FedQuad: Adaptive Layer-Wise LoRA Deployment and Activation Quantization for Federated Fine-Tuning

Jianchun Liu, *Member, IEEE*, Rukuo Li, Hongli Xu, *Member, IEEE*, Qianpiao Ma, Jiaming Yan, and Liusheng Huang, *Member, IEEE*

***Abstract*—Federated fine-tuning (FedFT) provides an effective paradigm for fine-tuning large language models (LLMs) in privacy-sensitive scenarios. However, practical deployment remains challenging due to the limited resources on end devices. Existing methods typically utilize parameter-efficient fine-tuning (PEFT) techniques, such as Low-Rank Adaptation (LoRA), to substantially reduce communication overhead. Nevertheless, significant memory usage for activation storage and computational demands from full backpropagation remain major barriers to efficient deployment on resource-constrained end devices. Moreover, substantial resource heterogeneity across devices results in severe synchronization bottlenecks, diminishing the overall fine-tuning efficiency. To address these issues, we propose FedQuad, a novel LoRA-based FedFT framework that adaptively adjusts the LoRA depth (the number of consecutive tunable LoRA layers from the output) according to devices' computational power, while employing activation quantization to reduce memory overhead, thereby enabling efficient deployment on resource-constrained devices. Specifically, FedQuad first identifies the feasible and efficient combinations of LoRA depth and the number of activation quantization layers based on device-specific resource constraints. Subsequently, FedQuad employs a greedy strategy to select the optimal configurations for each device, effectively accommodating system heterogeneity. Extensive experiments demonstrate that FedQuad achieves a 1.4–5.3× convergence acceleration compared to state-of-the-art baselines when reaching target accuracy, highlighting its efficiency and deployability in resource-constrained and heterogeneous end-device environments.**

***Index Terms*—Federated fine-tuning, resource constraint, system heterogeneity, low-rank adaptation, activation quantization.**

Received 6 July 2025; revised 21 October 2025; accepted 17 November 2025. Date of publication 25 November 2025; date of current version 6 April 2026. This work was supported in part by the National Science Foundation of China (NSFC) under Grants 62472401, Grant 62132019, and Grant 624B2136; in part by the Jiangsu Province Science Foundation for Youths under Grant BK20230275; in part by the Anhui Province Science Foundation for Youths under Grant 2408085QF185; and in part by the Fundamental Research Funds for the Central Universities under Grant WK2150110033 and Grant WK2150250044. Recommended for acceptance by H. Wu. *(Corresponding author: Hongli Xu.)*

Jianchun Liu, Rukuo Li, Hongli Xu, Jiaming Yan, and Liusheng Huang are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China, and also with the Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou 215123, China (e-mail: jcliu17@ustc.edu.cn; lirukuo@mail.ustc.edu.cn; xuhongli@ustc.edu.cn; jmyan@mail.ustc.edu.cn; lshuang@ustc.edu.cn).

Qianpiao Ma is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 211111, China (e-mail: maqianpiao@njust.edu.cn).

Digital Object Identifier 10.1109/TMC.2025.3637064

I. Introduction

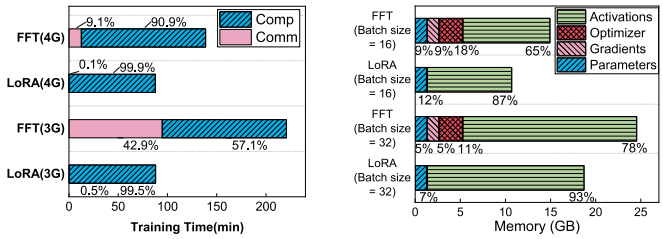
RECENT advances in large language models (LLMs), such as GPT-4 [1] and DeepSeek-V3 [2], have demonstrated remarkable modeling capabilities across various natural language processing (NLP) tasks. Serving as foundational models, LLMs naturally stimulate interest in fine-tuning for downstream domain-specific tasks, including question answering [3], sentiment analysis [4], and machine translation [5]. However, centralized fine-tuning methods rely on aggregating substantial amounts of domain-specific data from end devices, posing privacy risks [6] and facing constraints imposed by regulations such as GDPR [7]. To address these concerns, federated fine-tuning (FedFT), a distributed fine-tuning approach, has emerged [8], [9]. FedFT enables end devices to perform local model fine-tuning without sharing raw data, subsequently aggregating model parameters from individual devices at a parameter server (PS) to facilitate knowledge sharing across devices.

Nevertheless, the practical deployment of FedFT faces substantial challenges due to the mismatch between the enormous scale of LLMs and the limited resources (i.e., computational power, communication bandwidth, and memory) of end devices. For instance, standard NLP LLMs such as RoBERTa-large [10] demand approximately 4 GB of network traffic per training round and over 3,400 TFLOPs of computation. In contrast, typical end devices, such as the NVIDIA Jetson TX2 [11], provide less than 2 TFLOPS of computational power. In addition, the available bandwidth for end devices is typically below 100 Mbps, which is common in wide area network (WAN) environments [12]. As a result, the convergence process in real-world scenarios may extend to hundreds of hours [13]. Furthermore, fine-tuning RoBERTa-large requires around 25 GB of memory, exceeding the capabilities of most end devices, which typically have less than 16 GB of memory [14]. Recent studies on FedFT have primarily mitigated communication overhead by leveraging various parameter-efficient fine-tuning (PEFT) methods [9], [13], such as Adapter [15] and Low-Rank Adaptation (LoRA) [16]. Specifically, LoRA reduces the number of trainable parameters by freezing the base model and updating only trainable low-rank matrices inserted into transformer layers [16]. As these LoRA modules typically comprise less than 1% of the total parameters of the base LLM, LoRA substantially reduces communication overhead in FedFT.

Despite the substantial reduction in communication overhead enabled by LoRA, FedFT still faces two major challenges, i.e., resource constraints and system heterogeneity. *First*, resource (i.e., memory, computing power) constraints remain a critical bottleneck for deploying LLMs on end devices. We notice that LoRA does not fundamentally eliminate memory overhead, as fine-tuning LLMs still demands substantial memory to store intermediate activations. For example, fine-tuning RoBERTa-large [10] with LoRA still demands about 17.4 GB of memory, of which 15.9 GB is devoted to activation storage. Modern end devices typically have only 4–16 GB of RAM [14], exacerbating memory constraints. Additionally, due to the heavy computational demand of fine-tuning, resource-limited devices often exhibit extremely slow training speeds [17].

Second, participating devices typically possess heterogeneous system characteristics [12]. In particular, they show significant disparities in computational power (e.g., CPU frequency), with performance gaps often exceeding tenfold, causing synchronization delays as strong devices wait for weak ones during each training round [18]. For instance, the FLOPS capability of an iPhone 16 is only about 12% of that of a desktop GPU RTX 3090. Such imbalances often cause devices with weak capabilities to become stragglers, prolonging overall training time and impairing fine-tuning efficiency. Furthermore, compared to strong devices, weaker devices may be unable to load the full model and are thus either constrained to smaller sub-models or excluded from the training process altogether, leading to diminished overall performance.

Existing methods for addressing these challenges are typically classified into two major categories. The *first* category [19], [20] fine-tunes LoRA parameters on a subset of transformer layers and discards the rest. For example, Su et al. [19] propose FedRA, which constructs sub-models by randomly selecting transformer layers to accommodate resource constraints. Liu et al. [20] introduce InclusiveFL, assigning consecutive layers from the input based on device capabilities and employing momentum distillation to improve shallow model performance. However, these methods directly discard certain transformer layers, compromising the original model architecture, thereby limiting feature extraction capabilities and degrading accuracy and convergence speed [21]. Besides, memory constraints may limit the number of trainable layers on stronger devices, which restricts the utilization of their computational power. During synchronization, these devices often need to wait for weaker participants, resulting in significant delays and a degradation in overall fine-tuning efficiency. The *second* category [22], [23] retains all transformer layers while fine-tuning only a subset of LoRA parameters within them, such as by freezing some LoRA layers or reducing the LoRA rank. For instance, Sun et al. [22] propose selectively fine-tuning critical layers while freezing others to save resources, thereby avoiding accuracy degradation from discarded layers. However, the uneven distribution of resource consumption across layers limits the practical deployability of this approach and leads to longer waiting times. To enhance fine-tuning efficiency, Cho et al. [23] introduce HetLoRA, assigning different LoRA ranks based on device capabilities to alleviate system heterogeneity. Nonetheless, merely



(a) Training Time (b) Memory

Fig. 1. Resource Consumption Comparison: LoRA vs. Full Fine-Tuning.

adjusting LoRA rank does not fundamentally resolve LoRA's high computational overhead, and memory savings remain limited, hindering practical deployment. In this work, we propose FedQuad, a novel federated fine-tuning framework that adaptively determines the configuration of LoRA depth (i.e., the number of consecutive unfrozen LoRA layers from the output) and the number of activation quantization layers (i.e., consecutive transformer layers with quantized activations starting from the first unfrozen LoRA layer) to address challenges arising from resource constraints and system heterogeneity. As discussed in Section II, increasing LoRA depth generally improves fine-tuning performance but also increases resource consumption, making deployment on memory-limited devices impractical. Notably, due to the nature of backpropagation, freezing layers beyond the first fine-tuned LoRA layer yields little benefit in reducing memory consumption. While activation checkpointing reduces memory cost, it adds substantial computational overhead. To this end, FedQuad proposes to reduce memory usage via activation quantization, thereby enabling deeper LoRA configurations and improving fine-tuning performance. Specifically, FedQuad first identifies feasible and efficient configurations of LoRA depth and quantization layers that satisfy device-specific memory constraints. FedQuad then selects the optimal configuration by jointly considering device computational power, thereby striking a balance between training efficiency and model accuracy. Crucially, FedQuad maintains the full transformer architecture, ensuring both high model fidelity and practical deployability in heterogeneous federated environments. Even within memory-feasible configurations, the interplay between LoRA depth and activation quantization has a significant impact on overall fine-tuning efficiency. Increasing the number of quantized layers allows for deeper LoRA integration, thereby enhancing model accuracy. It also introduces additional computational overhead, potentially leading to prolonged synchronization delays across devices. Conversely, reducing the number of quantized layers lowers computational latency but restricts the allowable LoRA depth, ultimately limiting the model's representational capacity and fine-tuning performance. Therefore, under resource constraints, identifying an optimal trade-off between LoRA depth and the number of quantized layers remains a non-trivial yet essential task for balancing fine-tuning accuracy and efficiency. Our main contributions are summarized as follows:

- We propose FedQuad, an efficient LoRA-based FedFT framework that effectively addresses resource constraints

and system heterogeneity by jointly determining the LoRA depth and the number of activation quantization layers.
- We conduct a comprehensive analysis of the joint impact of LoRA depth and activation quantization on fine-tuning performance. Based on this insight, we design a greedy-based algorithm that dynamically selects optimal configurations according to device capabilities.
- Extensive experimental evaluations demonstrate that our proposed algorithm achieves 1.4–5.3× acceleration to reach the target accuracy, compared to existing solutions.

II. Background and Motivation

A. Federated Fine-Tuning LLMs With LoRA

LoRA for LLMs: With the rapid increase in the number of parameters in LLMs, traditional training methods, such as fine-tuning all model parameters, achieve excellent performance but are extremely resource-intensive and costly. Consequently, PEFT methods have emerged as essential approaches for the efficient deployment of LLMs. Among various PEFT methods, LoRA has gained widespread popularity as an effective and efficient strategy. LoRA reduces the number of trainable parameters by introducing low-rank adapters, enabling scalable and cost-effective fine-tuning of LLMs. Specifically, instead of updating the full parameter matrix of the pre-trained model, LoRA inserts two low-rank matrices, denoted as $\boldsymbol{A}$ and $\boldsymbol{B}$, to approximate the weight update. Let $\boldsymbol{W}_0$ and $\Delta\boldsymbol{W}$ denote the pre-trained model's weight matrix and the trainable weight update, respectively. The LoRA update can be formulated as:

$$\boldsymbol{W}_0 + \Delta\boldsymbol{W} = \boldsymbol{W}_0 + \boldsymbol{B}\boldsymbol{A} \tag{1}$$

where $\boldsymbol{W}_0 \in \mathbb{R}^{d\times k}$, $\boldsymbol{B} \in \mathbb{R}^{d\times r}$, and $\boldsymbol{A} \in \mathbb{R}^{r\times k}$, with $r \ll \min(d, k)$. Here, r is referred to as the LoRA rank, which determines the dimension of the subspace used for low-rank adaptation. This design approximates the weight update $\Delta\boldsymbol{W}$ via low-rank decomposition, where matrix $\boldsymbol{A}$ projects the input features into an r-dimensional subspace, and matrix $\boldsymbol{B}$ reconstructs the representation back to the output dimension. During training, inputs are simultaneously passed through both the frozen pre-trained model and the LoRA adapters. The outputs are then combined to form the final representation. For an input feature x_l in the l-th layer, the output h_l is computed as:

$$h_l = \boldsymbol{W}_l x_l + \Delta\boldsymbol{W}_l x_l = \boldsymbol{W}_l x_l + \boldsymbol{B}_l\boldsymbol{A}_l x_l \tag{2}$$

Only the parameters $\boldsymbol{A}_l$ and $\boldsymbol{B}_l$ are updated via gradient descent by minimizing the loss between the model output and the ground-truth labels. Therefore, LoRA significantly reduces the number of trainable parameters, typically to less than 1% of the pre-trained model size [9], while effectively preserving the generalization capability of the pre-trained model.

Federated Learning with LoRA: Federated learning (FL) is a distributed training paradigm that enables multiple end devices to collaboratively train a shared global model without sharing their local data. The global training objective in FL with LoRA is to find the optimal model parameters $\boldsymbol{W}^* = \{\Delta\boldsymbol{W}^*, \boldsymbol{W}_0\}$ that minimize the overall loss function $F(\boldsymbol{W})$. This objective can be formally defined as:

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}=\{\Delta\boldsymbol{W},\boldsymbol{W}_0\}}{\operatorname{argmin}} F(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|\mathbb{D}_i|}\sum_{\xi_i\in\mathbb{D}_i} F_i(\boldsymbol{W}_i;\xi_i) \tag{3}$$

where $F_i(\boldsymbol{W}_i;\xi_i)$ denotes the local loss function computed on device i with respect to a data sample ξ_i from its private dataset $\mathbb{D}_i$, and n is the total number of participating devices.

In each local training round h, end device i updates its local LoRA parameters using stochastic gradient descent (SGD) [24]:

$$\Delta\boldsymbol{W}_i^h = \Delta\boldsymbol{W}^h - \eta\cdot\nabla F_i(\Delta\boldsymbol{W}^h) \tag{4}$$

where η is the learning rate, and $\nabla F_i(\cdot)$ denotes the gradient of the local loss function with respect to the trainable parameters. After local training, each end device sends the updated parameters $\Delta\boldsymbol{W}_i^h$ to the central parameter server (PS), which performs model aggregation:

$$\Delta\boldsymbol{W}^{h+1} = \frac{1}{n}\sum_{i=1}^{n}\Delta\boldsymbol{W}_i^h \tag{5}$$

The server then distributes the aggregated updates to all end devices for the next training round.

In conventional FL, the entire pre-trained model's parameters must be synchronized between end devices and the server, which incurs high communication overhead, particularly for LLMs. When incorporating LoRA into FL, the pre-trained backbone $\boldsymbol{W}_0$ remains frozen throughout training, and only the low-rank adapter parameters $\Delta\boldsymbol{W}$ are updated and exchanged. This design substantially reduces the volume of transmitted data on resource-constrained devices, thus accelerating convergence and improving the feasibility of federated fine-tuning (FedFT) with LLMs.

B. Resource Bottlenecks of Fine-Tuning With LoRA

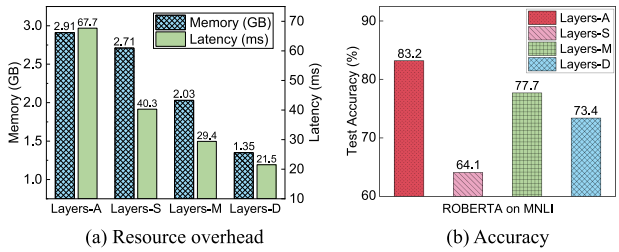
One of the key advantages of LoRA is its ability to significantly reduce the number of parameters that need to be updated during training. By introducing trainable low-rank matrices on top of the original weight matrices, LoRA constrains parameter updates to low-rank matrices, thereby greatly reducing communication overhead in FedFT. For instance, we measure the proportion of communication time and computation time per training round under 3G and 4G network conditions. As shown in Fig. 1(a), compared with full fine-tuning (FFT), LoRA only trains less than 1% of the full model parameters, resulting in a negligible communication overhead. Specifically, communication accounts for only 0.5% of the training time under 3G and 0.1% under 4G, confirming that communication overhead remains negligible even in low-bandwidth environments. Nevertheless, LoRA does not fully address the memory and computation bottlenecks during training. First, LoRA still demands substantial memory resources. Although reducing the number of trainable parameters alleviates memory usage for optimizer states and gradient storage, activations still need to be retained for gradient computation during the backward pass, which occupies a substantial portion of GPU memory. In particular, existing LoRA-based FedFT frameworks, such

as FedLoRA [25] and HetLoRA [23], typically update LoRA modules across all transformer layers. Consequently, activations from every layer must be preserved throughout training, exacerbating memory pressure and limiting the scalability of FedFT on memory-constrained devices. To further illustrate the severity of memory consumption caused by activations, we conduct a series of fine-tuning experiments on RoBERTa-large [10] using the MNLI dataset [26]. The batch size is set to 16 and 32, respectively, with a maximum sequence length of 256, and GPU memory usage is continuously monitored throughout training. As shown in Fig. 1(b), activations account for the majority of memory consumption in LoRA-based fine-tuning. With a batch size of 16, activations occupy 85.1% of total memory usage. When the batch size increases to 32, the total memory consumption reaches 17.4 GB, with activation memory usage rising to 15.9 GB, accounting for 91.5% of the total. Therefore, activation storage is the main cause of the memory bottleneck in LoRA-based fine-tuning.

Furthermore, computational power remains a critical bottleneck for LoRA. For instance, fine-tuning RoBERTa-large using LoRA on an A6000 GPU reduces per-batch latency from 1,002 ms to approximately 699 ms, which amounts to only a 30.2% reduction in computational time compared to full fine-tuning. Although LoRA dramatically reduces the number of trainable parameters and thus shortens gradient computation and optimizer update time, the forward and backward passes through the frozen backbone still remain computationally intensive. Consequently, resource-constrained devices within FedFT systems experience slow training speeds, resulting in significant synchronization delays and reduced overall fine-tuning efficiency. In summary, while LoRA substantially reduces communication costs, both memory and computational limitations continue to pose significant challenges for LoRA-based fine-tuning of LLMs on end devices with limited resources.

C. Opportunities for Resource-Efficient Fine-Tuning

1) Effect of LoRA Depth on Fine-Tuning: The memory and computational overhead of LoRA is largely determined by the positions of the unfrozen layers. Specifically, updating the parameters of a given layer during training requires storing the activations of that layer and all subsequent layers. Moreover, to compute the gradient for that layer, the backward pass must propagate gradients from the output layer back to it. Our observations further reveal that both the position and number of unfrozen LoRA layers have a significant impact on fine-tuning performance, i.e., resource consumption and model accuracy. In particular, consecutively fine-tuning LoRA layers starting from the output demonstrates superior efficiency in both resource usage and accuracy improvement. With this approach, although increasing the number of trainable layers continues to improve model accuracy, the marginal gains gradually diminish, while resource consumption grows nearly linearly. Based on these observations, we define the number of consecutively unfrozen LoRA layers starting from the output as the LoRA depth, which plays a critical role in balancing model accuracy and resource usage. To assess its impact, we conduct a series of experiments.



(a) Resource overhead (b) Accuracy

Fig. 2. The impact of LoRA position on fine-tuning.

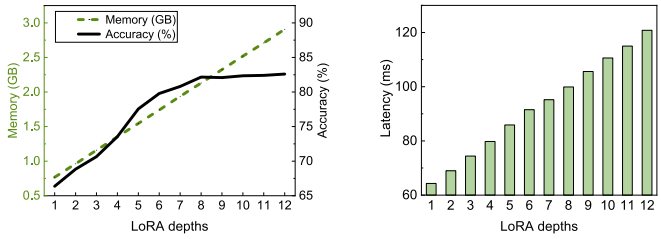
First, to verify the impact of the position of unfrozen LoRA layers on fine-tuning performance, we conduct a fine-tuning experiment using a 12-layer RoBERTa-base model [10] on the MNLI dataset [26] (detailed experimental settings can be found in Section IV). In this experiment, we selectively fine-tune LoRA layers at different positions while freezing the remaining layers. Specifically, we evaluate four configurations: all layers fine-tuned (denoted as Layers-A), shallow layers {0, 1, 2, 3} (Layers-S), middle layers {4, 5, 6, 7} (Layers-M), and deep layers {8, 9, 10, 11} (Layers-D). The experimental results in Fig. 2 lead to the following two key insights:

1) Consecutive fine-tuning from the output layer is more resource-efficient. As shown in Fig. 2(a), despite fine-tuning only one third of the layers, Layers-S consumes 93% of the memory used by Layers-A. In comparison, Layers-M and Layers-D consume 69.9% and 46.5% of the memory used by Layers-A, respectively. In LoRA fine-tuning, updating a given layer requires retaining the activations of all subsequent layers. As a result, although Layers-S fine-tunes only the first four layers, activations from all 12 layers must still be stored, leading to high memory usage. Moreover, Layers-S exhibits $1.37\times$ and $1.87\times$ higher computation latency than Layers-M and Layers-D, respectively. This is because the trainable layers in Layers-S are located near the input, resulting in a longer backward pass and thus higher computational overhead.

2) Shallow-layer fine-tuning provides limited performance gains. As shown in Fig. 2(b), Layers-M and Layers-D achieve accuracies of 77.7% and 73.4%, respectively, while Layers-S achieves only 64.1%. Furthermore, fine-tuning the last eight layers (i.e., Layers-M and Layers-D) achieves 82.8% accuracy, which is already close to the 83.2% obtained by full-layer fine-tuning (Layers-A). These results indicate that shallow layers contribute little to downstream task adaptation, whereas middle and deep layers play a more critical role in capturing task-relevant features and enhancing model performance.

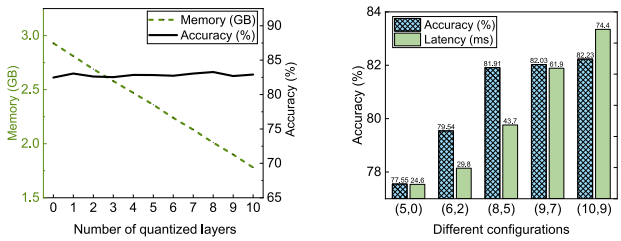
Therefore, fine-tuning consecutive LoRA layers from the output layer while freezing all other parameters is not only more resource-efficient but also straightforward and effective in preserving model accuracy. Then, we further investigate the relationship between LoRA depth and both resource usage and model accuracy. We conduct a series of experiments using RoBERTa-base on MNLI, with LoRA depth ranging from 1 to 12. The experimental results, shown in Fig. 3, reveal the following two key findings:

1) As LoRA depth increases, resource consumption exhibits an approximately linear growth trend. Specifically, each



(a) Accuracy & Memory usage (b) Computational latency

Fig. 3. The impact of LoRA depths on fine-tuning.



(a) Accuracy & Memory usage (b) Accuracy & Latency

Fig. 4. The impact of activation quantization on fine-tuning.

additional LoRA layer increases the computational latency by approximately 5 ms and memory usage by around 199MB.

2) Although model accuracy improves with increasing LoRA depth, the accuracy gain gradually diminishes, particularly in the shallower layers. From Fig. 3(a), we observe that when LoRA depth increases from 1 to 8, model accuracy improves by 15.8%, with an average accuracy gain of approximately 2.26% per additional LoRA layer. However, when the depth increases from 8 to 12, the accuracy gain is only 0.4%.

In summary, while model accuracy continues to improve with increasing LoRA depth, the marginal accuracy gain progressively diminishes, while memory usage and computational latency grow in a near-linear fashion. Therefore, selecting an appropriate LoRA depth based on the capabilities of end devices is essential to balance model accuracy and resource consumption.

2) Impact of Activation Quantization on Fine-Tuning: Previous observations indicate that stable model accuracy during training typically requires a larger LoRA depth. However, increasing the LoRA depth inevitably leads to greater resource consumption, especially in terms of memory usage, which may exceed the capacity of end devices and ultimately render the deployment of LLMs impractical in real-world scenarios. Despite the stringent resource constraints of heterogeneous devices, we identify opportunities to achieve resource-efficient federated fine-tuning. During training, activation storage is the primary contributor to the memory bottleneck. Existing approaches to mitigate activation-related memory overhead generally fall into two categories, i.e., activation checkpointing and activation quantization. Activation checkpointing reduces memory usage by discarding intermediate activations during the forward pass and recomputing them during the backward pass as needed. While this approach significantly lowers memory consumption, it introduces considerable computational overhead. For instance, fine-tuning a RoBERTa-base model on the MNLI dataset using an A6000 GPU shows that enabling activation checkpointing increases per-batch latency by approximately 83%. In contrast, activation quantization reduces memory usage by quantizing activations during the forward pass and dequantizing them during backpropagation for gradient computation. Although this method also introduces additional computational overhead, we optimize the quantization and dequantization operations using Triton [27], following the approach in Jetfire [28] (see Section IV for details), which results in only a 36% increase in per-batch latency. Additionally, we implement layer-wise quantization to better balance computational overhead.

We define the number of activation quantization layers as the count of consecutive transformer layers whose activations are quantized during training, starting from the first unfrozen LoRA layer. Taking RoBERTa-base as an example, as shown in Fig. 4(a), we enable all LoRA layers for fine-tuning and apply activation quantization sequentially from the first transformer layer onward. This approach results in an average memory reduction of 58% when fine-tuning the corresponding LoRA layers. Moreover, when the number of quantized layers reaches 8, the model achieves a 0.8% accuracy improvement compared to the non-quantized baseline. This performance gain is attributed to the stochastic noise introduced during quantization, which helps prevent overfitting and enhances the model's generalization capability.

To investigate the synergy between the number of activation quantization layers and LoRA depth under fixed memory constraints, we systematically vary the configuration, represented as (LoRA depth, number of quantized layers), and evaluate the resulting performance. For example, a configuration of (8, 5) represents a LoRA depth of 8 and the number of quantized layers of 5. As shown in Fig. 4(b), reallocating memory saved through activation quantization to support larger LoRA depth can substantially improve model accuracy. For instance, the configuration (5, 0) achieves an accuracy of 77.55%, while (8, 5) reaches 81.91%, yielding an absolute improvement of 4.36%. However, as both LoRA depth and the number of quantized layers continue to increase, the marginal accuracy gain gradually diminishes. For example, increasing from (8, 5) to (10, 9) improves accuracy by only 0.32%, indicating a clear saturation effect. Meanwhile, computational latency consistently increases. When moving from (8, 5) to (10, 9), the delay rises from 43.7 ms to 74.4 ms, representing a 70.3% increase.

While increasing the number of quantized layers can support a greater LoRA depth and improve model accuracy, it also introduces additional latency, which may cause slower devices to become stragglers and prolong synchronization time during training, thus reducing overall fine-tuning efficiency. Therefore, achieving efficient fine-tuning requires identifying the optimal configuration of the number of activation quantization layers and LoRA depth under heterogeneous resource constraints.

III. System Design

To clearly present the framework and design principles of FedQuad, this section details its architecture and workflow.

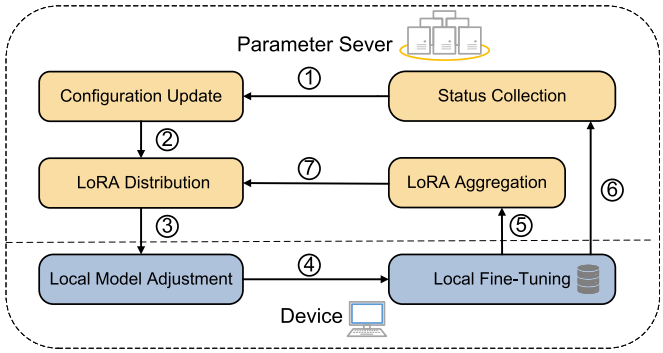


Fig. 5. Overview of FedQuad's workflow.

Section III-A introduces the overall system design, while Section III-B describes how device resources are collected and modeled. Section III-C presents the adaptive configuration mechanism for determining LoRA depth and activation quantization. Section III-D and III-E explain the local model adjustment and layer-wise aggregation strategies that ensure efficient updates across heterogeneous devices. Finally, Section III-F offers an illustrative example demonstrating how FedQuad adapts to diverse resource conditions.

A. System Overview

Inspired by the above observations, we propose FedQuad, a novel and efficient federated fine-tuning framework tailored for end devices with constrained resources and heterogeneous system characteristics. Specifically, FedQuad determines the optimal fine-tuning configurations for each device based on their status information. Upon completion of local fine-tuning, FedQuad performs adaptive aggregation of the updated parameters and decides the optimal configurations for the next training round. The overall workflow of FedQuad comprises six main steps (as illustrated in Fig. 5):

Status Collection: At the beginning of each training round, the parameter server (PS) collects status information from each device (①), including available memory and computational power to inform the configuration updates.

Configuration Update: The PS updates appropriate configurations, including LoRA depth and the number of activation quantization layers for each device based on their current memory availability and computational power (②). This ensures efficient resource utilization while simultaneously avoiding out-of-memory errors and degraded fine-tuning efficiency.

LoRA Dissemination: The PS disseminates the aggregated LoRA weights along with the updated configurations to every device (③).

Local Model Adjustment: Upon receiving its corresponding configuration, each device reconfigures its local model by adjusting unfrozen LoRA layers and quantized activation layers. After adjusting the settings of individual layers within the model, each device updates its local LoRA parameters to align with the received global parameters before starting the local fine-tuning phase (④).

Local Fine-Tuning: End devices perform local fine-tuning while monitoring runtime performance such as memory utilization and computational latency. After completing the local training, devices upload the updated LoRA parameters (⑤) and recorded performance status (⑥) to the PS.

LoRA Aggregation: The PS performs adaptive weighted aggregation of the received LoRA parameters, considering the varying LoRA depths used by individual devices, to obtain the updated global model (⑦).

The overall procedure iterates over multiple training rounds until the model satisfies a predefined convergence condition or achieves the target accuracy.

B. Status Collection

FedQuad effectively addresses resource constraints and system heterogeneity by allocating optimal training configurations to each device. Specifically, in the h-th training round, FedQuad determines a tailored configuration (d_i^h, a_i^h) for each device i according to its resource constraints, where d_i^h represents the assigned LoRA depth and a_i^h represents the number of activation quantization layers.

Before determining these configurations, each device uploads its current status, including available memory and computational power. The available memory of device i in round h is denoted by M_i^h, representing the portion of GPU or system memory that can be safely used for training without interfering with other local applications.

In round h, the computational power q_i^h denotes the floating-point operations per second (FLOPs) throughput of device i. The computational overhead of the fine-tuning task, denoted as $C(\cdot)$, is modeled as a linear function of the LoRA depth d and the number of quantized layers a:

$$C(d, a) = c_f + c_d \cdot d + c_a \cdot a \tag{6}$$

where c_f represents the computational cost of the forward pass, c_d denotes the additional computational overhead introduced by each extra LoRA-tunable layer in the backward pass, and c_a indicates the computational overhead associated with quantizing a single transformer layer. Accordingly, the local training time u_i^h for end device i during round h is calculated as:

$$u_i^h = \frac{C(d_i^h, a_i^h)}{q_i^h} = \frac{c_f + c_d \cdot d_i^h + c_a \cdot a_i^h}{q_i^h} \tag{7}$$

C. Configuration Update

Based on each device's status report, FedQuad constructs a unified model that incorporates accuracy, memory, and latency to guide configuration decisions. As discussed in Section II, fine-tuning performance is primarily determined by the LoRA depth d. Let $\mathcal{D}^h$ denote the set of LoRA depths assigned to all devices in the h-th training round, and let $\mathcal{D}$ be the union of all such depths across H training rounds, where H denotes the total number of training rounds:

$$\mathcal{D}^h = \left\{ d_i^h \mid i \in [1, n] \right\} \tag{8}$$

$$\mathcal{D} = \bigcup_{h=1}^{H} \mathcal{D}^h = \bigcup_{h=1}^{H} \left\{ d_i^h \mid i \in [1, n] \right\} \tag{9}$$

Let $\Delta \boldsymbol{W}^H(\mathcal{D})$ represent the aggregated global LoRA parameters after H training rounds under the set $\mathcal{D}$. The loss function of the global model after training can be expressed as $F(\boldsymbol{W_0}, \Delta \boldsymbol{W}^H(\mathcal{D}))$. The convergence requirement after training can be formulated as the following constraint:

$$F\left(\boldsymbol{W_0}, \Delta \boldsymbol{W}^H(\mathcal{D})\right) - F\left(\boldsymbol{W_0}, \Delta \boldsymbol{W}^*\right) \leq \epsilon \tag{10}$$

where $\epsilon > 0$ denotes the convergence threshold, and $F(W_0, \Delta W^*)$ represents the optimal fine-tuning performance. Ideally, ϵ should approach zero, indicating that the trained model closely approximates the optimal solution [29].

On resource-constrained devices, the number of LoRA layers that can be fine-tuned is typically limited. However, the memory saved through activation quantization can be reallocated to support a larger LoRA depth, enabling the fine-tuning of more layers and thereby improving fine-tuning accuracy. Therefore, let m_o denote the additional memory overhead incurred by increasing the LoRA depth by one layer, and m_q the memory savings achieved by quantizing the activations of a single transformer layer, and m_f the fixed memory overhead during training (e.g., model parameters). Then, the memory constraint can be formulated as:

$$m_f + m_o \cdot d_i^h - m_q \cdot a_i^h \leq M_i^h, \quad \forall i \in [1, n], \forall h \in [1, H] \tag{11}$$

The completion time t_i^h taken by device i in round h includes both local training time and communication time. Nevertheless, given the significantly small size of LoRA parameters, the communication time is omitted in subsequent analysis [9], [25]. Therefore, the completion time can be approximated as:

$$t_i^h = u_i^h \tag{12}$$

Strong devices have to wait for weak devices to complete training in each round. Let t_h denote the longest completion time across all participating devices in round h. We estimate the average waiting time across all devices in round h as:

$$w_h = \frac{1}{n} \sum_{i=1}^{n} \left(t_h - t_i^h\right) \tag{13}$$

A larger w_h indicates greater synchronization delay, which can slow down overall convergence. To mitigate this issue, we impose a constraint on the average waiting time in each round. Specifically, we require that:

$$w_h = \frac{1}{n} \sum_{i=1}^{n} \left(t_h - t_i^h\right) \leq \theta, \quad \forall i \in [1, n], \forall h \in [1, H] \tag{14}$$

where θ is a predefined threshold that limits the allowable average waiting time.

As both LoRA layer freezing and activation quantization are applied on a per-layer basis, the LoRA depth d and the number of quantized layers a must be non-negative integers. Specifically, d has to satisfy $d \in [1, L]$, where L denotes the total number of transformer layers of the pre-trained model, and a should satisfy $a \in [0, d-1]$, since we notice that quantizing the final output layer tends to degrade model accuracy while incurring additional memory overhead. Thus, the constraints can be formulated as:

$$d, a \in \mathbb{Z}_{\geq 0}, \quad d \in [1, L], \quad a \in [0, d-1] \tag{15}$$

where $\mathbb{Z}_{\geq 0}$ denotes the set of non-negative integers.

Given a FedFT task, FedQuad aims to assign an appropriate configuration (d_i^h, a_i^h) to each device i in each round h, based on its available resources, to achieve the target accuracy while minimizing the total fine-tuning time $\sum_{h=1}^{H} t_h$. Thus, we can formulate the problem as:

$$\begin{aligned} \min \quad & \sum_{h=1}^{H} t_h \\ \text{s.t.} \quad & (10),\ (11),\ (14),\ (15) \end{aligned} \tag{16}$$

To solve the problem in (16), we adopt a greedy-based Adaptive Configuration Selection (ACS) method, as detailed in Algorithm 1, to optimize per-device settings at each round. ACS first identifies the feasible and efficient configurations (d, a) for each device based on its memory constraint (Lines 1– 10). Specifically, the PS iterates through all possible LoRA depths d for each device i according to its available memory M_i^h, and adjusts the number of activation quantization layers a accordingly. For each candidate depth d, it identifies the minimal number of quantized layers a that satisfies the memory constraint in (11) while avoiding additional computational overhead. The corresponding configuration (d, a) is then added to the feasible set $\mathcal{C}_i^h$.

Next, FedQuad evaluates the training cost and performance gain of each candidate configuration (Lines 11– 16). Based on the current device's computational power, FedQuad estimates the completion time t_i^h for every feasible configuration (d, a). Additionally, FedQuad records the average completion time t_{avg}^{h-1} from the previous round as an estimate for the current round's average completion time. FedQuad aims to minimize the difference between each device's completion time and t_{avg}^{h-1}, enabling strong devices to fully utilize their computational resources while ensuring weak devices do not cause excessive synchronization delays.

Since gradient information inherently captures the marginal contribution of each layer to the loss function [22], [30], FedQuad quantifies the expected performance gain $G^h(d)$ for a LoRA depth of d in the h-th round based on layer-wise gradient norms. Specifically, FedQuad computes the aggregated gradient norm g_l^{h-1} for each LoRA layer $l \in [0, L-1]$ in the global model at round $h-1$, and then sums the norms of the top d layers. Formally, the expected performance gain at round h is defined as:

$$G^h(d) = \sum_{l=L-d}^{L-1} g_l^{h-1} \tag{17}$$

Finally, to balance performance gain and training cost, FedQuad defines a reward function to evaluate the trade-off of

Algorithm 1: Adaptive Configuration Selection (ACS) in FedQuad.

Input: Device memory constraint M_i^h at round h; Compute capacity q_i^h at round h; Average completion time t_{avg}^{h-1} in round $h-1$; Global gradient norm g_l^{h-1} for each layer l at round $h-1$;
Output: Optimal configuration (d_i^h, a_i^h) for each device i at round h

```
// Step 1: Identify feasible and efficient configurations (d, a) under memory constraint
for each device i ∈ [1, n] do
    C_i ← ∅;
    a_cur ← 0;
    for d ∈ [1, L] do
        for a ∈ [a_cur, d−1] do
            if the memory constraint in Eq. (11) holds then
                Add (d, a) to C_i;
                a_cur ← a;
                break ;   // Once a valid pair (d, a) is found, break out of the innermost loop
// Step 2: Estimate completion time for each feasible configuration
for each device i ∈ [1, n] do
    Estimate t_i^h for each (d, a) ∈ C_i using q_i^h (see Eqs. ((7) and 12));
// Step 3: Compute performance gain based on global gradient norms
for each LoRA depth d ∈ [1, L] do
    Compute G^h(d) = Σ_{l=L−d}^{L−1} g_l^{h−1};
// Step 4: Select the configuration that maximizes reward
for each device i ∈ [1, n] do
    (d_i^h, a_i^h) ← arg max_{(d,a)∈C_i} R(d, a);
return {(d_i^h, a_i^h) | i ∈ [1, n]};
```

each configuration (d, a) for each device i:

$$R(d,a) = \frac{G^h(d)}{\left|t_i^h(d,a) - t_{\text{avg}}^{h-1}\right| + c} \tag{18}$$

where the numerator $G^h(d)$ represents the contribution of each configuration to model convergence. The denominator captures the gap between the completion time under the current configuration (d, a) and the average completion time, with a small constant c (the detailed configuration is provided in Section IV) added to prevent division by zero. A smaller gap indicates that the selected configuration better matches the device's computational power, resulting in a higher reward. Therefore, a higher $R(d, a)$ reflects a more favorable trade-off between accuracy and efficiency. The pair (d, a) achieving the highest $R(d, a)$ is chosen as the optimal configuration for device i (Lines 17– 19).

D. Local Model Adjustment

In round h, device i receives the LoRA parameters $\Delta \boldsymbol{W}^h$ and the training configuration (d_i^h, a_i^h) from PS. Based on this configuration, the device first adjusts the trainable LoRA layers, represented by L_t, as:

$$L_t = \left\{l \,|\, l \in [L - d_i^h, L-1]\right\}, \forall i \in [1,n],\ \forall h \in [1,H].$$

The LoRA layers within L_t are set to be trainable, while all other layers remain frozen. Subsequently, the device configures the activation quantization transformer layers, denoted as L_q, for the current round as follows:

$$L_q = \left\{l \,|\, l \in [L - d_i^h, L - d_i^h + a_i^h - 1]\right\}, \forall i \in [1,n], \quad \forall h \in [1,H].$$

Once the fine-tuning configuration adjustments are complete, the device incorporates updates received from the PS to align its local model with the global model. Specifically, for a LLM with pre-trained parameters $\boldsymbol{W}_0 = \{\boldsymbol{W}_l^0 \,|\, l \in [0, L-1]\}$, the device i updates its local model as:

$$\boldsymbol{W}_i^h = \left\{\Delta \boldsymbol{W}^h, \boldsymbol{W}_0\right\},$$

where $\Delta \boldsymbol{W}^h$ denotes the global LoRA parameters in round h. After local model adjustment, device i trains the model on the local dataset $\mathcal{D}_i$.

E. LoRA Aggregation

Upon receipt of the updated LoRA parameters from all devices, the PS performs global aggregation. Due to heterogeneous LoRA depth across devices, each layer may receive updates from a different subset of devices. To address this heterogeneity, the PS employs an adaptive, layer-wise aggregation protocol. Specifically, the aggregation process for each layer only considers the updates from the devices that have valid updates for that particular layer, thereby enhancing the performance of the global model for that layer.

The PS aggregates the available updates layer by layer. Let the global LoRA update for layer l be denoted as $\Delta \boldsymbol{W}_l^{h+1}$, which is obtained by aggregating the updates from n_l devices. The adaptive layer-wise aggregation rule is defined as:

$$\Delta \boldsymbol{W}_l^{h+1} = \frac{1}{n_l} \sum_{i=1}^{n_l} \Delta \boldsymbol{W}_{i,l}^h \tag{19}$$

where $\Delta \boldsymbol{W}_{i,l}^h$ denotes the update to the LoRA layer l by device i during round h, and n_l represents the number of devices with valid updates for layer l. Here, $\Delta \boldsymbol{W}_l^{h+1}$ denotes the aggregated global update for layer l.

After aggregating the LoRA parameters, FedQuad initiates a new round of status collection to support subsequent configuration updates, thereby guiding the fine-tuning process on each device based on its latest resource constraints. By adaptively assigning larger LoRA depths and the corresponding number of quantized layers to stronger devices, and smaller configurations to weaker ones based on their resource limitations, FedQuad improves fine-tuning performance and efficiency. Consequently, it effectively addresses the dual challenges of resource constraints and system heterogeneity.

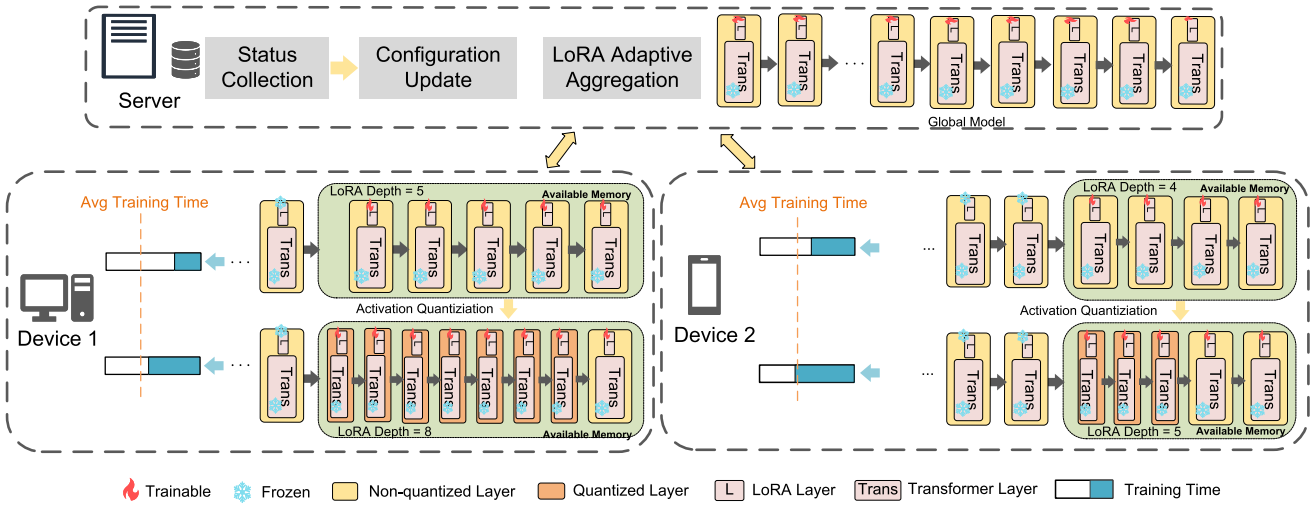


Fig. 6. An illustrative example of FedQuad.

F. An Illustrative Example of FedQuad

As illustrated in Fig. 6, FedQuad determines the optimal configuration for each device based on its resource constraints to accelerate overall convergence. For strong devices, although their greater memory capacity allows for larger LoRA depth during fine-tuning, the prolonged completion times of weaker devices often result in extended idle waiting for the strong devices. To mitigate this, FedQuad increases the number of activation quantization layers to support larger LoRA depths, thereby fully utilizing the computational resources of strong devices and improving fine-tuning performance. For example, as shown for Device 1 in Fig. 6, its available memory allows only limited LoRA layers, resulting in a relatively smaller contribution to model convergence and a much shorter completion time than the average, which indicates underutilized computing capacity. FedQuad increases its quantization layers to support a larger LoRA depth. Although this leads to a slight increase in training time, Device 1 still completes earlier than average, thus causing no synchronization delays and helping reduce overall idle time. For weaker devices, constrained in both memory and computational power, aggressively increasing quantization layers to enable larger LoRA depth may improve accuracy but risks introducing excessive synchronization delay. Therefore, FedQuad adjusts both activation quantization and LoRA depth, aiming to improve fine-tuning performance without overburdening the device. This prevents the device from becoming a straggler. For instance, FedQuad assigns Device 2 additional quantization layers to enhance fine-tuning performance, but limits the number to ensure that its completion time remains close to the average, avoiding significant slowdowns to the overall training process. After all end devices complete local training, they upload their parameter updates and status information to the PS. Upon receiving these updates, the PS performs adaptive aggregation of the LoRA parameters, while simultaneously leveraging the devices' status information to determine the optimal configurations for the subsequent round.

IV. Experimental Settings

System Implementation: We implement the FedQuad prototype based on the open-source FedPETuning framework [9], enhancing its functionality with approximately 2,500 lines of custom code. FedQuad leverages the modular APIs of the *Transformers* library [31] to support efficient LLM initialization and adaptation. To enable activation quantization, we modify the core LLM modules in the transformers.models package. Following the observations reported in SLIMFIT [32], we note that static activations from nonlinear operations such as GELU, MatMul, Softmax, and LayerNorm cannot be entirely eliminated by freezing the associated layers. Therefore, during the forward pass, we quantize the activations of these functions and subsequently apply dequantization during the backward pass. This approach effectively reduces memory overhead without compromising model accuracy. Our quantization process follows the approach proposed in Jetfire [28], and is implemented using Triton [27], a domain-specific programming language and compiler designed for developing efficient deep learning primitives. In our experiments, we set the block size to 32, i.e., B=32. We adopt PyTorch [33] as the training framework to manage the local fine-tuning workflow. GPU acceleration is enabled by CUDA v12.3 and cuDNN v9.1.0. For distributed communication between edge devices and the central server, we utilize torch.distributed, a native PyTorch library for parallel and distributed training [34].

Models: We primarily evaluate FedQuad on three popular LLMs: BERT-large [35] with 336 M parameters, RoBERTa-large [10] with 355 M parameters, and DeBERTaV3-large [36] with 435 M parameters. These models have been extensively adopted in prior federated fine-tuning studies [8], [9], [13], [20]. Each model consists of 24 transformer layers. Pre-trained weights are obtained from the Hugging Face repository [37].

Datasets: We conduct our experiments on four widely used NLP datasets from the GLUE benchmark [26] (summarized in Table II): (1) The Multi-Genre Natural Language Inference (MNLI) dataset is a large-scale dataset containing over 400,000

TABLE I
TECHNICAL SPECIFICATIONS OF JETSON KITS

	AI Performance	GPU Type
Jetson TX2	1.33 TFLOPS	256-core Pascal
Jetson NX	21 TOPS	384-core Volta
AGX Xavier	32 TOPS	512-core Volta
	CPU Frequency	**GPU Frequency**
Jetson TX2	1.2GHz	0.85GHz
Jetson NX	1.2GHz	0.8GHz
AGX Xavier	1.45GHz	0.9GHz

TABLE II
DATASETS AND NUMBERS OF SAMPLES USED IN THE EXPERIMENTS

Dataset	Application	# Train	# Test
MNLI	Textual Entailment	392,702	9,815
QQP	Semantic Equivalence	363,846	40,430
QNLI	Question Answering	104,743	5,463
SST-2	Sentiment Analysis	67,349	1,821

sentence pairs used for training and evaluating LLMs on natural language inference (NLI) tasks. In the NLI tasks, the goal is to determine whether a premise entails, contradicts, or is neutral with respect to a given hypothesis. (2) The Quora Question Pairs (QQP) dataset consists of more than 400,000 question pairs collected from the Quora platform. Each pair is labeled to indicate whether the two questions are paraphrases of each other or not. QQP is commonly utilized for training and evaluating LLMs on paraphrase detection tasks. (3) The Question Natural Language Inference (QNLI) dataset is derived from the Stanford Question Answering Dataset (SQuAD). It consists of over 100,000 sentence–question pairs labeled to indicate whether the sentence contains the correct answer to the corresponding question. (4) The Stanford Sentiment Treebank Binary Version (SST-2) dataset contains 70,042 movie review sentences annotated with binary sentiment labels. Each sentence is labeled as either positive or negative. To simulate non-independent and identically distributed (non-i.i.d.) data across devices, we follow prior work [8], [9], [13] by partitioning the dataset using a Dirichlet distribution. Specifically, for each device, we sample training data according to $D \sim \text{Dir}(\alpha)$, where α controls the degree of non-i.i.d. Unless otherwise specified, we set $\alpha = 10$ throughout all experiments.

Hardware: Consistent with previous federated learning literature [9], [13], [38], our experiments are conducted in a semi-simulated manner using an AMAX deep-learning workstation equipped with eight NVIDIA A6000 GPUs. On-device training times are measured on three representative edge devices (summarized in Table I): (1) Jetson TX2, a compact embedded computing platform designed specifically for AI applications at the edge. (2) Jetson NX, capable of accelerating computation by up to 21 TOPS and offering sufficient parallel computing power to support LLM workloads. (3) AGX Xavier, the most powerful among the three, delivering computational capabilities of up to 32 TOPS. Both TX2 and NX support four computational modes, whereas AGX supports eight modes. Devices operating in different modes demonstrate distinct performance levels.

System Setup: Our experiments involve a total of 100 devices, categorized into three types (strong, moderate, and weak) based on computational performance and memory capacity. To simulate the resource constraints of heterogeneous end devices, we adopt the following experimental setup:

1) *For Memory:* For clarity, we use the tunable LoRA depth in FedLoRA (i.e., federated fine-tuning with vanilla LoRA) to represent each device's available memory capacity. For all methods involved, we convert the LoRA depth to the corresponding memory capacity to ensure a fair comparison. Furthermore, we assign dynamic depth ranges to different device categories: strong devices are assigned LoRA depths in the range of 18–24, moderate devices in the range of 11–17, and weak devices in the range of 4–10. To further simulate fluctuations in memory availability under real-world conditions, we randomly adjust each device's tunable LoRA depth within its respective range after each training round.

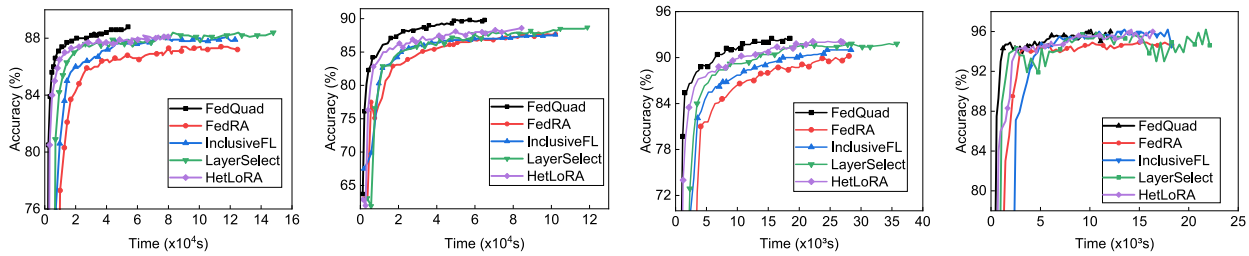
2) *For Compute:* We map the strong, moderate, and weak device categories to NVIDIA Jetson AGX, Jetson NX, and Jetson TX2 devices, respectively, representing varying levels of computational resources. Each Jetson device supports multiple operating modes corresponding to different computational capabilities and power configurations. We construct diverse computational profiles by setting various operation modes [39]. To realistically emulate fluctuations in computational resources, each device randomly switches its operating mode every 10 training rounds.

Baselines: To evaluate the effectiveness of FedQuad, we compare it against four baseline methods: (1) FedRA [19] randomly selects subsets of transformer layers based on device resource constraints to construct sub-models and fine-tunes these sub-models using LoRA. (2) InclusiveFL [20] allocates sub-models with varying numbers of consecutively stacked layers starting from the input layer, rather than selecting layers randomly, based on each device's resource constraints. To mitigate gradient loss in shallow models, it applies momentum distillation. (3) LayerSel [22] leverages local gradient information to select specific layers for fine-tuning across devices, while freezing the remaining layers instead of discarding them to conserve resources. Fine-tuning is then performed using LoRA. (4) HetLoRA [40] allows devices to fine-tune heterogeneous local models by allocating different LoRA ranks to tackle system heterogeneity.

Metrics: The following metrics are adopted to evaluate the performance of FedQuad and the baselines.

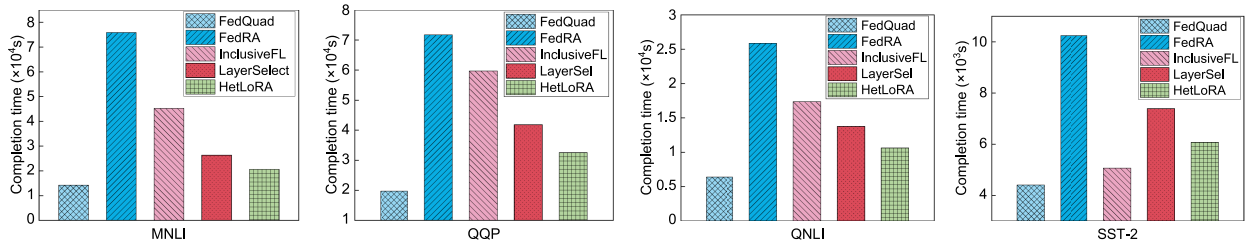
1) *Test Accuracy* reflects the accuracy of the models trained by different approaches on the test datasets, measured by the proportion of correctly predicted data. Specifically, we record the test accuracy of the global model (the model after aggregation at the PS) in each round.

2) *Time-to-Accuracy* represents the total wall-clock time required for training a model to achieve a target accuracy. For fair comparisons, we set the target accuracy as the highest achievable accuracy by FedQuad and all baselines. We record



(a) MNLI (b) QQP (c) QNLI (d) SST-2

Fig. 7. Test accuracy of four approaches on the four datasets.



(a) Time to reach 87.5% accuracy (b) Time to reach 87% accuracy (c) Time to reach 90% accuracy (d) Time to reach 95% accuracy

Fig. 8. Completion time of four approaches when achieving the target accuracy.

the completion time of each round, summing it up to obtain the total training time.

3) *Average Waiting Time* represents the average time each edge device spends waiting for global aggregation in each round, indicating the training efficiency of each method.

Experimental Parameters: By default, all experiments are carried out on our prototype system and run 50 rounds. Each device fine-tunes 1 epoch per round using AdamW [41] optimizer locally. The learning rate is set as 0.001 and decays according to a cosine scheduler. The batch size is fixed at 32 and the maximum sequence length is set to 128 for all experiments. The small constant c in (18) is set to 0.0001.

V. Performance Evaluation

A. Numerical Results

This section comprehensively evaluates FedQuad against representative baselines to verify its effectiveness and efficiency under diverse settings. Section V-A presents overall fine-tuning performance across datasets and models. Section V-B analyzes the impact of device heterogeneity, while Section V-C conducts ablation studies to assess the contributions of key components. Section V-D examines the sensitivity to major hyperparameters, and Section V-E summarizes additional findings and discussions.

Overall Performance: We conduct extensive experiments to evaluate the performance of FedQuad against several representative baselines. Figs. 7 and 8 illustrate the fine-tuning trajectories and overall completion time on various general language understanding tasks.

The results demonstrate that FedQuad achieves the fastest convergence across all tasks, significantly outperforming baselines. By adaptively allocating LoRA depth and activation quantization layers based on device-specific resource constraints, FedQuad enhances fine-tuning performance while reducing local training time. For instance, as shown in Figs. 7(a) and 8(a), FedQuad reaches 87.5% accuracy on MNLI within only 14,213 s, whereas FedRA, InclusiveFL, LayerSel, and HetLoRA require 75,928 s, 45,262 s, 26,350 s, and 20,642 s, respectively. Compared to FedRA, InclusiveFL, LayerSel, and HetLoRA, FedQuad provides 5.3×, 3.2×, 1.85×, and 1.45× speedups. Similarly, Figs. 7(b) and 8(b) show that FedQuad achieves 87% accuracy on QQP in 19,738 s, while FedRA, InclusiveFL, LayerSel, and HetLoRA take 71,707 s, 59,700 s, 41,845 s, and 32,575 s, respectively. The corresponding speedups are 3.6×, 3.0×, 2.1×, and 1.65×. Moreover, Figs. 7 and 8 further indicate that FedQuad consistently achieves superior completion times on QNLI and SST-2. As shown in Table III, FedQuad consistently achieves the fastest convergence across all evaluated models, significantly outperforming baseline methods. For instance, on BERT-large, FedQuad reduces convergence time by 80.2%, 83.0%, 53.5%, and 28.8% compared to FedRA, InclusiveFL, LayerSel, and HetLoRA, respectively. Similarly, on DeBERTa-large, FedQuad achieves convergence time reductions of 79.5%, 73.9%, 34.9%, and 22.4%, respectively. These results collectively confirm FedQuad's advantage in accelerating the fine-tuning process.

TABLE III
Comparison of Completion Time (In 10^3 S) Across Models on the MNLI Dataset Using Different Baselines

Model	Target Acc	FedQuad	FedRA	InclusiveFL	LayerSel	HetLoRA
BERT-large	81%	**14.1**	71.1	83.1	30.3	19.8
DeBERTa-large	88%	**23.5**	114.4	90.0	36.1	30.3
RoBERTa-large	87.5%	**14.2**	75.9	45.2	26.3	20.6

TABLE IV
COMPARISON OF COMPLETION TIME (IN 10^3 S) AND FINAL ACCURACY (%) ON MNLI UNDER DIFFERENT HETEROGENEITY LEVELS

Heter. Level	Low		Medium		High	
Target Acc	88.1		88.0		87.5	
Metric	Time	Final Acc	Time	Final Acc	Time	Final Acc
FedQuad	**5.2**	**88.9**	**8.7**	**88.7**	**14.2**	**88.8**
FedRA	7.8	88.1	21.4	88.0	75.9	87.5
InclusiveFL	7.2	88.3	15.7	88.2	45.2	87.9
LayerSel	6.5	88.6	14.6	88.4	26.3	88.4
HetLoRA	6.1	88.5	11.2	88.5	20.6	88.1

These results demonstrate that FedQuad achieves both improved fine-tuning performance and reduced convergence time. Unlike FedRA and InclusiveFL, which reduce resource consumption by directly skipping transformer layers, FedQuad employs a layer freezing strategy. This strategy avoids undesirable side effects such as output bias and accuracy degradation that often arise from layer dropping during forward propagation. Moreover, in both FedRA and InclusiveFL, memory limitations may hinder stronger devices from fully exploiting their computational power, leaving them idle as they wait for slower devices, thereby reducing overall fine-tuning efficiency and prolonging convergence time. Compared to more competitive baselines like LayerSel and HetLoRA, FedQuad also exhibits notable advantages. LayerSel selects layers for fine-tuning based on gradient norms, but the uneven distribution of resource consumption during fine-tuning leads to slower convergence and presents deployment challenges, as memory-constrained devices often cannot afford to compute gradient norms for all layers. HetLoRA, on the other hand, adapts LoRA ranks to device capabilities, but fails to fundamentally address the inherent resource overhead of PEFT. Adjusting only the LoRA rank is insufficient to overcome the system bottlenecks.

In contrast, FedQuad comprehensively models real-world device resource constraints. It employs a back-to-front layer selection strategy for fine-tuning, combined with activation quantization to mitigate memory constraints and fully leverage computational resources. This enables adaptive configuration of both LoRA depth and quantization layers for each device based on its resource constraints. As a result, FedQuad achieves substantially improved fine-tuning performance and significantly faster convergence.

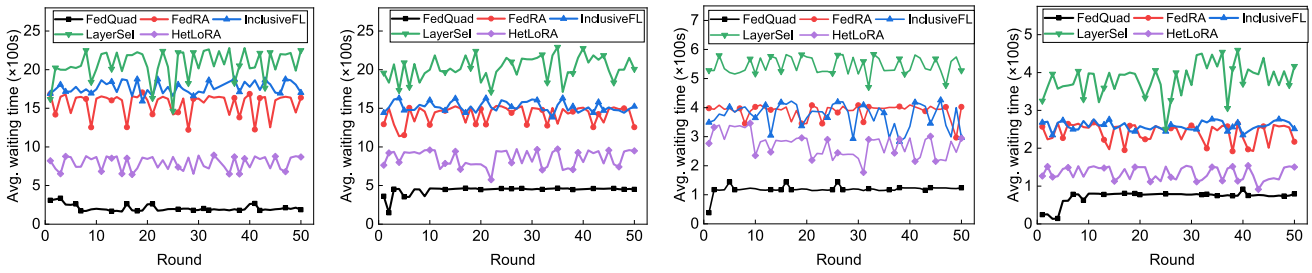
Effect of Device Heterogeneity: To comprehensively assess FedQuad's effectiveness under varying degrees of device heterogeneity, we evaluate FedQuad alongside baseline methods across three heterogeneity levels, i.e., Low, Medium, and High. Specifically, under the Low heterogeneity setting, we configure all devices as high-performance devices. For Medium, we equally distribute devices between high-performance and moderate-performance categories at a ratio of 1:1. In the High heterogeneity scenario, we assign the device proportions as 3:3:4 for strong-, moderate-, and weak-performance devices, respectively. As shown in Table IV, we observe that as device heterogeneity increases, the time required by all methods to reach the target accuracy grows significantly, which aligns with expectations when more resource-constrained devices are introduced into the system. However, compared with the baseline methods, FedQuad demonstrates remarkable advantages across all heterogeneity levels. For example, on the MNLI dataset under medium heterogeneity, FedQuad achieves speedups of approximately $2.5\times$, $1.8\times$, $1.7\times$, and $1.3\times$ over FedRA, InclusiveFL, LayerSel, and HetLoRA, respectively. Moreover, as heterogeneity rises from low to high, the performance gap between FedQuad and the other baselines further widens: FedQuad attains a $1.5\times$ speedup over FedRA in low heterogeneity scenarios, which increases to $5.35\times$ in high heterogeneity scenarios. These results indicate that FedQuad maintains robust and efficient fine-tuning capabilities across varying degrees of device heterogeneity.

To further assess FedQuad's capability in handling system heterogeneity, we report the average waiting time across four datasets in Fig. 9. FedQuad achieves the lowest average waiting time across all datasets. For example, on MNLI as illustrated in Fig. 9(a), FedQuad achieves the shortest average waiting time in each round. Furthermore, over all rounds, FedQuad reduces the average waiting time by 68.9%, 70.9%, 78.2%, and 48.1% compared to FedRA, InclusiveFL, LayerSel, and HetLoRA, respectively. On QQP, FedQuad achieves an average waiting time of 442.5 s, while the other baselines require 1423.1 s, 1520.4 s, 2026.1 s, and 852.4 s, corresponding to reductions of 68.9%, 70.9%, 78.2%, and 51.9%, respectively. This improvement stems from the fact that, although LayerSel selects the most important layers for fine-tuning, those layers can incur high computational overhead and thus long synchronization delays. FedRA and InclusiveFL adapt to heterogeneous devices by discarding layers but fail to fully leverage the computational capacity of more powerful devices under tight memory constraints, resulting in prolonged waiting times. HetLoRA mitigates heterogeneity by assigning devices to different LoRA levels but achieves only limited gains. In contrast, FedQuad adaptively determines the optimal combination of LoRA depth and activation quantization layers for each device based on its compute and memory resources, thereby maximizing resource utilization and minimizing synchronization delays for efficient fine-tuning.

B. Ablation Study

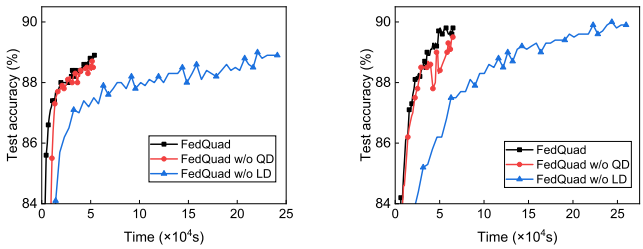
FedQuad incorporates two critical factors, i.e., LoRA depth and the number of activation quantization layers, which are specifically designed to enhance the efficiency and adaptability of FedLoRA in heterogeneous environments. In this section, we perform a series of ablation experiments on the MNLI and QQP datasets to evaluate the individual contributions of these two factors.

We compare three variants: (i) the full version of our method (FedQuad), (ii) a version without activation quantization (FedQuad w/o QD), and (iii) a version that applies the maximum possible number of activation quantization layers under memory constraints but removes adaptive LoRA depth (FedQuad w/o LD). As shown in Fig. 10, both LoRA depth and activation



(a) MNLI (b) QQP (c) QNLI (d) SST-2

Fig. 9. Average waiting time of four approaches on the four datasets.



(a) MNLI (b) QQP

Fig. 10. Effect of LoRA depth and activation quantization.

quantization play essential roles in FedQuad, though they affect system performance in different ways.

For instance, as illustrated in Fig. 10, both the FedQuad w/o QD and FedQuad w/o LD variants achieve final test accuracies comparable to that of FedQuad. However, when targeting an 88% accuracy on MNLI, FedQuad reduces training completion time by approximately 24.5% compared to FedQuad w/o QD and by 74.7% compared to FedQuad w/o LD. Similarly, on QQP, FedQuad accelerates the fine-tuning process by factors of $1.29\times$ and $3.53\times$ relative to FedQuad w/o QD and FedQuad w/o LD, respectively, when reaching the same 89% accuracy target.

These observations highlight the distinct roles and complementary benefits of the two factors. On the one hand, FedQuad w/o LD, which retains activation quantization, allows devices to fine-tune a larger number of LoRA layers, thereby improving final accuracy but at the cost of increased waiting time and slower convergence. On the other hand, FedQuad w/o QD, which removes activation quantization, restricts faster devices from utilizing their full capacity to fine-tune additional layers, limiting potential performance gains.

Overall, these results clearly validate the necessity of integrating both adaptive LoRA depth and activation quantization into FedQuad to achieve efficient and effective fine-tuning under heterogeneous system constraints.

VI. Related Work

Federated Fine-Tuning of LLMs: To fully leverage data on end devices while preserving user privacy, federated fine-tuning (FedFT) has emerged as a promising paradigm for adapting large language models (LLMs) to decentralized settings. For example, FedNLP [8] provides a benchmark framework for evaluating FedFT across various NLP tasks. To address communication overhead on devices, recent studies have turned to parameter-efficient fine-tuning (PEFT) techniques. Notably, FedAdapter [9] employs lightweight adapter modules to accelerate model convergence in FL, while FedLoRA integrates low-rank adapters [16] to improve training speed and accuracy.

Quantization of LLMs: While PEFT methods alleviate communication costs, the memory overhead associated with fine-tuning LLMs remains a major challenge. To alleviate the substantial memory overhead during the fine-tuning of LLMs, prior studies have proposed introducing quantization techniques during the training phase. A representative example is QLoRA [42], which reduces memory consumption by quantizing the weights of the pre-trained model and further applying a second-level quantization to the quantization factors. However, in practical deployment, memory usage associated with activations has emerged as a new bottleneck. SLIMFIT [32] significantly reduces memory consumption by quantizing activation tensors, but its coarse-grained quantization strategy often leads to notable degradation in fine-tuning performance. Jetfire [28] performs per-block quantization on activations, which significantly reduces memory overhead while preserving fine-tuning performance.

Resource-efficient FedFT Approaches: Substantial resource heterogeneity across devices results in severe synchronization bottlenecks, which significantly reduce the efficiency of FedFT. To address this challenge, existing works explore various fine-tuning strategies to tailor model architectures or training processes to device-specific resource profiles. For instance, FLAME [43] proposes a resource-adaptive federated fine-tuning framework based on a Sparse Mixture-of-Experts design. It keeps full LoRA parameters while varying the number of activated experts per client, and employs rescaling and activation-aware aggregation to address output imbalance. Dec-LoRA [44] proposes a decentralized coordination mechanism that allows clients to exchange parameter deltas through peer-to-peer topology, effectively removing reliance on a central server, but it does not address the substantial local memory footprint during fine-tuning. DropPEFT [45] designs a selective dropout mechanism over PEFT modules to balance accuracy and efficiency under device heterogeneity, yet stochastic dropout of adaptation layers may lead to performance degradation on complex tasks. AssyLLM [46] presents a modular assembling framework that reuses pre-trained Transformer blocks instead of full

backpropagation, achieving substantial reductions in both memory and communication overhead. Nevertheless, its assembly process is relatively rigid once the block pool is constructed, limiting flexibility for adaptive resource allocation. EcoLoRA [47] enhances communication efficiency in federated LoRA via round-robin segment sharing, adaptive sparsification, and lossless encoding, focusing on throughput rather than fine-grained memory optimization. PriFFT [48] combines arithmetic and function secret sharing for privacy-preserving fine-tuning, substantially lowering secure-computation overhead, though it emphasizes confidentiality over device-level efficiency. Collectively, these studies demonstrate significant progress toward scalable, resource-efficient, and privacy-aware federated fine-tuning of large models. However, few of them explicitly explore the joint optimization of LoRA depth and activation quantization layers under heterogeneous resource constraints.

VII. Conclusion

In this paper, we propose FedQuad, a novel and efficient federated fine-tuning framework designed to overcome the challenges posed by resource constraints and system heterogeneity. Specifically, FedQuad adaptively adjusts the LoRA depth to match the capabilities of individual devices and employs activation quantization to alleviate memory overhead, thus facilitating efficient deployment of large language models on resource-limited and heterogeneous devices. Additionally, we analyze the interplay between LoRA depth and the number of activation quantization layers, and devised a greedy-based algorithm to jointly determine optimal configurations for heterogeneous devices, significantly enhancing fine-tuning efficiency. Extensive experimental results demonstrate that FedQuad achieves superior convergence performance and accelerates fine-tuning by 1.4–5.3$\times$ compared to state-of-the-art baselines.

References

[1] J. Achiam et al., "GPT-4 technical report," 2023, *arXiv:2303.08774*.

[2] A. Liu et al., "Deepseek-V3 technical report," 2024, *arXiv:2412.19437*.

[3] Z. Jiang, F. F. Xu, J. Araki, and G. Neubig, "How can we know what language models know?," *Trans. Assoc. Comput. Linguistics*, vol. 8, pp. 423–438, 2020.

[4] W. Zhang, Y. Deng, B. Liu, S. J. Pan, and L. Bing, "Sentiment analysis in the era of large language models: A reality check," 2023, *arXiv:2305.15005*.

[5] B. Zhang, B. Haddow, and A. Birch, "Prompting large language model for machine translation: A case study," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 41092–41110.

[6] J. Huang et al., "Keystrokesniffer: An off-the-shelf smartphone can eavesdrop on your privacy from anywhere," *IEEE Trans. Inf. Forensics Secur.*, vol. 19, pp. 6840–6855, 2024.

[7] P. Voigt and A. Von dem Bussche, "The EU general data protection regulation (GDPR)," in *A Practical Guide*, 1st ed. Cham, Switzerland: Springer, 2017.

[8] B. Y. Lin et al., "FedNLP: Benchmarking federated learning methods for natural language processing tasks," 2021, *arXiv:2104.08815*.

[9] Z. Zhang et al., "FedPETuning: When federated learning meets the parameter-efficient tuning methods of pre-trained language models," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2023, pp. 9963–9977.

[10] Y. Liu et al., "RoBERTa: A robustly optimized bert pretraining approach," 2019, *arXiv: 1907.11692*.

[11] S. Mittal, "A survey on optimized implementation of deep learning models on the Nvidia jetson platform," *J. Syst. Archit.*, vol. 97, pp. 428–442, 2019.

[12] J. Liu, J. Yan, H. Xu, Z. Wang, J. Huang, and Y. Xu, "Finch: Enhancing federated learning with hierarchical neural architecture search," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 6012–6026, May 2024.

[13] D. Cai, Y. Wu, S. Wang, F. X. Lin, and M. Xu, "FedAdapter: Efficient federated learning for modern NLP," 2022, *arXiv:2205.10162*.

[14] A. Authority, "How much RAM does your Android phone really need in 2025?," 2025, Accessed: Mar. 16, 2025. [Online]. Available: https://www.androidauthority.com

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

[16] E. J. Hu et al., "LoRA: Low-rank adaptation of large language models," 2021, *arXiv:2106.09685*.

[17] J. Liu et al., "Federated learning with experience-driven model migration in heterogeneous edge networks," *IEEE/ACM Trans. Netw.*, vol. 32, no. 4, pp. 3468–3484, Aug. 2024.

[18] J. Liu et al., "Adaptive local update and neural composition for accelerating federated learning in heterogeneous edge networks," *IEEE Trans. Netw.*, vol. 33, no. 4, pp. 1438–1453, Aug. 2025.

[19] S. Su, B. Li, and X. Xue, "Fedra: A random allocation strategy for federated tuning to unleash the power of heterogeneous clients," in *Proc. Eur. Conf. Comput. Vis.*, 2024, pp. 342–358.

[20] R. Liu et al., "No one left behind: Inclusive federated learning over heterogeneous devices," in *Proc. 28th ACM SIGKDD Conf. Knowl. Discov. Data Mining*, 2022, pp. 3398–3406.

[21] S. Woo et al., "DropBP: Accelerating fine-tuning of large language models by dropping backward propagation," 2024, *arXiv:2402.17812*.

[22] Y. Sun, Y. Xie, B. Ding, Y. Li, and J. Zhang, "Exploring selective layer fine-tuning in federated learning," 2024, *arXiv:2408.15600*.

[23] Y. J. Cho, L. Liu, Z. Xu, A. Fahrezi, and G. Joshi, "Heterogeneous loRA for federated fine-tuning of on-device foundation models," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2024, pp. 12903–12913.

[24] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, pp. 400–407, 1951.

[25] X. Wu, X. Liu, J. Niu, H. Wang, S. Tang, and G. Zhu, "FedloRA: When personalized federated learning meets low-rank adaptation," 2024. [Online]. Available: https://openreview.net/forum

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

[27] P. Tillet, H.-T. Kung, and D. Cox, "Triton: An intermediate language and compiler for tiled neural network computations," in *Proc. 3rd ACM SIGPLAN Int. Workshop Mach.e Learn. Programm. Lang.*, 2019, pp. 10–19.

[28] H. Xi, Y. Chen, K. Zhao, K. J. Teh, J. Chen, and J. Zhu, "Jetfire: Efficient and accurate transformer pretraining with INT8 data flow and per-block quantization," 2024, *arXiv:2403.12422*.

[29] Y. Xu, Y. Liao, H. Xu, Z. Ma, L. Wang, and J. Liu, "Adaptive control of local updating and model compression for efficient federated learning," *IEEE Trans. Mobile Comput.*, vol. 22, no. 10, pp. 5675–5689, Oct. 2022.

[30] H. He, P. Ye, Y. Ren, Y. Yuan, and L. Chen, "GoRA: Gradient-driven adaptive low rank adaptation," 2025, *arXiv:2502.12171*.

[31] Hugging Face, "Transformers library," 2024, Accessed: Apr. 17, 2025. [Online]. Available: https://github.com/huggingface/transformers,

[32] A. Ardakani et al., "SlimFit: Memory-efficient fine-tuning of transformer-based models using training dynamics," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics - Hum. Lang. Technol.*, 2024, pp. 6218–6236.

[33] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.

[34] J. Liu et al., "Accelerating decentralized federated learning with probabilistic communication in heterogeneous edge computing," *IEEE Trans. Netw.*, early access, Aug. 25, 2025, doi: 10.1109/TON.2025.3600015.

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

[36] P. He, J. Gao, and W. Chen, "DeBERTav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing," 2021, *arXiv:2111.09543*.

[37] Hugging Face, "Pre-trained weights for LLMs," 2025, Accessed: Mar. 30, 2025. [Online]. Available: https://huggingface.co/models

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

[39] J. Liu et al., "Enhancing semi-supervised federated learning with progressive training in heterogeneous edge computing," *IEEE Trans. Mobile Comput.*, vol. 24, no. 3, pp. 2315–2330, Mar. 2025.
[40] Y. J. Cho, L. Liu, Z. Xu, A. Fahrezi, and G. Joshi, "Heterogeneous lora for federated fine-tuning of on-device foundation models," 2024, *arXiv:2401.06432*.
[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[42] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 10088–10115.
[43] K. Le, T. Tran, T. Hua, and N. V. Chawla, "FLAME: Towards federated fine-tuning large language models through adaptive smoe," 2025, *arXiv:2506.16600*.
[44] S. Ghiasvand, M. Alizadeh, and R. Pedarsani, "Decentralized low-rank fine-tuning of large language models," 2025, *arXiv:2501.15361*.
[45] S. Wang, J. Liu, H. Xu, J. Yan, and X. Gao, "Efficient federated fine-tuning of large language models with layer dropout," 2025, *arXiv:2503.10217*.
[46] S. Zhan, L. Li, and C. Xu, "{AssyLLM }: Efficient federated fine-tuning of { LLMs} via assembling pre-trained blocks," in *Proc. USENIX Annu. Tech. Conf.*, 2025, pp. 1677–1691.
[47] H. Liu et al., "EcoLoRA: Communication-efficient federated fine-tuning of large language models," 2025, *arXiv:2506.02001*.
[48] Z. You et al., "PriFFT: Privacy-preserving federated fine-tuning of large language models via hybrid secret sharing," 2025, *arXiv:2503.03146*.

Jianchun Liu (Member, IEEE) received the PhD degree from the School of Data Science, University of Science and Technology of China, in 2022. He is currently an associate researcher with the School of Computer Science and Technology, University of Science and Technology of China. His main research interests include software defined networks, network function virtualization, edge computing, and federated learning.

Rukuo Li received the BS degree from Dalian Maritime University, in 2023. He is currently working toward the master's degree with the School of Computer Science, University of Science and Technology of China (USTC). His main research interests include edge computing, deep learning, and federated learning.

Hongli Xu (Member, IEEE) received the BS degree in computer science from the University of Science and Technology of China (USTC), China, in 2002, and the PhD degree in computer software and theory from the USTC in 2007. He is a professor with the School of Computer Science and Technology, USTC. He was awarded the Outstanding Youth Science Foundation of NSFC, in 2018. He has won the best paper award or the best paper candidate in several famous conferences. He has published more than 100 papers in famous journals and conferences, including *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Parallel and Distributed Systems*, *Infocom* and ICNP, etc. He has also held more than 30 patents. His main research interests include software defined networks, edge computing, and Internet of Thing.

Qianpiao Ma received the BS degree in computer science and technology and the PhD degree in computer software and theory from the University of Science and Technology of China, Hefei, China, in 2014 and 2022, respectively. He is currently an associate professor with the School of Computer Science and Engineering, Nanjing University ofScience and Technology, Nanjing, China. His primary research interests include federated learning, edge computing, and distributed machine learning.

Jiaming Yan received the BS degree from the Hefei University of Technology, in 2021. He currently working toward the doctor's degree with the School of Computer Science, University of Science and Technology of China (USTC). His main research interests include edge computing, deep learning, and federated learning.

Liusheng Huang (Member, IEEE) received the MS degree in computer science from the University of Science and Technology of China, in 1988. He is currently a senior professor and a PhD Supervisor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored or co-authored six books and more than 300 journal/conference papers. His research interests include the areas of the Internet of Things, vehicular Ad-Hoc networks, information security, and distributed computing.