FLoRA: Federated Fine-Tuning Large Language Models with Heterogeneous Low-Rank Adaptations

Anonymous ACL submission

Abstract

The rapid development of Large Language Models (LLMs) has been pivotal in advancing AI, with pre-trained LLMs being adaptable to diverse downstream tasks through fine-tuning. Federated learning (FL) further enhances fine-tuning in a privacy-aware manner by utilizing clients' local data through in-situ computation, eliminating the need for data movement. However, fine-tuning LLMs, given their massive scale of parameters, poses challenges for clients with constrained and heterogeneous resources in FL. Previous methods employed low-rank adaptation (LoRA) for efficient federated fine-tuning but utilized traditional FL aggregation strategies on LoRA adapters. This approach led to mathematically inaccurate *aggregation noise*, reducing fine-tuning effectiveness and failing to address heterogeneous LoRAs. In this work, we first highlight the mathematical incorrectness of LoRA aggregation in existing federated fine-tuning methods. We introduce a new approach called FLoRA that enables federated fine-tuning on heterogeneous LoRA adapters across clients through a novel stacking-based aggregation method. Our approach is noise-free and seamlessly supports heterogeneous LoRAs. Extensive experiments demonstrate FLoRA's superior performance in both homogeneous and heterogeneous settings, surpassing state-of-the-art methods. We envision this work as a milestone for efficient, privacy-preserving, and accurate federated fine-tuning of LLMs.

1 Introduction

The Large Language Models (LLMs) have shown remarkable performance on various tasks, such as chatbots (Bill and Eriksson, 2023), virtual assistants (Dong et al., 2023), search engines (Kelly et al., 2023), and healthcare (Thirunavukarasu et al., 2023; Singhal et al., 2023). However, adapting pre-trained LLMs (*e.g.,* Llama 2 (Touvron et al., 2023b)) to downstream tasks requires tremendous computation resources to fine-tune all the model parameters. To mitigate this issue, a variety of parameter-efficient fine-tuning (PEFT) methods have been proposed. One of the most widely used PEFT methods is low-rank adaptation (LoRA) (Hu et al., 2021). As shown in the top of Figure 1, LoRA adds a parallel branch of trainable adapters $\mathbf{A}$ and $\mathbf{B}$ to compute the model update $\mathbf{\Delta W}$, where the ranks of $\mathbf{A}$ and $\mathbf{B}$ are much smaller than the pre-trained model parameter $\mathbf{W}$. When applying LoRA for fine-tuning, only $\mathbf{A}$ and $\mathbf{B}$ are updated while the entire $\mathbf{W}$ is frozen, thereby significantly reducing the GPU memory consumption.

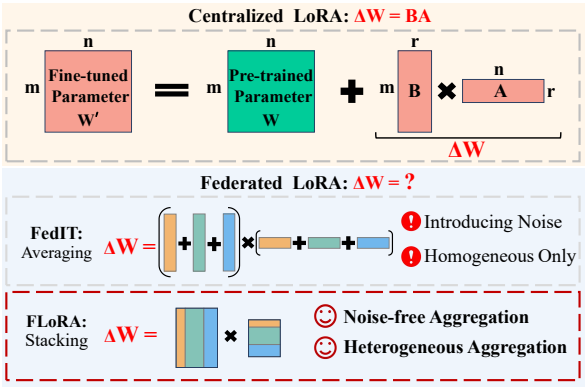


Figure 1: The overview of LoRA, FedIT and our FLoRA. The top row shows how LoRA updates the model in centralized fine-tuning. The middle and bottom rows show the global model updating strategies in FedIT and our FLoRA respectively.

Fine-tuning Large Language Models (LLMs) requires ample data for adaptation to specific downstream tasks. Often, this data is dispersed across a multitude of devices, harboring privacy concerns. For instance, aggregating medical data from hospitals for centralized LLM fine-tuning poses significant challenges. Consequently, to facilitate fine-tuning without compromising private data, federated learning (FL) becomes essential, enabling LLM fine-tuning across distributed clients while preserving data privacy (McMahan et al., 2017; Zhang et al., 2021). In this work, we focus on federated fine-tuning, enabling distributed clients

to collaboratively fine-tune LLMs for adaption to downstream tasks while preserving data privacy.

Prior work, FedIT, proposed a federated fine-tuning method (Zhang et al., 2023a), integrating LoRA with FedAvg (McMahan et al., 2017). In each FL round of FedIT, clients fine-tune LoRA modules using their local data and then send the fine-tuned modules to the server. The server averages all the local LoRA modules to obtain a global LoRA. Since only the weights of the LoRA modules are fine-tuned and communicated, FedIT effectively reduces both computation and communication costs. However, FedIT faces two key issues. First, **the naive averaging of local LoRA modules in FedIT introduces noise to the global model update.** Specifically, FedIT averages local $\mathbf{A}$ and $\mathbf{B}$ independently, which introduces mathematical errors to the global LoRA. In short,

$$\underbrace{\sum \mathbf{A} \times \sum \mathbf{B}}_{\text{FedIT}} \neq \underbrace{\sum \mathbf{A} \times \mathbf{B}}_{\text{mathematically correct}} .$$

We will elaborate on this issue in Section 2 with theoretical analysis. Such an inaccurate aggregation will hinder convergence, leading to higher fine-tuning costs. Second, due to the heterogeneous data distribution (Zhao et al., 2018; Li et al., 2019) and heterogeneous hardware resources, clients need to adapt LoRA ranks (Zhang et al., 2023b) according to the system and data heterogeneity. However, **FedIT cannot aggregate local LoRAs with heterogeneous ranks.**

In this work, we present FLoRA, an aggregation-noise-free federated fine-tuning method that supports heterogeneous LoRAs. Specifically, as shown in Figure 2, we propose to **stack** the local LoRA modules $\mathbf{A}_k$ and $\mathbf{B}_k$ separately to construct the global LoRA modules $\mathbf{A}$ and $\mathbf{B}$, where $\mathbf{A}_k$ and $\mathbf{B}_k$ denote the corresponding LoRA modules on the k-th client. This stacking method is theoretically proven to be accurate for the aggregation of local LoRA modules (Section 3.1). Additionally, it can naturally accommodate heterogeneous LoRA settings (Section 3.2), since stacking does not require the local LoRA modules to have identical ranks across clients. The noise-free aggregation of FLoRA accelerates convergence, which, in turn, improves the overall computation and communication efficiency of federated fine-tuning. Furthermore, FLoRA effectively caters to heterogeneous data and computational resources across clients, where heterogeneous ranks are applied. The noise-free aggregation of FLoRA accelerates convergence, which will in turn improve the overall computation and communication efficiency of federated fine-tuning. Furthermore, FLoRA can effectively cater to heterogeneous data and computational resources across clients, where heterogeneous ranks are applied. Our key contributions are summarized as follows:

- We propose FLoRA, a federated fine-tuning algorithm based on LoRA that can perform noise-free aggregation of local LoRA modules. Theoretical analysis shows that FLoRA eliminates the meaningless intermediate term in the global model update, leading to faster convergence and improved performance.
- The proposed stacking mechanism for aggregating LoRA modules supports heterogeneous LoRA ranks across clients, accommodating data and system heterogeneity in realistic settings. This encourages the broader participation of clients with heterogeneous data and resources in federated fine-tuning.
- We use FLoRA to fine-tune LLaMA, Llama2 (Touvron et al., 2023a) and TinyLlama (Zhang et al., 2024) on four benchmarks for two downstream tasks. Results show that FLoRA surpasses SoTA methods for both homogeneous and heterogeneous settings.

2 Preliminaries

Fine-tuning LLMs with LoRA. LoRA (Hu et al., 2021) uses two decomposed low-rank matrices to represent the update of the target module:

$$\mathbf{W}' = \mathbf{W} + \mathbf{\Delta W} = \mathbf{W} + \mathbf{BA}, \qquad (1)$$

where $\mathbf{W} \in \mathbb{R}^{m\times n}$ and $\mathbf{W}' \in \mathbb{R}^{m\times n}$ denote the pre-trained and fine-tuned parameters of target modules (*e.g.,* attention modules), respectively. $\mathbf{A}$ and $\mathbf{B}$ are low-rank decomposition of $\mathbf{\Delta W}$. where $\mathbf{A} \in \mathbb{R}^{r\times n}, \mathbf{B} \in \mathbb{R}^{m\times r}$, such that $\mathbf{\Delta W} = \mathbf{BA}$ with the identical dimensions as $\mathbf{W}$ and $\mathbf{W}'$. The rank of LoRA, denoted by r, is typically significantly smaller than m and n, leading to dramatic parameter reduction of $\mathbf{\Delta W}$. During the fine-tuning phase, LoRA optimizes matrices $\mathbf{A}$ and $\mathbf{B}$ instead of directly updating $\mathbf{W}$, thus achieving substantial savings in GPU memory usage. For example, in the context of the Llama-7b model (Touvron et al., 2023a), the original dimension of attention modules is 4096×4096 (i.e., $\mathbf{W} \in \mathbb{R}^{4096\times4096}$), setting

the LoRA rank to 16 reduces the decomposed matrices to $\mathbf{A} \in \mathbb{R}^{16\times4096}$ and $\mathbf{B} \in \mathbb{R}^{4096\times16}$. This approach decreases the number of trainable parameters to merely 0.78% of the entire parameter space of the pre-trained model, offering a significant efficiency boost in fine-tuning.

FedIT: Averaging Homogeneous LoRA. The most widely used FL algorithm, i.e., FedAvg (McMahan et al., 2017), aggregates all the local model updates by weighted averaging to update the global model in each communication round:

$$\mathbf{W}' = \mathbf{W} + \sum_{k=1}^{K} p_k \mathbf{\Delta W}_k = \mathbf{W} + \mathbf{\Delta W} \tag{2}$$

where $\mathbf{W}'$ and $\mathbf{W}$ denote the global model parameters before and after a communication round. $\mathbf{\Delta W}_k$ represents the local model update from the k-th client, with p_k being the corresponding scaling factor that is typically weighted by the local data size, and $\mathbf{\Delta W}$ represents the global model update.

FedIT (Zhang et al., 2023a) directly integrates FedAvg with LoRA to enable federated fine-tuning, where each client fine-tunes LoRA modules with the homogeneous rank. Specifically, the clients download the pre-trained LLM from the server. Then, the clients locally initialize and fine-tune the LoRA modules. After the local fine-tuning, the updated LoRA modules are sent to the server. The server finally updates the global LoRA modules $\mathbf{A}$ and $\mathbf{B}$ by independently applying the weighted averaging across all local modules $\mathbf{A}_k$ and $\mathbf{B}_k$:

$$\mathbf{A} = \sum_{k=1}^{K} p_k \mathbf{A}_k, \quad \mathbf{B} = \sum_{i=0}^{K} p_k \mathbf{B}_k. \tag{3}$$

This aggregation of FedIT is almost the same as FedAvg except that only the LoRA modules are trained and communicated. However, such a naive aggregation mechanism introduces additional problems for federated fine-tuning. First, each single module $\mathbf{A}$ or $\mathbf{B}$ is not the model update, and only $\mathbf{BA}$ represents the model update. Thus, averaging $\mathbf{A}_k$ and $\mathbf{B}_k$ *independently* to compute the aggregated gradients will introduce noises to the global model update. Here we use a simple example to explain how the noise is generated, and we assume that two clients are applying FedIT to perform federated fine-tuning. In a communication round, the two clients train $\mathbf{A}_0$, $\mathbf{B}_0$ and $\mathbf{A}_1$, $\mathbf{B}_1$ respectively. The local model updates $\mathbf{\Delta W}_0$ and $\mathbf{\Delta W}_1$ are the product of corresponding LoRA modules:

$$\mathbf{\Delta W}_k = \mathbf{B}_k\mathbf{A}_k, k \in \{0, 1\}. \tag{4}$$

According to Equation 2, the expected global model update $\mathbf{\Delta W}$ can be obtained by weighted averaging $\mathbf{\Delta W}_0$ and $\mathbf{\Delta W}_1$:

$$\begin{aligned}\mathbf{\Delta W} &= p_0\mathbf{\Delta W}_0 + p_1\mathbf{\Delta W}_1 \\ &= p_0\mathbf{B}_0\mathbf{A}_0 + p_1\mathbf{B}_1\mathbf{A}_1.\end{aligned} \tag{5}$$

However, according to Equation 3, FedIT aggregates $\mathbf{A}$ and $\mathbf{B}$ independently:

$$\begin{aligned}\mathbf{\Delta W} &= \mathbf{BA} = (p_0\mathbf{B}_0 + p_1\mathbf{B}_1)(p_0\mathbf{A}_0 + p_1\mathbf{A}_1) \\ &= p_0^2\mathbf{B}_0\mathbf{A}_0 + p_1^2\mathbf{B}_1\mathbf{A}_1 + \underline{p_0p_1(\mathbf{B}_0\mathbf{A}_1 + \mathbf{B}_1\mathbf{A}_0)}.\end{aligned} \tag{6}$$

The global model update in Equation 6 is different from the expected one in Equation 5, mainly due to the underlined intermediate term that is obtained by the cross-production of LoRA modules from different clients. This intermediate-term is unexpected noise in the model aggregation. With the number of clients increasing, this noisy term will become much larger than the real global updates, significantly slowing down the fine-tuning progress. In addition, FedIT applies the scaling factor p_k to both $\mathbf{A}_k$ and $\mathbf{B}_k$, resulting in a p_k^2 coefficient for the local model update $\mathbf{\Delta W_k}$, exacerbating the error of LoRA aggregation. As Figure 2 illustrates, the averaging algorithm in FedIT is an inaccurate aggregation method, leading to slower convergence and more computation cost.

The other deficiency of FedIT is that it cannot support aggregation on heterogeneous LoRA modules. The local data in FL may exhibit significant heterogeneity across clients (Zhao et al., 2018; Li et al., 2019). If a client configures a higher rank than the actual one required by the local data complexity, this may result in overfitting. Conversely, if the rank is too small, it may lack the necessary generalization capacity to effectively learn from the local dataset (Figure 4). Moreover, the heterogeneous computational resource across clients also requires heterogeneous rank deployment, *e.g.,* clients with smaller memory can only afford to train LoRA modules with smaller ranks. AdaLoRA (Zhang et al., 2023b) has been proposed to adapt LoRA ranks based on available computation resources. Therefore, deploying heterogeneous ranks across clients is a pressing requirement for accommodation to data and system heterogeneity. However, according to Equation 3, FedIT is only able to aggregate LoRA modules with the homogeneous rank.

3 Proposed Method: FLoRA

3.1 Stacking-based Noise-free Aggregation

Motivated by the aforementioned problem, we propose a novel aggregation mechanism that accu-

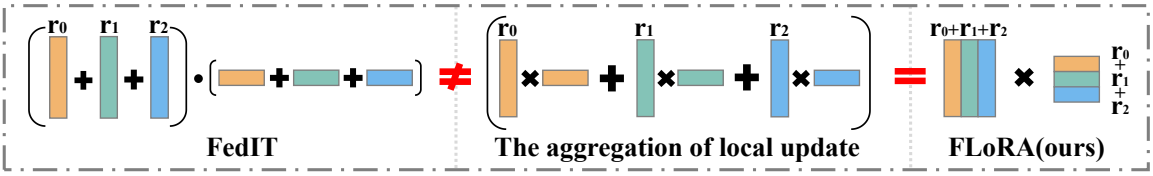


Figure 2: Module stacking in FLoRA is a noise-free aggregation for LoRA, while the module averaging in FedIT cannot accurately aggregate the local updates.

rately compute global model update $\mathbf{\Delta W}$ by aggregating local LoRA modules and effectively support the heterogeneous LoRA. According to matrix multiplication principles and the model update rule in LoRA (*i.e.,* Equation 1), the element at position (x, y) of the model update $\mathbf{\Delta W}$ is computed as the sum of the products of corresponding elements from the x-th column of $\mathbf{B}$ and the y-th row of $\mathbf{A}$:

$$\delta_{xy} = \sum_{i=0}^{r} a_{yi} b_{xi}, \tag{7}$$

where δ_{xy} represents the element at position (x, y) in $\mathbf{\Delta W}$. a_{yi}, b_{xi} are the elements at positions (y, i) and (x, i) in $\mathbf{A}$ and $\mathbf{B}$, respectively. According to Equation 7, the model update in LoRA can be expressed as the sum of the products of the corresponding rows of $\mathbf{A}$ and the columns of $\mathbf{B}$.

To illustrate this concept further, let us consider a simplified example where the dimensions of LoRA modules are given by $\mathbf{A} \in \mathbb{R}^{2\times 3}$ and $\mathbf{B} \in \mathbb{R}^{3\times 2}$. As described in Equation 8, $\mathbf{A}$ and $\mathbf{B}$ can be decomposed to two sub-matrices with rank $r = 1$, and the product of $\mathbf{A}$ and $\mathbf{B}$ then are computed as the sum of the products of two respective sub-matrices:

$$\begin{aligned}\mathbf{BA} &= \begin{bmatrix} b_{00}, b_{01} \\ b_{10}, b_{11} \\ b_{20}, b_{21} \end{bmatrix} \cdot \begin{bmatrix} a_{00}, a_{10}, a_{20} \\ a_{01}, a_{11}, a_{21} \end{bmatrix} \\ &= \begin{bmatrix} b_{00} \\ b_{10} \\ b_{20} \end{bmatrix} \cdot [a_{00}, a_{10}, a_{20}] + \begin{bmatrix} b_{01} \\ b_{11} \\ b_{21} \end{bmatrix} \cdot [a_{01}, a_{11}, a_{21}.]\end{aligned} \tag{8}$$

To address the aggregation challenge from an alternative perspective, let us consider the scenario where we have multiple pairs of LoRA modules, $\mathbf{A}_k$, $\mathbf{B}_k$, optimized by the clients. Each pair satisfies the dimensions $\mathbf{A}_k \in \mathbb{R}^{r_k \times n}$ and $\mathbf{B}_k \in \mathbb{R}^{m \times r_k}$. Similar to Equation 8, the sum of the products of these module pairs is the product of the stacked modules, *i.e.,* $\sum_{k=1}^{K} \mathbf{B}_k \mathbf{A}_k = \mathbf{BA}$, where $\mathbf{B}$ represents the *stacking* of all $\mathbf{B}_k$ modules aligned through dimension m and $\mathbf{A}$ is the *stacking* of all $\mathbf{A}_k$ aligned through dimension n. Figure 2 visually illustrates this concept, where the orange, green, and blue rectangles symbolize $\mathbf{A}_k$, $\mathbf{B}_k$, and their respective products. The aggregation of three products mirrors the product of the stacked $\mathbf{B}$ and $\mathbf{A}$ from all $\mathbf{B}_k$ and $\mathbf{A}_k$ pairs trained by clients. This mechanism demonstrates that, in the context of federated fine-tuning, we can achieve a noise-free aggregation of local updates by simply stacking the local LoRA modules. This process also circumvents the need for transmitting the full model parameters, thus reducing communication costs.

To facilitate our discussion, we introduce the stacking operation symbolized by "$\oplus$" to denote the module aggregation as depicted in Figure 2. This operation is mathematically defined as:

$$\begin{aligned}\mathbf{A} =& \mathbf{A}_0 \oplus \mathbf{A}_1 \oplus \mathbf{A}_2,\ \mathbf{B} = \mathbf{B}_0 \oplus \mathbf{B}_1 \oplus \mathbf{B}_2, \\ & \mathbf{A}_k \in \mathbb{R}^{r_k \times n}, \mathbf{A} \in \mathbb{R}^{(r_0+r_1+r_2)\times n}, \\ & \mathbf{B}_k \in \mathbb{R}^{m \times r_k}, \mathbf{B} \in \mathbb{R}^{m \times (r_0+r_1+r_2)}.\end{aligned} \tag{9}$$

In Equation 9, "$\oplus$" indicates that for $\mathbf{A}$, each subsequent module is vertically stacked below the preceding one, whereas for $\mathbf{B}$, each module is horizontally stacked to the right of the one before it.

We can now formalize our conclusion regarding the aggregation of LoRA modules. The sum of the products of K LoRA module pairs is equivalent to the product of their stacked matrices:

$$\sum_{k=0}^{K} \mathbf{B}_k \mathbf{A}_k = (\mathbf{B}_0 \oplus ... \oplus \mathbf{B}_K)(\mathbf{A}_0 \oplus ... \oplus \mathbf{A}_K) \tag{10}$$

This foundational principle will guide the design of FLoRA, as it allows for the efficient and effective aggregation of local updates without the transmission of entire model parameters.

3.2 FLoRA: Stacking-based Federated Fine-tuning for Heterogeneous LoRA

The stacking-based aggregation facilitates not only the accurate aggregation of LoRA modules but also inherently supports the heterogeneous LoRA ranks. This approach imposes no constraints on the ranks of each local LoRA module as long as each client fine-tunes the same pre-trained model, *i.e.,* they share the same dimension m and n.

By employing the stacking-based aggregation mechanism, we introduce FLoRA, an approach designed to facilitate federated fine-tuning of LLMs with heterogeneous LoRA. Let us use a concrete example to illustrate the key steps of applying

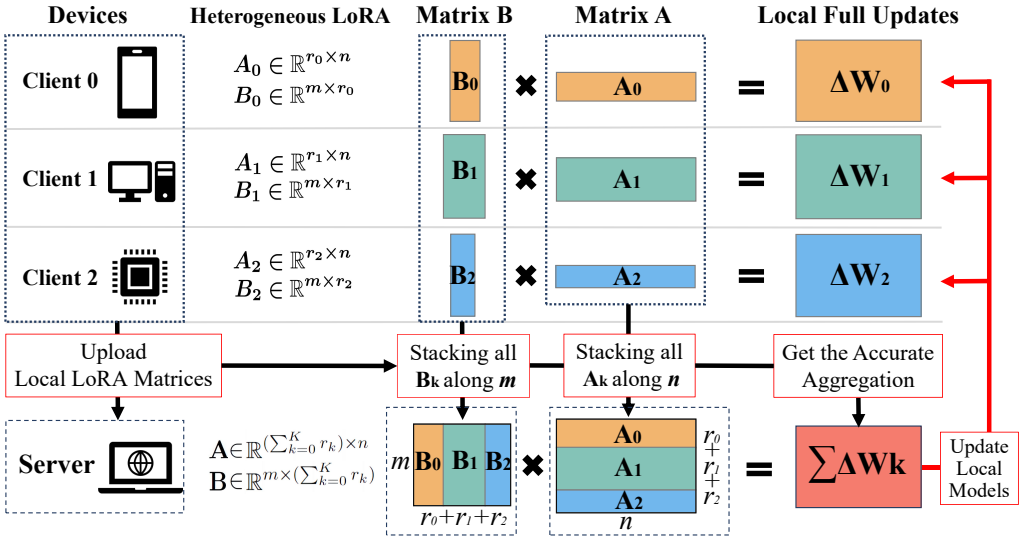


Figure 3: FLoRA workflow. The local LoRA modules are initialized and optimized each round, and stacked by the server to obtain the global LoRA modules. The global modules are then sent back to clients to update local models.

FLoRA, where K heterogeneous clients are involved in fine-tuning an LLM, and the pre-trained parameters are denoted by $\mathbf{W}$.

Initialization. The server first disseminates the pre-trained model parameters $\mathbf{W}$ to all K clients. Then, the clients initialize their local LoRA modules based on the complexity of local data and available local resources. The adaptation of LoRA ranks is beyond the scope of this paper, but existing work like AdaLoRA (Zhang et al., 2023b) can facilitate the rank adjustment.

Local Fine-tuning. Following initialization, the clients train their local LoRA modules with the local data for several iterations. Then, the clients send the local LoRA modules back to the server. Note that the clients initialize local LoRA modules *each* round before local fine-tuning.

Stacking-based LoRA Aggregation. Upon receiving the heterogeneous LoRA modules from participating clients, the server proceeds to aggregate them by stacking all $\mathbf{B}_k$ and $\mathbf{A}_k$ according to Equation 10, resulting in the global $\mathbf{A} \in \mathbb{R}^{(\sum_{k=0}^{K} r_k) \times n}$ and $\mathbf{B} \in \mathbb{R}^{m \times (\sum_{k=0}^{K} r_k)}$. The aggregation process of FLoRA can be described as follows:

$$\begin{aligned} &\mathbf{A} = p_0 \mathbf{A}_0 \oplus p_1 \mathbf{A}_1 \oplus \ldots \oplus p_K \mathbf{A}_K \\ &\mathbf{B} = \mathbf{B}_0 \oplus \mathbf{B}_1 \oplus \mathbf{B}_2 \oplus \ldots \oplus \mathbf{B}_K \\ &\mathbf{A}_k \in \mathbb{R}^{r_k \times n}, \mathbf{B}_k \in \mathbb{R}^{m \times r_k} \\ &\mathbf{A} \in \mathbb{R}^{(\sum_{k=0}^{K} r_k) \times n}, \mathbf{B} \in \mathbb{R}^{m \times (\sum_{k=0}^{K} r_k)}, \end{aligned} \tag{11}$$

where p_k represents the scaling factor for each local update, determined by the relative size of the local data to the global data:

$$p_k = \frac{len(D_k)}{len(\sum_{k=0}^{K} D_k)}. \tag{12}$$

Note that the scaling factor p_k should be only applied to one of $\mathbf{A}_k$ and $\mathbf{B}_k$ to avoid squaring the factor in the final model update $\mathbf{BA}$. This method ensures a noise-free aggregation mechanism as described in Equation 10.

Update Local Models. After each round of noise-free aggregation, the server redistributes the updated global LoRA modules $\mathbf{A}$ and $\mathbf{B}$ back to the clients. The clients then proceed to update the local models using $\mathbf{BA}$ and continue the fine-tuning. Using the stacking approach, the dimensions of updated global LoRA modules $\mathbf{A}$ and $\mathbf{B}$ are larger than those of FedIT, potentially leading to larger communication overhead in each round. However, empirical observations indicate that federated fine-tuning typically requires only a limited number of communication rounds to achieve satisfactory results, as detailed in Section 4. In addition, it is important to note that the LoRA modules $\mathbf{A}$ and $\mathbf{B}$ constitute a small fraction of the overall size of the pre-trained model, which is distributed to clients during the initialization phase. Thus, the additional communication overhead of the stacking approach is negligible and does not significantly impact the efficiency of federated fine-tuning.

4 Experiments

The key features of FLoRA are (i) noise-free aggregation and (ii) support for heterogeneous LoRA modules. In this section, we verify these key features across various LLM fine-tuning tasks. We first study the performance of FLoRA and compare it against FedIT under homogeneous settings to demonstrate the advantages of noise-free aggregation (Zhang et al., 2023a). Then, we examine

performance in a synthetic heterogeneous setup and compare FLoRAwith a vanilla *zero-padding* method. Finally, we conduct ablation studies on the scaling factor, the heterogeneity of LoRA ranks, and the extra communication overhead of FLoRA.

4.1 Experiment Setup

Models, Datasets and Experiment Settings. We employ three Llama-based models with different scales in our experiments: TinyLlama with 1.1 billion parameters (Zhang et al., 2024), and the 7 billion parameter versions of Llama (Touvron et al., 2023a) and Llama2 (Touvron et al., 2023b), evaluating FLoRA across different model capacities. Following the configurations in the original LoRA paper (Hu et al., 2021), the LoRA modules are applied to the self-attention layers only.

We use the Databricks-dolly-15k (Zhang et al., 2023a) instruction dataset, Alpaca dataset (Taori et al., 2023), and Wizard dataset (Luo et al., 2023) for the question-answering (QA) task, and Wizard and ShareGPT for the chat assistant task. We evaluate the federated fine-tuned models on MMLU (Hendrycks et al., 2020) for the QA task and MT-bench (Zheng et al., 2023) for the chat assistant task, respectively. We sample 10 clients uniformly at random following the non-IID setting in FedIT (Zhang et al., 2023a). The other experimental configurations are elaborated in Appendix A.

Baselines. We compare FLoRA with four baselines. (1) **FedIT:** It is the SOTA federated fine-tuning method (Zhang et al., 2023a) that integrates LoRA with FedAvg. We only apply FedIT to homogeneous LoRA experiments as it does not support heterogeneous LoRA. (2) **Zero-padding:** It is an approach that enables FedIT to support heterogeneous LoRA (Cho et al., 2023). It extends all the heterogeneous local ranks to the maximum rank among the clients and pads their remaining parts by 0. (3) **Centralized Fine-tuning:** we compare FLoRA with centralized LoRA fine-tuning with the same hyperparameters and configurations. (4) **Standalone:** the client fine-tunes the pre-trained model locally without federations.

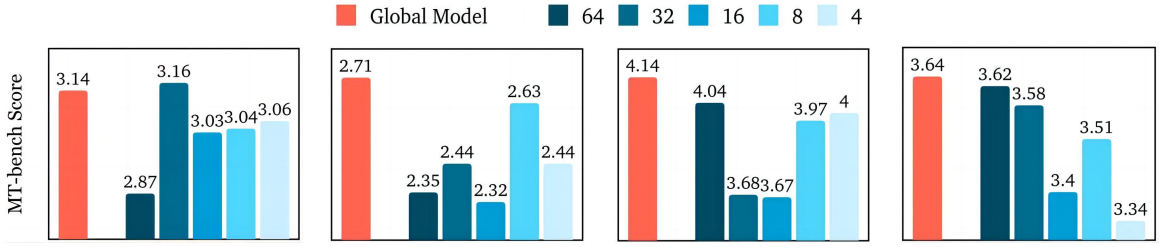
4.2 Experiment Results

Homogeneous LoRA. We first evaluate the performance of FLoRA with homogeneous LoRA. Specifically, all the clients share the identical LoRA rank of 16. As Table 1 depicts, FLoRA achieves consistently better performance than FedIT across all the evaluated models and tasks. This is evident in the MT-bench scores for both TinyLlama and Llama models, where FLoRA's performance exceeds that of FedIT by at least 0.2. A notable example is the MT-bench score for the Llama model fine-tuned with Wizard dataset, where FLoRA scores 4.21, surpassing FedIT's 3.07. On the MMLU test set, FLoRA outperforms FedIT in all the settings. For example, considering the TinyLlama model fine-tuned with Dolly, FLoRA nearly doubles the accuracy achieved by FedIT. While FedIT occasionally matches the performance of FLoRA, as observed with the Alpaca dataset on MMLU, the performance gap is marginal. Interestingly, in several scenarios, the performance of FLoRA not only outpaces FedIT but also exceeds the performance achieved by the centralized fine-tuning. This phenomenon, observed in the TinyLlama model fine-tuned with the Alpaca and Wizard datasets, suggests that the smaller data volume on clients for federated fine-tuning may help mitigate overfitting, thereby enhancing model generalization. The experiment results of the Llama2 model are presented in Appendix A, which reveal the same trend as that in TinyLlama and Llama. The consistent observations across the three models demonstrate that FLoRA consistently outperforms FedIT in the homogeneous LoRA setting.

Heterogeneous LoRA. Compared with FedIT, a distinctive strength of FLoRA lies in its inherent capability to accommodate heterogeneous LoRA configurations. In the heterogeneous LoRA settings, we apply varied local LoRA ranks, *i.e.,* [64, 32, 16, 16, 8, 8, 4, 4, 4, 4], to 10 clients, simulating a realistic scenario where clients have heterogeneous computational resources. As Table 1 and Table 4 illustrate, FLoRA not only adapts to heterogeneous ranks without performance degradation but also maintains consistency with the results observed in most homogeneous settings. This contrasts sharply with the performance of FedIT, where the application of zero-padding significantly degrades its performance on MMLU and MT-bench. It reveals that zero-padding exacerbates FedIT's inherent noise issues in the aggregation process, posing significant challenges in managing fine-tuning performance. For example, by applying the zero-padding method, the MMLU accuracy of Llama model fine-tuned with Alpaca dataset dramatically drops to 7.97%. The results demonstrate that FLoRA not only accommodates heterogeneous LoRA ranks effectively but also sustains robust training performance compared to baseline

Table 1: Comparison of FLoRA with baselines on MMLU and MT-bench. "Homo" represents the settings with homogeneous LoRA ranks, and "Heter" denote the settings with heterogeneous LoRA ranks.

Foundation model	Strategy	Fine-tuning algorithm	MMLU			MT-bench	
			Dolly	Alpaca	Wizard	Wizard	ShareGPT
TinyLlama	Centralized	LoRA	27.99	28.03	29.13	2.34	2.79
	Homo	FedIT	16.35	30.02	42.51	2.92	2.55
		FLoRA	**30.80**	**31.92**	**43.87**	**3.13**	**2.77**
	Heter	Zero-padding	15.76	29.56	40.79	1.56	1.29
		FLoRA	**18.45**	**29.69**	**41.48**	**3.14**	**2.71**
Llama	Centralized	LoRA	35.91	29.18	31.68	4.38	3.99
	Homo	FedIT	29.67	29.41	33.43	3.07	3.73
		FLoRA	**30.99**	**29.85**	**34.26**	**4.21**	**3.93**
	Heter	Zero-padding	26.46	7.97	26.98	3.51	3.26
		FLoRA	**28.50**	**29.54**	**27.91**	**4.14**	**3.64**



(a) TinyLlama - Wizard (b) TinyLlama - ShareGPT (c) Llama - Wizard (d) Llama - ShareGPT

Figure 4: The impact of heterogeneous LoRA ranks across clients. The red bars represent the global model performance and the blue bars represent the local model performance with varying LoRA ranks.

methods. It facilitates the participation of devices with varied computational capacities in heterogeneous federated fine-tuning tasks efficiently. Additionally, FLoRA can be seamlessly integrated with AdaLoRA (Zhang et al., 2023b), which dynamically adjusts the LoRA rank using on the clients, the results are presented in Appendix A.

The Impact of Scaling Factor. The scaling factor, denoted as p_k in Equation 12, playing a pivotal role in the efficacy of FL (Wang et al., 2023). To understand its impact on FLoRA, we conduct experiments investigating how varying scaling factors influence the performance of FLoRA. Given that the default scaling factor is set to 0.1 for all clients, assuming 10 clients with equal local dataset sizes as per Equation 12, we explored the effects of alternative scaling factors, namely 0.01, 0.05, and 0.2. The results are summarized in Figure 5. The results do not reveal a clear pattern or optimal scaling factor for federated fine-tuning across different settings. The efficacy of a specific scaling factor appears to be contingent upon the dataset, task, and model in use. For example, when fine-tuning TinyLlama on the Dolly dataset, a lower scaling factor of 0.01 yields the highest accuracy, significantly outperforming the 0.1 and 0.2 scaling factors. Conversely, the model fine-tuned on Wizard dataset demonstrates a preference for a higher scaling factor of 0.2, achieving the best performance, whereas the lowest scaling factor of 0.01 was the least effective. In the case of the Llama model, larger scaling factors consistently facilitated better fine-tuning performance. Applying FLoRA to Dolly and Alpaca shows the optimal performance with a scaling factor of 0.2. These observations suggest that the choice of an appropriate scaling factor is highly dependent on specific datasets and model characteristics, underscoring the necessity for a tailored approach in federated fine-tuning.

The Impact of Heterogeneous LoRA Ranks. Although the above results demonstrate FLoRA effectively enables the federated fine-tuning with heterogeneous LoRA, it is worth further investigating how the federated fine-tuning improves the local models with various ranks. Motivated by this, we evaluate MT-bench scores for local models with LoRA ranks of 64, 32, 16, 8, and 4, presenting the results in Figure 4. Global model scores are shown in red bars, while local models are in blue, with deeper shades indicating higher ranks. The results show that the global model outperforms all local models, except for a case with the TinyLlama model fine-tuned on the Wizard dataset, where the client with rank 32 slightly exceeds the global model. This demonstrates FLoRA's ability to synthesize knowledge from diverse clients effectively.

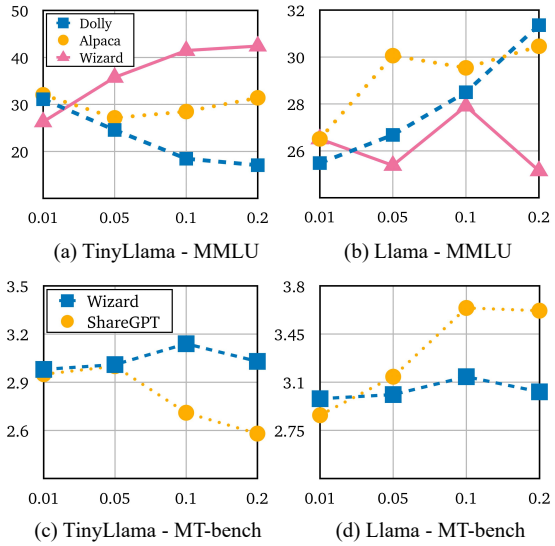


Figure 5: The impact of the scaling factor on FLoRA. The x-axis is the scaling factor, and the y-axis represents the MMLU accuracy for (a)-(b) and MT-bench score for (c)-(d). The results of Llama2 are in Appendix A.

Regarding the LoRA rank's impact, a rank of 8 consistently yields strong performance across various models and datasets. However, performance diverges at extreme ranks; for instance, the TinyLlama model fine-tuned on Wizzard with the LoRA rank of 64 underperforms the ones with smaller ranks, but the Llama model with the rank of 64 excels the counterparts with smaller ranks. This also demonstrates the heterogeneous rank deployment across clients is a realistic setting. These observations suggest a potential positive correlation between optimal LoRA rank and model capacity, motivating further exploration in future research.

Communication Efficiency. As discussed in Section 3, the server needs to send global LoRA modules to the clients in FLoRA, potentially raising concerns about increased communication overhead. To quantify this, we compare the communicated parameters of full fine-tuning, FedIT, and FLoRA over three communication rounds. As Figure 6 shows, although FLoRA transmits slightly more parameters than FedIT, it still significantly reduces the overhead of full fine-tuning. Despite the minor communication increase compared to FedIT, FLoRA enhances fine-tuning effectiveness and supports heterogeneous LoRA ranks, making it a preferable solution in federated fine-tuning.

5 Related Work

5.1 Parameter-efficient Fine-tuning of LLMs.

Parameter-efficient fine-tuning aims to reduce the number of trainable parameters. BitFit (Zaken et al., 2021) fine-tunes only the biases while

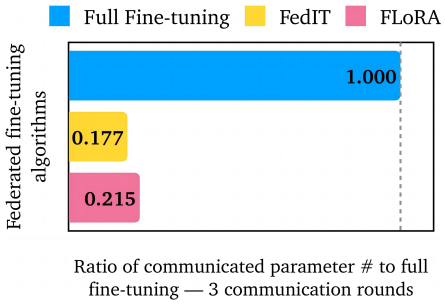


Figure 6: The ratio of communicated parameter numbers to full fine-tuning.

achieving similar accuracy with full fine-tuning. Houlsby et al. (2019) and Pfeiffer et al. (2020) apply transfer learning that adds pre-trained adapter layers between transformer blocks. LoRA (Hu et al., 2021) adopts the product of two low-rank matrices to represent the gradient in full fine-tuning, which achieves memory-efficient fine-tuning. AdaLoRA (Zhang et al., 2023b) optimizes LoRA by adaptively allocating the parameter budget, which enhances the flexibility of LoRA.

5.2 Federated Fine-tuning of LLMs.

Federated fine-tuning aims to extract knowledge from multiple on-device datasets while preserving data privacy. FedIT (Zhang et al., 2023a) leverages the FL framework for fine-tuning LLMs. It uses LoRA as the local fine-tuning strategy. However, concerns related to the deficiency in supporting heterogeneous LoRA limit its utilization. Cho et al. (2023) tries to solve this problem by zero-padding the local LoRA modules. However, this padding process causes additional computing overhead. Besides, it separately averages $\mathbf{A}$ and $\mathbf{B}$ modules, introducing noise to the global model.

6 Conclusion

In this work, we identified the limitations in current federated fine-tuning methods (*e.g.,* FedIT), and the challenges of applying federated fine-tuning in realistic settings, *i.e.,* the heterogeneous LoRA ranks across clients. To overcome these practical challenges and broaden the applicability of federated fine-tuning, we introduced FLoRA to enable the accurate aggregation on heterogeneous LoRA modules using the proposed stack-based LoRA aggregation mechanism. Our extensive experiments demonstrate that FLoRA outperforms the SOTA method in both homogeneous and heterogeneous LoRA settings. Moreover, our inspiring results provide valuable insights for future research in federated fine-tuning of large language models in a lightweight and accurate manner.

Broader Impacts and Ethics Statement

The objective of this paper is to improve the effectiveness of federated fine-tuning and does not involve sensitive information or ethical concerns related to AI and society. The experiments section of this paper uses publicly available LLMs and text datasets from the Internet with appropriate citations of their sources. The proposed algorithm can contribute to building up a privacy-preserving distributed fine-tuning framework. It encourages the research community to consider data privacy in fine-tuning LLMs with collected data.

Limitation

Our approach has the limitation that the server sends the stacked LoRA modules to the client, thereby increasing the communication costs. We discussed this limitation both theoretically and experimentally in Section 3 and Section 4, respectively. We believe that the increase in communication overhead is acceptable under the premise of improving fine-tuning effectiveness and accelerating convergence. In addition, due to constraints on computational resources and time, we only utilized Llama models in the experimental section. We aim to observe experimental phenomena of different types of LLM federated fine-tuning in future research and derive more general principles.

References

Desirée Bill and Theodor Eriksson. 2023. Fine-tuning a llm using reinforcement learning from human feedback for a therapy chatbot application.

Yae Jee Cho, Luyang Liu, Zheng Xu, Aldi Fahrezi, Matt Barnes, and Gauri Joshi. 2023. Heterogeneous lora for federated fine-tuning of on-device foundation models. In *International Workshop on Federated Learning in the Age of Foundation Models in Conjunction with NeurIPS 2023*.

Xin Luna Dong, Seungwhan Moon, Yifan Ethan Xu, Kshitiz Malik, and Zhou Yu. 2023. Towards next-generation intelligent assistants leveraging llm techniques. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5792–5793.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Dominique Kelly, Yimin Chen, Sarah E Cornwell, Nicole S Delellis, Alex Mayhew, Sodiq Onaolapo, and Victoria L Rubin. 2023. Bing chat: The future of search engines? *Proceedings of the Association for Information Science and Technology*, 60(1):1007–1009.

Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2019. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. 2023. Large language models encode clinical knowledge. *Nature*, 620(7972):172–180.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. 2023. Large language models in medicine. *Nature medicine*, 29(8):1930–1940.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ziyao Wang, Jianyu Wang, and Ang Li. 2023. Fedhyper: A universal and robust learning rate scheduler for federated learning with hypergradient descent. *arXiv preprint arXiv:2310.03156*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. 2021. A survey on federated learning. *Knowledge-Based Systems*, 216:106775.

Jianyi Zhang, Saeed Vahidian, Martin Kuo, Chunyuan Li, Ruiyi Zhang, Guoyin Wang, and Yiran Chen. 2023a. Towards building the federated gpt: Federated instruction tuning. *arXiv preprint arXiv:2305.05644*.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023b. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena.

A Additional Experiments and Setup Details

A.1 Datasets and Metric

Dolly dataset. The Dolly dataset is an open-source dataset with 15k text samples generated by Databricks employees. The topics include brainstorming, classification, closed QA, generation, information extraction, open QA, and summarization (Zhang et al., 2023a).

Alpaca dataset. The Alpaca dataset contains 52K instruction-following data used for fine-tuning the Alpaca model (Taori et al., 2023). This dataset is believed to be diverse enough for fine-tuning LLMs.

Wizard dataset. The Wizard dataset we use is the training data of the WizardLM model. It includes 70k pairs of instructions and outputs. The Wizard dataset generally features more complex instructions compared to the other datasets. Its fine-tuning results are typically better, which has been confirmed by our experiments, especially those evaluated by the MT-bench scores.

ShareGPT dataset. The ShareGPT dataset is a collection of approximately 52,000 conversations scraped via the ShareGPT API. The conversations in ShareGPT include both user prompts and responses from ChatGPT. In our experiments, we split the conversation dataset into question-answering pairs.

MMLU test set. The MMLU dataset is a widely used question-and-answer dataset in LLM fine-tuning. It has 14,024 questions in 57 different subjects, which can evaluate the logical reasoning capabilities of LLMs. We selected 1444 samples from the dataset for a quick and comprehensive evaluation.

MT-bench evaluation. MT-bench is a set of challenging multi-turn open-ended questions for evaluating chat assistants (Zheng et al., 2023). It evaluates the performance of LLMs by using the GPT-4 API to score the LLM-generated conversations. LLMs that behave more like GPT-4 will receive higher scores.

A.2 Hyperparameter Details

In all our experiments, the learning rate of fine-tuning is set to 0.0003; the batch size is 128 and the micro batch size is 16. Due to the large dataset and model sizes selected, federated fine-tuning consumes significant computational resources and time. Therefore, we opted for fewer fine-tuning rounds (even just one round) to ensure that we could observe enough data. Additionally, the MMLU dataset is prone to overfitting on these large datasets, resulting in a decrease in accuracy. Therefore, fewer training rounds ensure the effectiveness of the observed phenomena. Table 2 shows the fine-tuning rounds and local epochs we selected.

A.3 Supplementary Experiment Results

Integrating FLoRA with AdaLoRA All the observations about the impact of rank on the model performance, despite being influenced by data heterogeneity, still manage to reveal the importance of selecting an appropriate LoRA rank for a specific task. Thus, some algorithms such as

Table 2: The communication rounds and local epochs on each experiment setting. The Rounds column represents the number of communication rounds and the Epochs column represents the number of local fine-tuning epochs in each round.

Foundation	Datasets	Rounds	Epochs
TinyLlama	Dolly	3	1
	Alpaca	3	1
	Wizard	3	1
	ShareGPT	1	1
Llama	Dolly	3	3
	Alpaca	3	3
	Wizard	1	1
	ShareGPT	1	1
Llama2	Wizard	1	1
	ShareGPT	1	1

Table 3: The performance of FLoRA + AdaLoRA. AdaLoRA can reduce the rank while preserving the fine-tuning effectiveness.

Foundation model	Fine-tuning algorithm	Sum of local ranks	MT-bench score
TinyLlama	FLoRA	160	3.13
	FLoRA+AdaLoRA	120	3.14
Llama	FLoRA	160	4.21
	FLoRA+AdaLoRA	131	4.10
Llama2	FLoRA	160	4.17
	FLoRA+AdaLoRA	140	4.25

AdaLoRA (Zhang et al., 2023b) are designed to adaptively adjust the LoRA rank to optimize the model performance and save computational resources. With our support for heterogeneous LoRA, we can flexibly utilize AdaLoRA with adaptive LoRA ranks. We conducted corresponding experiments to demonstrate that we can use AdaLoRA to further improve the efficiency of federated fine-tuning. We implement AdaLoRA on each client to adjust LoRA modules during local fine-tuning. The results are shown in Table 3. The "Sum of local ranks" column means the sum of all local LoRA rank values *after* fine-tuning. Since our FLoRA does not adjust the rank, its value is 160, the same as the initial value. On the other hand, AdaLoRA dynamically adjusts the rank to maximize training effectiveness and minimize rank values to save resources. From Table Table 3, we can see that AdaLoRA on TinyLlama and Llama reduced the sum of local ranks to 120 and 131 from 160 respectively. We further conclude that FLoRA+AdaLoRA can further reduce the trainable parameter count while ensuring comparable or even improved performance compared to simply using LoRA on the clients. Our support for such rank adaptation further demonstrates the effectiveness and applicability of the FLoRA approach.

Table 4: Compare FLoRA with baselines in Llama2.

Strategy	Fine-tuning algorithm	Wizard	ShareGPT
Centralized	LoRA	4.24	3.99
Homo	FedIT	4.03	3.87
	FLoRA	4.22	3.96
Heter	Zero-padding	4.01	3.70
	FLoRA	4.17	3.91

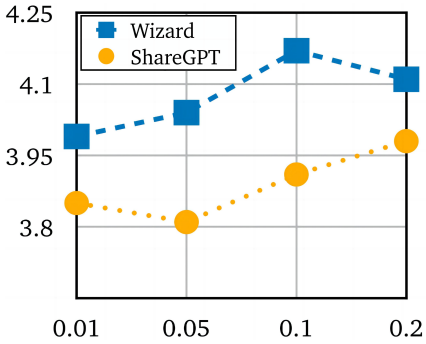


Figure 7: The impact of scaling factor on Llama2 model.

Experiment results of Llama2. Due to the inherently strong performance of Llama2, the improvement in the QA dataset is not significant. Therefore, we fine-tuned Llama2 using the Wizard and ShareGPT datasets. Overall, Llama2 exhibits similar experimental results to Tinyllama and Llama. Table 4 shows the comparison between FLoRA and our baselines. In the homogeneous and heterogeneous settings, the MT-bench scores of Wizard and ShareGPT all surpass those in FedIT and Zero-padding. As for the impact of scaling factors in Figure 7, Llama2 has a similar trend to the Llama-7b model, in which higher scaling factors exhibit better fine-tuning performance.

B Convergence Analysis

In this section, we demonstrate the convergence of FLoRA following the standard convergence analysis in Li et al. (2019). The FedAvg algorithm exhibits convergence to the global optimum at a rate of $O(1/T)$ for non-IID (independent and identically distributed) data under full client participation. This convergence is based on four assumptions mentioned in Li et al. (2019):

Assumption 1. Each local objective function is L- smooth, that is, for all x and y, $F_k(x) \leq F_k(y) + (x-y)^T \nabla F_k(y) + \frac{L}{2}\|x-y\|_2^2$

Assumption 2. Each local objective function is

μ - strongly convex that is, for all x and y, $F_k(x) \geq F_k(y) + (x-y)^T \nabla F_k(y) + \frac{\mu}{2}\|x-y\|_2^2$

Assumption 3. The variance of stochastic gradients in each client is bounded: $\mathbb{E}\|\nabla F_k(W_k^{(t)}, \xi_k^{(t)}) - \Delta W_k^{(t)}\|^2 \leq \sigma_k^2$ for $k = 1, ..., K$, where $\xi_k^{(t)}$ is the subset of training data randomly sampled from k-th client.

Assumption 4. The expected squared norm of stochastic gradients is uniformly bounded: $\mathbb{E}\|\nabla F_k(W_k^{(t)}, \xi_k^{(t)})\|^2 \leq G^2$ for all $k = 1, ..., K$ and $t = 1, ..., T$, where $\xi_k^{(t)}$ is the subset of training data randomly sampled from k-th client.

For the convergence analysis of FLoRA, we introduce an additional assumption 5 tailored to the specific dynamics of LoRA fine-tuning and its relation to traditional SGD-based full fine-tuning:

Assumption 5. (Unbiased LoRA Gradient). The updates applied to LoRA modules by each client serve as unbiased estimators of the gradient that would be directly computed on the base model through SGD: $\mathbf{B}_k^{(t+1)}\mathbf{A}_k^{(t+1)} - \mathbf{B}_k^{(t)}\mathbf{A}_k^{(t)} = \eta^{(t)} \nabla F_k(\mathbf{W}_k^{(t)} | \xi_k^{(t)})$. Note that we define the model parameter in t-th round by $\mathbf{W^{(t)}}$.

Theorem 1. Based on Assumptions 1-5, we choose $k = \frac{L}{\mu}$, $\gamma = \max\{8k, E\}$. The local learning rate $\frac{\alpha_k}{r_k} = \frac{2}{\mu(\gamma+t)}$. Then, we can deduce that the expectation of the fine-tuning error in FLoRA can be bounded by:

$$\delta^{(T)} \leq \frac{2k}{\gamma + T}\left(\frac{M}{\mu} + 2L\|\mathbf{W}^{(1)} - \mathbf{W}^*\|^2\right), \quad (13)$$

where $\delta^{(T)}$ is the fine-tuning error in T-th round. $\delta^{(T)}$ and M are defined as follows:

$$\begin{aligned} \delta^{(T)} &= \mathbb{E}[F(\mathbf{w}^{(\mathbf{T-1})} + \mathbf{B}^{(T)}\mathbf{A}^{(T)})] - F^*, \\ M &= \sum_{k=1}^{K} p_k^2 \sigma_k^2 + 6L\Gamma + 8(E-1)^2 G^2. \end{aligned} \quad (14)$$

where L, μ, σ_k, and G are defined by the assumptions 1-4. Γ is defined by $\Gamma = F* - \sum_{k=1}^{K} p_k F_k^*$ for quantifying the degree of non-iid. This theorem posits that as the number of rounds T approaches infinity, the expectation of the fine-tuning error $\delta^{(T)}$ converges to zero. In contrast, FedIT deviates from the FedAvg model updating rule as depicted in Equation 2, introducing non-gradient noises through its averaging process. Therefore, it fails to achieve convergence at the rate of $O(1/T)$ While this deviation does not invalidate FedIT's utility in federated fine-tuning, it significantly impairs its convergence rate and overall effectiveness.