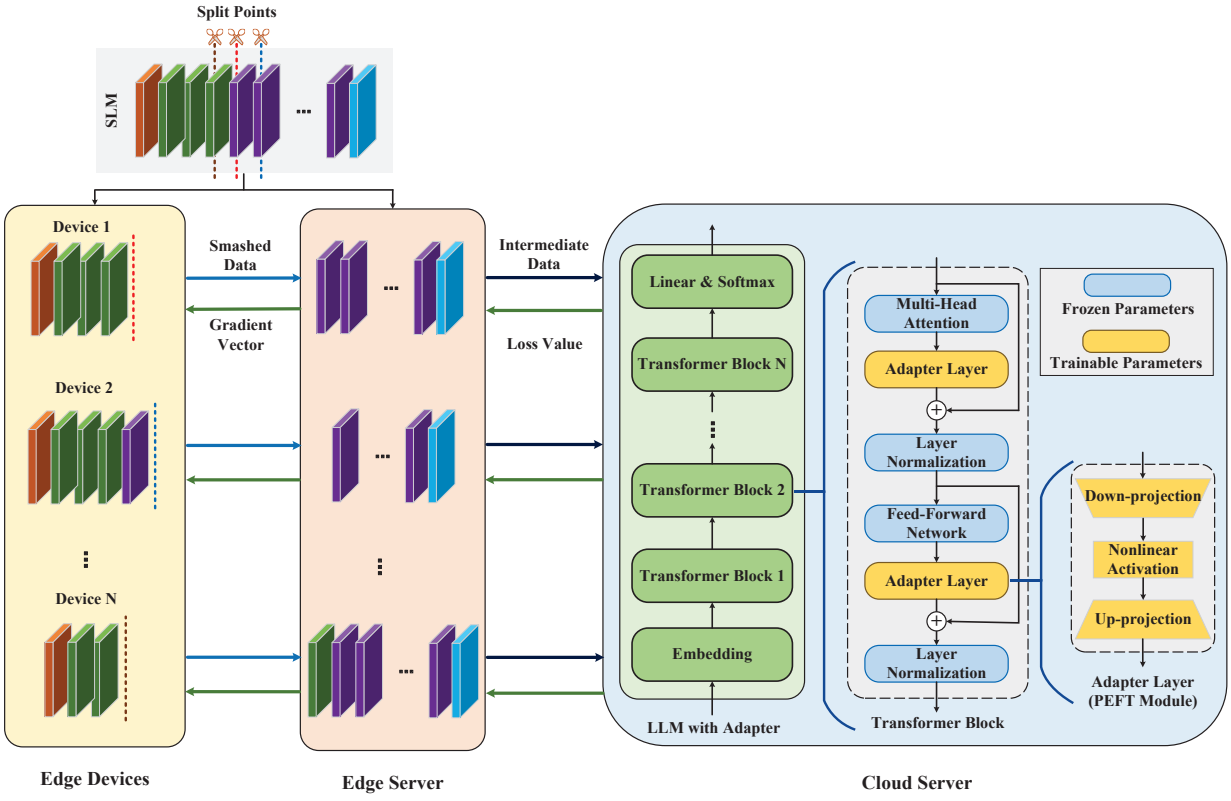

Fig. 4. Illustration of the training process in one round of the proposed cloud-edge-end collaborative training framework.

LLM, while the edge server and edge device update the parameters of SLM.

Considering that edge devices may join or leave the training process dynamically, we introduce the following mechanisms. When an edge device is not participating in training, it locally stores the complete SLM along with a backup copy. Upon joining the training, the edge device splits the locally stored SLM into two parts (i.e., the device-side sub-model and the server-side sub-model) based on the split point determined by its associated edge server. The server-side sub-model is then transmitted to the edge server, after which the device can participate in the training process. At the end of each training round, the edge server sends the updated parameters of the server-side sub-model back to the edge device, which then synchronizes these parameters to its local SLM backup. If the edge device voluntarily withdraws from training, the edge server first transmits the updated server-side sub-model to the device before it exits the training process. In the case of an unexpected or involuntary disconnection, the edge device will restore its local SLM backup as the new SLM and simultaneously generate a new backup copy.

It is worth noting that, for the sake of clarity, a simplified network model with a single edge server is used in this paper. In practice, the proposed SCCT can be easily extended to real-world scenarios involving multiple edge servers.

B. Latency Model

In SCCT, the latency of one training iteration consists of five components. These five components are computational latency on the edge device, transmission latency between the edge device and edge server, computational latency on the edge server, transmission latency between the edge server and cloud server, and computational latency on the cloud server, respectively. In the remainder of this subsection, we derive each component of the training latency for the j-th iteration in the t-th epoch of the r-th round.

1) Computation Latency on Edge Device: For $\forall u_n \in \mathcal{U}$, the total computation latency on u_n consists of FP and BP on the parameter set $\boldsymbol{w}_n^d(r,t,j)$ of its device-side sub-model. Thus the local computation latency of device u_n is

$$d_n^1(r,t,j) = \frac{(1+\epsilon)h_{s_n^r}^d b}{C_n(r,t)}, \tag{12}$$

where ϵ represents the ratio of the computational complexity of BP to the computational complexity of FP on the same parameter set, and $C_n(r,t)$ is the computational resource of u_n in the t-th epoch of the r-th round.

2) Transmission Latency between Edge Device and Edge Server: The transmission latency between u_n and edge server consists of two components. In FP, u_n needs to transmit the smashed data to the edge server. In BP, the edge server needs to transmit the gradient vector to u_n.

The transmission rate from u_n to the edge server is

$$R_n^d(r,t,j) = \beta_n^{r,t} B_s log_2(1 + \frac{G_n P_n}{N_0 \beta_n^{r,t} B_s}), \tag{13}$$

where G_n is channel gain, P_n is the transmit power of u_n, N_0 is the power spectral density of Gaussian white noise [73].

The transmission rate from the edge server to u_n is

$$R_n^e(r,t,j) = \beta_n^{r,t} B_s log_2(1 + \frac{G_n P_e}{N_0 \beta_n^{r,t} B_s}), \tag{14}$$

where P_e is the transmit power of the edge server.

Thus, the transmission latency between u_n and the edge server is

$$d_n^2(r,t,j) = \frac{z^s(I_{s_n^r}, b)}{R_n^d(r,t,j)} + \frac{z^g(O_{s_n^r})}{R_n^e(r,t,j)}, \tag{15}$$

where $z^s(I_{s_n^r}, b)$ is the size of smashed data in bits, and $z^g(O_{s_n^r})$ is the size of gradient vector in bits.

3) Computation Latency on Edge Server: In the j-th iteration of the t-th epoch of the r-th training round, the amount of computational resources that edge server allocates to u_n is $\alpha_n^{r,t} C_e$. Thus the computation latency on edge server for $\boldsymbol{w}_n^e(r,t)$ is

$$d_n^3(r,t,j) = \frac{(1+\epsilon) h_{s_n^r}^e b}{\alpha_n^{r,t} C_e}. \tag{16}$$

4) Transmission Latency between Edge Server and Cloud Server: In FP, after the edge server completes the forward propagation on N server-side sub-models, the edge server transmits the intermediate data, training data, and corresponding labels to the cloud server. In BP, the cloud server transmits N loss values to the edge server. The transmission rate between the cloud server and the edge server is denoted as $R_{c,e}$

Thus, the transmission latency between the edge server and the cloud server is

$$d^4(r,t,j) = \frac{z^e(N,b) + z^c(N)}{R_{c,e}}, \tag{17}$$

where $z^e(N,b)$ is the size of the data transmitted by the edge server in bits, and $z^g(N)$ is the size of N loss values in bits.

5) Computation Latency on Cloud Server: Generally, we consider the computation resources of cloud servers located in data centers to be sufficient for PEFT, with extremely low computation latency that can be considered negligible.

In addition, at the beginning of $\forall r \in \{1, 2, \cdots, R\}$ training round, the edge device or edge server may need to transmit the parameter set of several layers due to the update of the split point decision. The latency of parameter set transmission is

$$d_n^0(t) = \begin{cases} \boldsymbol{z}_{s_n^{t-1}, s_n^t} / R_n^e(t) & \text{if } s_n^{t-1} < s_n^t, \\ \boldsymbol{z}_{s_n^t, s_n^{t-1}} / R_n^d(t) & \text{if } s_n^{t-1} > s_n^t, \\ 0 & \text{if } s_n^{t-1} = s_n^t. \end{cases} \tag{18}$$

We denote the train latency of the j-th epoch of t-th round as $D^{r,t}$. And we have

$$D^{r,t} = I \cdot \Bigg[d^4(r,t,j) + \max_n \Big(d_n^1(r,t,j) + d_n^2(r,t,j) + d_n^3(r,t,j) \Big) \Bigg] \tag{19}$$

Thus the whole latency in r-th round is

$$\boldsymbol{D}^r = \max_n d_n^0(r) + \sum_{t=1}^{T} D^{r,t}. \tag{20}$$

A summary of key notations is presented in Table I.

C. Problem Formulation

With aim to reduce the training latency of the cloud-edge-end collaborative training mechanism, we need to optimize the split point decision and resource allocation. The split point decision consists of N split points of edge devices $\mathcal{U}$, which is denoted as $\mathcal{S} = \{s_1, s_2, \cdots, s_N\}$. Resource allocation refers to the allocation of available computing resources and bandwidth of the edge server to its associated N edge devices for parameter transmission and collaborative model training. The set of computational resource allocation variable is denoted as $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \cdots, \alpha_N\}$. The set of communication resource allocation variable is denoted as $\boldsymbol{\beta} = \{\beta_1, \beta_2, \cdots, \beta_N\}$. The optimization problem can be formulated as follows

$$\mathbf{P1}: \min_{\boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{\beta}} \sum_{r=1}^{R} \boldsymbol{D}^r \tag{21}$$

$$\text{s.t. } \alpha_n \in [0,1], \forall u_n \in \mathcal{U} \tag{21-1}$$

$$\sum_{n=1}^{N} \alpha_n \leq 1, \tag{21-2}$$

$$\beta_n \in \{0, 1, 2, \cdots, K\}, \tag{21-3}$$

$$\sum_{n=1}^{N} \beta_n \leq K, \tag{21-4}$$

$$s_n \in \mathcal{L}, \forall u_n \in \mathcal{U}, \tag{21-5}$$

$$\boldsymbol{z}_{s_n}^d \leq M_n, \forall u_n \in \mathcal{U}, \tag{21-6}$$

$$\sum_{n=1}^{N} \boldsymbol{z}_{s_n}^e \leq M_e. \tag{21-7}$$

Problem 21 is a mixed-integer nonlinear programming problem. The optimization objective is to minimize the maximal training latency. Constraints 21-1 and 21-2 pertain to the computational resource constraints of the edge server. Constraints 21-3 and 21-4 present the communication resource constraints between the edge server and end devices. Constraint 21-5 defines the feasible split point constraints. Constraint 21-6 states the memory constraints of edge devices, while constraint 21-7 states the memory constraints of the edge server. Taking into account the dynamic nature of device resources and wireless channels, split point decision $\boldsymbol{s}$, computational resources allocation decision $\boldsymbol{\alpha}$, and spectrum resources allocation decision $\boldsymbol{\beta}$ need to be updated periodically. 21 can be considered as

a two-timescale optimization problem, because $\boldsymbol{s}$ is updated at the beginning of each training round while $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are updated at the beginning of each epoch.

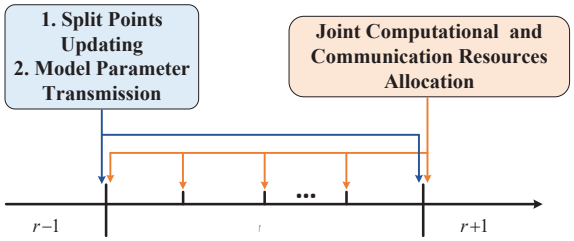

Fig. 5. Illustration of the two-timescale optimization algorithm.

IV. Two-timescale Optimization Algorithm

To solve problem **P1**, we design a two-timescale optimization algorithm to minimize training latency. As shown in Fig.5, we optimize the split point decision $\boldsymbol{s}$ on a large timescale (i.e., one round), while we optimize the joint allocation of computational resources $\boldsymbol{\alpha}$ and bandwidth $\boldsymbol{\beta}$ on a small timescale (i.e., one epoch). It is worth noting that the proposed two-timescale optimization algorithm is deployed on the edge server, as the edge server possesses more abundant computational resources and has access to specific information about its associated edge devices.

Algorithm 1: Joint Optimize Computational and Communication Resources Allocation

Input: Edge devices' computational resources and channel conditions
Output: Optimal $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$

Initialize the subcarrier allocation $\boldsymbol{\beta} = [1, 1, \cdots, 1]$;
for $iteration = 1$ *to* I **do**
  ▷ **Communication Resources Allocation**
  for $iteration = 1, 2, \cdots, K - N$ **do**
    $\Phi = D(\boldsymbol{s}^*, \boldsymbol{\alpha}', \boldsymbol{\beta})$;
    for $n = 1, 2, \cdots, N$ **do**
      $\hat{\beta}_n = \beta_n + 1$;
      $\hat{\boldsymbol{\beta}}_n = \left\{\beta_1, \beta_2, \cdots, \hat{\beta}_n, \cdots, \beta_N\right\}$;
      $\Phi_n = D(\boldsymbol{s}^*, \boldsymbol{\alpha}, \hat{\boldsymbol{\beta}}_n)$;
    end
    $n_* = argmax_n\{\Phi - \Phi_n\}$;
    $\boldsymbol{\beta} = \hat{\boldsymbol{\beta}}_{n^*}$;
  end
  ▷**Computational Resources Allocation**
  Initialize a solution $\boldsymbol{\alpha}_0$;
  repeat
    Based on the *KKT* conditions of (25) and *Newton's method*, calculating the searching direction $\Delta\boldsymbol{\alpha}$;
    Update the solution by $\boldsymbol{\alpha}_{k+1} = \boldsymbol{\alpha}_k + \zeta\Delta\boldsymbol{\alpha}$;
  until stop-criterion met;
end

A. Small Timescale: Joint Computational and Communication Resources Allocation

Considering the dynamic nature of the network, it is necessary to jointly adjust the allocation of computational and communication resources at the beginning of each epoch to minimize the training latency. To solve this joint optimization problem, we further decompose it into a communication resource allocation subproblem and a computational resource allocation subproblem. Then, by iteratively solving these two subproblems, we obtain the optimal joint resource allocation scheme.

1) Communication Resources Allocation Subproblem: With a given computational resources allocation scheme $\boldsymbol{\alpha} = \boldsymbol{\alpha}'$, the communication resources allocation subproblem can be presented as follows.

$$\begin{aligned} \mathbf{P1-1}: \min_{\boldsymbol{\beta}} \ & D^{r,t} \\ \text{s.t. } & (21-3), \\ & (21-4). \end{aligned} \tag{22}$$

This is a simple combinatorial optimization problem that can be solved using a low-complexity greedy algorithm. First, we set the initial value of the optimization variable $\boldsymbol{\beta}$ as $[1, 1, \cdots, 1] \in \mathbb{R}^{1\times N}$, meaning that each edge device is initially allocated one subcarrier. Then, the remaining $K - N$ subcarriers are allocated using a greedy approach, where each subcarrier is assigned sequentially to the edge device with the highest marginal gain.

2) Computational Resource Allocation Subproblem: With a given communication resources allocation scheme $\boldsymbol{\beta} = \boldsymbol{\beta}'$, the computational resources allocation subproblem can be presented as follows.

$$\mathbf{P1-2}: \min_{\boldsymbol{\alpha}} \left\{\max_n \left[\frac{(1+\epsilon)\boldsymbol{h}^e_{s^r_n} b}{\alpha_n C_e} + \left(d^1_n + d^2_n\right)\right]\right\} \tag{23}$$

$$\text{s.t. } \alpha_n \le 1, n = 1, 2, \cdots, N, \tag{23-1}$$

$$-\alpha_n \le 0, n = 1, 2, \cdots, N, \tag{23-2}$$

$$\sum_{n=1}^{N} \alpha_n \le 1. \tag{23-3}$$

Obviously, the problem **P1-2** is a minimax problem. To simplify this problem, we introduce an auxiliary variable V. We assume that $V \ge (1+\epsilon)\boldsymbol{h}^e_{s^r_n} b/\alpha_n C_e + d^1_n + d^2_n, \forall n \in \{1, 2, \cdots, N\}$. Thus, problem **P1-2** can be transformed into

$$\mathbf{P1-3}: \min_{\boldsymbol{\alpha}, V} \ V \tag{24}$$

$$\text{s.t. } \alpha_n \le 1, n = 1, 2, \cdots, N, \tag{24-1}$$

$$-\alpha_n \le 0, n = 1, 2, \cdots, N, \tag{24-2}$$

$$\sum_{n=1}^{N} \alpha_n \le 1, \tag{24-3}$$

$$\frac{(1+\epsilon)\boldsymbol{h}^e_{s^r_n} b}{\alpha_n C_e} + d^1_n + d^2_n \le V, n = 1, 2, \cdots, N. \tag{24-4}$$

This is a nonlinear programming problem with a continuous optimization variable space. It can be solved by using *Lagrange Multiplier Method*. The *Lagrange* function of problem **P1-3** is

$$\begin{aligned}\mathcal{L}(\boldsymbol{\alpha},\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\varepsilon},\xi,V) = M + \xi\left(\sum_{n=1}^{N}\alpha_n - 1\right) + \\ \sum_{n=1}^{N}\lambda_n\left(\alpha_n - 1\right) + \sum_{n=1}^{N}\mu_n\left(\alpha_n - 1\right) + \\ \sum_{n=1}^{N}\varepsilon_n\left(\frac{(1+\epsilon)\boldsymbol{h}_{s_n^r}^e b}{\alpha_n^{r,t}C_e} + d_n^1 + d_n^2 - V\right).\end{aligned} \tag{25}$$

Then we can obtain the *KKT conditions*. By iteratively solving and using Newton's method to update the search direction during the solving process, the optimal solution $\boldsymbol{\alpha}^*$ can be obtained.

The joint communication and computation resources allocation algorithm is present in Algorithm 1.

B. Large Timescale: Split Point Optimization Algorithm

The update of the split point decision involves parameter transmission, which may result in excessive communication overhead. Thus, we update the split point decision at the beginning of each round to minimize the training latency. Because the computational resources $\boldsymbol{C} = \{C_1, C_2, \cdots, C_N\}$ and wireless channel gain $\boldsymbol{G} = \{G_1, G_2, \cdots, G_N\}$ are independent in different rounds, the split point optimization subproblem can be formulated as follows.

$$\begin{aligned}\mathbf{P1-5}:\ \min_{\boldsymbol{s}}\ &\left\{\max_n d_n^0(r) + \sum_{t=1}^{T} D^{r,t}\right\} \\ \text{s.t. }&(21-5),\\ &(21-6),\\ &(21-7).\end{aligned} \tag{26}$$

The objective function in problem **P1-5** can be approximated by

$$\max_n d_n^0(r) + \sum_{t=1}^{T} D^{r,t} \approx \max_n d_n^0(r) + T\mathbb{E}_{\boldsymbol{C},\boldsymbol{G}}[D(\boldsymbol{s},\boldsymbol{\alpha},\boldsymbol{\beta})], \tag{27}$$

where $\mathbb{E}_{\boldsymbol{C},\boldsymbol{G}}[D(\boldsymbol{s},\boldsymbol{\alpha},\boldsymbol{\beta})]$ represents the average per-epoch training latency.

Then, we adopt a sampling-based average approximation approach to approximate the $\mathbb{E}_{\boldsymbol{C},\boldsymbol{G}}[D(\boldsymbol{s},\boldsymbol{\alpha},\boldsymbol{\beta})]$. This approach is based on the *Law of Large Numbers*, which means that the sample average of a random variable can be used to approximate its mathematical expectation. Therefore, we extract a batch of computational resources and channel gain data samples from the historical records, where the batch size is denoted as Q. Given a specified split point decision $\boldsymbol{s}$ and any data sample $(\boldsymbol{C}_q, \boldsymbol{G}_q)$, the optimal resource allocation decision $(\boldsymbol{\alpha}_q^*, \boldsymbol{\beta}_q^*)$ can be obtained by using Algorithm 1. Then the approximate average latency can be derived as follows. When the value of Q is large enough, such an approximation is valid.

$$\mathbb{E}_{\boldsymbol{C},\boldsymbol{G}}[D(\boldsymbol{s},\boldsymbol{\alpha},\boldsymbol{\beta})] \approx \frac{1}{Q}\sum_{q=1}^{Q} D(\boldsymbol{s},\boldsymbol{\alpha}_q^*,\boldsymbol{\beta}_q^*). \tag{28}$$

We can replace the optimization objective of problem **P1-5** by using 28, then the problem **P1-5** can be transformed into:

$$\begin{aligned}\mathbf{P1-6}:\ \min_{\boldsymbol{s}}\ &\left\{\max_n d_n^0(r) + \frac{T}{Q}\sum_{q=1}^{Q} D(\boldsymbol{s},\boldsymbol{\alpha}_q^*,\boldsymbol{\beta}_q^*)\right\} \\ \text{s.t. }&(21-5),\\ &(21-6),\\ &(21-7).\end{aligned} \tag{29}$$

For this complex combinatorial optimization problem **P1-6**, the *adaptive large neighborhood search* (ALNS) algorithm [74], a classic heuristic algorithm, can be employed to address it. The core idea of ALNS is to generate new solutions by applying a pair of destroy and repair operators to the current solution, and explore the solution space by probabilistically accepting worse solutions based on the simulated annealing criterion. Additionally, ALNS improves the efficiency of solution space exploration by adaptively updating the weights of the destroy and repair operators during the search process. The split point optimization algorithm based on ALNS is presented as Algorithm 2. Below, we briefly introduce the key notations and the algorithmic procedure used in Algorithm 2.

In Algorithm 2, the sets of destroy and repair operators are denoted by $\Psi^- = \{\psi_1^-, \psi_2^-, \cdots, \psi_P^-\}$ and $\Psi^+ = \{\psi_1^+, \psi_2^+, \cdots, \psi_{P'}^+\}$ respectively. The corresponding weight sets are denoted by Θ^- and Θ^+, while the corresponding score sets are denoted by Ξ^- and Ξ^+, respectively. The number of times each operator is selected is tracked by the sets Φ^- and Φ^+ respectively. The notations involved in simulated annealing include the initial temperature δ_s, the terminal temperature δ_e, the current temperature δ, and the cooling rate ρ. The feasible solution sets of split points used in Algorithm 2 include the initial solution $\boldsymbol{s_0}$, the current solution $\boldsymbol{s}$, the best-known solution $\boldsymbol{s}^*$, and the newly generated solution $\boldsymbol{s}'$. Additionally, the feasible score set used in the scoring process is denoted by $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, where $\sigma_1 > \sigma_2 > \sigma_3 > \sigma_4$.

Algorithm 2 is based on a loop structure, with the termination condition being that the current temperature drops to the terminal temperature. Each iteration of the loop consists of four steps. First, a pair of destroy-and-repair operators is selected by using a roulette wheel mechanism and applied to the current solution to generate a new solution. Second, Algorithm 2 determines whether to accept the new solution based on a specific criterion. Third, a score is assigned to the selected pair of operators based on whether the new solution was accepted in the previous step. Finally, Algorithm 2 updates relevant parameters.

V. Simulation Setting and Numerical Results

In this section, we conduct simulation experiments to validate the effectiveness of the proposed cloud-edge-end collaborative training framework SCCT and the two-timescale optimization algorithm.

Algorithm 2: Split Point Optimization Algorithm

Input: $\boldsymbol{s_0}$, Ψ^-, Ψ^+, Θ^-, Θ^+,Ξ^-,Ξ^+,τ,ρ, δ_s,δ_e
Output: $\boldsymbol{s}^*$

Initialize $\boldsymbol{s} = \boldsymbol{s_0}$, $\boldsymbol{s}^* = \boldsymbol{s_0}$, $i = 1$, $\delta = \delta_s$;
Initialize $\Theta^- = \Theta^+ = \mathbf{1}$, $\Xi^- = \Xi^+ = \mathbf{0}$;
while $\delta > \delta_e$ **do**
  ▷ **Modify the current solution**
  Select a pair of destroy-and-repair operators $(\psi_p^-, \psi_{p'}^+)$ from Ψ^- and Ψ^+ by using roulette wheel selection ;
  $\phi_p^- = \phi_p^- + 1$, $\phi_{p'}^+ = \phi_{p'}^+ + 1$;
  Obtain the new solution $\boldsymbol{s}' = \psi_{p'}^+(\psi_p^-(\boldsymbol{s}))$;
  ▷ **Evaluate the new solution**
  if $D(\boldsymbol{s}') < D(\boldsymbol{s})$ **then**
    $\boldsymbol{s} = \boldsymbol{s}'$;
    if $D(\boldsymbol{s}') < D(\boldsymbol{s}^*)$ **then**
      $\boldsymbol{s}^* = \boldsymbol{s}'$;
    end
  end
  else if $rand[0,1] < e^{-((D(\boldsymbol{s}') - D(\boldsymbol{s}))/\delta)}$ **then**
    $\boldsymbol{s} = \boldsymbol{s}'$;
  end
  ▷ **Score the selected operators**
  if $\boldsymbol{s}' == \boldsymbol{s}^*$ **then**
    $\xi_p^- = \xi_p^- + \sigma_1$, $\xi_{p'}^+ = \xi_{p'}^+ + \sigma_1$;
  end
  else if $\boldsymbol{s}' == \boldsymbol{s}$ ***and*** $D(\boldsymbol{s}') < D(\boldsymbol{s})$ **then**
    $\xi_p^- = \xi_p^- + \sigma_2$, $\xi_{p'}^+ = \xi_{p'}^+ + \sigma_2$;
  end
  else if $\boldsymbol{s}' == \boldsymbol{s}$ ***and*** $D(\boldsymbol{s}') \geq D(\boldsymbol{s})$ **then**
    $\xi_p^- = \xi_p^- + \sigma_3$, $\xi_{p'}^+ = \xi_{p'}^+ + \sigma_3$;
  end
  else
    $\xi_p^- = \xi_p^- + \sigma_4$, $\xi_{p'}^+ = \xi_{p'}^+ + \sigma_4$;
  end
  ▷ **Update parameters**
  if $i == \tau$ **then**
    $\xi_p^- = \xi_p^- / \phi_p^-, \forall \psi_p^- \in \Psi^-$;
    $\xi_{p'}^+ = \xi_{p'}^+ / \phi_{p'}^+, \forall \psi_{p'}^+ \in \Psi^+$;
    Set $\Xi^- = \Xi^+ = \mathbf{0}$;
    Set $\Phi^- = \Phi^+ = \mathbf{0}$;
    $i = 0$;
  end
  $\delta = \rho\delta$, $i = i + 1$;
end
return $\boldsymbol{s}^*$.

A. Simulation Setting

To implement the SCCT, we use RoBERTa-Large [75] as the small language model (SLM) deployed across edge devices and the edge server, while Llama2-70B, equipped with inserted Adapter modules, serves as the large language model (LLM) deployed on the cloud server. RoBERTa-Large is mainly responsible for enriching/refining the input text based on users' personalized characteristics, while Llama2-70B performs inference and generates responses based on the refined text. The dataset used for the simulation is SQuAD [76], a classic question-answering dataset which is appropriate for personalized fine-tuning. The main parameters used in the simulations are summarized in Table III.

TABLE III
PARAMETERS OF SIMULATION

Parameter	Value
N	[10, 20, 30]
$\mathcal{L}$	[1,2,3,4,5,6,7,8,9,10,11,12]
P_d	$23dBm$
G_d	$5dBi$
B_s	$2MHz$
N_0	$-174dBm/Hz$
P_e	$50dBm$
G_e	$20dBi$
C_e	$312TFLOPS$
M_e	$6TB$
$R_{c,e}$	$1000Mbps$
K	200
R	100
T	60
J	60
η	0.0001

For simplicity, we abbreviate SCCT enhanced by the two-timescale optimization algorithm as SCCTT. We employ the following three fine-tuning mechanisms as comparison references.

- *SCCTL*: We adopt the SCCT framework, and evenly allocate the communication and computational resources of the edge server among the edge devices. Then the split points are optimized by Algorithm 2 (i.e., large timescale optimization algorithm).
- *SCCTS*: We adopt the SCCT framework, set the split points of all edge devices to the same value, and keep them fixed throughout the training process. Then the joint allocation of the communication and computational resources of the edge server is optimized by Algorithm 1 (i.e., small timescale optimization algorithm).
- *CloudO*: Fine-tuning LLM only on the cloud, i.e., the centralized deployment and training paradigm.

SCCTT, SCCTL, and SCCTS are all based on the proposed SCCT framework but differ in split points optimization and resource allocation strategies. CloudO represents the most common LLM deployment architecture and serves well as a baseline. The effectiveness of the small-timescale optimization algorithm is validated by comparing the performance of SCCTT and SCCTL, while the effectiveness of the large-timescale optimization algorithm is assessed through the comparison between SCCTT and SCCTS. Moreover, the effectiveness of the proposed cloud-edge-end collaborative training framework SCCT, is demonstrated by comparing SCCTT with CloudO.

B. Numerical Results

1) Convergence of the Two-timescale Algorithm: Given that the proposed small-timescale optimization algorithm follows an alternating optimization framework, while the large-timescale optimization algorithm adopts a heuristic approach (i.e., ALNS), verifying their convergence is essential.

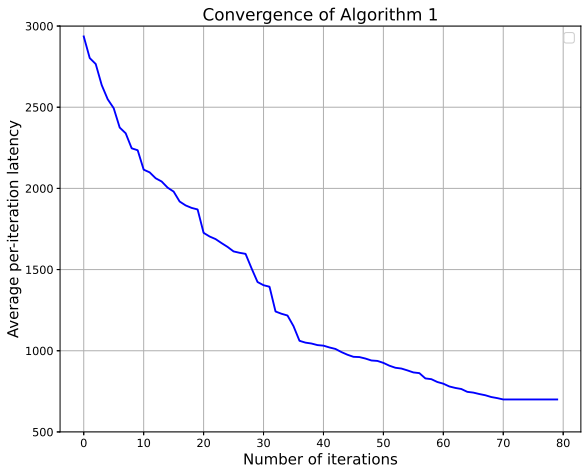

Fig. 6. Convergence of Algorithm 1.

In the first experiment, we examine the convergence of the small-timescale optimization algorithm. The numerical results are also shown in Fig.6, where the horizontal axis represents the number of iterations, and the vertical axis represents the average per-iteration training latency. It can be observed that the small-timescale optimization algorithm exhibits a decreasing trend and reaches convergence after 70 iterations. This means that a better joint communication and computation resource allocation decision is made after each iteration, which reduces the objective function value (i.e., the average per-iteration latency). This numerical result demonstrates the convergence of the proposed small-timescale optimization algorithm.

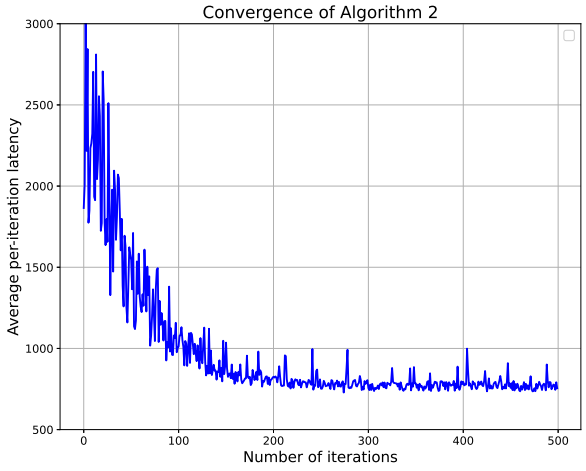

Fig. 7. Convergence of Algorithm 2.

In the second experiment, we verify the convergence of the large-timescale optimization algorithm. The numerical results are shown in Fig.7, with the same axis definitions as the first experiment. It can be observed that the large-timescale optimization algorithm exhibits an oscillatory convergence trend and reaches convergence at approximately 200 iterations. This numerical result demonstrates the convergence of the proposed large-timescale optimization algorithm.

2) Performance of SCCTT: After verifying the asymptotic optimality of the proposed two-timescale optimization algorithm, we further conduct the following four simulation experiments to examine the performance of the SCCTT.

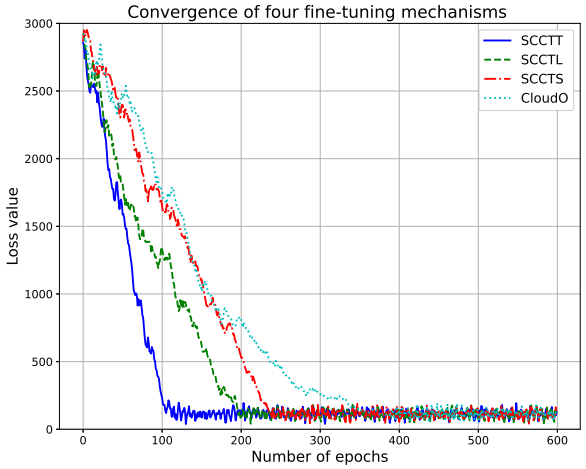

Fig. 8. Convergence of four fine-tuning mechanisms.

In the third experiment, we investigated the convergence behavior of four fine-tuning mechanisms. The numerical results are shown in Figure.8. SCCTT demonstrates the fastest convergence, followed by SCCTL, then SCCTS, while CloudO converges the slowest. Compared to deploying the LLM solely in the cloud, the SCCT architecture deploys SLM across the edge server and edge devices, thereby enhancing the ability to acquire knowledge from specific datasets. As a result, all three fine-tuning mechanisms based on the SCCT framework achieve faster convergence than CloudO. Due to the efficient resource utilization enabled by the two-timescale optimization algorithm, SCCTT outperforms SCCTL and SCCTS in training efficiency, resulting in the fastest convergence overall.

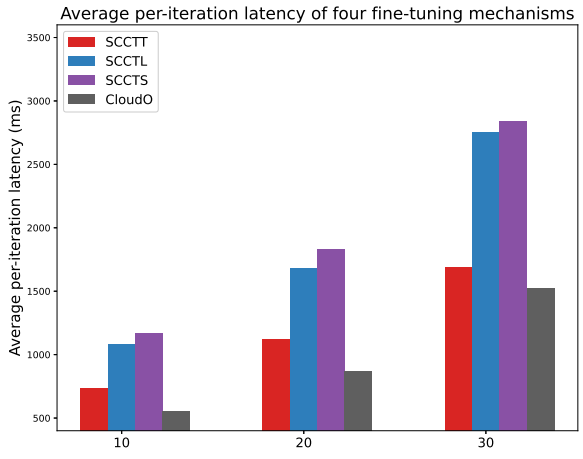

Fig. 9. Average per-iteration training latency of four fine-tuning mechanisms under different numbers of edge devices.

In the fourth experiment, we investigate the average per-iteration latency of the four fine-tuning mechanisms under different numbers of edge devices (i.e., $N = 10/20/30$). The purpose of this experiment is to investigate how the number

of edge devices affects training efficiency and to compare the training efficiency across the fine-tuning mechanisms. The numerical results are shown in Fig.9. We can see that the average per-iteration latency also increases with the number of edge devices. This is because a larger number of devices results in fewer resources allocated to each device, leading to increased latency. Within each group of bar charts, we observe that CloudO exhibits the lowest latency since it does not involve edge devices or edge servers in computation, relying solely on the cloud server for fine-tuning. SCCTT has the second-lowest latency, lower than SCCTL and SCCTS, due to its highest resource utilization.

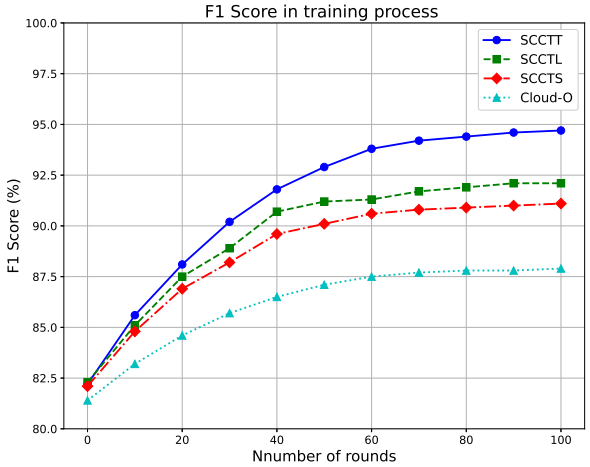

Fig. 10. *F1 Score* of four fine-tuning mechanisms during the training process.

In the fifth experiment, we use *F1 Score* [77] as the metric to evaluate the comprehensive performance of four fine-tuning mechanisms during the training process. The F1 score is the harmonic mean of precision and recall based on the word-level overlap between the predicted answers and the reference answers. The numerical results are shown in Fig.10. We can observe that the *F1 Score* gradually increases with the number of training rounds. The SCCTT algorithm achieves the highest *F1 Score*, followed by SCCTL, then SCCTS, and finally CloudO. This means that the SCCTT has the best comprehensive performance during the training process. This is because the RoBERTa in SCCT enhances the personalization capability of Llama in the fine-tuning process. Meanwhile, the two-timescale optimization algorithm effectively mitigates the effects of heterogeneity and dynamics of networks by adjusting the split points on large timescale as well as adjusting the resource allocation on small timescale, thus improving the training efficiency.

In the sixth experiment, we use *Exact Match* (EM) as the metric to examine the strict test accuracy during training for the four fine-tuning mechanisms. For a test set, EM is defined as the percentage of fully correct predicted samples out of the total number of samples [78]. EM is a stricter metric than the F1 score, as it only counts the proportion of fully correct predicted samples. When evaluating model performance, EM is typically used as a complementary metric of *F1 Score*. The numerical results are shown in Fig.11. We can observe that the

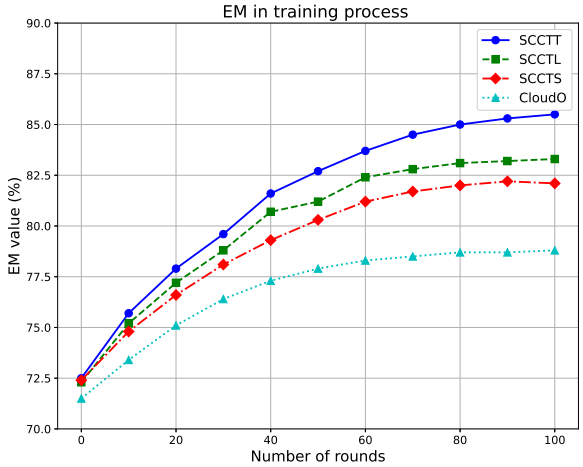

Fig. 11. EM value of four fine-tuning mechanisms during the training process.

EM value also gradually increases with the number of training rounds. With the same results as in the fourth experiment, the SCCTT algorithm achieves the highest EM value, followed by SCCTL, then SCCTS, and finally CloudO. This is because the SCCT architecture offers advantages over cloud-based architectures, and the two-timescale optimization algorithm in SCCTT provides greater flexibility and higher resource utilization efficiency compared to SCCTL and SCCTS.

VI. Conclusion

In this paper, to enhance the personalized service capabilities of pretrained large language models (LLMs) in heterogeneous network environments, we have proposed a split-learning-based cloud–edge–end collaborative training framework (SCCT). By deploying a small language model (SLM) under a split-learning paradigm across edge devices and the edge server, and a large language model (LLM) with parameter-efficient fine-tuning (PEFT) modules on the cloud, SCCT enables collaborative training and inference between the large and small models. Owing to this unique architecture, SCCT effectively leverages the abundant data and computational resources on massive edge devices to enhance the personalization capability of an LLM without compromising its generalization capability. Considering the dynamic nature of networks, we have designed a two-timescale optimization algorithm to minimize the training latency of SCCT. Numerical results show that the proposed two-timescale optimization algorithm efficiently reduces the training latency of SCCT, and the two-timescale optimization algorithm assisted SCCT outperforms the state-of-the-art LLM deployment framework in terms of fine-tuning performance. This indicates that the SCCT, empowered by the proposed two-timescale optimization algorithm, successfully pushes LLM fine-tuning from the cloud to the network edge, efficiently leveraging massive distributed datasets and computational resources at the network edge. For future work, we would investigate the performance optimization of SCCT in the case of considering the mobility of the edge devices.

APPENDIX A
PARAMETER SIZE AND COMPUTATIONAL COMPLEXITY OF A TRANSFORMER ENCODER LAYER

A transformer encoder layer mainly consists of the multi-head attention (MHA) module and feed-forward neural network (FFN) [72]. It is worth noting that the number of heads in MHA neither alters the parameter size nor the computational complexity. Therefore, to streamline the derivation, we will not refer to the number of attention heads below.

A. Parameter Size

The parameters of the multi-head attention module consist of four matrices: the query projection matrix $\boldsymbol{W}^Q \in \mathbb{R}^{d_{hid}\times d_{hid}}$, the key projection matrix $\boldsymbol{W}^K \in \mathbb{R}^{d_{hid}\times d_{hid}}$, the value projection matrix $\boldsymbol{W}^V \in \mathbb{R}^{d_{hid}\times d_{hid}}$, and the output projection matrix $\boldsymbol{W}^O \in \mathbb{R}^{d_{hid}\times d_{hid}}$. The number of parameters of these four matrices is

$$z_1 = 4d_{hid}^2.$$

The feed-forward neural network is a three-layer fully connected neural network, with the number of neurons in the three layers being d_{hid}, $4d_{hid}$, and d_{hid}, respectively. The weight parameters of these three fully connected layers consist of two weight matrices, denoted as $\boldsymbol{W}^1 \in \mathbb{R}^{d_{hid}\times 4d_{hid}}$ and $\boldsymbol{W}^2 \in \mathbb{R}^{4d_{hid}\times d_{hid}}$. The number of parameters of $\boldsymbol{W}^1$ and $\boldsymbol{W}^2$ is

$$z_2 = 8d_{hid}^2.$$

In addition, there are two *LayerNorm* modules in a transformer encoder layer. The parameters of each *LayerNorm* module include two vectors with dimension d_{hid}. Therefore, the number of parameters of two *LayerNorm* modules is

$$z_3 = 4d_{hid}.$$

The number of parameters of one transformer encoder layer is $z_1 + z_2 + z_3 = 12d_{hid}^2 + 4d_{hid}$. The number of bits needed to save one parameter is denoted as ξ. The value of ξ depends on the parameter precision. For example, when the parameter precision is *float32*, we have $\xi = 32$. Therefore, the parameter size (in bits) of one transformer encoder layer is

$$z = \xi\left(12d_{hid}^2 + 4d_{hid}\right).$$

B. Computational Complexity

We next analyze the computational complexity (in FLOPs) of the transformer encoder layer step by step according to its computation procedure. It is worth noting that in the following computational complexity analysis, we will discard lower-order terms from the algebraic expressions to simplify the final result, since their impact on overall complexity is entirely negligible.

1) Computational Complexity of MHA: When the data sample $\boldsymbol{X} \in \mathbb{R}^{d_{in}\times d_{hid}}$ is fed into the transformer encoder layer, it first needs to be multiplied by the matrices $\boldsymbol{W}^Q \in \mathbb{R}^{d_{hid}\times d_{hid}}$, $\boldsymbol{W}^K \in \mathbb{R}^{d_{hid}\times d_{hid}}$, and $\boldsymbol{W}^V \in \mathbb{R}^{d_{hid}\times d_{hid}}$, respectively. This process can be expressed as

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q, \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K, \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V.$$

The computational complexity of this step is

$$h_1 = 3d_{in}d_{hid}(d_{hid} + d_{hid} - 1) \approx 6d_{in}d_{hid}^2.$$

The second step is to obtain the weight matrix $\boldsymbol{A} \in \mathbb{R}^{d_{in}\times d_{in}}$ by

$$\boldsymbol{A} = softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_{hid}}}\right) \in \mathbb{R}^{d_{in}\times d_{in}}$$

The computational complexity of this step is

$$h_2 = d_{in}d_{in}(d_{hid} + d_{hid} - 1) \approx 2d_{in}^2 d_{hid}.$$

In the third step, the weight matrix $\boldsymbol{A}$ is multiplied by the value matrix $\boldsymbol{V}$, that is,

$$\boldsymbol{Z} = \boldsymbol{A}\boldsymbol{V} \in \mathbb{R}^{d_{in}\times d_{hid}}$$

The computational complexity of this step is

$$h_3 = d_{in}d_{hid}(d_{in} + d_{in} - 1) \approx 2d_{in}^2 d_{hid}$$

In the fourth step, the matrix $\boldsymbol{Z}$ is first multiplied by the output projection matrix $\boldsymbol{W}^O \in \mathbb{R}^{d_{hid}\times d_{hid}}$, then the residual term $\boldsymbol{X}$ is added, and finally the result is passed through a *LayerNorm* layer to produce the output of the MHA module as follows

$$\boldsymbol{Output}_{MHA} = LayerNorm\left(\boldsymbol{Z}\boldsymbol{W}^O + \boldsymbol{X}\right).$$

The computational complexity of this step is

$$h_4 = d_{in}d_{hid}(d_{hid} + d_{hid} - 1 + 1) = 2d_{in}d_{hid}^2$$

2) Computational Complexity of FFN: In the fifth step, the output of the MHA module is fed into the FFN, and then the forward propagation is performed on the $\boldsymbol{W}^1$, that is,

$$\boldsymbol{H}_1 = GELU\left(Output_{MHA}\boldsymbol{W}^1 + \boldsymbol{b}^1\right).$$

where $GELU$ is the activation function, $\boldsymbol{b}^1$ is the bias vector. The computational complexity of this step is

$$h_5 = 4d_{in}d_{hid}(d_{hid} + d_{hid} - 1 + 1) = 8d_{in}d_{hid}^2.$$

In the sixth step, the forward propagation on $\boldsymbol{W}^2$ can be expressed as

$$\boldsymbol{H}_2 = \boldsymbol{H}_1\boldsymbol{W}^2 + \boldsymbol{b}^2$$

where $\boldsymbol{b}^2$ is the bias vector. The computational complexity of this step is

$$h_6 = d_{in}d_{hid}(4d_{hid} + 4d_{hid} - 1 + 1) = 8d_{in}d_{hid}^2.$$

In the last step, the output of FFN is obtained by

$$\boldsymbol{Output}_{FFN} = LayerNorm\left(\boldsymbol{H}_2 + \boldsymbol{Output}_{MHA}\right).$$

The computational complexity of this step is negligible.

Therefore, the computational complexity of one transformer encoder layer is

$$\begin{aligned} h &= h_1 + h_2 + h_3 + h_4 + h_5 + h_6 \\ &= 24d_{in}d_{hid}^2 + 4d_{in}^2 d_{hid}. \end{aligned}$$

REFERENCES

[1] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM transactions on intelligent systems and technology*, vol. 15, no. 3, pp. 1–45, 2024.

[2] O. Friha, M. A. Ferrag, B. Kantarci, B. Cakmak, A. Ozgun, and N. Ghoualmi-Zine, "LLM-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness," *IEEE Open Journal of the Communications Society*, 2024.

[3] H. Zhou, C. Hu, Y. Yuan, Y. Cui, Y. Jin, C. Chen, H. Wu, D. Yuan, L. Jiang, D. Wu *et al.*, "Large language model (LLM) for telecommunications: A comprehensive survey on principles, key techniques, and opportunities," *IEEE Communications Surveys & Tutorials*, 2024.

[4] J. Wang, H. Du, D. Niyato, J. Kang, S. Cui, X. S. Shen, and P. Zhang, "Generative AI for integrated sensing and communication: Insights from the physical layer perspective," *IEEE Wireless Communications*, 2024.

[5] R. Zhang, H. Du, Y. Liu, D. Niyato, J. Kang, Z. Xiong, A. Jamalipour, and D. I. Kim, "Generative AI agents with large language model for satellite networks via a mixture of experts transmission," *IEEE Journal on Selected Areas in Communications*, 2024.

[6] Z. Lin, G. Qu, Q. Chen, X. Chen, Z. Chen, and K. Huang, "Pushing large language models to the 6G edge: Vision, challenges, and opportunities," *arXiv preprint arXiv:2309.16739*, 2023.

[7] G. O. Boateng, H. Sami, A. Alagha, H. Elmekki, A. Hammoud, R. Mizouni, A. Mourad, H. Otrok, J. Bentahar, S. Muhaidat *et al.*, "A survey on large language models for communication, network, and service management: Application insights, challenges, and future directions," *IEEE Communications Surveys & Tutorials*, 2025.

[8] S. Long, F. Tang, Y. Li, T. Tan, Z. Jin, M. Zhao, and N. Kato, "6G comprehensive intelligence: network operations and optimization based on large language models," *IEEE Network*, 2024.

[9] Q. Cui, X. You, N. Wei, G. Nan, X. Zhang, J. Zhang, X. Lyu, M. Ai, X. Tao, Z. Feng *et al.*, "Overview of AI and communication for 6G network: Fundamentals, challenges, and future research opportunities," *Science China Information Sciences*, vol. 68, no. 7, p. 171301, 2025.

[10] F. Jiang, C. Pan, L. Dong, K. Wang, M. Debbah, D. Niyato, and Z. Han, "A comprehensive survey of large AI models for future communications: Foundations, applications and challenges," *arXiv preprint arXiv:2505.03556*, 2025.

[11] F. Zhu, X. Wang, X. Li, M. Zhang, Y. Chen, C. Huang, Z. Yang, X. Chen, Z. Zhang, R. Jin *et al.*, "Wireless large AI model: Shaping the AI-native future of 6G and beyond," *arXiv preprint arXiv:2504.14653*, 2025.

[12] Y.-J. Liu, H. Du, X. Xu, R. Zhang, G. Feng, B. Cao, D. Niyato, D. I. Kim, A. Jamalipour, K. B. Letaief *et al.*, "A survey of integrating generative artificial intelligence and 6G mobile services: Architectures, solutions, technologies and outlooks," *IEEE Transactions on Cognitive Communications and Networking*, 2025.

[13] J. Wang, H. Du, Y. Liu, G. Sun, D. Niyato, S. Mao, D. I. Kim, and X. Shen, "Generative AI based secure wireless sensing for ISAC networks," *IEEE Transactions on Information Forensics and Security*, 2025.

[14] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[15] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[16] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, "Parameter-efficient fine-tuning for large models: A comprehensive survey," *arXiv preprint arXiv:2403.14608*, 2024.

[17] H. Wu, X. Chen, and K. Huang, "Device-edge cooperative fine-tuning of foundation models as a 6G service," *IEEE Wireless Communications*, vol. 31, no. 3, pp. 60–67, 2024.

[18] Z. Lyu, Y. Li, G. Zhu, J. Xu, H. V. Poor, and S. Cui, "Rethinking resource management in edge learning: A joint pre-training and fine-tuning design paradigm," *IEEE Transactions on Wireless Communications*, 2024.

[19] Y. Xin, S. Luo, H. Zhou, J. Du, X. Liu, Y. Fan, Q. Li, and Y. Du, "Parameter-efficient fine-tuning for pre-trained vision models: A survey," *arXiv preprint arXiv:2402.02242*, 2024.

[20] L. Xu, H. Xie, S.-Z. J. Qin, X. Tao, and F. L. Wang, "Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment," *arXiv preprint arXiv:2312.12148*, 2023.

[21] G. I. Kim, S. Hwang, and B. Jang, "Efficient compressing and tuning methods for large language models: A systematic literature review," *ACM Computing Surveys*, vol. 57, no. 10, pp. 1–39, 2025.

[22] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for NLP," in *International conference on machine learning*. PMLR, 2019, pp. 2790–2799.

[23] E. B. Zaken, S. Ravfogel, and Y. Goldberg, "BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models," *arXiv preprint arXiv:2106.10199*, 2021.

[24] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "LoRA: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[25] H. Ao, H. Tian, W. Ni, G. Nie, and D. Niyato, "Semi-asynchronous federated split learning for computing-limited devices in wireless networks," *IEEE Transactions on Wireless Communications*, 2025.

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

[27] C. Thapa, M. A. P. Chamikara, and S. A. Camtepe, "Advancements of federated learning towards privacy preservation: From federated learning to split learning," *Federated Learning Systems: Towards Next-Generation AI*, pp. 79–109, 2021.

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

[29] L. Luo and X. Zhang, "Federated split learning via mutual knowledge distillation," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 3, pp. 2729–2741, 2024.

[30] Y. Wen, G. Zhang, K. Wang, and K. Yang, "Training latency minimization for model-splitting allowed federated edge learning," *IEEE Transactions on Network Science and Engineering*, 2025.

[31] Z. Lin, G. Qu, X. Chen, and K. Huang, "Split learning in 6G edge networks," *IEEE Wireless Communications*, 2024.

[32] C. C. S. Balne, S. Bhaduri, T. Roy, V. Jain, and A. Chadha, "Parameter efficient fine tuning: A comprehensive analysis across applications," *arXiv preprint arXiv:2404.13506*, 2024.

[33] Y. Shen, J. Shao, X. Zhang, Z. Lin, H. Pan, D. Li, J. Zhang, and K. B. Letaief, "Large language models empowered autonomous edge AI for connected intelligence," *IEEE Communications Magazine*, vol. 62, no. 10, pp. 140–146, 2024.

[34] F. Cai, D. Yuan, Z. Yang, and L. Cui, "Edge-LLM: A collaborative framework for large language model serving in edge computing," in *2024 IEEE International Conference on Web Services (ICWS)*. IEEE, 2024, pp. 799–809.

[35] S. Zhang, G. Cheng, Z. Li, and W. Wu, "SplitLLM: Hierarchical split learning for large language model over wireless network," *arXiv preprint arXiv:2501.13318*, 2025.

[36] B. Picano, D. T. Hoang, and D. Nguyen, "A matching game for LLM layer deployment in heterogeneous edge networks," *IEEE Open Journal of the Communications Society*, 2025.

[37] Y. Chen, R. Li, X. Yu, Z. Zhao, and H. Zhang, "Adaptive layer splitting for wireless large language model inference in edge computing: A model-based reinforcement learning approach," *Frontiers of Information Technology & Electronic Engineering*, vol. 26, no. 2, pp. 278–292, 2025.

[38] Y. Chen, R. Li, Z. Zhao, C. Peng, J. Wu, E. Hossain, and H. Zhang, "NetGPT: An AI-native network architecture for provisioning beyond personalized generative services," *IEEE Network*, pp. 1–1, 2024.

[39] A. Borzunov, M. Ryabinin, A. Chumachenko, D. Baranchuk, T. Dettmers, Y. Belkada, P. Samygin, and C. A. Raffel, "Distributed inference and fine-tuning of large language models over the internet," *Advances in neural information processing systems*, vol. 36, pp. 12 312–12 331, 2023.

[40] M. Zhang, X. Shen, J. Cao, Z. Cui, and S. Jiang, "Edgeshard: Efficient LLM inference via collaborative edge computing," *IEEE Internet of Things Journal*, 2024.

[41] B. Yuan, Y. He, J. Davis, T. Zhang, T. Dao, B. Chen, P. S. Liang, C. Re, and C. Zhang, "Decentralized training of foundation models in heterogeneous environments," *Advances in Neural Information Processing Systems*, vol. 35, pp. 25 464–25 477, 2022.

[42] X. Chen, W. Wu, Z. Li, L. Li, and F. Ji, "LLM-empowered IoT for 6G networks: Architecture, challenges, and solutions," *arXiv preprint arXiv:2503.13819*, 2025.

[43] W. Zhao, W. Jing, Z. Lu, and X. Wen, "Edge and terminal cooperation enabled LLM deployment optimization in wireless network," in *2024 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*. IEEE, 2024, pp. 220–225.

[44] K. Lv, Y. Yang, T. Liu, Q. Gao, Q. Guo, and X. Qiu, "Full parameter fine-tuning for large language models with limited resources," *arXiv preprint arXiv:2306.09782*, 2023.

[45] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig, "Towards a unified view of parameter-efficient transfer learning," *arXiv preprint arXiv:2110.04366*, 2021.
[46] J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych, "Adapterfusion: Non-destructive task composition for transfer learning," *arXiv preprint arXiv:2005.00247*, 2020.
[47] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," *arXiv preprint arXiv:2101.00190*, 2021.
[48] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," *arXiv preprint arXiv:2104.08691*, 2021.
[49] D. Guo, A. M. Rush, and Y. Kim, "Parameter-efficient transfer learning with diff pruning," *arXiv preprint arXiv:2012.07463*, 2020.
[50] N. Lawton, A. Kumar, G. Thattai, A. Galstyan, and G. V. Steeg, "Neural architecture search for parameter-efficient fine-tuning of large pre-trained language models," *arXiv preprint arXiv:2305.16597*, 2023.
[51] Y.-L. Sung, V. Nair, and C. A. Raffel, "Training neural networks with fixed sparse masks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24 193–24 205, 2021.
[52] M. Valipour, M. Rezagholizadeh, I. Kobyzev, and A. Ghodsi, "DyLoRA: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation," *arXiv preprint arXiv:2210.07558*, 2022.
[53] Q. Zhang, M. Chen, A. Bukharin, N. Karampatziakis, P. He, Y. Cheng, W. Chen, and T. Zhao, "AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning," *arXiv preprint arXiv:2303.10512*, 2023.
[54] A. X. Yang, M. Robeyns, X. Wang, and L. Aitchison, "Bayesian low-rank adaptation for large language models," *arXiv preprint arXiv:2308.13111*, 2023.
[55] Y. Lin, X. Ma, X. Chu, Y. Jin, Z. Yang, Y. Wang, and H. Mei, "LoRA dropout as a sparsity regularizer for overfitting control," *arXiv preprint arXiv:2404.09610*, 2024.
[56] A. Ansell, E. M. Ponti, A. Korhonen, and I. Vulić, "Composable sparse fine-tuning for cross-lingual transfer," *arXiv preprint arXiv:2110.07560*, 2021.
[57] Z. Li, W. Wu, S. Wu, and W. Wang, "Adaptive split learning over energy-constrained wireless edge networks," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2024, pp. 1–6.
[58] W. Wu, M. Li, K. Qu, C. Zhou, X. Shen, W. Zhuang, X. Li, and W. Shi, "Split learning over wireless networks: Parallel design and resource management," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 4, pp. 1051–1066, 2023.
[59] X. Shen, Y. Liu, H. Liu, J. Hong, B. Duan, Z. Huang, Y. Mao, Y. Wu, and D. Wu, "A split-and-privatize framework for large language model fine-tuning," *arXiv preprint arXiv:2312.15603*, 2023.
[60] S. Zhang, G. Cheng, X. Huang, Z. Li, W. Wu, L. Song, and X. Shen, "Split fine-tuning for large language models in wireless networks," *arXiv preprint arXiv2501.09237*, 2025.
[61] J. Ma, X. Lyu, J. Jiang, Q. Cui, H. Yao, and X. Tao, "SplitFrozen: Split learning with device-side model frozen for fine-tuning LLM on heterogeneous resource-constrained devices," *arXiv preprint arXiv:2503.18986*, 2025.
[62] K. Zhao and Z. Yang, "Efficient federated split learning for large language models over communication networks," *arXiv preprint arXiv:2504.14667*, 2025.
[63] Z. Li, S. Wu, L. Li, and S. Zhang, "Energy-efficient split learning for fine-tuning large language models in edge networks," *IEEE Networking Letters*, 2025.
[64] Z. Lin, X. Hu, Y. Zhang, Z. Chen, Z. Fang, X. Chen, A. Li, P. Vepakomma, and Y. Gao, "SplitLoRA: A split parameter-efficient fine-tuning framework for large language models," *arXiv preprint arXiv:2407.00952*, 2024.
[65] S. O. Semerikov, T. A. Vakaliuk, O. B. Kanevska, M. V. Moiseienko, I. I. Donchev, and A. O. Kolhatin, "LLM on the edge: the new frontier," in *CEUR Workshop Proceedings*, 2025, pp. 137–161.
[66] W. Wang, W. Chen, Y. Luo, Y. Long, Z. Lin, L. Zhang, B. Lin, D. Cai, and X. He, "Model compression and efficient inference for large language models: A survey," *arXiv preprint arXiv:2402.09748*, 2024.
[67] S. Guo, J. Xu, L. L. Zhang, and M. Yang, "Compresso: Structured pruning with collaborative prompting learns compact large language models," *arXiv preprint arXiv:2310.05015*, 2023.
[68] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang, "A survey on model compression for large language models," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 1556–1577, 2024.
[69] X. Ma, G. Fang, and X. Wang, "LLM-pruner: On the structural pruning of large language models," *Advances in neural information processing systems*, vol. 36, pp. 21 702–21 720, 2023.
[70] R. Agrawal, H. Kumar, and S. R. Lnu, "Efficient LLMs for edge devices: Pruning, quantization, and distillation techniques," in *2025 International Conference on Machine Learning and Autonomous Systems (ICMLAS)*. IEEE, 2025, pp. 1413–1418.
[71] C. Thapa, P. C. Mahawaga Arachchige, S. Camtepe, and L. Sun, "SplitFed: When federated learning meets split learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, pp. 8485–8493, Jun. 2022.
[72] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
[73] J. Li, G. Sun, L. Duan, and Q. Wu, "Multi-objective optimization for UAV swarm-assisted IoT with virtual antenna arrays," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 4890–4907, 2023.
[74] S. T. W. Mara, R. Norcahyo, P. Jodiawan, L. Lusiantoro, and A. P. Rifai, "A survey of adaptive large neighborhood search algorithms and applications," *Computers & Operations Research*, vol. 146, p. 105903, 2022.
[75] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.
[76] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.
[77] M. Sokolova, N. Japkowicz, and S. Szpakowicz, "Beyond accuracy, F-score and ROC: A family of discriminant measures for performance evaluation," in *Australasian joint conference on artificial intelligence*. Springer, 2006, pp. 1015–1021.
[78] X. Ho, J. Huang, F. Boudin, and A. Aizawa, "LLM-as-a-Judge: Reassessing the performance of LLMs in extractive QA," *arXiv preprint arXiv:2504.11972*, 2025.

Jian Wang received the bachelor's degree in Internet of Things Engineering from the School of Information and Communication Engineering, University of Electronic Science and Technology of China, in 2019. He is currently pursuing the Ph.D. degree with the National Key Laboratory of Wireless Communications, University of Electronic Science and Technology of China. His current research interests include next generation mobile networks, distributed learning, and edge computing.

Gang Feng received his BEng and MEng degrees in Electronic Engineering from the University of Electronic Science and Technology of China (UESTC), in 1986 and 1989, respectively, and the Ph.D. degrees in Information Engineering from The Chinese University of Hong Kong in 1998. He joined the School of Electric and Electronic Engineering, Nanyang Technological University as an assistant professor and an associate professor in 2000 and 2005 respectively. At present he is a professor with the National Key Laboratory of Wireless Communications, UESTC of China. Dr. Feng is the recipient of IEEE Communications Society Technical Committee on Transmission Access and Optical Systems (TAOS) Best Paper Award (2019), ICC 2019 best paper award, and the Best Paper Award of IEEE Internet of Things Journal 2022. His research interests include AI Enabled wireless networking, distributed and collaborative machine learning, next generation cellular networks, etc. He is a Senior Member of IEEE.

Yi-Jing Liu is currently a postdoctoral researcher in the University of Electronic Science and Technology of China, Chengdu, China. She received the B.S. degree from the College of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, in 2017, and the Ph.D. degree from the National Key Laboratory of Wireless Communications, University of Electronic Science and Technology of China, in 2023. She is awarded the fellowship of China National Postdoctoral Program for Innovative Talents in 2023. Her current research interests include AI enabled wireless networking, distributed and collaborative machine learning, and edge AI.

Xinyi Xu received the B.S. degree from the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu, China, where she is currently pursuing the Ph.D. degree with the National Key Laboratory of Science and Technology on Wireless Communications. Her current research interests include intelligent wireless networking and machine learning.

Lei Cheng received the B.E. degree in Communication Engineering from the University of Electronic Science and Technology of China (UESTC) in 2019. She is now pursuing her Ph.D. degree with National Key Laboratory of Wireless Communications, UESTC. From 2023 to 2024, she was a Visiting Student with the Singapore University of Technology and Design (SUTD), Singapore. Her research interests include distributed learning and resource scheduling for space-air-ground/satellite-terrestrial integrated networks. She has served as a TPC member of VTC, ICC workshops, etc.

Wei Jiang received the B.S. degree in School of Communication and Information Engineering at Chongqing University of Posts and Telecommunications (CQUPT) in 2013, and the Ph.D. degree in School of Communication and Information Engineering at University of Electronic Science and Technology of China (UESTC) in 2019. She was a visiting Ph.D. student at Pennsylvania State University (PSU) from 2017 to 2018 and a postdoctoral researcher at Shenzhen University from 2020 to 2022. She is currently an associate professor with the Institute of Cyberspace Security, Zhejiang University of Technology. She has won the Best Paper Award of IEEE Transactions on Services Computing in 2023. Her current research interests include next generation mobile communication systems, mobile edge computing, and content caching.

Li Ping Qian received the PhD degree in Information Engineering from the Chinese University of Hong Kong in 2010. She is currently a Full Professor with the Institute of Cyberspace Security, Zhejiang University of Technology, Hangzhou, China. Her research interests include wireless communication and networking, resource management in wireless networks, massive IoTs, mobile edge computing, emerging multiple access techniques, and machine learning oriented towards wireless communications. She was a co-recipient of the IEEE Marconi Prize Paper Award in Wireless Communications in 2011, and the Best Paper Awards from IEEE ICC 2016, IEEE Communication Society GCCTC 2017, the Digital Communications and Networking in 2021, and IEEE WCNC 2023. She is the Distinguished Lecturer of IEEE Vehicular Technology Society (2024-2026), and is currently on the Editorial Board of IEEE Transactions on Cognitive Communications and Networking, IEEE Internet of Things Journal, and IEEE Wireless Communications.