Resource-Efficient Personal Large Language Models Fine-Tuning With Collaborative Edge Computing

Shengyuan Ye, *Graduate Student Member, IEEE*, Bei Ouyang, Tianyi Qian, Liekang Zeng, *Member, IEEE*, Jingyi Li, Jiangsu Du, Xiaowen Chu, *Fellow, IEEE*, Guoliang Xing, *Fellow, IEEE*, and Xu Chen

***Abstract*—Large language models (LLMs) have enabled transformative applications at the network edge, such as intelligent personal assistants. However, data privacy and security concerns necessitate a shift from cloud-centric paradigms to edge-based fine-tuning for personal LLMs. This transition is significantly hindered by intensive computational requirements and inherent resource scarcity, creating a "resource wall" that compromises training efficiency and feasibility. While current parameter-efficient fine-tuning (PEFT) and resource management strategies attempt to mitigate these constraints, they remain insufficient for the limited capacities of individual edge devices. To address these challenges, we propose PAC+, a resourceefficient collaborative edge AI framework for in-situ personal LLM fine-tuning. PAC+ overcomes the resource bottlenecks through a sophisticated algorithm-system codesign: (1) Algorithmically, PAC+ introduces a fine-tuning technique optimized for parameters, time, and memory. It utilizes Parallel Adapters to circumvent the need for a full backward pass through the LLM backbone. Furthermore, an activation cache mechanism streamlines the process by negating redundant forward passes across multiple epochs. (2) Systematically, PAC+ aggregates proximate edge devices into a collective resource pool, employing hybrid data and pipeline parallelism to orchestrate distributed training. By leveraging the activation cache, PAC+ enables the exclusive fine-tuning of Parallel Adapters via data parallelism, effectively bypassing the backbone's constraints. Extensive evaluation of the prototype implementation demonstrates that PAC+ significantly outperforms existing collaborative edge training systems, achieving up to a 9.7× end-to-end speedup. Furthermore, compared to mainstream LLM fine-tuning algorithms, PAC+ reduces memory footprint by up to 88.16%.**

***Index Terms*—Edge intelligence, personal LLM fine-tuning, collaborative edge computing, distributed training.**

Received 13 February 2025; revised 10 December 2025; accepted 11 January 2026. Date of publication 16 January 2026; date of current version 30 January 2026. This work was supported by the Longgang District Shenzhen's "Ten Action Plan" for Supporting Innovation Projects under Grant LGKCSDPT 2024004. Recommended for acceptance by R. Tolosana.*(Shengyuan Ye and Bei Ouyang contributed equally to this work.) (Corresponding author: Xu Chen.)*

Shengyuan Ye, Bei Ouyang, Tianyi Qian, Jingyi Li, and Jiangsu Du are with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510275, China (e-mail: yeshy8@mail2.sysu.edu.cn; ouyb9@mail2.sysu.edu.cn; qianty@mail2.sysu.edu.cn; lijy573@mail2.sysu.edu.cn; dujiangsu@mail.sysu.edu.cn).

Liekang Zeng and Guoliang Xing are with the The Hong Kong University of Science and Technology (Guangzhou), Guangzhou 511458, China (e-mail: lkzeng@cuhk.edu.hk; glxing@cuhk.edu.hk).

Xiaowen Chu is with the The Hong Kong University of Science and Technology (Guangzhou), Guangzhou 511458, China (e-mail: xwchu@ust.hk).

Xu Chen is with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510275, China, and also with the Shenzhen Institute of Artificial Intelligence and Robotics for Society, Shenzhen 518172, China (e-mail: chenxu35@mail.sysu.edu.cn).

Digital Object Identifier 10.1109/TPDS.2026.3654957

I. Introduction

LARGE language models (LLMs) [1], [2], [3] have ushered in a revolution in machine intelligence, owing to their exceptional capabilities in a wide range of machine learning tasks. While born on datacenter warehouse, LLMs have quickly sunk to edge devices and facilitated a range of intelligent applications at the network edge, such as *intelligent personal assistants* (IPAs) which are software agents that can augment individuals' abilities, complete complicated tasks, and even satisfy emotional needs. A recent survey [4] targeting LLM-based IPAs has revealed that over 80% of industry experts believe that, owing to the sensitive and privacy-critical nature of user data, personal LLMs should be fully (or primarily) hosted at the edge in order to enable privacy-preserving model personalization and serving. Fig. 1 illustrates the scenario of hosting a personal LLM-based intelligent agent within a smart home. A personal LLM agent provides users with high-performance, privacy-preserving intelligent services. Meanwhile, the agent also tracks user interactions, learns from experiences, and extracts knowledge to fine-tune the personal LLMs and further enhance the service quality.

While the serving of LLMs on edge devices has been made feasible through careful engineering [5], [6], [7], fine-tuning these models remains significantly challenging due to the resource-intensive nature of LLM training. Towards alleviating the resource challenges, some research works [8], [9] have explored parameter-efficient fine-tuning (PEFT) techniques, such as Adapters [10] and LoRA [11], which modify less than 2% of the model's parameters, thereby reducing resource requirements. Although these techniques are highly parameter-efficient, our analysis observes that they are not resource-efficient enough for edge environments. The inefficiency stems from Adapters and LoRA embedding trainable structures in the LLM backbone, necessitating complete backward passes through the LLMs during backpropagation. As we will empirically show in Section II, fine-tuning a popular LLM of T5-Base (0.25B) by Google with PEFT techniques can only reduce computational overhead by up to 30% compared to full model fine-tuning. In practice, fine-tuning the T5-Base on a typical edge device (e.g., NVIDIA Jetson Nano [12]) still demands a minimum of 72.6 minutes per training epoch employing LoRA. Moreover, on-device fine-tuning is severely hindered by the memory wall of a single device. Predominant techniques require substantial memory to accommodate both model parameters and intermediate results.

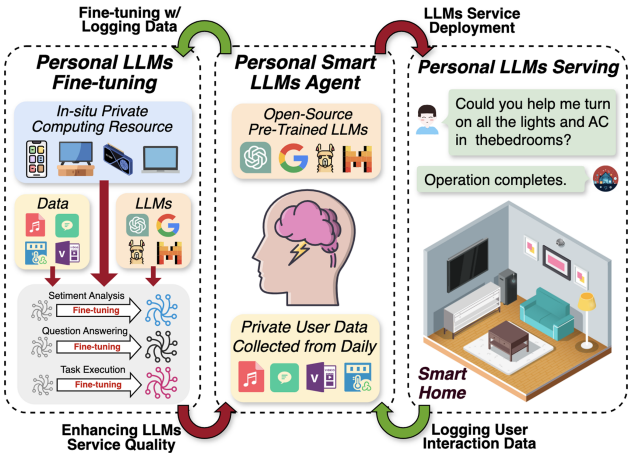


Fig. 1. An illustration of hosting personal LLM-based intelligent agents within a smart home.

The observed memory expense for fine-tuning LLMs like T5-Large (0.74B), which exceeds 7.1 GB with LoRA and 6.8 GB with Adapters, is often unaffordable as typical mobile devices only possess 4-12 GB DRAMs in total to run both system software and applications.

Other leading researchers have explored designing sophisticated resource management mechanisms (e.g., CPU-DSP co-execution [13], memory budget adapting [14], [15]) to leverage native resources, but are still bottlenecked by the intrinsic resource shortage of single device. To break the resource wall of a single device, we alternatively observe that prevalent edge environments like smart homes usually comprise a group of trusted idle devices beyond a single terminal (e.g., phones and smart-home devices). These accompanying devices are typically in physical proximity and can be associated as a resource augmentation for in-situ personal LLMs fine-tuning.

As motivated, in this paper, we introduce **PAC+**, a resource-efficient collaborative edge AI framework for personal LLMs fine-tuning. PAC+'s contribution goes beyond merely leveraging distributed edge devices, instead it breaks the resource wall of in-situ personal LLMs fine-tuning with a sophisticated algorithm-system co-design:

• *(Algorithm):* We evaluate two predominant PEFT techniques, Adapters and LoRA, and reveal that although parameter efficient, these techniques do not achieve sufficient resource efficiency. In light of the side-tuning [16] techniques, PAC+ leverages the existing Parallel Adapters approach [17], [18] to realize not only parameter but also time and memory-efficient personal LLMs fine-tuning. This technique provides a dedicated gradient "highway" for the trainable parameters, significantly reducing the computational and memory overhead of backpropagation on the LLM backbone. By employing low-precision quantization on the LLM backbone and tailored initialization methods for the Parallel Adapters, PAC+ further minimize the memory and time costs of LLM fine-tuning. Additionally, our Parallel Adapters stand out from other PEFT techniques by preserving the invariant intermediate activations from the LLM backbone for any given input sequence. By reusing these cached activations across multiple epochs, PAC+ increases resource efficiency and reduces fine-tuning latency by eliminating repetitive forward propagation through the LLM backbone.

• *(System):* PAC+ leverage edge devices in physical proximity and associate them as an edge resource pool for in-situ personal LLMs fine-tuning. Our fine-tuning process can be divided into two phases: (1) For the first epoch, the LLMs backbone, augmented with Parallel Adapters, is fine-tuned across multiple edge devices. To enhance scalability and training throughput, a hybrid parallelism approach that combines the merits of both data and pipeline parallelism is employed by PAC+ as a principle to manage collaborative training across multiple edge devices. A novel heterogeneity-aware planning algorithm is introduced for multidimensional resources optimization. (2) In subsequent fine-tuning epochs, the activation cache obviates the need for forward propagation through the LLM backbone, allowing for the exclusive fine-tuning of our Parallel Adapters using data parallelism.

We implement PAC+ in realistic testbeds with a cluster of edge devices. Extensive evaluations across three LLMs demonstrate that PAC+ not only accelerates fine-tuning, achieving up to a 9.7× speedup over existing collaborative edge training systems, but also reduces memory footprint by up to 88.16% compared to mainstream LLM fine-tuning algorithms.

The main contributions are summarized as follows.

- We conduct extensive measurement studies on prevalent PEFT techniques, demonstrating their inefficiency, and design a not only parameter but also time and memory efficient LLM fine-tuning technique for resource-limited edge environments.
- We analyze and design low-precision quantization for the LLM backbone and tailored initialization methods for the Parallel Adapters, enabling further optimization of both memory and time overhead in LLM fine-tuning.
- We design a hybrid parallel fine-tuning architecture to leverage edge devices in physical proximity, complemented by a novel heterogeneity-aware planning algorithm for optimizing multidimensional resources.
- We design a system cache mechanism for invariant intermediate activations, eliminating the need for a forward pass through the LLM backbone and enabling exclusive fine-tuning of the Parallel Adapters using data parallelism.
- We implement PAC+ and evaluate it in realistic edge testbeds. Experimental results show up to 9.7× fine-tuning acceleration and 88.16% memory reduction without sacrificing performance compared to state-of-the-art methods.

II. Motivation and Preliminaries

A. Transformer-Based LLMs and Fine-Tuning

Transformer-Based LLMs: Transformer-based LLMs have gained prominence in various language-related applications due to their impressive performance. These models consist of multiple Transformer layers, each comprising two main components: the Multi-head Attention and the Feed Forward block. The Multi-head Attention block utilizes linear layers to generate query (Q), key (K), and value (V) matrices for each attention

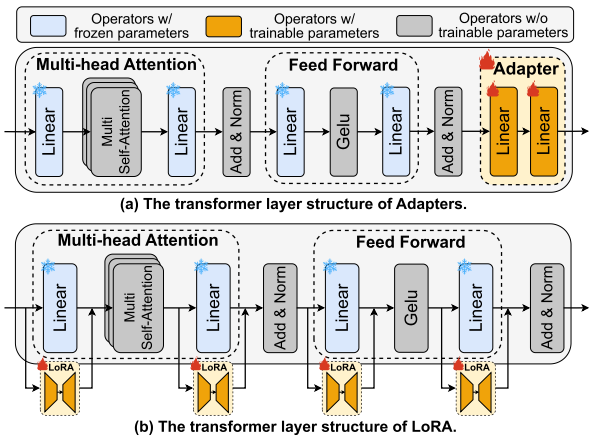


Fig. 2. Illustration of the model structures with two PEFT.

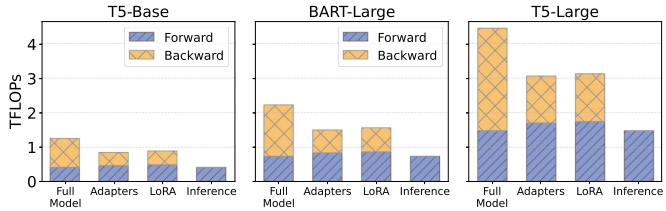


Fig. 3. The comparison of floating point of operations (FLOPs). Mini-batch size: 16; sequence length: 128.

TABLE I
THE BREAKDOWN OF MEMORY FOOTPRINT. "ACTIVATIONS" CONTAIN THE INTERMEDIATE RESULTS AND OPTIMIZER STATES. MODEL: T5-LARGE; MINI-BATCH SIZE: 16; SEQUENCE LENGTH: 128.

Techniques	Trainable Parameters	Memory Footprint (GB)			
		Weights	Activations	Gradients	Total
Full	737M (100%)	2.75	5.33	2.75	10.83
Adapters	12M (1.70 %)	2.80	4.04	0.05	6.89
LoRA	9M (1.26%)	2.78	4.31	0.04	7.13
Inference	/	2.75	/	/	2.75

head, allowing for independent self-attention computations. The outputs of these attention heads are then concatenated and processed through a final linear layer. The Feed Forward block involves two linear operations that expand the hidden dimension size then reduce it back.

Personal LLMs Fine-Tuning: The training of LLMs typically consists of two stages: pre-training and fine-tuning. Before being deployed for specific tasks, language models are often pre-trained on extensive text datasets containing vast linguistic data. The pre-training process enables the model to acquire a general understanding of linguistic structure and patterns that are widely applicable. The fine-tuning adapts the pre-trained model to various, concrete downstream language tasks such as intelligent personal assistants. During actual deployment, the data required for fine-tuning is often generated at the user end, which can carry significant concerns regarding data security and privacy. In recent years, in-situ learning on edge devices [14], [15], [19], [20] has emerged as a promising approach for customizing LLMs while preserving user data fully in-situ.

Full model fine-tuning updates all parameters of an LLM for a specific downstream task. However, it is impractical for adapting an LLM to multiple distinct downstream tasks, as each target task would require maintaining a separate LLM with whole parameters. Some leading researchers have proposed parameter-efficient fine-tuning (PEFT) techniques [10], [11], [21], [22] which adapt a small subset of the LLM parameters or a set of newly added parameters for each new task. Adapters [10] and LoRA [11] are two of the most widely used PEFT techniques. Fig. 2 illustrates how the transformer layer structure incorporates these two techniques. Specifically, adapters are compact bottleneck modules inserted at the end of each transformer layer. Similarly, LoRA injects trainable low-rank matrices into a frozen pre-trained model. These decompose the weight matrix parameter updates into two learnable low-rank matrices. Extensive experiments have demonstrated that these PEFT techniques can achieve performance comparable to full fine-tuning.

Although these PEFT techniques can greatly reduce the number of trainable parameters (around 98%), our analysis has revealed that they do not significantly decrease the computational and memory requirements during training. Fig. 3 illustrates the floating point of operations (FLOPs) of different fine-tuning techniques and inference. Adapters and LoRA exhibit a limited reduction in computation (around 30%). Table I summarizes the memory footprint breakdown for T5-Large. Although Adapters and LoRA minimize the gradient memory footprint by restricting the number of trainable parameters, the memory consumed by activations still constitutes substantial overhead, with a maximum reduction of only 36%. The reason is that both Adapters and LoRA introduce trainable structures within the LLM backbone, such as at the end of each transformer block or as bypasses to linear layers. Computing gradients for trainable parameters via backpropagation involves traversing the LLM backbone, compromising the efficiency of PEFT techniques due to the additional computational overhead and memory required to maintain considerable intermediate activations in LLM backbone.

B. Personal LLMs Fine-Tuning With Resource-Constrained Edge Devices

On-device fine-tuning enables leveraging idle resources at the edge while fully preserving user data privacy [14], [15], [19], [20]. This paradigm is widely adopted in privacy-sensitive edge computing applications. However, the resource-intensive nature of LLMs fine-tuning presents two significant challenges for resource-limited edge devices: **(1) The computational capabilities of edge devices are constrained.** Edge devices often face stark computational constraints compared to the powerful accelerators available in cloud datacenters. The Jetson Nano [12], a specialized platform for edge AI, peaks at a mere 0.47 TFLOPS, a tiny fraction of the 312 TFLOPS achievable with NVIDIA's A100 GPU typically found in data centers. Fine-tuning a T5-Base model with Adapters on a single Jetson Nano requires an epoch time of 72.6 minutes, which is $175.5\times$ longer than that in a NVIDIA A100 GPU, showing the fundamental contradiction between intensive LLM fine-tuning workload and constrained on-board resources. **(2) On-device fine-tuning is hindered by**

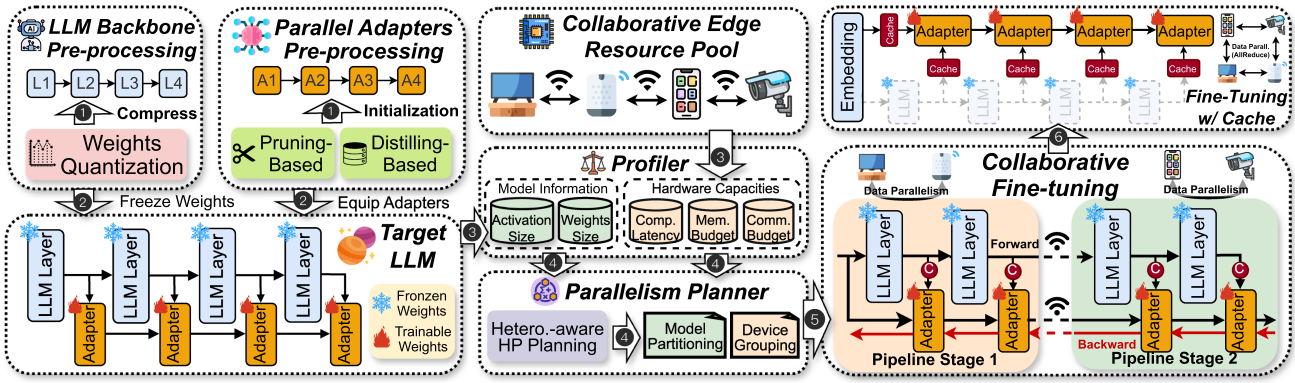


Fig. 4. PAC+ workflow.

the memory wall. As shown in Table I, fine-tuning the T5-Large model incurs a peak memory footprint that is often unaffordable for edge devices. For instance, although PEFT techniques such as Adapters and LoRA adjust only approximately 2% of the parameters, they still require substantial memory 6.89 GB and 7.13 GB respectively. Compared to full model fine-tuning, which requires over 10 GB, these techniques reduce memory usage by only 36%, often insufficient for typical mobile devices with 4-12 GB DRAM to run system software and applications.

To break the resource wall of a single edge device, in our work, we alternatively observe that prevalent edge scenarios usually comprise a group of trusted idle devices beyond a single terminal. These accompanying devices are typically located in close physical proximity, such as being connected to the same local area network (LAN), and can be utilized as a resource augmentation for in-situ LLMs fine-tuning acceleration. While several pioneering research works [7], [23] have delved into collaborative edge computing to overcome resource limitations faced by edge devices, the majority of these works primarily focus on LLMs inference. Other studies [8], [24] employing federated learning for fine-tuning LLMs with collaborative edge devices primarily address the dissolution of data silos, rather than resource augmentation within LANs.

C. Technical Challenges

According to the aforementioned analysis and discussion, we can summarize two key challenges inherent to fine-tuning personal LLMs on resource-constrained edge devices:

1) How to boost the time and memory efficiency for edge fine-tuning? The current predominant PEFT techniques like LoRA and Adapters, while parameter-efficient, are not inherently resource-efficient. These approaches still require substantial computational and memory resources for deployment. This poses a fundamental contradiction, as edge devices are typically constrained by cost and power considerations, and thus possess highly limited computational and memory capabilities. Therefore, we need to exploit a PEFT technique that is inherently more time and memory efficient, and well-suited for edge environments.

2) How to orchestrate resource-efficient collaborative computing across multiple edge devices? Leveraging multiple edge devices for distributed fine-tuning is not trivial. Careful selection of parallel architectures and parallelism planning strategies is necessary to maximize resource utilization and avoid out-of-memory issues. Furthermore, an algorithm-system co-design approach is required to seamlessly integrate the resource-efficient PEFT techniques with our collaborative edge system. This holistic design is crucial for enabling scalable and high-performance personal LLMs fine-tuning in resource-constrained edge environments.

III. System Overview

PAC+ is a resource efficient collaborative framework for personal LLMs fine-tuning across multiple edge devices. Fig. 4 illustrates the PAC+ workflow. PAC+ first constructs a lightweight Parallel Adapter for parameter-efficient LLM fine-tuning, developed as a lightweight version of the target transformer-based LLM. Next, PAC+ optimize resource efficiency during fine-tuning by pre-processing both the LLM and Parallel Adapters. This includes quantizing LLM parameters to a lower bit-width representation for reduced memory footprint, and applying initialization techniques such as structural pruning or knowledge distillation to accelerate convergence (Step ❶). PAC+ then integrate the Parallel Adapters module into the LLM, freezing the LLM backbone parameters and fine-tuning only the lightweight adapter for parameter efficient adapting (Step ❷). PAC+ profiler fine-tunes the LLM using a calibration dataset on edge devices to record the runtime profile required for parallelism planning (Step ❸). PAC+ planner then takes the profiling results as input and generates planning configurations, including LLM partitioning points and device grouping strategies. These configurations comprehensively addresses challenges including memory budget, limited communication capacity, and resource heterogeneity (Step ❹). The parallel configurations generated by the PAC+ planner are then applied to the edge devices, enabling time, memory, and parameter-efficient hybrid data and pipeline parallelism fine-tuning of the target LLM (Step ❺). Since the LLM backbone parameters remain fixed, the intermediate activations generated by the backbone model are invariant for a

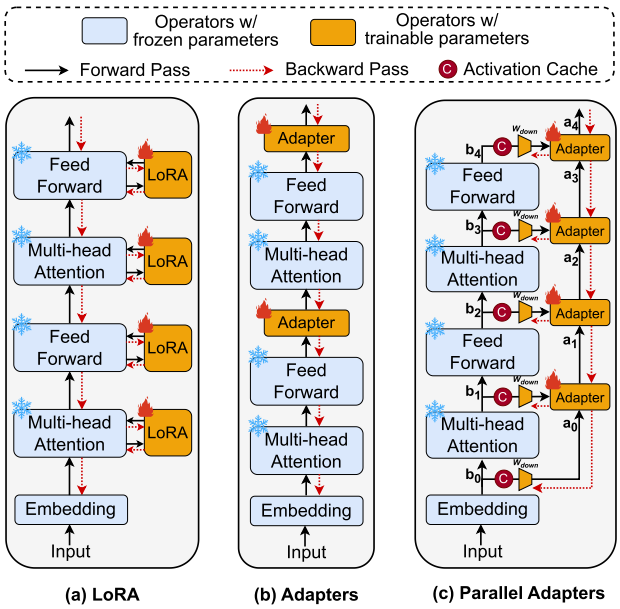


Fig. 5. Comparison between LLMs fine-tuning with LoRA, Adapters, and our Parallel Adapters.

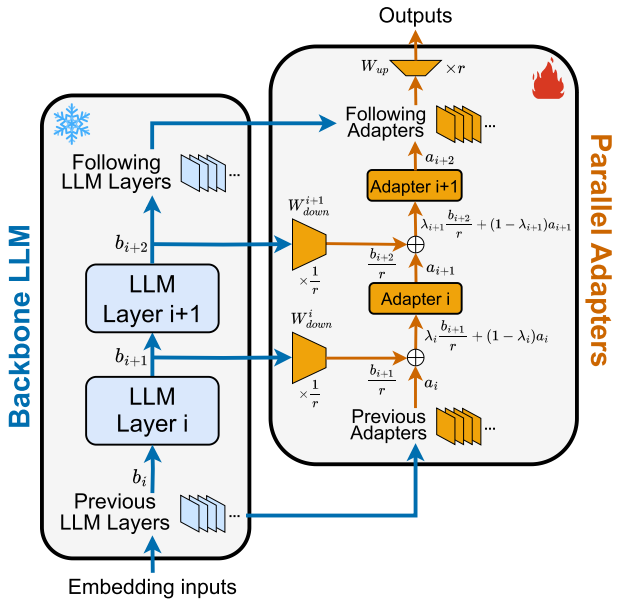


Fig. 6. Detailed illustration of the Parallel Adapters architecture.

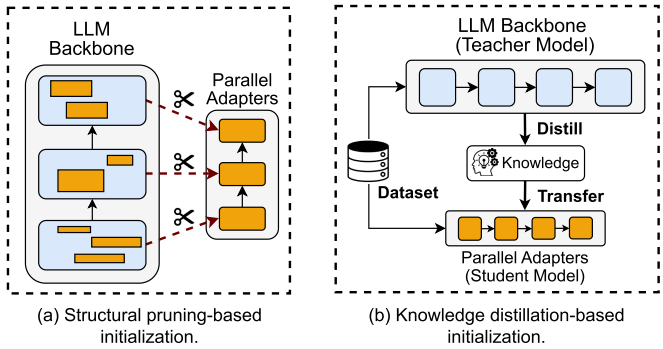


Fig. 7. Illustration of weight initialization for Parallel Adapters.

given input sequence. The PAC+ maintains a cache of these invariant activations. Through leveraging the cached activations, the efficiency of the fine-tuning process can be accelerated (Step ❻).

IV. Time, Memory and Parameter Efficient Fine-Tuning Algorithm

A. Fine-Tuning LLMs With Parallel Adapters

Observation and Key Insight: As discussed in Section II, while techniques such as LoRA [11] and Adapters [10] reduce the number of parameters that need to be updated during fine-tuning, they do not significantly reduce the computational and memory requirements during the training on edge devices. This is because the parameters being updated are still inside the LLM backbone. To calculate the gradients for backpropagation, the full backward passes through the entire pre-trained model are still necessary, as illustrated in Fig. 5(a) and (b). In the research field of AI, side-tuning [16] is a specialized fine-tuning technique. It adds a trainable side network that runs in parallel to the backbone model, with the side network's representation summed with the backbone's output in the final layer. Crucially, side-tuning only updates the side network, without backpropagating through the backbone model.

Parallel Adapters Architecture: In light of side-tuning, we employ a time and memory efficient personal LLMs fine-tuning technique with Parallel Adapters. The overall structure is illustrated in Fig. 5(c). Specifically, we decouple conventional Adapters [10] from the LLM backbone, avoiding their integration at the end of each transformer layer. Instead, we provide a dedicated parallel highway for our trainable adapters network, which takes intermediate activations from the backbone transformer as input and generates the final predictions. In this way, backpropagation through the LLM backbone is free, reducing memory demands for massive activations and computational burdens, thereby enhancing time and memory efficiency over techniques like Adapters and LoRA. Observing Fig. 5(c), the training paradigm of our Parallel Adapters can be viewed as a shift from fine-tuning the original LLM backbone to fine-tuning a lightweight proxy network that operates in parallel with the backbone. In our design, this lightweight proxy network (i.e., Parallel Adapter) is developed as a lightweight version of the backbone transformer-based LLM [17], [18]. Additionally, our adapter module also demonstrates broad compatibility with established LLM fine-tuning architectures, including the commonly used linear layers for upward and downward projections [10].

Fig. 6 provides a detailed illustration of the workflow of our Parallel Adapters architecture. To ensure the lightweight and resource-efficient nature of our proxy network, the hidden dimension of our Parallel Adapters will be $\frac{1}{r}$ times of the original hidden dimension in backbone. Consider an LLM backbone with L layers, where the intermediate outputs $\mathrm{b}_1, \mathrm{b}_2, \ldots, \mathrm{b}_L$ each contain n tokens, with a hidden dimensionality of d, such that $\mathbf{b}_i \in \mathbb{R}^{n \times d}$. We denote the embedding input sequence as

$\mathrm{b}_0 \in \mathbb{R}^{n\times d}$. Assuming adapters are inserted after every transformer layer of backbone LLM, Parallel Adapters consist of L adapters, yielding L intermediate outputs $\mathrm{a}_1, \mathrm{a}_2, \ldots \mathrm{a}_L$, $\mathbf{a}_i \in \mathbb{R}^{n\times \frac{d}{r}}$. To align the dimension of b_{i+1} with a_i, we learn a linear projection $\mathbf{W}^i_{down} \in \mathbb{R}^d \rightarrow \mathbb{R}^{\frac{d}{r}}$ to downsample ($\times\frac{1}{r}$) the intermediate activations from the LLM backbone, resulting in $\frac{\mathrm{b}_{i+1}}{r} \in \mathbb{R}^{n\times\frac{d}{r}}$. Next, we combine the downsampled LLM activations $\frac{\mathrm{b}_{i+1}}{r}$ with the intermediate results of the Parallel Adapters a_i using a learnable parameter λ_i: $\lambda_i \frac{b_{i+1}}{r} + (1 - \lambda_i)a_i$. This combined result is then used as the input to the i-th adapter, with λ_i initialized to 0.5. At the end of the Parallel Adapters, we learn a linear projection $\mathbf{W}_{up} \in \mathbb{R}^{\frac{d}{r}} \rightarrow \mathbb{R}^d$ to upsample the final outputs to match the dimension of the original language model head. Our evaluation in Section VI reveals that Parallel Adapters can achieve comparable model performance to mainstream fine-tuning techniques while being more resource-efficient and better suited for resource-constrained edge environments.

B. Activation Cache for Parallel Adapters

Observation and Opportunities: Leveraging Parallel Adapters substantially diminishes the computational and memory demands by circumventing backward propagation through the LLM backbone. However, for edge environments with limited resources, forward propagation calculations on the backbone of LLMs also require substantial computational resources. Fig. 3 demonstrates that the computational overhead for forward propagation constitutes 54% and 56% of the total overhead when fine-tuning the T5-Large with Adapters and LoRA, respectively.

To minimize the computational demand, we identify two distinct opportunities for utilizing Parallel Adapters in in-situ fine-tuning of LLMs: (1) During the pre-training phase of LLMs, due to the vast volumes of data involved, researchers typically train for only one epoch, meaning each sequence input is processed by the model a single time. However, in typical in-situ LLM fine-tuning scenarios, users often utilize small datasets collected from their specific context, repeatedly training the models with these inputs until achieving model convergence. (2) When employing parallel adapters to fine-tune LLMs, the parameters of the LLM backbone remain fixed. Unlike other PEFT techniques, the LLM backbone operates independently of the intermediate outputs generated by Parallel Adapters. Consequently, for a given input sequence, the activations generated by the LLM backbone are always invariant.

Fine-Tuning Parallel Adapters with PAC+ Activation Cache: Our key idea leverages the frozen parameters of the backbone model, enabling the caching of activations produced during the forward propagation of the same input sequence, thereby facilitating their reuse across multiple epochs [8]. As discussed in Section IV-A, the parallel adapters are a lightweight, separate network that takes the intermediate activations from the backbone transformer as input and generates predictions. During the first epoch, when processing a new input sequence, we cache all the input activations required by the Parallel Adapters that are obtained from the LLM backbone, as illustrated in Fig. 5(c), highlighted by the red circle. In subsequent fine-tuning epochs using the same input sequence, we can skip the forward propagation through the LLM backbone entirely, since the required activations have already been cached. The combination of Parallel Adapters and activation caching allows efficient fine-tuning of the LLMs without the need for both forward and backward propagation through the backbone network, thereby (1) significantly accelerating the fine-tuning process and (2) reducing the memory footprint by allowing the release of the memory space occupied by the LLM parameters.

C. Analysis of Weight Initialization for Parallel Adapters

Observation and Insights: As outlined in Section IV-A, our Parallel Adapters fine-tune a lightweight proxy network alongside the original LLM backbone. Proper initialization of the proxy network's weights is essential for accelerating convergence and reducing computational cost and latency, particularly in resource-constrained edge environments.

We begin by observing the PEFT technique employed by LoRA [11]. LoRA introduces low-rank matrices A and B, representing the weight update of the original model as the product of these two matrices, i.e., $\Delta W = BA$. In forward propagation, the model output is expressed as $h = Wx + \Delta Wx = Wx + BAx$. In backward propagation, the LLM backbone weights W are frozen, while the parameters of the low-rank matrices A and B are updated. At the beginning of fine-tuning, LoRA initialized the learnable parameter $\Delta W = 0$, with matrix A initialized using a random Gaussian distribution with zero mean, and matrix B initialized as a zero matrix. The rationale behind this initialization strategy is to ensure that the PEFT initial state remains as closely aligned as possible with the pre-trained model, minimizing random perturbations to the model output during the early stages of training [10], [11]. This helps ensure a smooth transition from the pre-trained state to the fine-tuning phase.

We design our initialization scheme based on this insight. As shown in Fig. 7, directly applying commonly used random Gaussian initialization or zero initialization to our lightweight parallel adapters results in a significant discrepancy between the initial outputs of the proxy network and the original pre-trained LLM, causing substantial perturbations during the early stages of model fine-tuning. Therefore, we aim for the initialization technique we employ to align the initial states of the lightweight proxy network with frozen LLM backbone. Given that our proxy network is designed as a lightweight version of the backbone LLM, its initialization weights can be derived from model compression techniques, specifically structural pruning and knowledge distillation. In this work, we design and analyze two initialization methods based on above two techniques.

Structural Pruning-Based Initialization: Structural pruning refers to the process of simplifying architectural components, such as dense layers, attention heads, or convolutional channels, while preserving their overall functionality [25], [26], [27]. It is widely employed to achieve model compression, effectively reducing memory footprint and accelerating inference. We observe that the concept of structural pruning can also be applied

to initialize the parameters of our Parallel Adapters, where the pruning algorithm takes the backbone LLM as input and outputs its lightweight version for initialization. We implemented the pruning algorithm in our study using Torch-Pruning [27], [28], a popular open-source toolkit for structural pruning atop PyTorch. The fundamental insight of the Torch-Pruning toolkit [28] lies in using a practical norm-based criterion to quantify parameter importance and prune low-significance weights.

Knowledge Distillation-Based Initialization: Knowledge distillation is another widely used model compression technique, where knowledge is transferred from larger, more competent teacher models to smaller student models that are more suitable for resource-efficient deployment [29], [30], [31]. Similarly, in the context of our task, the backbone LLM can be regarded as the larger teacher model, while the Parallel Adapters function as the smaller student model. Specifically, we first prompt the teacher model to generate labels along with rationales justifying the labels for an open dataset. We then leverage these labels, in addition to the rationales, to train smaller student models [32]. We implemented our distillation method based on a popular distillation toolkit open-sourced by Google Research [33]. In practice, since the distillation process does not require the use of private user data, it can be conducted in cloud data centers equipped with more powerful computational resources.

D. Further Memory Efficiency Through Fine-Tuning With Quantized LLMs

The memory footprint of LLM fine-tuning is primarily contributed by the *backbone LLM parameters*, *intermediate activations*, and *parameter gradients*. Our Parallel Adapters design reduces gradient memory overhead by significantly lowering the number of trainable parameters and minimizes intermediate activation memory overhead by avoiding backpropagation through the backbone model. However, Parallel Adapters do not account for the memory footprint of these substantial backbone model parameters, which can be substantial, especially for LLMs with parameter counts often exceeding hundreds of millions. Quantization reduces memory usage by mapping model weights from higher (e.g., 32-bit or 16-bit floating points) to lower (e.g., 8-bit or 4-bit integers) bit-width representations. This technique is widely adopted for model compression in LLMs serving and fine-tuning [34], [35], [36], [37]. For example, quantizing a 32-bit floating-point (FP32) tensor $\mathbf{X}^{32b}$ into an 8-bit integer (INT8) tensor $\mathbf{X}^{8b}$:

$$\begin{aligned}\mathbf{X}^{8b} &= \text{round}\left(\frac{127}{\text{absmax}(\mathbf{X}^{32b})}\mathbf{X}^{32b}\right)\\ &= \text{round}\left(c^{32b}\cdot\mathbf{X}^{32b}\right),\end{aligned} \tag{1}$$

where 127 is the maximum value of the INT8 data type and $\text{absmax}(\mathbf{X}^{32b})$ represents the maximum absolute value in $\mathbf{X}^{32b}$. c^{32b} is the quantization constant for 32-bit tensor $\mathbf{X}^{32b}$. Correspondingly, dequantization is given by:

$$\text{dequant}\left(c^{32b},\mathbf{X}^{8b}\right)=\frac{\mathbf{X}^{8b}}{c^{32b}}=\mathbf{X}^{32b}. \tag{2}$$

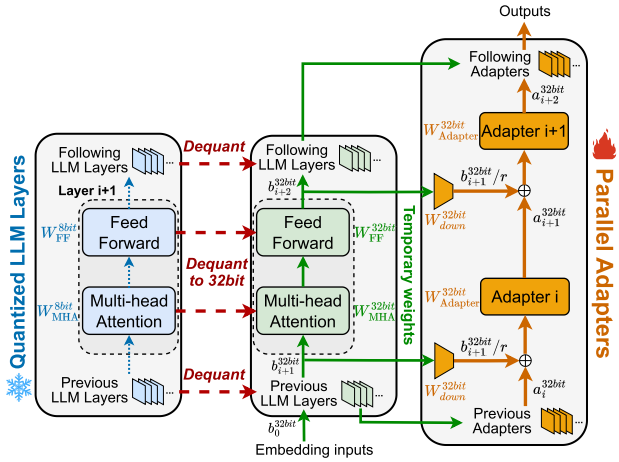


Fig. 8. The workflow of mixed-precision computation performed by our Parallel Adapters.

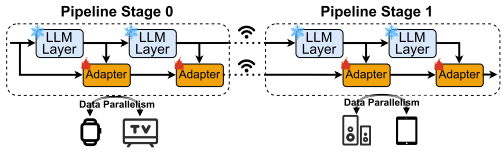
This quantization method can be intuitively understood as mapping the range of values representable by FP32 to the range of INT8. However, the issue with this approach is that the presence of large-magnitude values (i.e., outliers) in the input tensor leads to inefficient utilization of the available bit combinations, resulting in precision loss. To address the outlier issue, recent pioneering works [35] inspire the adoption of block-wise quantization algorithms to reduce quantization loss from large-magnitude outliers in the input tensor. We divide the input tensor $\mathbf{X}$ into contiguous blocks and quantize them independently with (1).

In our work, we introduce quantization techniques [35] to reduce the significant memory footprint of backbone LLM parameters and perform our fine-tuning algorithm in a mixed-precision computation paradigm, as illustrated in Fig. 8. We have one storage data type (typically a lower-bit data type, e.g., INT8) and a computation data type (typically a higher-bit data type, e.g., FP32). Initially, we quantize the backbone LLM to an 8-bit format for memory-efficient storage, while retaining the lightweight Parallel Adapters in 32-bit precision. During the forward and backward passes, we dequantize the backbone LLM from storage data type to the computation data type to perform FP32-based tensor operations. This mixed-precision design simultaneously offers substantial memory reduction (INT8) on the backbone LLM while provisioning high-precision (FP32) tensors for fine-tuning operations.

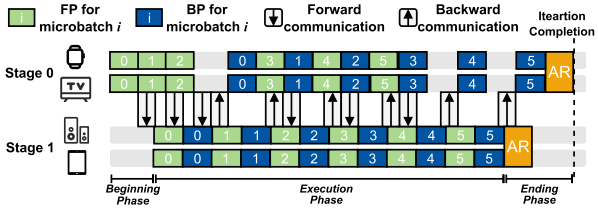
V. Collaborative Edge AI System for Efficient Personal LLMs Fine-Tuning

In PAC+, we leverage edge devices in physical proximity and associate them as a resource pool to boost in-situ fine-tuning. Specifically, the fine-tuning procedure comprises two phases: (1) In the initial epoch, the backbone of LLMs, enhanced with Parallel Adapters, undergoes fine-tuning across multiple edge devices through a blend of data and pipeline parallelism (Section V-A); (2) In subsequent epochs, the activation cache eliminates the necessity for forward propagation within the backbone, thereby enabling the exclusive fine-tuning of our Parallel Adapters utilizing data parallelism (Section V-B).

Fig. 9. Different parallelism plans of a computation graph, where different colors represent different devices and dashed boxes represent stages. For instance, data parallelism involves ingesting data batches into four devices.



(a) The LLM transformer layers is partitioned into two stages, where both Stage 0 and 1 are replicated on a device group with two devices for intra-stage data parallelism.



(b) Fine-tuning pipeline of 6 micro-batches. The numbers in the cells represent micro-batch ids. AllReduce (AR) is performed in both Stage 0 and 1 for model synchronization.

Fig. 10. An instance of hybrid parallelism in PAC+.

TABLE II
TABLE OF NOTATIONS FOR PARALLELISM PLANNING

Notation	Definition
L	Total number of layers in the DNN model.
$\mathcal{D}$	An ordered set of all devices involved in training.
$\mathcal{D}_n$	The subset of first n devices in $\mathcal{D}$.
M	Number of micro-batches in a mini-batch.
B	Size of each micro-batch.
u_d	Memory budget available on device d.
m_d	The sum of the memory footprint of LLM parameters, parameter gradients, and activations of device d.
G_n	A device group involving n edge devices.
$t_f^{d,l}(\beta)$ / $t_b^{d,l}(\beta)$	Time to perform FP / BP for layer l on device d with a batch size of β.
$c_f^s(i)$ / $c_b^s(i)$	FP / BP communication time between stage i and $i+1$.
W_s	Balanced paratition configurations for pipeline with s stages.
$e_f^s(i)$ / $e_b^s(i)$	Time taken for FP / BP in step i with W_s.
L_b^s / L_e^s / L_n^s	Execution time for the Beginning / Execution / Ending phases in W_s.
$\mathrm{AR}^s(i)$	AllReduce time of stage i in W_s.

A. Resource-Efficient Collaborative Orchestration for LLMs Fine-Tuning

Observation of Data and Pipeline Parallelism at the Edge: When collaborating on LLM fine-tuning among edge devices, the principle question is which type of parallelism should be used. We analyze different parallelism plans in Fig. 9. The most common way to train models in parallel is *data parallelism* (DP) [38]. However, DP necessitates that each device maintains a replica of the entire model, a requirement difficult to meet for LLMs with extensive parameter sizes, often surpassing the capacity of a single device. *Pipeline parallelism* (PP) [39] is further proposed to address this problem. In PP, the model is partitioned into multiple consecutive stages and each stage is mapped to a separate device. Consequently, PP enables the training of increasingly large models by deploying more devices. Nonetheless, PP encounters scalability constraints as the addition of edge devices results in more stages. This not only results in a significant presence of pipeline bubbles but also amplifies the impact of inter-stage communication latency, thereby hindering efficiency. The above observation motivates us to employ a *hybrid parallelism* (HP) architecture that incorporates the best of both DP and PP, so as to achieve superior performance and scalability in resource-constrained edge environments.

Hybrid Parallelism Architecture in PAC+: As illustrated in Fig. 10(a), PAC+ first divides an LLM into multiple *stages* where each contains a *stage model* composed of a set of consecutive transformer layer. Edge devices are allocated into several device groups, each comprising one or more devices. PAC+ maps each stage to a group, with the stage model replicated across all devices within that group. Throughout the fine-tuning process, a mini-batch is divided into several *micro-batches* for concurrent processing to enhance parallelism. If a device cluster hosts multiple devices, micro-batches are further subdivided. Each device is responsible for executing the forward (FP) and backward passes (BP) for its assigned stage model and aggregates gradients across all micro-batches for every mini-batch. Upon completing a mini-batch, gradient synchronization within each device group is achieved through *AllReduce*. Since the majority of parameters in LLMs are frozen, AllReduce synchronizes only the lightweight parallel adapters, ensuring a swift process. We adopt the *one-forward-one-backward* (1F1B) micro-batch scheduling [40] which schedules the BP early to release the activation memory produced by FP for reuse. Fig. 10(b) depicts a well-structured hybrid parallelism, encompassing FP, BP, and inter-stage communication.

Profiling: To enable parallelism planning, PAC+ profiler first fine-tunes the target LLM using calibration datasets to record the runtime profile required for planning. We define $t_f^{d,l}(\beta)$ and $t_b^{d,l}(\beta)$ as the FP and BP execution times for layer l on device d with batch size of β, respectively. u_d denotes the memory budget of device d. The size of output activations, input gradients, and weight parameters in bytes will also be collected to calculate memory footprint. Table II summarizes the notation for parallelism planning.

Planning Algorithm for Hybrid Parallelism: The global throughput of a pipeline is determined by the execution time of the slowest stage. Consequently, our algorithm endeavors to partition the model into balanced stages. We consider an LLM consisting of L layers and denote $\mathcal{D}$ as an ordered set of all devices involved in planning, while $\mathcal{D}_n = \{d_0, \ldots d_{n-1}\}$ as the subset of first n devices in $\mathcal{D}$. $W(x \to y, \mathcal{D}_n, s)$ denote the time

taken by the slowest stage in the optimally balanced sub-pipeline between layer x to y with $\mathcal{D}_n$, when divided into s stages. To solve this partitioning problem, we break the pipeline into sub-pipelines and leverage the idea of dynamic programming. The formula of the dynamic programming algorithm can be written as:

$$W(0 \rightarrow y, \mathcal{D}_n, s) = \min_{0 \leqslant q < y} \min_{1 \leqslant m < n} \max\{W(0 \rightarrow q, \mathcal{D}_{n-m}, s-1), T(q+1 \rightarrow y, \{d_{n-m} \ldots, d_{n-1}\})\}, \quad (3)$$

where the first term inside the max is the time of the optimally balanced sub-pipeline between layers 0 to q with $n-m$ devices. The second term represents the time required by the single stage comprising layers $q+1$ to y across m devices. The notation $T(x \rightarrow y, \mathcal{G})$ denotes the time required for a single stage to execute FP and BP in a data-parallel manner across the device group $\mathcal{G}$. In allocating samples of a mini-batch to devices within $\mathcal{G}$, the goal is to minimize the inference latency determined by the slowest device while ensuring that individual devices do not encounter out-of-memory (OOM) exceptions. We define $H_{x \rightarrow y}(b, \mathcal{G}_n)$ as the optimal time required by the slowest device in the device group $\mathcal{G}_n = \{d_0, d_1, \ldots, d_{n-1}\}$ to execute the stage model (layers from x to y) when distributing b samples across the group. To solve the aforementioned sample dispatching problem, we also employ a dynamic programming algorithm to search for the optimal sample dispatching strategy. The transition equation for the dynamic programming algorithm can be written as:

$$H_{x \rightarrow y}(b, \mathcal{G}_n) = \min_{0 \leq i \leq b} \max \begin{cases} H_{x \rightarrow y}(b-i, \mathcal{G}_n \setminus \{d_{n-1}\}), \\ \sum_{l=x}^{y} \left[t_f^{d_{n-1}, l}(i) + t_b^{d_{n-1}, l}(i) \right] \end{cases}. \quad (4)$$

By solving (4) with dynamic programming, we ultimately obtain $H_{x \rightarrow y}(B, \mathcal{G})$, which represents the desired value of $T(x \rightarrow y, \mathcal{G})$, where B denotes the micro-batch size. We define the peak memory footprint of device d, denoted as m_d, as the sum of memory used by the LLM parameters, parameter gradients, and intermediate activations. Increasing the number of samples allocated to a device enlarges the intermediate activations, potentially leading to OOM exceptions. To exclude OOM cases during the allocation algorithm, the FP and BP time for device d is set to positive infinity. If the algorithm ultimately yields a result of positive infinity for $H_{x \rightarrow y}(B, \mathcal{G})$, it indicates that the collective memory of this group of edge devices cannot accommodate the target stage model. During the dynamic programming, we will record pipeline planning configurations, including *LLM segmentation points, micro-batch splitting results*, and *device groupings strategies*.

Upon the completion of dynamic programming process, we obtain a set of balanced partition configurations for various number of pipeline stages: $\{W_s | \text{ config. of } W(0 \rightarrow L, \mathcal{D}, s), s \in \{1, 2, \ldots, |\mathcal{D}|\}\}$. The next step is to determine the optimal number of stages. Using recorded configurations, we can profile FP and BP execution time of stage i in W_s as $e_f^s(i)$ and $e_b^s(i)$. Similarly, forward and backward communication time between stages i and $i+1$ are represented as $c_f^s(i)$ and $c_b^s(i)$. $\text{AR}^s(i)$ represents the AllReduce time of stage i in W_s. M is the number of micro-batch. As shown in Fig. 10(b), we can divide per mini-batch training of W_s into three phases: *beginning phase*, *execution phase*, and *ending phase* with corresponding latency denoted as L_b^s, L_e^s, L_n^s:

$$L_b^s = \sum_{i=1}^{s-1} \left[e_f^s(i) + c_f^s(i) \right], \quad L_e^s = M \cdot \left(e_f^s(s) + e_b^s(s) \right), \quad (5)$$

$$L_n^s = \max_{i \in \{1 \ldots, s\}} (\text{AR}^s(i) + \sum_{j=i}^{s-1} (e_b^s(j) + c_b^s(j)), \quad (6)$$

$$\sigma = \text{argmin}_s (L_b^s + L_e^s + L_n^s). \quad (7)$$

Our algorithm aims to minimize this total latency by optimally determining the number of stages s. We summarize our dynamic programming planning in Algorithm 1. We remark that our parallelism planning is an offline procedure that runs once before deployment. The time complexity for our dynamic programming algorithm exhibits an upper bound of $O(BL^2|\mathcal{D}|^3)$. In our experiment, the whole planning time is within several minutes on an edge device.

Algorithm 1: Planning for Hybrid Parallelism.

Input: The target LLM model with L layers. An ordered set of all devices $\mathcal{D}$.
Output: The optimal planning configurations W_σ.

```
for s ← 1 to min(L, |D|) do
    for y ← 1 to L do
        for n ← 1 to |D| do
            for q ← 0 to y do
                for m ← 0 to n do
                    Get T(q+1 → y, {d_{n-m}, ..., d_{n-1}})
                      by solving Eq. (4);
                end
            end
            Update W(0 → y, D_n, s) with Eq. (3);
        end
    end
    Calculate L_b^s, L_e^s, L_n^s using Eq. (5) and (6);
    Update the optimal stage number σ with Eq. (7);
end
Return W_σ;
```

B. Cache-Enabled Collaborative Edge Fine-Tuning of Parallel Adapters

Data-Parallel Fine-Tuning for Parallel Adapters: The computationally lightweight nature of the Parallel Adapters precludes the use of pipeline parallelism to fine-tuning with activation cache, as it would result in unoverlapable inter-stage communication latency. Therefore, we employ data parallelism to exclusively fine-tune our Parallel Adapters. Specifically, after the first training epoch, the activation cache for all samples is already collected. We then perform collective communication to redistribute the Parallel Adapters parameters and locally cached activations across all devices, ensuring each device receives the complete set of adapter parameters and corresponding activations. The devices then utilize this shared information to

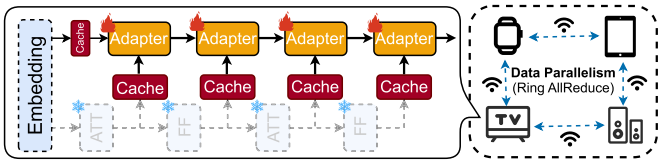


Fig. 11. An instance of fine-tuning with activation cache.

fine-tune the parallel adapters in a data-parallel manner. In our experiments, fine-tuning the BART-Large model on the MRPC dataset for three epochs, the redistribution of parameters and activations only contributed to approximately 8% of the total training time. Notably, the overhead of this process can be further amortized over additional training epochs. An instance of personal LLMs fine-tuning with activation cache is depicted in Fig. 11.

Storage Cost Analysis: Employing activation caching can reduce the computational requirements of forward propagation; however, it incurs additional storage overhead for activations. Specifically, the storage overhead is $s \times h \times l$ per sequence, where s denotes the sequence length, h represents the transformer's internal feature dimension, and l corresponds to the number of transformer layers. For T5-Base model, the activation caching requires less than 1 GB to store the activations for 500 training samples with sequence length of 30. Such cost is no more than 1% of the storage of a modern mobile device, e.g., hundreds of GB. During fine-tuning, the activation cache is reloaded from disk per micro-batch, a process that takes no more than tens of milliseconds on embedded flash storage. The cache will be cleared once the fine-tuning process finishes.

VI. Evaluation

A. Implementation and Setups

Implementation of PAC+: We have fully implemented the prototype framework of PAC+ and baselines with ∼2,000 LoC in Python atop Pytorch [41]. PAC+'s idea is also portable and can work well with other lightweight ML frameworks such as MNN [42] and TF-Lite [43]. Our Parallel Adapters is a lightweight version of the backbone model. The size of Parallel Adapters is determined by the reduction factor r. All weights and hidden state dimensions of the Parallel Adapters are $\frac{1}{r}$ times the corresponding weights and hidden states of the backbone model. In our experiments, the reduction factor r is set to 8. The weights of the Parallel Adapters are initialized based on structural pruning, using the weights of the backbone model. We insert Parallel Adapters at the end of each transformer layer.

Models and Datasets: We evaluate PAC+ with three typical transformer based LLM with parameters ranging from 0.25B to 0.74B, as detailed in Table III, which are widely considered for IPA and edge deployments [4], [44]. All experiments were performed under conditions using FP32 precision to ensure fine-tuning performance. We employ two variants of the T5 model [2], specifically T5-Base and T5-Large with differing parameter sizes. We also compare PAC+ with baseline methods with BART-Large [3] as the backbone for our parallel adapters. We evaluate our fine-tuned LLMs with four tasks from GLUE benchmark. The four tasks evaluate models on multiple diverse tasks over sentiment analysis (SST2), similarity and paraphrase (MRPC, STS-B) and natural language inference (QNLI).

TABLE III
LLM Model Specifications Used for Experiments. "en-De" Indicates Encoder-Decoder LLM Structure

Model	Structure	Layers	Heads	Hidden Size	Param. Count
T5-Base [2]	en-de	12	12	768	0.25B
BART-Large [3]	en-de	12	16	1024	0.41B
T5-Large [2]	en-de	24	16	1024	0.74B

TABLE IV
Specifications of Edge Devices in Evaluation

Edge Device	GPU Processor	Memory Budget	GPU Frequency	Power Mode
Jetson Nano [12]	128-core Maxwell	4GB	921MHz	10W (**H**)
			640MHz	5W (**L**)
Jetson TX2 [45]	256-core Pascal	8GB	1.3GHz	15W (**H**)
			850MHz	7.5W (**L**)

Edge Environment Setup: We conduct experiments using two off-the-shelf edge devices listed in Table IV, simulating four heterogeneous platforms by adjusting their power modes. We evaluate PAC+'s performance in two realistic edge environments, incorporating both homogeneous and heterogeneous configurations. *Homogeneous Environment A* (**Env. A**) consists of 4×Nano-H, while *Heterogeneous Environment B* (**Env. B**) comprises 1×Nano-H, 1×Nano-L, 1×TX-H and 1×TX-L. We simulate common network conditions in edge environments (e.g., smart homes) by setting the intra-cluster network bandwidth to 1000 Mbps.

Baseline Methods: To evaluate the high performance and robustness of our PAC+ system, we comprehensively compare it with a wide range of baseline methods, including those we build ourselves and existing end-to-end collaborative edge training systems.

To thoroughly evaluate the advantages of our PAC+ in both algorithmic and system design, we select **3** classic collaborative edge computing paradigms and **3** widely adopted LLM fine-tuning methods, combining them in pairs to form **9** distinct baselines. The three collaborative edge computing paradigms are as follows:

1) *Standalone* means fine-tuning LLMs on a single edge device. We compare with it to analyze the scalability performance of PAC+.
2) *Data Parallelism (DP)* [46] is a classic cluster computing paradigm that distributes batch data across devices for simultaneous processing. EDDL [38] is a representative work that applies DP to edge computing, and we reference its architecture to implement our baseline.
3) *Pipeline Parallelism (PP)* [47] enables collaborative training across an edge device cluster by segmenting LLMs into sequential stages for pipeline processing. We implement our baseline framework based on the representative work Eco-FL [39].

TABLE V
TRAINING DURATIONS (IN HOURS) FOR DIFFERENT METHODS: 3 EPOCHS FOR MRPC AND STS-B, AND 1 EPOCH FOR SST-2 AND QNLI

Fine-tuning Techniques	Baseline Methods	T5-Base				BART-Large				T5-Large			
		MRPC	STS-B	SST-2	QNLI	MRPC	STS-B	SST-2	QNLI	MRPC	STS-B	SST-2	QNLI
Full Model	Standalone	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
	PP (Eco-FL)	0.45	0.71	2.74	4.32	2.41	3.78	14.56	22.98	OOM	OOM	OOM	OOM
	DP (EDDL)	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
Adapters	Standalone	1.21	1.9	7.29	11.51	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
	PP (Eco-FL)	0.39	0.61	2.35	3.71	0.54	0.85	3.27	5.16	2.75	4.31	16.59	26.19
	DP (EDDL)	0.34	0.53	2.06	3.25	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
LoRA	Standalone	1.21	1.89	7.28	11.49	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
	PP (Eco-FL)	0.41	0.64	2.45	3.87	0.55	0.87	3.33	5.26	2.73	4.28	16.48	26.02
	DP (EDDL)	0.31	0.48	1.86	2.94	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
Parallel Adapters	**PAC+ (Ours)**	**0.14**	**0.22**	**1.34**	**2.12**	**0.29**	**0.45**	**2.69**	**4.25**	**0.69**	**1.09**	**8.88**	**14.02**

The three LLM fine-tuning methods are as follows:

1) *Full model fine-tuning* means that all the LLM parameters are updated for a downstream task.
2) *LoRA* [11] is a widely-used PEFT technique that decomposes the parameter update for a weight matrix into two trainable low-rank matrices. We apply LoRA to W_q and W_v following the settings from [11].
3) *Adapters* [10] is another widely-used PEFT technique that injects small trainable modules at the end of each transformer layer following the settings from [10].

In addition to our self-built baselines, we also compare PAC+ with two existing end-to-end edge training systems optimized for resource heterogeneity in edge environments:

1) *Asteroid* [48] is a collaborative DNN training framework designed for heterogeneous edge device clusters. It employs a hybrid data and pipeline parallelism architecture, similar to PAC+, but is designed for full-parameter fine-tuning of both vision and language models.
2) *HetPipe* [49] is a distributed training framework designed for heterogeneous consumer-grade GPU clusters. It facilitates asynchronous parallel training by treating sub-groups of GPUs as virtual workers, employing intra-worker PP and inter-worker communication.

B. Analysis of Algorithmic and System Design Benefits

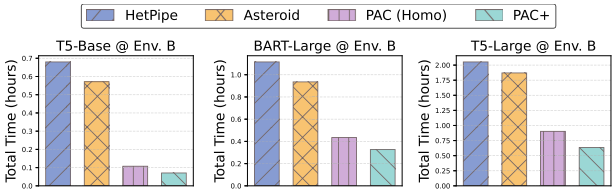
1) Comparison of End-to-End Elapsed Fine-Tuning Time Between PAC+ and Baselines: To thoroughly evaluate the strengths of PAC+ in both algorithmic and system design, we select three classic collaborative edge computing paradigms and equip them with three prevalent LLM fine-tuning techniques as baselines in the edge environment Env. A. For the Eco-FL, we divided one mini-batch into 4 micro-batches to enhance pipeline concurrency, whereas the Standalone and EDDL methods were fine-tuned strictly at the mini-batch granularity. Table V summarizes the end-to-end fine-tuning elapsed time on the GLUE datasets for both PAC+ and the baseline methods. For fine-tuning smaller datasets in GLUE, MRPC and STS-B, we conduct training over three epochs, with the latter two epochs benefiting from the PAC+ activation cache. In contrast, for larger datasets in GLUE, STS-2 and QNLI, a single epoch of fine-tuning is sufficient to achieve satisfactory performance.

As shown in Table V, PAC+ significantly speeds up the training process while preserving convergence performance. PAC+ achieves an acceleration ranging from $1.21\times$ to $5.44\times$ on SST-2 and QNLI without utilizing activation cache. In comparison to Standalone and DP, these two baselines often encounter Out-of-Memory (OOM) issues, even when integrating PEFT techniques such as LoRA and Adapters. This issue stems from the training requirement for each edge device to host the entire target model. Particularly for T5-Large, a single Jetson Nano is inadequate to accommodate LLM parameters, not to mention the intermediate activations. Compared to PP, PAC+ achieves an acceleration ranging from $1.21\times$ to $5.41\times$ on SST-2 and QNLI, without utilizing activation cache. Parallel Adapters not only alleviate the memory footprint of LLM parameters but also intermediate activations. Pipeline parallel strategies allow each edge device to host only a portion of the model parameters. However, these devices still bear a substantial memory footprint from intermediate activations, even when employing PEFT technologies such as LoRA and Adapters. Therefore, the PP approach necessitates the use of smaller micro-batch sizes or a reduction in the number of micro-batches simultaneously input into the pipeline. This results in decreased concurrency in pipeline parallelism and lowers the training throughput. Moreover, our hybrid parallelism merges the benefits of both data and pipeline parallelism, providing an expanded search space for parallel architectures to accommodate complex edge environments. Our method enables the identification of the most efficient parallel configuration with maximum throughput within the constraints of available resources. With the integration of our activation cache mechanism, PAC+ achieves speedups of up to $8.64\times$ on the MRPC and STS-B datasets. As discussed in Section IV-A, our Parallel Adapters constitute a lightweight, independent network. We can skip both the forward and backward passes through the LLM backbone, since the required activations have already been calculated and stored. Consequently, training overhead can be markedly reduced in the second and third fine-tuning epochs.

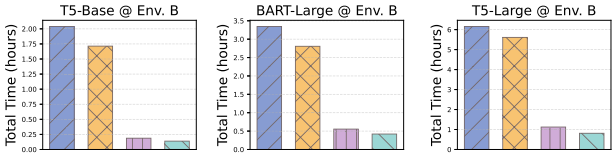
2) Analysis of Fine-Tuned LLM Performance with PAC+: Table V demonstrates that our PAC+ achieves higher fine-tuning throughput compared to all baselines. However, it is still necessary to further validate that our PAC+ design does not compromise model accuracy in exchange for lower elapsed time. Table VI reports the model performance of LLMs fine-tuned using three classic LLM fine-tuning algorithms and our Parallel Adapters approach. All accuracy metrics were evaluated on a server environment with sufficient computational and memory resources to ensure consistent performance reporting, irrespective of potential OOM constraints. Fine-tuning involves 3 epochs for the smaller MRPC and STS-B datasets, and 1 epoch for

TABLE VI
COMPARISON OF FINAL PERFORMANCE BETWEEN DIFFERENT FINE-TUNING TECHNIQUES ACROSS FOUR DATASETS. WE REPORT THE AVERAGE OF F1 SCORE AND ACCURACY FOR MRPC. WE USE PEARSON-SPEARMAN CORRELATION AS THE METRIC FOR STS-B. FOR SST-2 AND QNLI, WE REPORT ACCURACY. THE MEAN VALUE IS THE AVERAGE PERFORMANCE OF FULL MODEL, ADAPTERS AND LORA.

Fine-tuning Techniques	T5-Base				BART-Large				T5-Large			
	MRPC	STS-B	SST-2	QNLI	MRPC	STS-B	SST-2	QNLI	MRPC	STS-B	SST-2	QNLI
Full Model	89.71	90.94	94.03	93.08	88.16	91.10	95.64	94.40	92.78	91.08	95.30	93.30
Adapters	88.73	90.51	93.58	93.04	86.63	90.24	94.93	93.27	91.86	90.58	96.10	94.07
LoRA	86.27	90.73	93.69	93.30	87.46	90.36	95.23	94.48	90.27	92.08	95.53	94.18
Mean Value	88.24	90.73	93.77	93.14	87.42	90.57	95.27	94.05	91.64	91.25	95.64	93.85
Parallel Adapters (Ours)	88.24	90.43	93.46	93.25	87.71	90.54	95.25	93.68	91.70	91.57	95.76	93.70
Difference from Mean	+0.00	-0.30	-0.31	+0.11	+0.29	-0.03	-0.02	-0.37	+0.06	+0.32	+0.12	-0.15



(a) Total time required to fine-tune 1 epoch on the MRPC dataset in Env. B.



(b) Total time required to fine-tune 3 epoch on the MRPC dataset in Env. B. PAC+ further benefits from the activation cache design.

Fig. 12. Comparison of total fine-tuning time across different baselines. PAC+ (Homo) refers to a version of the PAC+ algorithm that does not consider device resource heterogeneity.

the larger SST-2 and QNLI datasets. We can observe from Table VI that PAC+ achieves comparable or superior performance to full model fine-tuning and baseline PEFT techniques across various models and datasets. The largest discrepancy in mean performance metrics between PAC+ and these baseline methods is only -0.37, a negligible difference. Notably, PAC+ frequently outperforms these methods and achieves the highest performance on the SST-2 dataset with the T5-Large model.

C. Comparison With Existing Collaborative Training Systems Under Heterogeneous Edge Environments

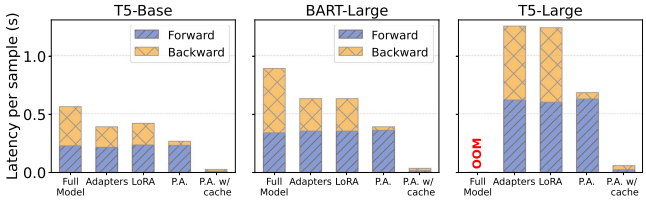
To evaluate the superiority and robustness of our proposed PAC+ system compared to existing collaborative edge training systems, we select two state-of-the-art training frameworks designed for heterogeneous device clusters, Asteroid [48] and HetPipe [49], as baselines for comparison with PAC+. The experiments are conducted in the heterogeneous edge environment Env. B. In addition to comparing PAC+ with the two baselines, we also included its older PAC version [50], which does not account for device heterogeneity, as part of our ablation study. This comparison aims to demonstrate the effectiveness of our heterogeneity-aware algorithm design. We evaluated all three models on the MRPC dataset, conducting fine-tuning for one epoch (Fig. 12(a)) and three epochs (Fig. 12(b)), respectively.

As shown in Fig. 12(a), under the one-epoch fine-tuning setting, the PAC+ architecture achieves a $3.2\times$–$9.7\times$ improvement over HetPipe and a $2.9\times$–$8.1\times$ improvement over Asteroid, attributed to the co-design of the PAC+ algorithm and system. Specifically, HetPipe adopts a hybrid parallelism architecture with inter-group DP and intra-group PP, combining asynchronous updates to address device heterogeneity. However, this parallelism architecture has been shown in existing studies [48] to require higher inter-device communication, making it unsuitable for bandwidth-constrained edge clusters. Asteroid adopts the same inter-group PP and intra-group DP architecture as PAC+, along with fine-grained resource partitioning and orchestration to address heterogeneity. However, both Asteroid and HetPipe lacks optimizations for LLM fine-tuning algorithms and relies solely on full-parameter fine-tuning, resulting in higher computational resource requirements and longer training times. Furthermore, the activation cache design in PAC+ enables even greater performance improvements during three-epoch fine-tuning compared to the baselines, as shown in Fig. 12(b). Specifically, PAC+ achieves a $7.6\times$–$14.7\times$ improvement over HetPipe and a $6.9\times$–$12.3\times$ improvement over Asteroid. Our designed PAC+ algorithm can achieve up to a 35% latency reduction compared to algorithms that do not account for device heterogeneity.

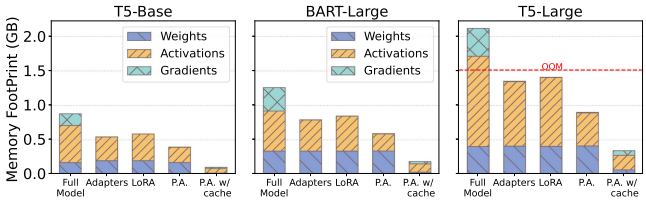
D. Further Breakdown of the Benefits of Parallel Adapters

We further conduct experiments to break down the training latency and memory footprint, assessing the time and memory efficiency of Parallel Adapters at the edge. We conduct experiments on an edge cluster consisting of 8 Jetson Nano-H. All baselines are implemented using the same hybrid parallel architecture and equipped with different fine-tuning algorithms, with 1F1B micro-batch scheduling disabled. "Activations" contain the intermediate results and optimizer states. Fig. 13 illustrates that Parallel Adapters outperform other fine-tuning techniques regarding both time and memory efficiency.

1) Parallel Adapters markedly reduce per-sample training time: Fig. 13(a) presents the average sample training time across different fine-tuning techniques. Without activation cache, Parallel Adapters can reduce the average sample training time by 31.94% to 56.24% compared to baseline methods, primarily owing to a substantial decrease in backward propagation overhead. Both Adapters and LoRA incorporate trainable structures into the backbone model, thus necessitating backpropagation across the entire backbone model for gradient computation of these parameters. Consequently, regarding backward time, Adapters and

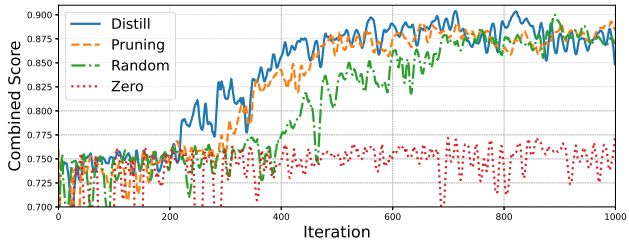


(a) The comparison of average sample training time of different fine-tuning techniques.

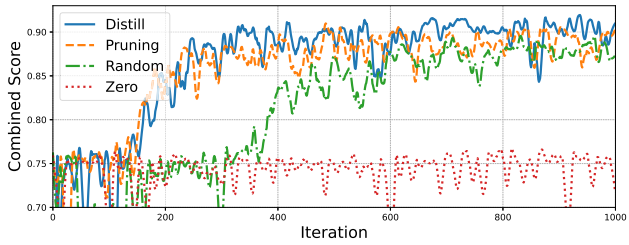


(b) Maximum total memory consumption per device across the edge cluster.

Fig. 13. Comparison of different fine-tuning techniques. P.A. indicates our Parallel Adapters technique. Mini-batch size 16; sequence length: 128.



(a) Fine-tuning performance of different initialization strategies on BART-Large.



(b) Fine-tuning performance of different initialization strategies on T5-Large.

Fig. 14. The ablation study on different initialization strategies for Parallel Adapters using MRPC dataset.

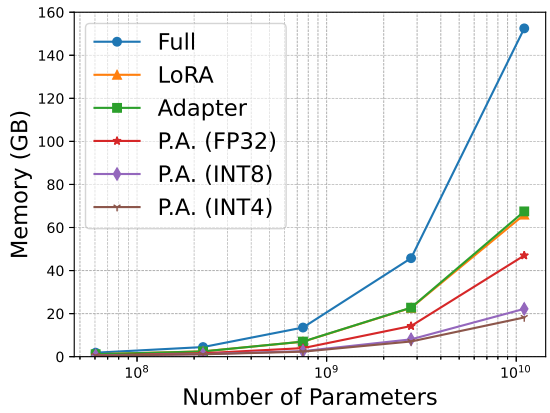


Fig. 15. Memory footprint analysis of baselines and Parallel Adapters with various quantization precision.

LoRA can only achieve approximately a 49% reduction compared to full fine-tuning. In contrast, backpropagation through the backbone model is unnecessary with Parallel Adapters, leading to a more substantial reduction in backward time, nearly 92% compared to full fine-tuning. Moreover, Parallel Adapters With activation cache mechanism can further decrease the average sample training time up to 96.39%. These results demonstrate the substantial reduction in training time achieved by Parallel Adapters.

2) Parallel Adapters yield a substantial reduction in memory usage: Fig. 13(b) depicts the breakdown of the memory footprint for different fine-tuning techniques. We report the peak memory consumption per device across edge clusters. Without activation cache, Parallel Adapters can reduce memory usage by 25.27% to 60.49%. Adapters and LoRA demonstrate parameter efficiency but do not exhibit significant memory efficiency. While these techniques notably decrease the memory footprint of gradients by reducing the number of trainable parameters, the memory usage associated with activations remains considerable. However, intermediate activations often become the primary memory bottleneck during training, especially with larger batch sizes. For PEFT techniques such as Adapters and LoRA, activation memory can be reduced by up to 28.15% compared to full fine-tuning across the three models. In contrast, Parallel Adapters achieve a more significant reduction, reaching up to 58.87%. With activation cache, Parallel Adapters can decrease the peak memory footprint from 74.57% to 88.16% compared to baselines. This is because it's sufficient to store only the lightweight Parallel Adapters, eliminating the need to host the entire LLM backbone in memory.

E. Analyzing the Impact of Weight Initialization on Parallel Adapters

To investigate the impact of weight initialization on the fine-tuning performance of Parallel Adapters, we compare four initialization methods: knowledge distillation, structural pruning, random Gaussian initialization, and zero initialization. The fine-tuning performance of the BART-Large and T5-Large models on the MRPC dataset is illustrated in Fig. 14. We implemented our distillation method using Google Research's open-source distillation toolkit [33] and applied structural pruning with Torch-Pruning, a popular open-source toolkit for structural pruning built on PyTorch [27], [28]. Random Gaussian initialization is widely used in popular PEFT techniques, such as LoRA [11]. We can observe from Fig. 14 that using distillation or pruning-based initialization for Parallel Adapters allows the model to reach the target accuracy more quickly during fine-tuning compared to random Gaussian or zero initialization. Specifically, using distillation and pruning-based initialization, the BART-Large model achieves a combined score of 0.875 after approximately 410 and 457 iterations, respectively, while random Gaussian initialization requires 605 iterations. Reducing the required fine-tuning iterations can significantly lower the time and computational resources required for fine-tuning on resource-constrained edge clusters.

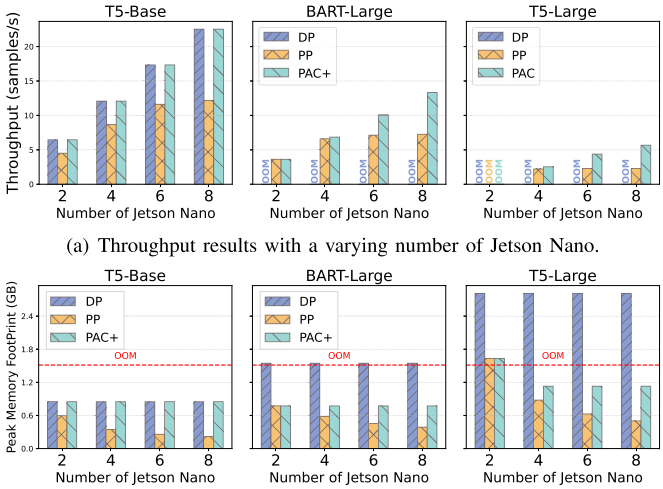
TABLE VII
COMPARISON OF FINAL PERFORMANCE BETWEEN DIFFERENT QUANTIZATION PRECISION ACROSS FOUR DATASETS. WE REPORT THE AVERAGE OF F1 SCORE AND ACCURACY FOR MRPC. WE USE PEARSON-SPEARMAN CORRELATION AS THE METRIC FOR STS-B. FOR SST-2 AND QNLI, WE REPORT ACCURACY. THE MEAN VALUE IS THE AVERAGE PERFORMANCE OF FULL MODEL, ADAPTERS AND LORA.

Backbone LLM	T5-Base				BART-Large				T5-Large			
Precision	MRPC	STS-B	SST-2	QNLI	MRPC	STS-B	SST-2	QNLI	MRPC	STS-B	SST-2	QNLI
FP32	88.24	90.43	93.46	93.25	87.71	90.54	95.25	93.68	91.70	91.57	95.76	93.70
FP16	88.42	89.33	93.35	92.58	87.76	90.34	94.83	92.62	91.24	91.17	95.43	92.88
INT8	87.45	89.12	92.78	91.97	87.66	90.03	94.64	92.10	91.03	90.17	94.73	92.23
INT4	87.24	88.25	92.66	91.10	87.52	89.87	94.49	91.27	89.69	90.08	94.72	91.71

F. Analysis of Fine-Tuning With Quantized LLMs

1) Memory Footprint Analysis: We compare the memory footprint during the fine-tuning process of LLMs with different number of parameters, using full-parameter fine-tuning, LoRA, Adapter, and Parallel Adapters algorithms with various quantization precisions. We obtained a series of T5 models with varying parameter sizes by adjusting parameters such as hidden size, number of layers, and number of attention heads. Experimental results in Fig. 15 show that Parallel Adapters with lower quantization precisions, such as INT4 and INT8, can significantly reduce the memory footprint during fine-tuning. Specifically, compared to full-parameter fine-tuning, the Parallel Adapter algorithm with INT4 precision reduces memory footprint by up to 88%, and compared to existing PEFT algorithms, it can reduce memory footprint by up to 73%.

2) Analysis of Fine-Tuning Performance with Quantized LLMs: Building on Table VI, we further evaluate and report the final fine-tuned performance of Parallel Adapters with different quantization precisions across four datasets. The results are presented in Table VII, with the FP32 precision results for Parallel Adapter directly taken from Table VI. Experimental results show that fine-tuning with low-precision backbone LLM does not significantly degrade model performance and they still maintain competitive accuracy. Our previous experiments have demonstrated that quantization can significantly reduce memory footprint during fine-tuning, making it suitable for resource-constrained edge environments.



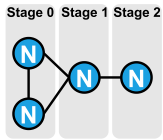
(a) Throughput results with a varying number of Jetson Nano.

(b) Peak memory consumption of LLM weights per device.

Fig. 16. Comparison of different collaborative edge computing architecture.

Model	Number of Jeton Nano			
	2	4	6	8
T5-Base	[2]	[4]	[6]	[8]
BART-Large	[1-1]	[2-2]	[3-3]	[4-4]
T5-Large	[1-1]	[2-1-1]	[2-2-2]	[3-3-2]

(a) PAC's device grouping results.



(b) An instance of [2-1-1].

Fig. 17. Device grouping results of PAC+'s hybrid parallelism for experiments in Fig. 16. "N" indicates Jetson Nano.

G. Analysis the Benefits and Scalability of the Hybrid Parallelism Architecture.

Compared to standalone data parallelism or pipeline parallelism, the hybrid parallelism architecture adopted by PAC+ integrates the strengths of both DP and PP, enabling superior performance and scalability in resource-constrained edge environments. To explore the scalability advantages of PAC+'s hybrid parallelism over DP and PP, we compared the throughput of these methods when fine-tuning collaboratively across 2 to 8 Jetson Nano-H devices. The batch size was consistent with the number of devices, and the sequence length of each sample was fixed at 128. We implement DP and PP using the Parallel Adapters technique to ensure a fair comparison. Note that none of the three methods utilizes activation cache.

Fig. 16(b) illustrates the maximum per device memory footprint of model weights across edge cluster. For DP, each device must host a complete LLM, preventing the reduction of the parameters' memory footprint through scaling up the number of devices. Therefore, as shown in Fig. 16(a), the DP method exhibits OOM errors with both the BART-Large and T5-Large models. Conversely, PAC+ and PP partition the model into multiple stages with each handled by different devices. This approach allows for scaling the number of devices to reduce the peak memory footprint. PAC+'s hybrid parallelism offers a broader search space for parallel strategies compared to standalone PP. Our planning algorithm for PAC+ is capable of identifying more efficient hybrid parallel configurations within memory constraints, enhancing resource utilization. Although PAC+ may incur higher memory overhead in some instances, it achieves greater system throughput. Specifically, when compared to PP, PAC+ exhibits an increase in throughput from 39.50% to 84.79%.

To more clearly illustrate the parallel strategies adopted by PAC+, we present the device grouping configurations for PAC+ across various LLMs and numbers of devices in Fig. 17. On the left, a table displays all the grouping results across three models.

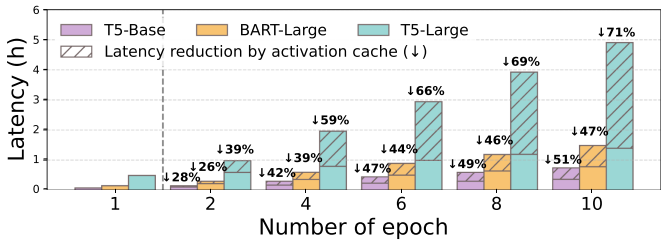


Fig. 18. Fine-tuning time with PAC+. Time without activation cache is represented by bars. The corresponding reduction in time achieved utilizing activation cache is represented by shaded areas. Dataset: MRPC.

On the right, an instance is shown where a model is divided into three stages, with two devices handling the first stage to perform data parallelism. Specifically, when fine-tuning BART-Large with eight devices, DP encounters OOM issues because a single Jetson Nano cannot accommodate a complete BART-Large. PP addresses this problem by dividing the model into eight stages and employing straight pipeline parallelism for training. On the contrary, our PAC+ approach divides BART-Large into two stage models, with each stage replicated across four devices. This configuration significantly reduces the number of stages in the pipeline, thereby minimizing inter-stage data dependencies and communication latency, which in turn enhances the pipeline's concurrent efficiency. These results demonstrate that our hybrid parallel approach offers a larger search space for parallel configurations, providing enhanced scalability and robustness across varying numbers of devices and workloads.

H. Comparison of PAC+ With and Without Activation Cache.

We further investigated how our activation cache mechanism benefits the required fine-tuning latency. By leveraging activation cache, the fine-tuning latency per epoch can decrease up to 79.51%. Fig. 18 shows the fine-tuning latency reduction as the number of epochs increases with the use of the activation cache mechanism. We can observe that by leveraging the activation cache, fine-tuning latency is reduced by 26% to 71%. Moreover, this reduction in latency increases with the number of epochs. For example, with the T5-Large model, training for two epochs reduces latency by 39%, whereas training for ten epochs increases the reduction to 71%. This reduction can be attributed to the fact that the Parallel Adapters constitute a lightweight, independent network, resulting in a significant decrease in training cost compared to the LLM backbone. We can bypass both the forward and backward passes through the LLM backbone since the necessary activations are already cached.

VII. Related Work

Parameter-Efficient Fine-Tuning for LLM: Prompt tuning [21] proposes to prepend the model input embeddings with a trainable tensor. Adapters tuning [10] adds domain-specific layers after attention and FFN layers in transformer. LoRA [11] decomposes the parameter update for a weight matrix into two trainable low-rank matrices. To further reduce the memory overhead, pioneering studies explore fine-tuning techniques that obviate the need for backpropagation through the backbone model. Y-tuning [22] learns additional task-specific label representations, which are integrated with the output of the backbone model to circumvent backpropagation. LST [18] involves the use of pruned lightweight transformer structures from the backbone as a side network. E ^{3} VA [51] extends the concept of the side network into the realm of computer vision.

Model Compressing Techniques for LLM Fine-tuning: Recent pioneering techniques focus on compressing LLMs to mitigate resource challenges. For instance, structural pruning, exemplified by LLM-Pruner [26], selectively removes non-critical coupled structures based on gradient information, thereby maximally preserving the majority of the LLM's functionality. Similarly, researchers at Google [33] have demonstrated transferring knowledge from a large billion-parameter T5 teacher model to a student model with only millions of parameters via knowledge distillation, while retaining near-comparable performance. By reducing the model's volume through compression, computational and memory resources required for full parameter fine-tuning can also be decreased.

The Parallel Adapter approach utilized in our work is categorized as a PEFT technique. However, our design is orthogonal to aforementioned model compression methods. Specifically, when the resource demands for the LLM backbone are substantial, the backbone can initially undergo appropriate compression to fit within the constraints of the target edge environment. Then, our Parallel Adapter technology can be further applied to significantly reduce the resources needed for fine-tuning, while simultaneously providing superior pluggable characteristics and mitigating catastrophic forgetting—critical advantages over full fine-tuning.

On-device DNN Fine-Tuning: POET [19] achieves the fine-tuning of a BERT model on embedded devices, optimizing for both training speed and energy consumption. Lin et al. [20] enable training directly on devices with a minimal memory requirement of only 256 KB. Sage and Melon [14], [15] implement hybrid memory management and conservation strategies, including operator fusion and the use of a dedicated memory pool, to mitigate memory limitations. Additionally, Mandheling [13] incorporates mixed-precision training along with DSP offloading to enhance the speed of learning.

Collaborative Edge Computing for DNN Fine-Tuning: Federated Learning (FL) has been a promising paradigm in distributed machine learning that enables in-situ model fine-tuning. FwdLLM [24] designs a backpropagation-free fine-tuning FL protocol to enhance efficiency. AdaFL [8] proposes an FL framework for fine-tuning LLMs that features adaptable depth and width in its adapters modules. Breaking through the conventional paradigm of FL, Ye et al. [39], [52] devise a pipeline parallel architecture that facilitates the collaborative fine-tuning of DNNs across multiple edge devices. EDDL [38] adopts data parallelism training across embedded devices in a local area network. Asteroid [48] also employs HPP across multiple edge devices for DNN training, but it does not specifically address the parameter-efficient fine-tuning of LLMs.

This paper is an extension of the shorter conference version [50]. In this work, we further explore low-precision

quantization of the LLM backbone and weight initialization for Parallel Adapters, enabling further optimization of both memory and time overhead in LLM fine-tuning. Also, we design a novel heterogeneity-aware parallelism planning algorithm to fully exploit the computational resource of heterogeneous edge clusters. Additionally, we provide a more detailed discussion of the design and implementation of our Parallel Adapters architecture.

VIII. Conclusion

This paper proposes PAC+, a time and memory efficient collaborative edge AI framework for personal LLMs fine-tuning. PAC+ breaks the resource wall of personal LLMs fine-tuning with a sophisticated algorithm-system co-design. Extensive evaluation of the prototype implementation demonstrates that PAC+ outperforms existing collaborative edge training systems, achieving up to a $9.7\times$ end-to-end speedup and reducing memory footprint by up to 88.16% compared to mainstream LLM fine-tuning algorithms.

References

[1] A. Vaswani et al., "Attention is all you need," in *Proc. NeurIPS*, 2017, vol. 30, pp. 5998–6008.

[2] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.

[3] M. Lewis et al., "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics*, 2020, pp. 7871–7880.

[4] Y. Li et al., "Personal LLM agents: Insights and survey about the capability, efficiency and security," 2024, *arXiv:2401.05459*.

[5] L. Guo, W. Choe, and F. X. Lin, "STI: Turbocharge NLP inference at the edge via elastic pipelining," in *Proc. 28th ACM Int. Conf. Architectural Support Programm. Lang. Oper. Syst.*, 2023, vol. 2, pp. 791–803.

[6] D. Xu et al., "EdgeLLM: Fast on-device LLM inference with speculative decoding," *IEEE Trans. Mobile Comput.*, vol. 24, no. 4, pp. 3256–3273, 2025.

[7] S. Ye et al., "Galaxy: A resource-efficient collaborative edge ai system for in-situ transformer inference," in *Proc. IEEE Conf. Comput. Commun.*, 2024, pp. 1001–1010.

[8] D. Cai, Y. Wu, S. Wang, F. X. Lin, and M. Xu, "Efficient federated learning for modern NLP," in *Proc. 29th Annu. Int. Conf. Mobile Comput. Netw.*, 2023, pp. 1–16.

[9] X. Miao, G. Oliaro, X. Cheng, M. Wu, C. Unger, and Z. Jia, "FlexLLM: A system for co-serving large language model inference and parameter-efficient finetuning," 2024, *arXiv:2402.18789*.

[10] N. Houlsby et al., "Parameter-efficient transfer learning for NLP," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2790–2799.

[11] E. J. Hu et al., "LORA: Low-rank adaptation of large language models," in *Proc. Int. Conf. Learn. Representations*, 2021.

[12] "Jetson-nano," 2019.[Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[13] D. Xu et al., "Mandheling: Mixed-precision on-device DNN training with DSP offloading," in *Proc. 28th Annu. Int. Conf. Mobile Comput. Netw.*, 2022, pp. 214–227.

[14] I. Gim and J. Ko, "Memory-efficient DNN training on mobile devices," in *Proc. 20th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2022, pp. 464–476.

[15] Q. Wang et al., "Melon: Breaking the memory wall for resource-efficient on-device machine learning," in *Proc. 20th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2022, pp. 450–463.

[16] J. O. Zhang, A. Sax, A. Zamir, L. Guibas, and J. Malik, "Side-tuning: A baseline for network adaptation via additive side networks," in *Proc. IEEE Eur. Conf. Comput. Vis.*, 2020, pp. 698–714.

[17] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, "Parameter-efficient fine-tuning for large models: A comprehensive survey," *Trans. Mach. Learn. Res.*, vol. 2024, 2024.

[18] Y.-L. Sung, J. Cho, and M. Bansal, "LST: Ladder side-tuning for parameter and memory efficient transfer learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, vol. 35, pp. 12991–13005.

[19] S. G. Patil, P. Jain, P. Dutta, I. Stoica, and J. Gonzalez, "POET: Training neural networks on tiny devices with integrated rematerialization and paging," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 17573–17583.

[20] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256KB memory," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, vol. 35, pp. 22941–22954.

[21] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," 2021, *arXiv:2104.08691*.

[22] Y. Liu, C. An, and X. Qiu, "Y-tuning: An efficient tuning paradigm for large-scale pre-trained models via label representation learning," *Front. Comput. Sci.*, vol. 18, no. 4, 2024, Art. no. 184320.

[23] Y. Wei et al., "Communication-efficient model parallelism for distributed in-situ transformer inference," in *Proc. Design, Automat. Test Europe Conf. Exhib.*, 2024, pp. 1–6.

[24] M. Xu, D. Cai, Y. Wu, X. Li, and S. Wang, "FwdLLM: Efficient federated finetuning of large language models with perturbed inferences," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC 24)*, 2024, pp. 579–596.

[25] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. 5th Int. Conf. Learn. Representations*, Toulon, France, Apr. 24–26, 2017, pp. 1683–1695.

[26] X. Ma, G. Fang, and X. Wang, "LLM-pruner: On the structural pruning of large language models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, vol. 36, pp. 21702–21720.

[27] G. Fang, X. Ma, M. Song, M. B. Mi, and X. Wang, "DepGraph: Towards any structural pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 16091–16101.

[28] VainF, "Torch-pruning: A structured pruning toolkit for pytorch," *GitHub Repository*, 2023. [Online] Available: https://github.com/VainF/Torch-Pruning

[29] G. Hinton, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[30] Y. Gu, L. Dong, F. Wei, and M. Huang, "MiniLLM: Knowledge distillation of large language models," in *Proc. 12th Int. Conf. Learn. Representations*, 2024, pp. 13 278–13 301.

[31] L. Beyer, X. Zhai, A. Royer, L. Markeeva, R. Anil, and A. Kolesnikov, "Knowledge distillation: A good teacher is patient and consistent," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 10925–10934.

[32] C.-Y. Hsieh et al., "Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes," 2023, *arXiv:2305.02301*.

[33] G. Research, "Distilling step-by-step," 2021. Accessed: Jan. 15, 2025. [Online]. Available: https://github.com/google-research/distilling-step-by-step

[34] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "GPT3.int8(): 8-bit matrix multiplication for transformers at scale," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, vol. 35, pp. 30318–30 332.

[35] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2024, vol. 36, pp. 10088–10115.

[36] T. Dettmers and L. Zettlemoyer, "The case for 4-bit precision: K-bit inference scaling laws," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 7750–7774.

[37] Z. Zhang et al., "Quantized side tuning: Fast and memory-efficient tuning of quantized large language models," in *Proc. 62nd Annu. Meeting Assoc. Comput. Linguistics*, 2024, vol. 1, pp. 1–17.

[38] P. Hao and Y. Zhang, "EDDL: A distributed deep learning system for resource-limited edge computing environment," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2021, pp. 1–13.

[39] S. Ye, L. Zeng, Q. Wu, K. Luo, Q. Fang, and X. Chen, "ECO-FL: Adaptive federated learning with efficient edge collaborative pipeline training," in *Proc. 51st Int. Conf. Parallel Process.*, 2022, pp. 1–11.

[40] D. Narayanan et al., "PipeDream: Generalized pipeline parallelism for DNN training," in *Proc. 27th ACM Symp. Oper. Syst. Princ.*, 2019, pp. 1–15.

[41] "Pytorch," 2019. [Online]. Available: https://github.com/pytorch/pytorch

[42] X. Jiang et al., "MNN: A universal and efficient inference engine," in *Proc. Mach. Learn. Syst.*, 2020, vol. 2, pp. 1–13.

[43] "On-device training with tensorflow lite," 2021. [Online]. Available: https://www.tensorflow.org/lite/ examples/on_device_training/overview

[44] J. Yuan et al., "Mobile foundation model as firmware," in *Proc. 30th Annu. Int. Conf. Mobile Comput. Netw.*, 2024, pp. 279–295.

[45] "Jetson-tx2," 2017. [Online]. Available: https://developer.nvidia.com/embedded/jetson-tx2

[46] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, vol. 27, pp. 19–27.
[47] Y. Huang et al., "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. NeurIPS*, 2019, vol. 32, pp. 103–112.
[48] 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. 30th Annu. Int. Conf. Mobile Comput. Netw.*, 2024, pp. 312–326.
[49] J. H. Park et al., "{HetPipe }: Enabling large { DNN} training on (whimpy) heterogeneous {GPU} clusters through integration of pipelined model parallelism and data parallelism," in *Proc. USENIX Annu. Tech. Conf.*, 2020, pp. 307–321.
[50] B. Ouyang, S. Ye, L. Zeng, T. Qian, J. Li, and X. Chen, "Pluto and charon: A time and memory efficient collaborative edge AI framework for personal LLMs fine-tuning," in *Proc. 53rd Int. Conf. Parallel Process.*, 2024, pp. 762–771.
[51] D. Yin, X. Han, B. Li, H. Feng, and J. Bai, "Parameter-efficient is not sufficient: Exploring parameter, memory, and time efficient adapter tuning for dense predictions," in *Proc. 32nd ACM Int. Conf. Multimedia*, 2024, pp. 1398–1406.
[52] L. Zeng, S. Ye, X. Chen, and Y. Yang, "Implementation of big AI models for wireless networks with collaborative edge computing," *IEEE Wireless Commun.*, vol. 31, no. 3, pp. 50–58, Jun. 2024.

Shengyuan Ye (Graduate Student Member, IEEE) received the BE degree in 2021 from the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China, where he is currently working toward the PhD degree. His research interests include mobile edge computing, resource-efficient mobile AI systems, anddistributed machine learning systems.

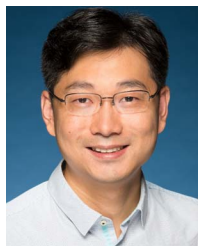
Bei Ouyang received the BS degree in computer science in 2023 from the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China, where she is currently working toward the master's degree with the School of Computer Science and Engineering. Her research interests include mobile edge computing and distributed computing.

Tianyi Qian received the BS degree in computer science and technology from the Honors College of Engineering Interdisciplinary Excellence Program, Northwestern Polytechnical University, Xi'an, China, in 2023. She is currently working toward the master's degree with the School of Computer Science, Sun Yat-sen University, Guangzhou, China. Her research interests include mobile edge computing and distributed computing.

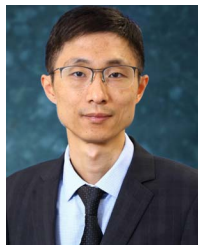
Liekang Zeng (Member, IEEE) received the BE and PhD degrees from Sun Yat-sen University, Guangzhou, China. His research interests include edge intelligence, mobile computing, and distributed machine learning systems.

Jingyi Li received the BE and BBA degrees from the South China University of Technology in 2020. He is currently working toward the PhD degree with the School of Computer Science and Engineering, Sun Yat-sen University. His research interests include edge intelligence and privacy preservation.

Jiangsu Du received the BSc degree from Wuhan University in 2016, the MSc degree from Edinburgh Parallel Computing Center, University of Edinburgh in 2017, and the PhD degree from the School of Computer Science and Engineering, Sun Yat-sen University in 2022. He is currently a postdoctoral researcher with the School of Computer Science and Engineering, Sun Yat-sen University. His research interests include high performance computing, parallel and distributed artificial intelligent system.



Xiaowen Chu (Fellow, IEEE) received the BEng degree in computer science from Tsinghua University, Beijing, China, in 1999, and the PhD degree in computer science from The Hong Kong University of Science and Technology, Hong Kong, in 2003. He is currently a professor with Data Science and Analytics Thrust, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China. His research interests include graphics processing unit computing, distributed machine learning, and wireless networks.



Guoliang Xing (Fellow, IEEE) received the PhD degree from Washington University, St. Louis, MO, USA, in 2006. He was a faculty member with Michigan State University, East Lansing, MI, USA. He is currently a professor of information engineering with The Chinese University of Hong Kong, China. His research interests include Internet of Things, smart health, cyber-physical systems, security, and wireless networking. He was the recipient of the three Best Paper Awards and six Best Paper Nominations at leading international conferences. He has led several multi-disciplinary research projects on smart health systems for activity recognition, family wellness, and sleep monitoring. Several mobile health technologies developed in his lab won Best App Awards at the MobiCom conference and were successfully transferred to the industry.

Xu Chen received the PhD degree in information engineering from the Chinese University of Hong Kong in 2012. He is currently a full professor with Sun Yat-sen University, Guangzhou, China, and the vice director of National and Local Joint Engineering Laboratory. His research interests include edge computing, edge intelligence, edge robotics, AI for networking, mobile social networks game theory, deep learning, dynamic optimization, and resource allocation.