MobiLLM: Enabling On-Device Fine-Tuning of Billion-Sized LLMs via Server-Assisted Side-Tuning

Liang Li*, *Member, IEEE,* Xingke Yang*, *Student Member, IEEE,* Wen Wu, *Senior Member, IEEE,* Hao Wang, *Member, IEEE,* Tomoaki Ohtsuki, *Senior Member, IEEE,* Xin Fu, *Senior Member, IEEE,* Miao Pan, *Senior Member, IEEE,* and Xuemin (Sherman) Shen, *Fellow, IEEE*

***Abstract*—On-device fine-tuning of large language models (LLMs) has attracted a lot of attention because of its tailoring personalized models while retaining user data locally on the mobile device. However, it faces significant challenges due to prohibitive memory requirements and slow training speeds. In this paper, we propose MobiLLM, a novel scheme enabling memory-efficient LLM fine-tuning on a single mobile device via server-assisted side-tuning. Particularly, MobiLLM strategically offloads backpropagation computations to an edge server while allowing the resource-constrained mobile device to retain merely a pretrained backbone model with frozen parameters during finetuning. It constructs a backpropagation bypass via parallel adapters decoupled from the backbone. During forward propagation, the device employs low bitwidth quantization for transmitting intermediate activations to the server to reduce communication overhead. The advantage of MobiLLM lies in: 1) confining training data strictly to the mobile device, and 2) eliminating on-device backpropagation while overlapping local computations with server execution. Collectively, MobiLLM ensures the data never leaves the local mobile device while significantly reducing mobile memory and computational burdens. We implement MobiLLM on several popular mobile devices, including NVIDIA Jetson Xavier NX and CPU-only laptops. Extensive experimental results demonstrate that MobiLLM can enable a resource-constrained mobile device to fine-tune billion-sized LLMs, achieving up to $4\times$ memory reduction and $2.3\times$ faster convergence as compared to state-of-the-art baselines.**

***Index Terms*—Large language model, Transformer, on-device fine-tuning, memory efficiency.**

I. Introduction

Transformer-based large language models (LLMs), like BERT [1], LLaMa [2], and GPT [3], have led to significant progress in artificial intelligence (AI) by enabling unprecedented capabilities in natural language processing (NLP), computer vision (CV), and beyond. These models typically follow a pre-training then fine-tuning paradigm: they are first trained on massive public datasets to learn general knowledge patterns, then fine-tuned to specific downstream tasks. However, real-world user data on mobile devices, such as private messages and reviews, often differs substantially from public training data and contains sensitive information. This combination of data divergence and privacy concerns necessitates on-device fine-tuning to tailor personalized models while ensuring the user data never leaves the local mobile device.

Simultaneously, rapid advances in mobile hardware are converging to make on-device computation increasingly viable. Modern devices, such as the iPhone 17, NVIDIA AGX platform, and MacBook Pro, coupled with mobile system-on-chip innovations, now integrate dedicated AI acclerators, such as NPUs and GPUs, explicitly optimized for complex machine learning workloads [4], [5]. Recently, there have been great hardware efforts specifically targeting LLM applications on mobile devices. For example, Qualcomm's Snapdragon 8 Elite can execute LLMs at a remarkable speed of 70 tokens per second [6], unlocking the potential for real-time on-device LLM applications. Google is also revolutionizing its smart home ecosystem by integrating Gemini AI - a multi-modal LLM developed by DeepMind [7] - into Google Assistant, Google Nest devices, and the Google Home APP [8]. These hardware evolution paves the way for intelligent, personalized mobile applications powered by on-device LLMs.

Despite the remarkable potential of mobile LLMs, contemporary LLMs have grown enormously in size, commonly reaching billion-scale parameters (e.g., billions to hundreds of billions), which presents substantial barriers to on-device fine-tuning. The vast gap between stringent mobile resource constraints - including memory, compute, and energy - and the prohibitive demands of training these billion-sized models renders conventional cloud-based fine-tuning solutions impractical for resource-limited mobile devices.

Recent research on parameter-efficient fine-tuning (PEFT) aims to alleviate the computational burden by keeping most of a pre-trained model frozen and updating only a minimal set of parameters [9]. This is typically achieved either by inserting lightweight trainable modules at various layers or by selectively updating a portion of the original parameters. Although PEFT techniques can reduce the required number of trainable parameters by orders of magnitude versus full

L. Li and W. Wu are with the Frontier Research Center, Pengcheng Laboratory, Shenzhen, China (e-mails: {lil03, wuw02}@pcl.ac.cn);

X. Yang, X. Fu, and M. Pan are with the Department of Electrical and Computer Engineering, University of Houston, Houston, USA (e-mails: xyang56@cougarnet.uh.edu, xfu8@central.uh.edu, mpan2@uh.edu);

H. Wang is with the Department of Electrical and Computer Engineering, Stevens Institue of Technology, Hoboken, USA (e-mail: hwang9@stevens.edu);

T. Ohtsuki is with the Department of Information and Computer Science, Keio University, Tokyo, Japan (e-mail: ohtsuki@keio.jp);

X. Shen is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Canada (e-mail: sshen@uwaterloo.ca).

*Contribute equally.

The work of L. Li and W. Wu was supported in part by National Natural Science Foundation of China under grants 62201071 and 62201311; in part by the Basic and Frontier Research Project of PCL under grants 2025QYB041 and 2025QYA002. The work of X. Yang and M. Pan was supported in part by the US National Science Foundation under grants CNS-2107057, CNS-2318664, CSR-2403249, and CNS-2431596.

fine-tuning methods [10]–[12], its residual computational and memory demands remain prohibitive for billion-parameter LLMs on a single mobile device. To overcome this limitation, some pioneering approaches explore multi-device cooperative training paradigms. These approaches distribute the resource-intensive task of fine-tuning LLMs across an edge network of connected mobile devices, thereby collectively alleviating the computational and memory burden of any individual device. For instance, the pipeline parallel training paradigm partitions an LLM into a sequence of smaller sub-models. These sub-models are then concatenated and deployed sequentially across cooperating mobile devices [13], [14]. During training, activations and gradients propagate through devices in multi-stage relay patterns [15]–[17]. Such fundamental dependence on multi-device orchestration renders these solutions infeasible for single-device scenarios while complicating practical deployment.

On-device LLM fine-tuning faces three primary challenges. *Firstly*, existing collaborative LLM fine-tuning schemes typically rely on layer-wise model partitioning and cross-device pipeline training, which requires all devices to be within a local area network with stable peer-to-peer links. It is, however, often impractical in edge networks with limited capable mobile devices and vulnerable to the failure of any single device in the pipeline, not to mention high error rates of multi-hop transmissions [18]. *Secondly*, training billion-sized LLMs demands enormous memory usage, which often exceeds the capacity of mobile devices (e.g., $>$ 20 GB for full fine-tuning OPT-1.3B model vs. typical 4–12 GB memory of mobile devices). Although PEFT methods help to reduce memory footprint up to 30% [19], they cannot fundamentally resolve memory issues since gradients for backpropagation still require passing through the entire pre-trained model. That makes the PEFT schemes inadequate for enabling LLM fine-tuning on the memory-constrained mobile device. *Thirdly*, fine-tuning process typically follows an alternating forward and backward propagation policy, where backward propagation is more computation-intensive. On-device accelerators, e.g., Google Edge TPU or Qualcomm Hexagon, are typically optimized for inference (i.e., forward propagation only) and lack acceleration support for backpropagation-specific operations. That may significantly prolong training time or even disable fine-tuning on resource-constrained mobile devices.

In this paper, we introduce MobiLLM, a novel scheme and corresponding system implementation that enables LLM fine-tuning on a single mobile device via server-assisted side-tuning. Inspiring by the side-tuning concept [20], MobiLLM's core design principles are to: (i) divide LLM fine-tuning into a frozen pre-trained backbone network and a side network consisting of trainable modules; (ii) partition the LLM fine-tuning workload between the mobile device and a server, i.e., deploying the frozen backbone model on the mobile device and the trainable side network on the server; and (iii) facilitate the exchange of inter-layer activations between the mobile device and the server with low communication overhead. In this way, MobiLLM ensures a memory- and computation-efficient forwardpropagation using the LLM backbone on the resource-constrained mobile device, while offloading the computation-intensive and memory-heavy backpropagation to the high-performance edge server, while keeping raw data on the local mobile device. With these design innovations, MobiLLM enables the fine-tuning of billion-sized LLMs on a mobile device - even those relying solely on CPUs - with the help of the server. It effectively leverages the established mobile edge network architecture to decouple forward and backward propagation, thereby overcoming fundamental memory and computational barriers inherent to mobile devices. Our salient contributions are summarized as follows:

- We propose MobiLLM, an on-device LLM fine-tuning scheme that offloads dominant computational and memory burdens to an edge server while confining mobile devices to forward propagation only. By executing mobile-friendly computational operations on a frozen backbone network, MobiLLM enables the mobile device to fine-tune LLMs while keeping their data locally on the device.
- We develop a quantized adapter side-tuning method for the MobiLLM system. It constructs a trainable side-network by stacking a set of parallel adapter modules, which is decoupled from the frozen pre-trained LLM backbone and thereby creating a backpropagation bypass. During forward propagation, activations from each backbone block undergo low-precision quantization and propagate to the side-network, enabling task adaptation through using user-specific intermediate representations.
- We implement MobiLLM on several popular mobile devices, including NVIDIA Jetson Xavier NX and CPU-only laptops, and evaluate its performance using both *sub-billion-sized LLM* (e.g., OPT-350M) and *billion-sized LLM* (e.g., OPT-1.3B), across various AI tasks and system configurations. The results show that MobiLLM enables billion-sized LLM fine-tuning on memory-constrained mobile devices like NVIDIA Xavier (with 4.6 GB available memory only). Compared with SOTA methods, MobiLLM remarkably reduces its memory usage by up to $4\times$ and convergence time by up to $2.3\times$.

The remainder of the paper is organized as follows. Section II reviews related work. Section III unveils the core motivation and challenges driving our design. Section IV proposes the MobiLLM scheme, detailing its modular architecture, runtime workflow, and fundamental benefits. Section V presents the system implementation, and Section VI comprehensively evaluates performance against state-of-the-art benchmarks. Section VII finally concludes the paper with contributions and future directions.

II. Related Work

A. Parameter-Efficient Fine-Tuning

PEFT is a feasible direction that adapts a pre-trained model to a downstream task by only training a few trainable parameters, instead of updating all parameters [9]. Current research streams primarily adopt two paradigms: *modular augmentation* and *sparse parameter update*. In modular augmentation, lightweight trainable components are integrated into the frozen backbone. Initial work proposed Adapters, which insert bottleneck layers, typically two linear layers with

a nonlinearity, within transformer blocks. This requires only 3-4% additional parameters per layer to match full fine-tuning accuracy when trained with layer normalization [10]. LoRA adopts a matrix decomposition perspective, injecting trainable low-rank matrices to approximate weight updates without altering original parameters [11]. Contrastingly, prompt-based methods (e.g., prefix tuning) embed trainable tokens into input sequences, steering frozen model behavior through learned soft prompts [21]. Sparse update strategies exploit parameter redundancy by selectively updating subsets of existing parameters. BitFit, for instance, updates only bias terms while freezing all weights [12]. While PEFT methods aim to achieve competitive results with minimal parameter updates, they still require significant GPU memory and can be slow during the fine-tuning process. This is because the updated parameters are inside the pre-trained large model, necessitating a full backward pass through the model to compute gradients for backpropagation. This prevents PEFT methods from being directly applied to many real-world applications with limited computational resources, and thus does not help to fit for fine-tuning a large language model on a GPU of a resource-constrained mobile device.

B. LLM Fine-Tuning on Mobile Devices

On-device fine-tuning of personal data is a powerful solution for adapting LLMs to user-specific tasks while maintaining privacy, as all data storage and computation occur locally without any data leaving the device [22]. To overcome severe resource constraints and fit an LLM on mobile devices, two primary research directions have evolved: *model compression* and *collaborative fine-tuning*. Model compression compresses a backbone network to a smaller one, making it more manageable for training. Techniques like pruning and quantization reduce the number of weights and the bit-width needed to represent these weights, respectively [23]. However, excessive compression can lead to significant drops in performance, necessitating a careful balance between resource savings and model accuracy. Collaborative fine-tuning enables resource-constrained devices to distribute the computational and memory load of LLM fine-tuning by leveraging available computational nodes, which may include other mobile devices or edge/cloud servers, by establishing direct communication links [24], [25]. This is often based on exchanging necessary intermediate activations during training. For example, multi-device pipeline parallelism splits the LLM by layers into smaller sub-models that are deployed across different devices, where forward and backward propagation computations are performed in a relay fashion across these devices [13], [14]. This pipeline can be further accelerated by partitioning input data into micro-batches and feeding them continuously across devices to enhance parallelism [15], [17]. This collaborative fine-tuning paradigm effectively leverages the computational and transmission capabilities of neighboring nodes within a network, allowing flexible task deployment and execution under individual computational and memory constraints.

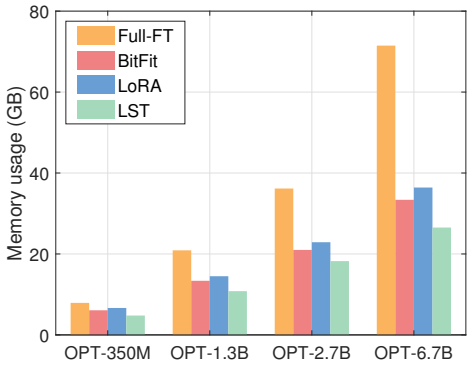


Fig. 1: Peak memory footprint of different methods.

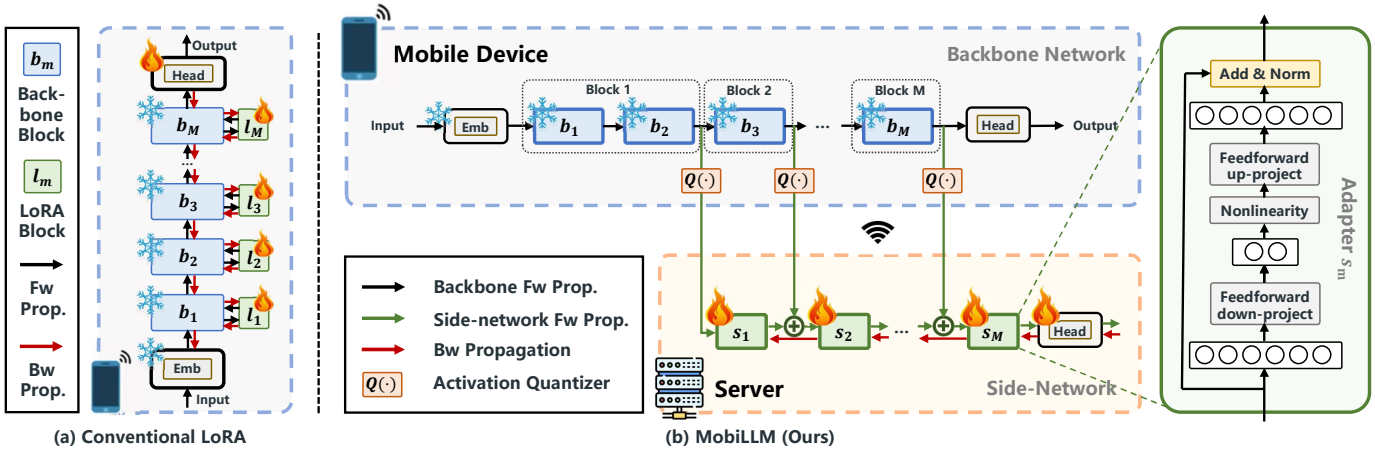
TABLE I: The breakdown of memory footprint. (Model: OPT-1.3B; batch size: 16; sequence length: 256; Full-FT: full fine-tuning.)

Method	Trainable Para.	Memory Footprint (GB)			
		Model	Act.	Opt.	Total
Full-FT	1.3B	2.509	10.859	7.527	20.895
BitFit	0.52M	2.509	10.859	0.003	13.371
LoRA	12.35M	2.533	11.964	0.072	14.569
Inference	—	2.509	1.894	—	4.403

III. Motivation

In this section, we identify three core difficulties in on-device LLM fine-tuning through preliminary experiments, motivating our key improvements for practical deployment.

Observation 1: Limitations of pipeline-based cooperative fine-tuning. Most of SOTA cooperative LLM fine-tuning schemes rely on layer-wise model partitioning-whether across multiple devices or between a device and a server-combined with sequential pipeline training. For those cross-device schemes, they typically require stable end-to-end multi-hop transmission links among participating mobile devices within a local area network. However, in real-world networks, there may not be enough capable mobile devices to form a stable collaboration loop, and single-device failures in the chain can disrupt the system or even ruin the overall LLM fine-tuning. Besides, multi-hop transmissions are notorious for the low delivery ratio and high error rates [26]–[29]. Even under ideal network conditions, the sequential nature of such pipeline execution inevitably introduces idle time at each stage, as devices (or the server) must wait for intermediate results from preceding devices in the pipeline. While some parallel training methods may mitigate devices' idle time by using stale model parameters for concurrent computation, this can impair model convergence and incur significant coordination overhead. Furthermore, since each device in the pipeline maintains only a local portion of the LLM, individual mobile device cannot employ the entire fine-tuned model to perform inference independently, which hinders local on-device LLM service provisioning.



(a) Conventional LoRA (b) MobiLLM (Ours)

Fig. 2: Comparative illustration of on-device LLM fine-tuning schemes. (a) Conventional LoRA: Trainable LoRA modules integrated throughout the backbone network require full-model backpropagation. (b) Our MobiLLM design: Decoupling the frozen backbone from the trainable side-network enables backpropagation-free on-device computation, utilizing only forward propagation with one-way activation offloading to the server.

Observation 2: Challenges imposed by memory bottlenecks. Figure 1 shows the peak memory usage in training various LLMs with a batch size of 16. Table I further provides a detailed breakdown of the memory footprint for fine-tuning an OPT-1.3B model. The observed memory requirements are often unaffordable for mobile devices (e.g., more than 70 GB for OPT-6.7B model vs. typical 4–12 GB DRAM of mobile devices). For smartphones, this situation is even worse, since only a fraction of the memory can be allocated to training tasks without affecting the user's experiences. In contrast, inference (forward propagation only) consumes much less memory (e.g., 4.4GB) because it does not need to store intermediate activations for all layers, which are necessary for backpropagation during training. In fine-tuning, those activations take up most of the memory, and this memory usage quickly scales up with batch size, sequence length, hidden layer dimensions, and the number of layers. Consequently, given the fact that LLMs typically have large dimensions and numerous layers, a slight increase in sequence length may lead to memory overflow for mobile devices, which can support LLM fine-tuning with smaller sequence-length data samples. Those harsh memory challenges result in unreliable on-device LLM fine-tuning, where tasks may occasionally succeed or fail. It is also difficult to assess whether a particular mobile device can support local LLM fine-tuning.

Observation 3: Inefficiency of on-device backpropagation computing. Fine-tuning process typically follows an alternating forward and backward propagation policy, where backward propagation is more computation-intensive — roughly twice the cost of forward propagation. While pipeline parallel training [15] enables multi-batch parallel processing, concurrent forward and backward computation on each device can lead to competition for computational resources. These factors can severely prolong processing time or even cause task failure on resource constrained mobile devices. Advanced mobile devices may be equipped with powerful and fast-evolving DNN accelerators (NPUs), e.g., Google Edge TPU and Qualcomm Hexagon, but these accelerators are tailored for inference rather than training. They lack good support for backpropagation-specific operations like dynamic gradient updating [30], which limits their accelerating performance for LLM fine-tuning tasks.

In summary, current mobile devices are incapable of handling LLM fine-tuning tasks due to the prohibitively high demands on hardware resources. Simply freezing pre-trained model parameters or distributing computational workloads yields limited improvements. Achieving practical and resource-efficient on-device LLM fine-tuning necessitates co-designing model architecture, training paradigms, and network support - an integrated approach we pioneer in MobiLLM.

IV. MobiLLM Design

A. MobiLLM Overview

MobiLLM is a device-server collaborative framework designed to enable memory-efficient LLM fine-tuning on the mobile device. The key idea is to create a backpropagation bypass parallel to the frozen backbone LLM, and split the LLM fine-tuning between the mobile device (the fixed forward propagation) and the server (the memory and computation intensive backward propagation), while keeping local data on the device.

Figure 2 presents an overview of the MobiLLM system. MobiLLM allows the mobile device to retain the local training dataset and pre-trained LLM backbone. Meanwhile, a side-network is deployed and trained on the server connected to the mobile device via stable wireless transmission links. During the fine-tuning, the mobile device performs only forward propagation on the backbone model, where intermediate activations are extracted and served as the inputs for the side-network via shortcut connections. In this way, MobiLLM completely separates the mobile device from those expensive backpropagation computations and saves substantial on-device memory (intermediate activations and optimizer states) during LLM fine-tuning. As a potential cost, MobiLLM introduces

some communication overheads and transmission latencies due to the transfer of activation values from the mobile device to the server. To tackle those issues, we provide a communication-efficient side-tuning method using activation quantization (Sec. IV-B) and optimize the device-server training procedures (Sec. IV-C) to effectively reduce transmission latencies in the proposed MobiLLM system, ensuring memory-friendly, computation-light and time-efficient LLM fine-tuning on the mobile device.

B. Quantized Adapter Side-Tuning

In Fig. 2, the blue boxes are the original transformer-based LLM backbone, and the green boxes outline the side-network that is composed of a set of trainable modules. Specifically, we separate the trainable modules from the backbone network rather than plug them in, thus providing a dedicated "highway" where all trainable parameters are on this highway. The remainder of this subsection elaborates on the design of the side-network and its connection with the backbone.

1) Parallel adapter based side-network: MobiLLM establishes a side-network parallel to the main backbone, which refines the backbone output to produce more accurate representations. In particular, we design our side-network by using stacked parallel adapter modules. The input to each adapter combines the intermediate activation output of the transformer layer in the backbone and the activation of the previous adapter. This ensures that our parallel adapters can refine the feature representations from the original backbone when fine-tuning towards a new task.

Each adapter contains down-projection matrix $\mathbf{W}_{down} \in \mathbb{R}^{d*r}$ to project the input to a lower-dimensional space, followed by a non-linear activation function $\sigma(\cdot)$, and an up-projection matrix $\mathbf{W}_{up} \in \mathbb{R}^{r*d}$. Here, d represents the dimension of the hidden layer, and r serves as the adapter bottleneck dimension, a hyperparameter used in configuring the adapters. Denoting $\mathbf{h}_{in}$ as the input to the adapter, the computation within the adapter module (with residual) can be summarized as follows:

$$\mathbf{h}_{in} \leftarrow \mathbf{h}_{in} + \sigma(\mathbf{h}_{in}\mathbf{W}_{down})\mathbf{W}_{up}. \tag{1}$$

Compared with previous side-tuning methods using lightweight transformer structures pruned from the backbone as the side network, our side-network design eliminates multi-head attention mechanisms and feed-forward networks. As a result, it further reduce amount of trainable parameters and increase the fine-tuning overhead.

2) Shortcut connection with quantized activation: MobiLLM utilizes shortcut connections from the intermediate activations of the backbone model to the side network. The outputs of the frozen backbone's final layer and the side network are combined and fed in the LLM head to predict.

MobiLLM allows the transformer layers of the pre-trained backbone to be grouped into M blocks, with each block connected to a parallel adapter in the task-specific side network. Accordingly, there are M shortcuts that feed intermediate activations from the backbone to the corresponding adapters in the side network. Let $\mathbf{a}_1, \ldots, \mathbf{a}_m, \ldots, \mathbf{a}_M$, where $\mathbf{a}_m \in \mathbb{R}^{n\times d}$, denote the activation outputs with each consisting of n tokens with a hidden dimension of d. To reduce communication overhead, MobiLLM applies low-precision quantization on these intermediate activations before feeding in the side-network. This process involves converting the original data format of activations (e.g., 32-bit or 16-bit floating-point) into a lower-bit data type. Typically, the input data is normalized by the absolute maximum value of its elements to fit within the range of the target data type. Taking FP4 quantization as an example, a 16-bit floating-point tensor is quantized into a 4-bit format with a range of $[0, 15]$ through

$$\mathbf{X}^{\mathrm{FP4}} = \mathrm{round}\left(\frac{15}{\mathrm{absmax}\left(\mathbf{X}^{\mathrm{FP16}}\right)}\mathbf{X}^{\mathrm{FP16}}\right), \tag{2}$$

where $\mathbf{X}^{\mathrm{FP16}}$ is the floating-point tensor, $\mathbf{X}^{\mathrm{4bit}}$ is the quantized counterpart with $\mathrm{INT4}$ data type. Alternatively, by using NF4 data type, MobiLLM also allows for the following quantization process:

$$\mathbf{X}^{\mathrm{NF4}} = \mathbf{T}\left(\frac{\mathbf{X}^{FP16}}{\mathrm{absmax}(\mathbf{X}^{FP16})}\right). \tag{3}$$

Here, the $\mathbf{X}^{\mathrm{FP16}}$ is first normalized using its maximum absolute value, and then the 4-bit quantized counterpart is obtained by mapping to a quantile table (denoted by $\mathbf{T}(\cdot)$). The quantile table $\mathbf{T}(\cdot)$ is well constructed to ensure that each quantization bin has an equal number of values assigned from the input tensor [31]. MobiLLM is potentially compatible with various quantization techniques and data formats, which may help mitigate information distortion by leveraging the distribution characteristics of activation values. We tested both FP4 and NF4 data types in our experiments and empirically demonstrated that NF4 achieves better performance (Section 6.4).

We note that it is not mandatory to partition the backbone into blocks with an identical number of transformer layers. A practical approach is to group the top layers (near the backbone's output) more finely, while grouping the bottom layers more coarsely. The rationale is that the bottom layers of the pre-trained backbone often learn general low-level feature representations shared across downstream tasks, requiring minimal adjustment during fine-tuning. This eliminates the need for attaching tunable adapters to every transformer layer, further reducing communication overhead for uploading activations to the server. Moreover, this design enhances data privacy, as the activation outputs from each backbone block almost distort the input embeddings, making it difficult to infer the original input samples.

C. MobiLLM Procedure

This subsection details the training procedure of the proposed MobiLLM. We start with MobiLLM training within each iteration, and then present MobiLLM overlapping training strategy across iterations for better time efficiency.

1) MobiLLM partitions the training computations within each iteration between the mobile device and the edge server: For each iteration, MobiLLM splits the training workloads between the mobile device and the server. Traditional split

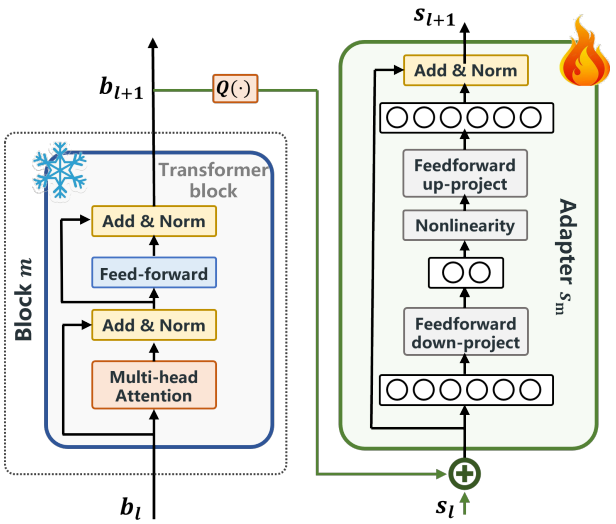


Fig. 3: Schematic diagram of the quantized adapter side-tuning structure.

learning involves device-side forward propagation, server-side forward and backward propagation, and device-side backward propagation, requiring bidirectional exchanges of smashed data at the cut layer. Different from the traditional one, MobiLLM simplifies this process by delegating backpropagation computations entirely to the server, thanks to its design separating the trainable side network from the frozen backbone network as shown in Fig. 2 and Fig. 3. Therefore, MobiLLM only requires a one-way transfer of intermediate activations (i.e., smashed data) from the mobile device to the server during the forward pass. Without loss of generality, we present the detailed training process of MobiLLM within an iteration as follows:

1) *Initialization (mobile device & server):* The mobile device retrieves a pre-trained LLM backbone that is suitable for its local computational and memory capabilities. Then, the server initializes a side-network, where the weights in the adapter modules were drawn from a zero-mean Gaussian with a well-selected standard deviation.
2) *Local backbone forward propagation (mobile device):* The mobile device samples a mini-batch of data from its local dataset and performs forward propagation through the frozen backbone model, generating intermediate activation outputs corresponding to each transformer block.
3) *Activation transmission (mobile device → server):* The mobile device quantizes the intermediate activation outputs of the backbone model ($\overline{\mathbf{a}_1}, \ldots, \overline{\mathbf{a}_m}, \ldots, \overline{\mathbf{a}_M}$) into a compressed format. The mobile device then transmits the quantized activations along with the corresponding labels and metadata (e.g., batch indices) to the server.
4) *Forward propagation (server):* Upon receiving the activations, the server integrates them with the input to its adapter modules, which serve as the side-network. The server performs forward propagation through the side-network, generating outputs needed for the training process.
5) *Loss calculation (server):* The server computes the training loss by comparing the side-network's output with the ground-truth labels received from the mobile device. The loss metric depends on the specific downstream task (e.g., cross-entropy loss for classification or mean squared error for regression).
6) *Backward propagation and model updates (server):* The server performs backward propagation to update the side-network.
7) The mobile device continues sampling new data batches and feeding them into the local backbone model. The above steps are repeated iteratively to tune the side-network until it converges.

Figure 3 illustrates the MobiLLM's forward propagation process, using a GPT-style transformer architecture [3] as an example. For simplicity, key components within each transformer layer are denoted as follows: Multi-Head Self-Attention (MSA) is f_{B_1}, and Feed Forward Network (FFN) is f_{B_3}. The two Normalization (LN2) layers are denoted by f_{B_2} and f_{B_4}, respectively. In the side-network, the down/up projections and non-linear layer in an adapter module are simplified as f_{A_1}, while Layer Normalization is represented by f_{A_2}. In the l-th transformer layer, the model generates two distinct outputs - one for the backbone and one for the side-network, which can be expressed as

$$b_{l+1} = f_{B_4}(f_{B_3}(f_{B_2}(f_{B_1}(b_l)+b_l))+f_{B_2}(f_{B_1}(b_l)+b_l)), \quad (4)$$

and

$$s_{l+1} = f_{A_2}(f_{A_1}(s_l + b_{l+1}) + s_l + Q(b_{l+1})). \quad (5)$$

Here, $Q(\cdot)$ denotes the quantization operator for the shortcut activation connections.

In our MobiLLM, the forward propagation calculations in Eq. 4 and Eq. 5 are executed by the mobile device and server, respectively. Importantly, we observe the updates to the adapters are decoupled from the backpropagation of the backbone (i.e., Eq. 4 operates independently of Eq. 5), because the adapters are not embedded within the backbone layers. Besides, the adapters take the intermediate activations from the backbone as their inputs, which ensures that the side-network benefits from the pre-trained representations during training.

Next, we present the gradient calculation for the trainable parameters θ_l of the l-th adapter, where $\theta_l = \{\theta_l^1, \theta_l^2, ..., \theta_l^n\}$. We use a_{l+1} to denote the concatenation l-th layer output of the backbone (i.e., b_{l+1}) and the side-network (i.e., s_{l+1}). In backpropagation with loss L, the gradient with respect to θ_l is:

$$\begin{aligned}\frac{\partial L}{\partial \theta_l} &= \frac{\partial L}{\partial a_{l+1}} \frac{\partial a_{l+1}}{\partial f_A} \frac{\partial f_A}{\partial \theta_l} \\ &= \frac{\partial L}{\partial s_{l+1}} \left(\frac{\partial s_{l+1}}{\partial f_l^1} \frac{\partial f_l^1}{\partial f_l^2} \frac{\partial f_l^2}{\partial f_l^3} \cdots \frac{\partial f_l^m}{\partial f_A} \right) \sum_{k=1}^{n} \frac{\partial f_A}{\partial \theta_l^k},\end{aligned} \quad (6)$$

where f_l^m is the intermediate process from the output of the l-the layer to f_A. Most of existing PEFT methods only reduce the number of trainable parameters θ_l, i.e., $\left|\sum_{k=1}^{n} \frac{\partial f_A}{\partial \theta_l^k}\right|$, resulting in a slight reduction in memory usage. In contrast, MobiLLM takes a step further. It simplifies the intermediate

Algorithm 1 MobiLLM Training Procedure

Input: Local dataset $\mathcal{D}$, pre-trained backbone f_B with frozen weights, quantization function Q, learning rate η
Initialization:
  Device: Load f_B parameters and send backbone config to server
  Server: Initialize trainable side-network f_S with parameters θ
while convergence criterion not met **do** ▷ Training loop
  On Mobile Device: ▷ Executed per mini-batch
    Sample data batch: $(x_t, y_t) \sim \mathcal{D}$
    Backbone network forward propagation: $a_t = f_B(x_t)$
    Quantize activations: $\tilde{a}_t = Q(a_t)$
    Transmit $(\tilde{a}_t, y_t)$ to server
    Proceed to next data batch ▷ Non-blocking pipeline
  On Edge Server: ▷ Concurrent with device
    On receive $(\tilde{a}_t, y_t)$:
      Side-network forward propagation: $\hat{y}_t = f_S(\tilde{a}_t)$
      Compute loss: $\mathcal{L}_t = \mathcal{L}(\hat{y}_t, y_t)$
      Update parameters: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_t$
Model deployment:
  Server $\rightarrow$ Device: Transfer f_S parameters
  Device: Assemble joint model: $f_{\text{joint}} = \{f_B, f_S\}$ for local LLM inference

process $\frac{\partial a_{l+1}}{\partial f_A}$ and offloads the necessary gradient calculations to the memory-rich server along a parallel gradient highway. Therefore, MobiLLM saves considerable memory and computation time for the mobile device. In addition, MobiLLM effectively bypasses the inefficiencies of mobile accelerators in handling training-specific operations and ensures high utilization of available hardware resources on the mobile device. Since the server has much higher processing speeds than the mobile device, the server-side operations also speed up the LLM fine-tuning for resource-constrained mobile devices.

2) MobiLLM overlaps device-side and server-side training across iterations: MobiLLM diverges from the conventional one-forward-one-backward interleaved updating rule by enabling uninterrupted forward pass computations on the mobile device. In parallel with intermediate activation transmission and side-network training, the mobile device overlaps the process by continuously feeding data batches into the backbone for successive training iterations. This is grounded in the fact that the device-side backbone model remains fixed throughout the fine-tuning process, eliminating the need to wait for parameter updates. As a result, multi-batch parallelism is achieved without introducing model staleness, which is a common drawback in other overlapping training methods. Besides, the server-side side-network training, powered by high-performance computation, typically outpaces the device's execution and transmission speeds by several times. This streamlined workflow guarantees the timely reception of up-to-date intermediate outputs by the server, eliminating unnecessary waiting times and accelerating collaborative fine-tuning. We summarize the procedure in Algorithm 1, which outlines the parallel execution across the device and server.

Algorithm 2 Adaptive Activation Quantization Algorithm.

Input: B_t (current bandwidth), τ (latency threshold), $\{\mathcal{I}^{(m)}\}$ (layer importance)
$D_{\max} \leftarrow B_t \cdot \tau$ ▷ Transmission budget
for each layer m **do**
  $Q^{(m)} \leftarrow Q_{\min}$ ▷ Minimum precision initialization
$D_{\text{remain}} \leftarrow D_{\max} - \sum_m \text{Size}(Q^{(m)})$
while $D_{\text{remain}} > 0 \wedge \exists Q^{(m)} < Q_{\max}$ **do**
  $\Delta U_{\max} \leftarrow 0,\ k^* \leftarrow \emptyset$
  for each layer m where $Q^{(k)} < Q_{\max}$ **do**
    $\delta\epsilon \leftarrow \epsilon(Q^{(m)}) - \epsilon(Q^{(m)}+1)$
    $\Delta U \leftarrow \delta\epsilon \times \mathcal{I}^{(m)}$
    if $\Delta U > \Delta U_{\max}$ **then**
      $\Delta U_{\max} \leftarrow \Delta U,\ m^* \leftarrow m$
  $\Delta D \leftarrow \text{Size}(Q^{(m^*)}+1) - \text{Size}(Q^{(m^*)})$
  if $\Delta D \leq D_{\text{remain}}$ **then**
    $Q^{(m^*)} \leftarrow Q^{(m^*)} + 1$
    $D_{\text{remain}} \leftarrow D_{\text{remain}} - \Delta D$
  else
    Break
Output: Quantization bitwidths $\{Q^{(m)}\}_{\forall m}$

D. Importance-Aware Adaptive Activation Quantization

The transmission of intermediate activations from mobile devices to the server may constitute a potential bottleneck in MobiLLM's collaborative fine-tuning paradigm, particularly under poor network conditions with fluctuated channel quality [32]. To mitigate this limitation, we introduce an online adaptive quantization method that optionally replaces the fixed-precision quantization scheme in the vanilla MobiLLM implementation. Its core idea is to strategically adjust quantization precision across backbone transformer blocks. This method dynamically adjusts bitwidth assignments in real-time, responsively adapting to both instantaneous network conditions and layer-specific feature sensitivity.

Mechanistically, the system initiates each iteration by profiling current uplink bandwidth B_t through periodic probe packets, establishing a transmissible data budget as:

$$D_{max} = B_t \tau. \tag{7}$$

Here, τ represents a configurable latency threshold that accommodates side-network computation at the server by scaling the transmission windows. Importance characterization employs a generalized metric $\mathcal{I}(m)$, applicable to various importance measures (e.g., Fisher information, gradient magnitude, or attention entropy), to quantify the sensitivity of layer m's activations to precision distortion.

The bitwidth allocation algorithm optimizes within budget constraints through marginal utility maximization. Starting from minimal bitwidth for all layers ($Q^{(m)} = Q_{min}, \forall m$), the algorithm iteratively upgrades precision through utility

maximization. The marginal utility $\Delta U^{(m)}$ for upgrading layer m combines precision gain and importance:

$$\Delta U^{(m)} = \left[\epsilon(Q^{(m)}) - \epsilon(Q^{(m)} + 1)\right] \cdot \mathcal{L}(m), \tag{8}$$

where $\epsilon(Q^{(m)})$ represents the quantization error at bitwidth Q. The optimization terminates when transmission budget exhaustion occurs (i.e., $D_{remain} \leq 0$) or all layers reach maximum precision (i.e., $Q^{(m)} = Q_{max}, \forall m$).

The above adaptive quantization method prioritizes precision allocation to sensitivity-critical layers, thereby preserving high-information features. The calibrated latency thresholds maintain minimal computational idle time at the server, while dynamically adapting to network conditions. Collectively, the system achieves an optimal balance between quantization error suppression and transmission efficiency.

E. Advantages of MobiLLM

In this subsection, we first analyze how MobiLLM helps to save memory of LLM fine-tuning on the mobile device, and then discuss MobiLLM's other merits.

1) Analysis of memory saving: MobiLLM restricts device-side operations to forward propagation only, drastically reducing the mobile device's memory usage. Specifically, MobiLLM reduces two main contributors of the memory footprint for the mobile device during the fine-tuning process, i.e., intermediate activations, optimizer states and model states, which are analyzed as follows.

(i) Intermediate activations: During forward propagation, only the activations of a few selected layers are temporarily stored on the device and can be discarded after transmission to the server. Consider an L-layer LLM, where the activation size of each layer is denoted by $A = SBH$. Here, S, B, and H represent the sequence length, batch size, and hidden-layer dimension, respectively. Traditional fine-tuning methods require storing activations for all layers on the mobile device, which consumes approximately LA of memory. In contrast, MobiLLM requires only γA, where γ ($\gamma \ll L$) is the number of selected layers for temporary storage. This reduces memory usage for activations by approximately L/γ-fold.

(ii) Optimizer state: Classical optimization algorithms like Adam require storing additional states (e.g., momentum and second-order statistics), typically doubling or tripling the memory needed for trainable model parameters. In MobiLLM, these computations and states are fully offloaded to the server, avoiding that memory consumption on the mobile device.

Those optimizations in MobiLLM design collectively reduce the memory footprint, enabling mobile devices even with limited memory capacities to fine-tune LLMs efficiently.

2) Advantages beyond memory efficiency: MobiLLM has other merits that enhance its practicality and performance in resource-constrained settings, which are listed as follows.

(i) Enhanced system reliability. Rather than relying on fragile multi-hop pipeline collaboration among multiple mobile devices [13], [14], MobiLLM only employs a single server with sufficient computational resources for help. Such a simple device-server design aligns with existing mobile edge network architecture, which has mature physical and network layer protocols to support terminal-server collaboration with high speed transmissions. For instance, 5G/6G base stations often feature high-performance computational units to provide nearby services for mobile users, and smart hubs in home IoT networks can manage workloads for connected devices via high-speed Wi-Fi links. Thus, the setup of MobiLLM is inherently more reliable and practical. In addition, MobiLLM ensures user data never leaves the local mobile device throughout the LLM fine-tuning process.

(ii) Versatility besides fine-tuning. Unlike pipeline-based methods that partition LLM across devices, MobiLLM allows each mobile device to keep a full LLM. That enables mobile devices to independently run local inference tasks besides on-device LLM fine-tuning. In other words, the mobile device may respond to on-demand and latency-sensitive inference requests during a continuous fine-tuning process via local resource sharing, i.e., simultaneously serving LLM fine-tuning and inference. In addition, the device can periodically update its model by fetching and merging task-specific adapters trained on the server, and the server can host multiple side-networks for different downstream tasks. That empowers a mobile device to switch between LLM fine-tuning tasks by simply altering the side-networks required at the server side. This flexibility enhances mobile devices' utility both during and beyond LLM fine-tuning process.

TABLE II: Training parameters.

Parameter	Value	Parameter	Value
Training precision	FP16	Epochs	20
Batch size	16	Learning rate	5×10^{-4}
Sequence length	256	Transmission rate	60 Mbps

V. Experimental Setup

A. MobiLLM Implementation

The proposed MobiLLM system is deployed on a testbed consisting of a server and a mobile device. The server is equipped with a NVIDIA A100 GPU (6912-core Ampere GPU with 40GB memory). On the mobile device side, we consider two classical types of devices: (1) NVIDIA Jetson Xavier NX with 6-core NVIDIA Carmel ARM CPU, 384-core NVIDIA Volta GPU, and 8GB RAM. (2) Huawei Matebook laptop with Intel 13th Gen Core i5-13500H and 16GB RAM, which is a CPU-only device. The communication between the mobile device and the server is based on Wi-Fi 5 connections, which uses the WebSocket communication protocol.

In particular, LLMs' fine-tuning on Jetson Xavier NX contributes the major experimental results in this work. Since Xavier's GPU and CPU share 8GB RAM, its maximum available GPU memory for training is 4.6G. The Matebook laptop is employed to test the training acceleration performance of those CPU-only mobile devices (Sec. VI-C).

B. Baselines

The MobiLLM scheme is benchmarked against the following state-of-the-art baselines across different LLM tasks.

TABLE III: Experiment results on GLUE benchmark. Model: OPT-350M (Batch size=16, Sequence length=256)

Method	Hidden Size/ r	Trainable Parameters (%)	Memory Usage (GB)		Accuracy (%)								
			Training	Inference	SST-2	QNLI	QQP	MRPC	CoLA	MNLI	RTE	STS-B	Avg
Full-FT	—	100	7.910	1.580	92.9	**84.2**	**85.9**	**83**	**61.9**	76.4	**76**	85.6	**80.7**
LoRA	64	1.86	6.700	1.580	92.7	84.1	85.1	82.2	61.1	**76.5**	75.8	85.9	80.4
	16	0.47	6.655	1.580	92.2	83.7	84.3	81.7	60.4	76.1	75.4	85.6	79.9
BitFit	—	0.08	6.081	1.580	**93.8**	**84.2**	84.6	82.2	61.4	76.3	**76**	**86.2**	80.6
LST	16	0.84	4.932	1.624	91.9	84	84.5	82.3	60.2	75.8	73.9	84.9	79.7
	64	0.15	4.787	1.588	91.4	83.8	84.3	81.9	60.2	75.6	73.2	84.3	79.3
MobiLLM-L	64	1.12	**2.452**	1.623	90.2	83.7	84.2	81.8	59.7	75.2	73.4	84.2	79.1
	16	0.42	**2.421**	1.605	90	83.4	84	81.6	59.3	75.1	72.9	83.7	78.8
MobiLLM	64	0	**1.635**	1.623	89.9	83.5	83.9	81.7	59.4	75.1	73	84	78.8
	16	0	**1.621**	1.605	89.8	83.1	83.5	81.6	59.3	74.7	72.7	83.5	78.5

TABLE IV: Experiment results on GLUE benchmark. Model: OPT-1.3B (Batch size=16, Sequence length=256)

Method	Hidden Size/ r	Trainable Parameters (%)	Memory Usage (GB)		Accuracy (%)								
			Training	Inference	SST-2	QNLI	QQP	MRPC	CoLA	MNLI	RTE	STS-B	Avg
Full-FT	—	100	20.895	4.403	**95.9**	85	**86.9**	**84**	**63.9**	81	**83**	**89.6**	**83.6**
LoRA	64	0.95	14.569	4.403	94.4	84.9	86.6	83.3	62.5	**81.1**	82.3	89.2	83.0
	16	0.24	14.497	4.403	94.2	84.6	86.1	82.9	61.9	79.7	82.1	88.7	82.5
BitFit	—	0.04	13.371	4.403	95.4	85.5	86.4	83.3	62.2	79.9	81.7	88.8	82.9
LST	16	0.85	11.122	4.533	94.5	**85.8**	86.4	83.3	60.8	77.6	81.6	88.4	82.3
	64	0.15	10.804	4.427	94.4	85.6	86.2	83.1	60.3	77.4	81.2	88.1	82.0
MobiLLM-L	64	0.49	**6.132**	4.472	94.1	84.7	85.8	81.7	59.9	76.9	80.7	87.9	81.5
	16	0.13	**6.088**	4.461	94.1	84.3	85.6	81.2	59.4	76.7	80.4	87.5	81.2
MobiLLM	64	0	**4.495**	4.472	94	84.5	85.4	81.7	59.7	76.8	80.6	87.3	81.3
	16	0	**4.487**	4.461	93.8	84.1	85.3	81.4	59.4	76.6	80.2	87.3	81.0

- **Full-FT**: Updates all parameters of the pre-trained LLM, serving as the performance upper bound at significant computational expense.
- **LoRA [11]**: Inserts trainable low-rank decomposition matrices into transformer layers while freezing backbone weights, with r (reported as hidden size) indicating the low-rank dimension.
- **BitFit [12]**: Trains exclusively the bias terms within linear/attention layers while keeping all other weights frozen.
- **LST [20]**: Uses lightweight transformer structures pruned from the backbone as the side-network. The reduction factor r determines dimension shrinkage of side-network (e.g., $r = 16$ implies $1/16$ of the backbone's width).
- **SL [18]**: Layer-wise model partitioning with BitFit fine-tuning, where initial transformer layers are processed on the device and remaining layers on the server.
- **MobiLLM-L**: Trains both the backbone network and the side-network entirely on-device, isolating the impact of our server-assisted computation partitioning.

C. Models, Datasets and Parameters

The OPT-350M and OPT-1.3B, two popular decoder-only LLMs from the OPT series, are exploited to evaluate MobiLLM's performance. The pre-trained weights of the above models are from Huggingface [33].

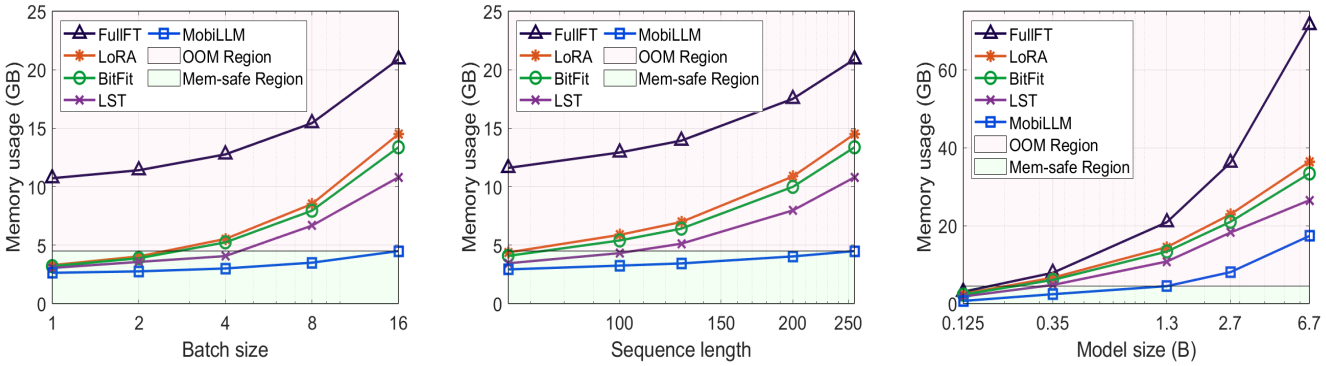
The MobiLLM and several baselines are compared on natural language understanding (NLU) tasks. We use the GLUE [34] benchmark, which consists of seven classification and one regression task. The benchmark evaluate models on multiple diverse tasks over linguistic acceptability (CoLA [35]), sentiment analysis (SST-2 [36]), similarity and paraphrase (MRPC [37], QQP [38], STS-B [39]) and natural language inference (MNLI [40], QNLI [41], RTE [36]). We report accuracy on MNLI, QQP, QNLI, SST-2, MRPC, and RTE, Pearson correlation coefficients on SST-B, and Mathews correlation coefficients [42] on CoLA.

Unless otherwise specified, MobiLLM and all baseline methods follow the same simulation configuration, as summarized in Table II.

VI. Evaluation Results & Analysis

A. End-to-End Performance

Performance of different baselines on GLUE Benchmark. We first study the performance of MobiLLM on the OPT-350M model. Table III summarizes the experimental results of MobiLLM and other baselines on GLUE benchmark under the default setting. Overall, MobiLLM achieves the lowest memory footprint among all methods while maintaining comparable accuracy. Particularly, MobiLLM outperforms Full-FT by a significant reduction of 79.5% in memory usage,



(a) Impact of batch size (Model: OPT-1.3B) (b) Impact of sequence length (c) Impact of model size

Fig. 4: Performance on various training configurations. (Green area indicates the memory-safe region for Xaiver with peaking 4.6 GB GPU RAM.)

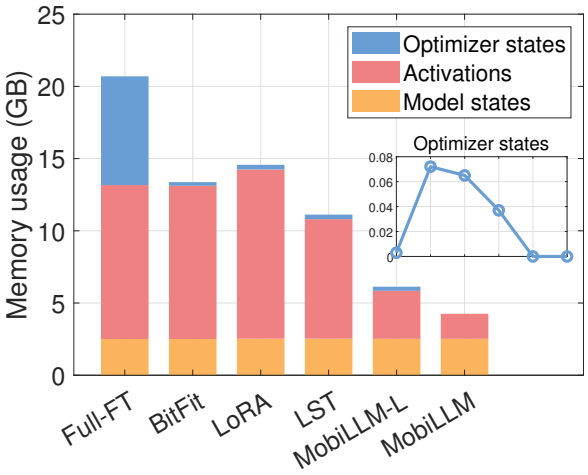


Fig. 5: The breakdown of memory footprint.

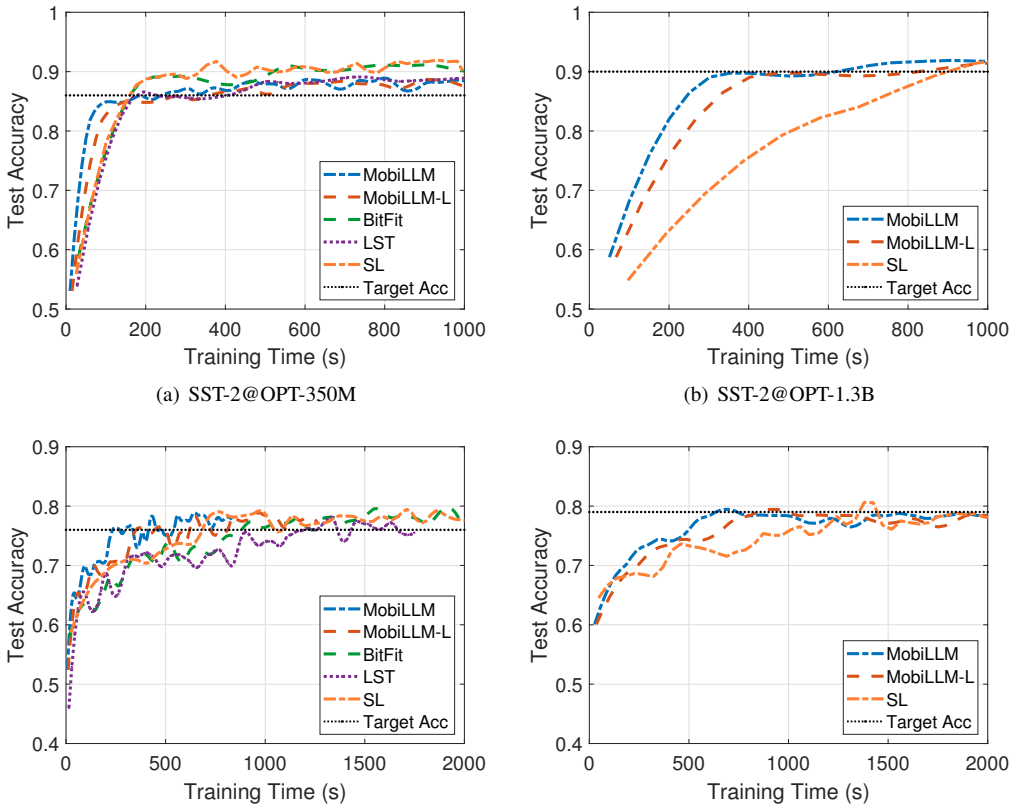
while only introducing a 2% accuracy drop. Against other PEFT baselines, i.e., LoRA and BitFit, MobiLLM is much better than them by reducing averaged 75% memory requirements. Using a local deployment without server assistance, MobiLLM-L reduces memory usage by 2.48 GB compared to LST, thanks to its lightweight adapter-based side-network design. With server-assisted side-tuning, MobiLLM further reduces memory usage to 1.621 GB. The significant improvement over all the baselines is mainly attributed to MobiLLM's design, which offloads dominant computational and memory burdens to the high-performance server. It then allows the mobile device to handle only the forward pass, in contrast to other baselines that all require the backward pass on the mobile device and can easily result in out-of-memory errors. Moreover, MobiLLM enables the mobile device to fine-tune LLM with a memory cost close to that of on-device inference, as verified in Table III. That makes on-device LLM fine-tuning practical, especially for memory-constrained mobile devices.

MobiLLM makes the billion-sized LLM model fine-tuning affordable to the mobile device. We further evaluate MobiLLM using a billion-sized model, OPT-1.3B, and report the results in Table IV. LoRA, BitFit and LST have excessively huge memory footprints exceeding 10 GB, rendering them infeasible for most mainstream mobile devices. In contrast, our MobiLLM effectively supports fine-tuning such sizable models on a single mobile device. For instance, MobiLLM requires only 4.487 GB of memory on the mobile device to run the fine-tuning task smoothly on Xavier with 4.6 GB DRAM. Those results demonstrate that even resource-constrained mobile devices can reliably fine-tune such billion-sized LLMs without the risk of memory overflow.

B. Advantages in Memory Efficiency

MobiLLM achieves near-stable memory usage across diverse training configurations. Figure 4(a)-4(b) demonstrate how batch size, sequence length, and model size affect the memory usage. Fig. 4(a) shows that while memory usage increases with batch size for all methods, MobiLLM consistently maintains the lowest memory footprint among all. Besides, MobiLLM exhibits slower memory growth compared to other baselines as the batch size increases. This is because, from a memory usage perspective, the batch size primarily affects the amount of intermediate activations that need to be stored for backpropagation, whereas our MobiLLM frees the mobile device from performing backpropagation entirely. Fig. 4(b) shows memory usage for varying sequence lengths. Similar to the impact of batch size, LST and MobiLLM alleviate the growth rate of memory footprint of intermediate activations. Notably, MobiLLM requires only 41% of the device-side memory usage of LST, when the sequence length is set to 256.

We further evaluates the impact of different model sizes, as shown in Fig. 4(c), testing our approach at five different model scales: OPT-125M, OPT-350M, OPT-1.3B, OPT-2.7B, and OPT-6.7B. The results indicate that MobiLLM consistently outperforms baselines, with its memory advantage widening as model size grows. These results verify that MobiLLM makes the mobile device's memory usage insensitive to training configurations such as batch size and sequence length, having a near-consistent memory cost. This alleviates the common "occasional success or failure" issue for fine-tuning specific



(a) SST-2@OPT-350M (b) SST-2@OPT-1.3B

(c) MRPC@OPT-350M (d) MRPC@OPT-1.3B

Fig. 6: Convergence performance on various models and tasks (On Xavier).

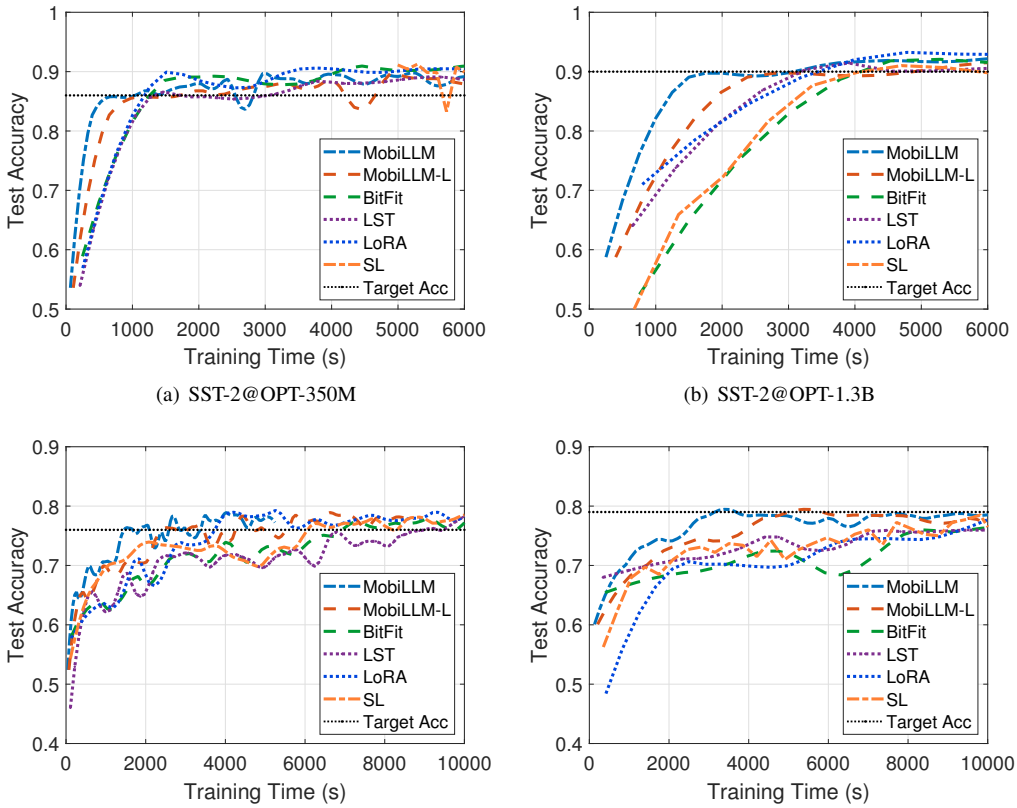
models under varying settings. It further helps to prevent memory overflows on mobile devices even when processing larger batch sizes and longer sequences, ensuring reliable fine-tuning performance.

MobiLLM excels in memory saving. Figure 5 further shows the breakdown of memory footprint for different methods when fine-tuning OPT-1.3B. Both LoRA and BitFit significantly reduce memory consumption by decreasing the number of trainable parameters, thereby remarkably lowering the memory required for optimizer states, as shown in the zoomed-in view in Fig. 5. MobiLLM and LST further optimize the memory usage by establishing backpropagation highways for side-tuning, which reduces the need to store intermediate activations for the backward pass. As a result, it achieves a total memory reduction of $2-4$ GB. MobiLLM also achieves an additional reduction of $6.4-6.7$ GB memory compared to LST, owing to two key factors: 1) MobiLLM offloads the intermediate activation memory burden of the side-network to the server. 2) MobiLLM eliminates the need for storing the optimizer states at the mobile device since it makes the mobile device only execute forward propagation.

C. Advantages in Fine-tuning Acceleration

MobiLLM accelerates on-device fine-tuning. Figure 6 and Fig. 7 show training curves that evaluate the accuracy-to-time performance of MobiLLM and other baselines. Specifically, the results in Fig. 6 are obtained by fine-tuning OPT-350M/1.5B models (batch size=12) on a Xavier device equipped with a GPU offering 4.6 GB DRAM for model training. Fig. 7 shows results from experiments on a CPU-only laptop, which, unlike Xavier, lacks hardware acceleration for neural network computations but benefits from larger available memory. Here, we exclude the results of the Full-FT method from comparison as its memory requirements exceed the capabilities of both devices.

Overall, the proposed MobiLLM consistently outperforms its peer designs across various LLMs and tasks, achieving significant training speedups while maintaining comparable accuracy. Compared with the LoRA baseline, MobiLLM expedites the fine-tuning to the target test accuracy by an average of $2.3\times$ on the laptop. Crucially, Fig. 6(b) and Fig. 6(d) verifies that MobiLLM-L (the fully localized variant of our MobiLLM) stands out as the only non-split solution capable of fine-tuning the billion-scale OPT-1.3B model under Xavier's 4.6 GB memory constraint-other methods that do not employ model partitioning fail to initialize under this limitation. This advantage stems from its training strategy (as discussed in Sec. IV-C): unlike LoRA and BitFit, which compute gradients through the frozen backbone, MobiLLM-L updates only the side-network, avoiding backpropagation through the backbone and thus reducing memory footprint and speeding up tuning. Moreover, MobiLLM-L also surpasses LST in efficiency. Although LST uses scaled-down transformer-based side networks, their structure still entails



(a) SST-2@OPT-350M (b) SST-2@OPT-1.3B

(c) MRPC@OPT-350M (d) MRPC@OPT-1.3B

Fig. 7: Convergence performance on various models and tasks (On CPU-only laptop).

high FLOPs due to compute-intensive self-attention and feed-forward layers. In contrast, MobiLLM's adapter-style side-network is computationally lighter.

As shown in Fig. 6 and Fig. 7, MobiLLM achieves additional speedup over its localized variant (MobiLLM-L) by offloading the side-network's training to the server, thereby leveraging the server's high-performance compute for the associated forward and backward passes. Moreover, although both MobiLLM and SL adopt a form of model partitioning, MobiLLM's design decouples the device-side forward pass from the server-side side-network training, enabling parallel computation. This eliminates device idle time and leads to better acceleration than the sequential execution scheme in SL. In addition, since the mobile device in MobiLLM focus solely on inference-related computations, cutting-edge inference-centric hardware acceleration techniques can be effortlessly integrated to further achieve speedups.

TABLE V: Performance comparison with different activation quantizers. (Batch size=16, Sequence length=256)

Model	Quantizer Configuration	Data Size (MB)	Accuracy
OPT-350M	No Act. Quant.	190	79.1
	FP8 Act. Quant.	99.6	78.6
	FP4 Act. Quant.	49.2	78.1
	NP4 Act. Quant.	49.2	78.8
OPT-1.3B	No Act. Quant.	400	81.5
	FP8 Act. Quant.	200.3	81.2
	FP4 Act. Quant.	100.2	80.4
	NP4 Act. Quant.	100.2	81.3

D. Sensitivity Study

Sensitivity to activation quantization levels. Table V illustrates how varying quantization levels of intermediate activation values influence the performance of MobiLLM. The results show that MobiLLM achieves similar accuracy performance across all quantization settings. When adopting the FP4 data type, MobiLLM incurs an average accuracy drop of only 1% compared to non-quantized methods while reducing per-iteration transmission volume by approximately $4\times$. When adopting the NP4 data format, the accuracy remains even more stable. The results validate that MobiLLM can balance the trade-off between fine-tuning accuracy and transmission burden by appropriately selecting the activation quantization level. Furthermore, MobiLLM is compatible with diverse quantization techniques and benefits from their performance gains, which allows MobiLLM to tolerate quantization noise within an acceptable range to achieve the desired fine-tuning performance.

Sensitivity to transmission rates. By fixing the activation

TABLE VI: Sensitivity to different transmission rate. (Model: OPT-350M)

Time Per Iteration (s)	10 Mbps	60 Mbps	100 Mbps	Device only
Batch size=12	7.3	5.5	5.4	6.8
Batch size=16	9.8	7.48	7.5	8.4
Batch size=24	14.7	10.1	10	—
Batch size=30	18.4	14	14.1	—

quantization level to be 4-bit, we also study the impacts of different transmission rates on the convergence time of MobiLLM, as reported in Table VI. To ensure the side-network effectively learns from the device's data, the mobile device occasionally reports its intermediate activations, whose size is proportional to the product of batch size and sequence length. At relatively higher uplink rates (e.g., 60 Mbps and 100 Mbps), MobiLLM converges faster than the device-only method despite introducing transmission delays. This is achieved through our overlapping training and activation quantization designs, which allow the mobile device to continuously conduct forward propagation concurrently with low-bit activation transmission. Moreover, the quantization of activations effectively reduces the transmission delays. The server's efficient side-network computation further accelerates training compared to using a mobile device alone, mitigating the impact of transmission delays. Even at a slow transmission rate (e.g., 10 Mbps), MobiLLM maintains acceptable training time (only a 7% increase compared to device-only), where the transmission delay emerges as a bottleneck. Notably, with a batch size larger than 24, MobiLLM can still support on-device fine-tuning, although the longer transmission delay slightly prolongs the training time. In contrast, device-only methods exceed the device's memory capacity and fail to perform fine-tuning under those settings. Future deployment or wide coverages of high-speed networks (e.g., 5G beyond, 6G, or Wi-Fi 6) will strengthen MobiLLM's power to enable or accelerate LLM fine-tuning on resource-constrained mobile devices.

E. Additional Discussion

Energy Efficiency Implications: Although a direct measurement of energy consumption is reserved for future work, the performance improvements demonstrated by MobiLLM, particularly in computational and memory efficiency, are consistent with improved energy utilization. On mobile Systems-on-Chips (SoCs), reductions in computational load and memory operations typically correlate with lower power consumption. Thus, the efficiency gains observed in our experiments suggest that MobiLLM offers a favorable direction for energy-efficient on-device fine-tuning.

Compatibility with Model Quantization: To further alleviate on-device memory requirements, MobiLLM is fully compatible with model quantization. Our additional experiments reveal that with the backbone model quantized to INT4, MobiLLM limits on-device memory usage to 1.176 GB when fine-tuning OPT-350M with batch size 16 and sequence length 256. For fine-tuning an OPT-1.3B model, the memory usage is remarkably reduced to 2.751 GB (over a $5\times$ reduction compared to LoRA fine-tuning) while maintaining a negligible accuracy drop of just 2%. In addition, low-precision data are typically faster to execute on modern accelerators by utilizing the integer kernels, which are supported by a wide range of hardware (e.g., NVIDIA GPUs, Intel CPUs, Qualcomm DSPs, etc.). These facts highlight the practical value of combining MobiLLM with quantization for deploying LLMs in resource-limited settings.

VII. Conclusion and Future Work

In this paper, we have proposed MobiLLM, a novel scheme for on-device LLM fine-tuning. MobiLLM separates parallel adapters from the backbone network, and offloads dominant computational and memory burdens in the fine-tuning process to the edge server. It allows the mobile device to retain a frozen backbone model and perform only forward propagation parallel to the server-side execution, while keeping their data locally on the device. Comparative analyses have revealed that MobiLLM outperforms benchmarks, including a $4\times$ reduction in memory footprint and a $2.3\times$ convergence speedup.

Crucially, MobiLLM fundamentally rethinks workload partitioning in device-server collaborative learning system - moving beyond naive layer-wise LLM splitting - by innovatively distributing computations based on operator-level characteristics and memory access patterns. This principled approach unlocks the potential for on-device fine-tuning of billion-sized LLMs, such as OPT-1.3B, demonstrating viability on commodity mobile hardware, even a CPU-only one. For the future work, we will explore integrating MobiLLM with post-training quantization and selective activation transmission to further reduce memory and communication overhead.We will also conduct a more comprehensive evaluation under highly dynamic network conditions to further verify the system's robustness in practical environments.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[3] 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," *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 1877–1901, 2020.

[4] L. Li, D. Shi, R. Hou, H. Li, M. Pan, and Z. Han, "To talk or to work: Flexible communication compression for energy efficient federated learning over heterogeneous mobile edge devices," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, May 2021, pp. 1–10.

[5] R. Chen, Q. Wan, X. Zhang, X. Qin, Y. Hou, D. Wang, X. Fu, and M. Pan, "EEFL: High-speed wireless communications inspired energy efficient federated learning over mobile devices," in *Proc. of The 21st ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Helsinki, Finland, June 2023, pp. 544–556.

[6] Qualcomm, "Snapdragon 8 elite mobile platform," https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-elite-mobile-platform/, accessed Nov. 2024.

[7] G. DeepMind, "Gemini," https://gemini.google.com/, accessed Nov. 2024.

[8] Google, "Gemini help," https://support.google.com/gemini/, accessed Nov. 2024.

[9] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen *et al.*, "Parameter-efficient fine-tuning of large-scale pre-trained language models," *Nature Machine Intelligence*, vol. 5, no. 3, pp. 220–235, 2023.

[10] 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 *Proc. International Conference on Machine Learning (ICML)*, Long Beach, CA, Jun. 2019, pp. 1–10.

[11] 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," Apr. 2022.

[12] 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.

[13] 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," vol. 36, 2023, pp. 12 312–12 331.

[14] Z. Wang, Y. Zhou, Y. Shi, and K. B. Letaief, "Federated fine-tuning for pre-trained foundation models over wireless networks," *IEEE Transactions on Wireless Communications*, vol. 24, no. 4, pp. 3450–3464, 2025.

[15] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "PipeDream: Generalized pipeline parallelism for dnn training," in *Proc. 27th ACM Symposium on Operating Systems Principles (SOSP)*, Ontario, Canada, 2019, pp. 1–15.

[16] Y. Chen, Q. Yang, S. He, Z. Shi, J. Chen, and M. Guizani, "FTPipeHD: A fault-tolerant pipeline-parallel distributed training approach for heterogeneous edge devices," *IEEE Transactions on Mobile Computing*, vol. 23, no. 4, pp. 3200–3212, 2023.

[17] S. Ye, L. Zeng, X. Chu, G. Xing, and X. Chen, "Asteroid: Resource-efficient hybrid pipeline parallelism for collaborative dnn training on heterogeneous edge devices," in *Proc. the 30th Annual International Conference on Mobile Computing and Networking (MobiCom)*, Washington D.C., Oct. 2024, pp. 1–15.

[18] 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.

[19] B. Liao, S. Tan, and C. Monz, "Make pre-trained model reversible: From parameter to memory efficient fine-tuning," vol. 36, Dec. 2023, pp. 15 186–15 209.

[20] Y.-L. Sung, J. Cho, and M. Bansal, "LST: Ladder side-tuning for parameter and memory efficient transfer learning," *Proc. of Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 12 991–13 005, 2022.

[21] H. Zhao, W. Du, F. Li, P. Li, and G. Liu, "Fedprompt: Communication-efficient and privacy-preserving prompt tuning in federated learning," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Rhodes Island, Greek, 2023, pp. 1–5.

[22] 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," *IEEE Journal of Selected Topics in Signal Processing*, 2025, early access.

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

[24] L. Li, X. Chen, and W. Wu, "RingAda: Pipelining large model fine-tuning on edge devices with scheduled layer unfreezing," *arXiv preprint arXiv:2502.19864*, 2025.

[25] S. Zhang, W. Wu, L. Song, and X. Shen, "Efficient model training in edge networks with hierarchical split learning," *IEEE Transactions on Mobile Computing*, 2025, to appear.

[26] X. Shen, J. Gao, W. Wu, M. Li, C. Zhou, and W. Zhuang, "Holistic network virtualization and pervasive network intelligence for 6G," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 1, pp. 1–30, 2022.

[27] W. Wu, C. Zhou, M. Li, H. Wu, H. Zhou, N. Zhang, X. S. Shen, and W. Zhuang, "AI-native network slicing for 6G networks," *IEEE Wireless Communications*, vol. 29, no. 1, pp. 96–103, 2022.

[28] T. Xiang, Y. Bi, X. Chen, Y. Liu, B. Wang, X. Shen, and X. Wang, "Federated learning with dynamic epoch adjustment and collaborative training in mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 4092–4106, 2023.

[29] X. Chen, W. Wu, L. Li, and F. Ji, "LLM-empowered iot for 6g networks: Architecture, challenges, and solutions," *IEEE Internet of Things Magazine*, 2025, to appear.

[30] M. Xu, D. Cai, Y. Wu, X. Li, and S. Wang, "Federated fine-tuning of billion-sized language models across mobile devices," *arXiv preprint arXiv:2308.13894v2*, 2024.

[31] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," vol. 36, 2023, pp. 10 088–10 115.

[32] Z. Zhao, Y. Mao, Z. Shi, Y. Liu, T. Lan, W. Ding, and X.-P. Zhang, "Aquila: Communication efficient federated learning with adaptive quantization in device selection strategy," *IEEE Transactions on Mobile Computing*, vol. 23, no. 6, pp. 7363–7376, 2023.

[33] T. Wolf, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[34] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[35] A. Warstadt, "Neural network acceptability judgments," *arXiv preprint arXiv:1805.12471*, 2019.

[36] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Seattle, USA, Oct. 2013, pp. 1631–1642.

[37] B. Dolan and C. Brockett, "Automatically constructing a corpus of sentential paraphrases," in *Proc. 3rd International Workshop on Paraphrasing (IWP)*, Jeju Island, Korea, 2005.

[38] S. Iyer, N. Dandekar, K. Csernai *et al.*, "First Quora dataset release: Question pairs," *data. quora. com*, 2017.

[39] D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia, "Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation," *arXiv preprint arXiv:1708.00055*, 2017.

[40] A. Williams, N. Nangia, and S. R. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," *arXiv preprint arXiv:1704.05426*, 2017.

[41] P. Rajpurkar, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[42] B. W. Matthews, "Comparison of the predicted and observed secondary structure of T4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

Liang Li is an assistant researcher at the Frontier Research Center, Pengcheng Laboratory, Shenzhen, China. She received her Ph.D. degree in Information and Communication Engineering from Xidian University, China, in 2021. She was a visiting Ph.D. student with the Department of Electrical and Computer Engineering, University of Houston, TX, USA, from 2018 to 2020. She was a postdoctoral faculty member with the School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing, China, from 2021 to 2023. Her research interests include 6G networks, mobile LLM, distributed machine learning.

Xingke Yang received the B.Eng. degree from Southwest Jiaotong University, Chengdu, China, in 2022. He is currently pursuing the Ph.D. degree with the University of Houston, Houston, TX, USA. His research focuses on efficient fine-tuning of large language models on mobiles, retrieval-augmented generation, and distributed machine learning.

Wen Wu received the Ph.D. degree in Electrical and Computer Engineering from the University of Waterloo, Waterloo, ON, Canada, in 2019. He is an Associate Researcher at the Frontier Research Center, Pengcheng Laboratory, Shenzhen, China. His research interests include 6G networks, edge LLM, split learning, network intelligence, and network virtualization. Dr. Wu received the OJVT Best Paper Award from the IEEE Vehicular Technology Society in 2025, and the Award for Excellence (Early Career Researcher) from the IEEE Hyper-Intelligence Technical Committee in 2022. He also received the World Top 2% Scientist Award in 2023-2025, USENIX Security Distinguished Paper Award in 2024, ComSoc TEA Excellence Camp Runner-Up Award in 2023, IEEE ICCT Best Student Paper Award in 2025, and IEEE CIC/ICCC Best Paper Award in 2022. Dr. Wu serves as Track Co-Chair for IEEE VTC 2022-2025, and Workshop Co-Chair for IEEE INFOCOM 2022-2025, IEEE Globecom 2024, IEEE MASS 2025, and IEEE ICCC 2023-2025. He serves as Editor or Guest Editor for IEEE TMC, IEEE Networking Letters, Hindawi WCMC, China Communications, Electronics, and Springer PPNA. He has published over 130 refereed IEEE journal and conference papers, as well as 6 books/book chapters, filed 10+ PCT patents, and issued 2 US patents.

Hao Wang is an Assistant Professor in the Department Electrical and Computer Engineering at Stevens Institute of Technology, Hoboken, NJ, USA. He received both his B.E. degree in Information Security and M.E. degree in Software Engineering from Shanghai Jiao Tong University, Shanghai, China, in 2012 and 2015 respectively, and the Ph.D. degree in the Department of Electrical and Computer Engineering at the University of Toronto, Canada in 2020. His research interests include distributed ML systems, AI security and forensics, privacy-preserving data analytics, serverless computing, and high-performance computing. He is a recipient of the NSF CRII Award.

Tomoaki Otsuki (Ohtsuki) received the B.E., M.E., and Ph. D. degrees in Electrical Engineering from Keio University, Yokohama, Japan in 1990, 1992, and 1994, respectively. From 1994 to 1995, he was a post-doctoral fellow and a Visiting Researcher in Electrical Engineering at Keio University. From 1993 to 1995, he was a Special Researcher at the Fellowships of the Japan Society for the Promotion of Science for Japanese Junior Scientists. From 1995 to 2005, he was with the Science University of Tokyo. In 2005, he joined Keio University. He is now a Professor at Keio University. From 1998 to 1999, he was with the department of electrical engineering and computer sciences, University of California, Berkeley. He is engaged in research on wireless communications, optical communications, signal processing, and information theory. Dr. Ohtsuki is a recipient of the 1997 Inoue Research Award for Young Scientist, the 1997 Hiroshi Ando Memorial Young Engineering Award, Ericsson Young Scientist Award 2000, 2002 Funai Information and Science Award for Young Scientist, IEEE the 1st Asia-Pacific Young Researcher Award 2001, the 5th International Communication Foundation (ICF) Research Award, 2011 IEEE SPCE Outstanding Service Award, the 27th TELECOM System Technology Award. He has published more than 325 journal papers and 550 international conference papers.

He served as a Chair of IEEE Communications Society, Signal Processing for Communications and Electronics Technical Committee. He served as a technical editor of the IEEE Wireless Communications Magazine and an editor of Elsevier Physical Communications.He is now serving as an Area Editor of the IEEE Transactions on Vehicular Technology and an editor of the IEEE Communications Surveys and Tutorials. He is also serving as the IEEE Communications Society, Asia Pacific Board Director. He was Vice President and President of the Communications Society of the IEICE, also he was a distinguished lecturer of the IEEE. He is a fellow of the IEICE, a Fellow of Asia-Pacific Artificial Intelligence Association (AAIA), a senior member of the IEEE, and a member of the Engineering Academy of Japan.

Xin Fu received the Ph.D. in Computer Engineering from University of Florida in 2009. She was an NSF Computing Innovation Fellow with the CS Department, University of Illinois at Urbana-Champaign from 2009 to 2010. From 2010 to 2014, she was an Assistant Professor at the EECS Department, University of Kansas. Currently, she is a Professor at the ECE Department, University of Houston. Her research interests include energy-efficient computing, machine learning, edge/mobile computing, high-performance computing. Dr. Fu is a recipient of 2014 NSF Faculty Early CAREER Award, 2012 Kansas NSF EPSCoR First Award, and 2009 NSF Computing Innovation Fellow.

Miao Pan is a Full Professor in the Department of Electrical and Computer Engineering at University of Houston. He was a recipient of NSF CAREER Award in 2014. Dr. Pan received his Ph.D. degree in Electrical and Computer Engineering from University of Florida in August 2012. Dr. Pan's research interests include wireless for AI, mobile AI systems, LLM security & privacy, deep learning privacy, new biometric based authentication, quantum computing privacy, wireless networks, and underwater IoT networks. He has published 3 books and book chapters, 5 patents, more than 150 papers in prestigious journals/magazines and more than 160 papers in top conferences. He has also been serving as a TPC Co-Chair for Mobiquitous 2019, ACM WUWNet 2019, Local Chair for MobiHoc 2025, etc. Dr. Pan is an Associate Editor for ACM Computing Surveys, IEEE Journal of Oceanic Engineering, IEEE Open Journal of Vehicular Technology and IEEE Internet of Things (IoT) Journal (Area 5: Artificial Intelligence for IoT), and used to be an Associate Editor for IEEE Internet of Things (IoT) Journal (Area 4: Services, Applications, and Other Topics for IoT) from 2015 to 2018. Dr. Pan is a member of AAAI, a senior member of ACM, and a senior member of IEEE and IEEE Communications Society.

Xuemin (Sherman) Shen (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990. He is a University Professor with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research focuses on network resource management, wireless network security, Internet of Things, 5G and beyond, and vehicular networks. Dr. Shen is a registered Professional Engineer of Ontario, Canada, an Engineering Institute of Canada Fellow, a Canadian Academy of Engineering Fellow, a Royal Society of Canada Fellow, a Chinese Academy of Engineering Foreign Member, and an Engineering Academy of Japan International Fellow.

Dr. Shen received the "West Lake Friendship Award" from Zhejiang Province in 2023, the Canadian Award for Telecommunications Research from the Canadian Society of Information Theory (CSIT) in 2021, the R.A. Fessenden Award in 2019 from IEEE, Canada, Award of Merit from the Federation of Chinese Canadian Professionals (Ontario) in 2019, James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, Joseph LoCicero Award in 2015 and Education Award in 2017 from the IEEE Communications Society (ComSoc), and Technical Recognition Award from Wireless Communications Technical Committee (2019) and AHSN Technical Committee (2013). He has also received the Excellent Graduate Supervision Award in 2006 from the University of Waterloo and the Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada. He serves/served as the General Chair for the 6G Global Conference'23, and ACM Mobihoc'15, Technical Program Committee Chair/Co-Chair for IEEE Globecom'24, '16 and '07, IEEE Infocom'14, and IEEE VTC'10 Fall. Dr. Shen is the Past President of the IEEE ComSoc, the Vice President for Technical & Educational Activities, Vice President for Publications, Member-at-Large on the Board of Governors, Chair of the Distinguished Lecturer Selection Committee, and Member of IEEE Fellow Selection Committee of the ComSoc. Dr. Shen served as the Editor-in-Chief of the IEEE IoT JOURNAL, IEEE Network, and IET Communications.