SLoRA: Federated Parameter Efficient Fine-Tuning of Language Models

Sara Babakniya*[1], Ahmed Roushdy Elkordy*[1], Yahya H. Ezzeldin[1],
Qingfeng Liu[2], Kee-Bong Song[2], Mostafa El-Khamy[2], Salman Avestimehr[1]

[1] University of Southern California, [2] SoC R&D Samsung Semiconductor Inc.

Abstract

Transfer learning via fine-tuning pre-trained transformer models has gained significant success in delivering state-of-the-art results across various NLP tasks. In the absence of centralized data, Federated Learning (FL) can benefit from distributed and private data of the FL edge clients for fine-tuning. However, due to the limited communication, computation, and storage capabilities of edge devices and the huge sizes of popular transformer models, efficient fine-tuning is crucial to make federated training feasible. This work explores the opportunities and challenges associated with applying parameter efficient fine-tuning (PEFT) methods in different FL settings for language tasks. Specifically, our investigation reveals that as the data across users becomes more diverse, the gap between fully fine-tuning the model and employing PEFT methods widens. To bridge this performance gap, we propose a method called SLoRA, which overcomes the key limitations of LoRA in high heterogeneous data scenarios through a novel data-driven initialization technique. Our experimental results demonstrate that SLoRA achieves performance comparable to full fine-tuning, with significant sparse updates with approximately $\sim 1\%$ density while reducing training time by up to 90%.

1 Introduction

With the popularity of smartphones and personal gadgets, valuable user data is distributed more than ever. This data can help different companies and service providers improve their products and make them more efficient and personalized. However, privacy is a growing and crucial concern that is inevitable to avoid. Users care about the performance of their applications but do not want their private data to be accessed by everyone. Federated Learning (FL) (McMahan et al., 2017; Konečnỳ et al., 2016) is a new paradigm that can solve both problems simultaneously. Users collaboratively train a common model locally using their private dataset and only share their model update with a parameter server that orchestrates the training process over multiple training rounds without sharing their private data.

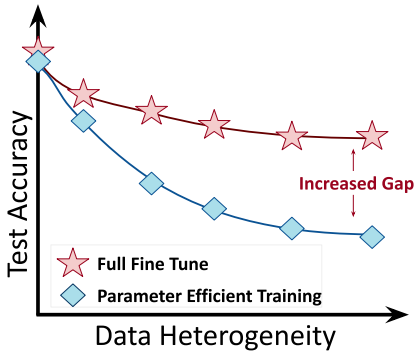


Figure 1: The impact of client data distribution on the performance of full fine-tuning vs. Parameter efficient Fine-tuning. While heterogeneity has an adverse effect on both of them, parameter efficient methods are more vulnerable and experience more accuracy drop in more heterogeneous settings.

Although FL has already proven beneficial in various domains (Kairouz et al., 2021), such as next-word prediction and healthcare, it still has critical challenges to be deployed on large scales. Here we mainly focus on the efficiency of FL and problems of clients' heterogeneous data distribution for pre-trained language models.

Large pre-trained language models have proven to perform very well even in the zero-shot setup (Brown et al., 2020). However, as the tasks get more specialized, these models require fine-tuning to enhance their performance on these domain-specific tasks (Hu et al., 2021). To provide the required privacy while fine-tuning, we can benefit from the FL paradigm. But asking the clients to fine-tune the models and communicate the update has downsides.

The problem is that fine-tuning can be computationally expensive as it might change the parame-

* Equal contribution (alphabetical order on the last name).

ters of the entire model. Besides, language models are not used in one application, and clients need to fine-tune them on several tasks. As a result, supporting multi-tasks can also be challenging, especially in memory-constrained scenarios (e.g., for edge devices), because the required memory for the fine-tuned models grows linearly with the number of tasks.

The next concern of shifting the fine-tuning to the client side is its overhead on their already limited resources. Edge devices usually have very little bandwidth (especially up-link) as multiple users share the same resource. Furthermore, both communicating and training can be highly energy-consuming. Therefore, the direct use of FL for NLP tasks may limit its applicability.

Recently, Parameter Efficient Fine Tuning (PEFT) (Hu et al., 2021; Li and Liang, 2021; Lester et al., 2021) has emerged as an alternative training strategy that does not require fine-tuning of all parameters of the pre-trained model but *only* updates a small portion of the parameters (task-specific parameters) while freezing most of the pre-trained weights of the model to their initial pre-trained values. This approach has been shown to maintain task performance while reducing the parameter budget needed in the centralized setting. Furthermore, if one can successfully design a federated PEFT can also benefit from reduced communication costs.

In this work, we first explore the performance of the existing centralized PEFT method in FL. We observe that the gap between Full Fine Tuning (FFT) and PEFT increases, that the clients' data distribution gets more heterogeneous (Fig. 1). To this aim, we propose a new algorithm called Primed-LoRA designed for FL where its efficient variant, SLoRA, can achieve first parameter efficiency, second, reduce training and communication cost, and finally closes the gap between PEFT and FFT.

2 Related Work

In the following, we summarize areas in the literature closely related to federated parameter efficient fine-tuning.

Parameter Efficient Fine Tuning (PEFT). In general, PEFT methods can be broadly classified into two main categories based on the nature of the tuned parameter. The first category fine-tunes a subset of existing parameters, including the classification head, bias term (Zaken et al., 2021), and sparse subnetworks within the original pre-trained model for each task (Guo et al., 2020). The second category is module-based fine-tuning, where an additional set of parameters (e.g., modules) are added for each task. These modules are fine-tuned while freezing the entire pre-trained model.

Different methods have been proposed depending on the place where the modules are inserted into the model. One class adds bottleneck trainable modules serially to the model components such as adapters (Houlsby et al., 2019) and its variants (Pfeiffer et al., 2022; He et al., 2021). Another approach is to introduce modules added in parallel to model parameters such as LoRA (Hu et al., 2021) and prefix or prompt tuning added in parallel to the attention heads (Li and Liang, 2021) or embeddings (Lester et al., 2021). Recently, there have been several approaches for data-driven PEFT configuration selection for adding adapter modules (He et al., 2021; Zhou et al., 2023; Wang et al., 2022).

PEFT in Federated Learning. Recent studies (Sun et al., 2022; Zhang et al., 2022) have investigated the performance of various PEFT methods within the context of Federated Learning (FL) for vision tasks. These studies considered different aspects of federated learning, such as client stability, data distribution, and differential privacy settings. The findings indicated that PEFT could replace FFT without compromising performance while significantly reducing communication costs.

While the previous works focused on vision and vision-language models, our study differs in several key aspects. Firstly, we specifically study the application of PEFT in the context of language models and additionally examine the effect of data heterogeneity across clients on the performance of PEFT for NLP tasks. Secondly, our work extends beyond benchmarking different PEFT methods in the federated setting to propose an approach that yields comparable performance to FFT even in extreme non-IID settings.

Efficient Training in FL. Efficient training in FL has been extensively studied in the literature (Diao et al., 2020; Horvath et al., 2021; Alam et al., 2022; Niu et al., 2022; Kang et al., 2020; Li et al., 2020, 2021). Efficient training in FL employs sparse training at different levels. Some approaches only apply sparse training at the client side to update a full-size model retained at the server (Diao et al., 2020; Horvath et al., 2021; Alam et al., 2022; Niu et al., 2022). Other approaches utilize sparse training to

optimize a model that is sparse both at the client and server sides (Bibikar et al., 2022; Babakniya et al., 2022; Qiu et al., 2022; Li et al., 2021).

Although efficient learning and PEFT seem very similar at first glance, given they both share a common goal of reducing the training complexity for the clients. PEFT also focuses on the unique aspect of storage load when considering multiple tasks.

Applying efficient sparse training methods on (pre-trained) large language models typically can only retain good performance with a moderate level of sparsity (Frantar and Alistarh, 2023; Gordon et al., 2020; Chen et al., 2020). This can result in a huge storage penalty as sparsification patterns can be heterogeneous across different tasks. On the other hand, PEFT retains the full pre-trained model but applies extremely sparse with ($\sim 1\%$) density updates (Hu et al., 2021; Houlsby et al., 2019; Zaken et al., 2021; Pfeiffer et al., 2022; He et al., 2021) to the pre-trained model per task, which allows for substantial storage savings and significant reduction on the communication cost. Importantly, PEFT does this while providing a strong comparable performance to the fully fine-tuned model.

3 Preliminaries

3.1 PEFT Baselines in Centralized Learning

We investigate Pfieffer, LoRA, Holusby, and BitFit as state-of-the-art PEFT methods in PEFT. The first three methods add a separate bottleneck module (a down projection dense layer followed by an up projection) with a dimension r to the model. Their difference is where the module is added. Holusby places a bottleneck module after the multi-head attention and feed-forward block in each Transformer layer. Pfieffer places a bottleneck module only after the feed-forward block in each Transformer layer. In LoRA, the bottleneck module can be parallel to any dense layer in the model. Finally, BitFit is a simple method that only allows fine-tuning the bias terms.

3.2 Observation: PEFT is challenged when data distribution gets non-IID

One of the biggest challenges in FL is the degradation in performance when training in scenarios with heterogeneous client distributions (Kairouz et al., 2021). While this is a well-documented phenomenon reflected in FFT in FL, we observe in this work that the penalty to performance is even more substantial when using PEFT compared to FFT. In particular, after benchmarking different models and datasets, we observe that the higher the level of heterogeneity, the more significant the performance gap between FFT and PEFT methods. Thus, a simple naive adaptation of applying PEFT methods locally in an FL setting can lead to potentially huge performance loss.

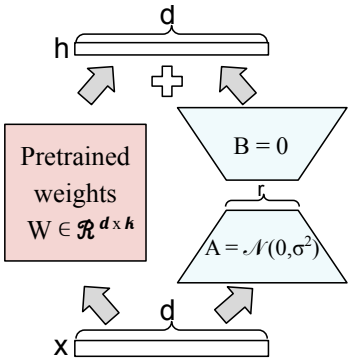


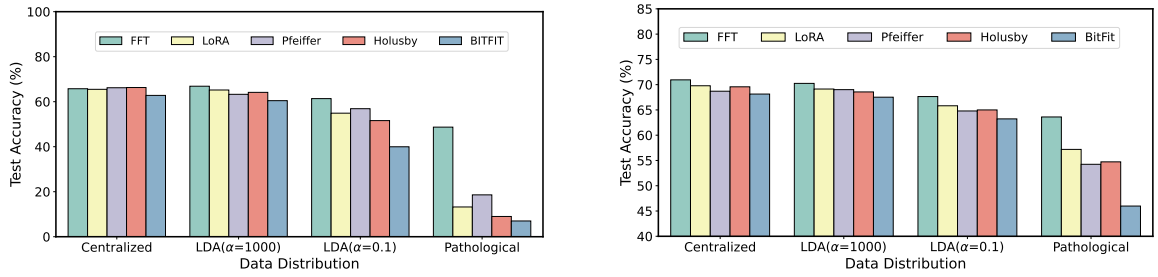
Figure 2: LoRA Block

Our focus in the remainder of the paper is on developing approaches to reduce the gap between FFT and PEFT while efficiently using clients' resources regarding communication and storage loads. Before proposing our approach in Section 4, we summarize the approach that shows the greatest promise, which is the SOTA PEFT approach, LoRA.

3.3 Low-Rank Adaption: LoRA

We focus on presenting Low-Rank Adaptation (LoRA) since it's the state-of-the-art method for PEFT of large pre-trained language models (LMs), and our proposed algorithm adopts this method. The key idea of LoRA is that instead of fully fine-tuning the pre-trained weight matrix $\mathbf{W}_0 \in \mathbb{R}^{d\times k}$, its update is constrained with a low-rank decomposition $\mathbf{W}_0 + \Delta\mathbf{W} = \mathbf{W}_0 + \mathbf{BA}$, where $\mathbf{B} \in \mathbb{R}^{d\times r}$, $\mathbf{A} \in \mathbb{R}^{r\times k}$, and $r << \min(k, d)$, and only $\mathbf{B}$ and $\mathbf{A}$ are trained while freezing the pre-trained weight $\mathbf{W}_0$.

The way the LoRA is implemented is shown in Fig. 2, where a parallel module of a down projection matrix $\mathbf{B}$ followed by the up projection matrix $\mathbf{A}$ in parallel to original pre-trained weight matrix. A random Gaussian initialization for $\mathbf{A}$ and zero for $\mathbf{B}$ are used to ensure the modified model and the original model are equivalent (e.g., $\Delta\mathbf{W} = \mathbf{BA}$ is zero at the beginning of training). The modified forward pass after adding the LoRA module is given



(a) 20News group dataset on Albert

(b) 20News group dataset on DistilBERT

Figure 3: Performance of PEFT methods in the centralized and federated setting with different data distributions for 20News group dataset.

as follows

$$\mathbf{h} = \mathbf{W}_0\mathbf{x} + \frac{\beta}{r}\mathbf{BAx}, \tag{1}$$

where r is LoRA rank and β is a constant in r. Therefore, according to (eq. 1), the output from the LoRA module is added coordinate-wise to the output of the original model. The scaling $\frac{\beta}{r}$ can be used to reduce the need to re-tune hyper-parameters when varying r.

4 Our Proposed approach (Primed-LoRA)

In centralized learning, LoRA consistently has shown promising performance in different tasks and closely follows the FFT accuracy. As shown in Fig. 3, this still holds for federated settings with more homogeneous data distribution (larger α). However, in highly non-IID data distribution, LoRA can fail to reach close to the FFT performance or suffer from a slower convergence rate compared to FFT. We hypothesize that one of the reasons behind this is the way that LoRA blocks are initialized (Fig. 2).

As described in Hu et al. (2021), LoRA initializes the A matrix with random independent Gaussian coefficients and B with 0. This initialization works in a centralized setting where the data is ample and concentrated. However, these random and zero initializations in FL can potentially slow down the fine-tuning process.

Data-driven priming of LoRA. Based on our hypothesis, picking a better starting point for LoRA might improve its performance. Therefore, we propose a two-stage parameter efficient fine-tuning, Primed-LoRA, based on the LoRA algorithm.

In Stage 1, the FL clients collaboratively find a mature starting point to prime the LoRA blocks. Then, in Stage 2, we run the LoRA algorithm with our learned initializers from Stage 1. In the remainder of the section, we discuss different ways of priming in Primed-LoRA and the properties of the different priming methods.

4.1 Full fine-tuning for priming LoRA

A straightforward approach for Stage 1 of Primed-LoRA is to perform a full fine-tuning for a few rounds in Stage 1 of Primed-LoRA and then use SVD matrix decomposition t extract a good initialization for Stage 2. We call this variant of Primed-LoRA as FFT-LoRA or **FLoRA**.

Formally, we use ΔW to denote the accumulated change in the model parameters after Stage 1. This weight difference is, conceptually, the same as what we have in each LoRA block, but ΔW is of size $d \times k$ matrix. We use SVD to arrive at a low-rank approximation of $\Delta W = \mathbf{B} \times \mathbf{A}$, with $\mathbf{B} \in \mathbb{R}^{d \times r}$, $\mathbf{A} \in \mathbb{R}^{r \times k}$ that is used to prime LoRA in Stage 2. A description of how we create $\mathbf{A}$ and $\mathbf{B}$ using SVD is delegated to Appendix A.

After converting ΔW into A and B, the training goes to the second stage, where clients only update the LoRA blocks and share those parameters with the server. As shown in Fig. 7, FLoRA can improve the global model's performance, especially with more training rounds in Stage 1. However, as we discuss next, using full fine-tuning to prime LoRA comes at a cost.

Cost of FLoRA In FLoRA, Stage 1 successfully enhances the performance while achieving parameter efficiency. However, in terms of the training cost, the communication and computation cost of training at this stage is the same as full fine-tuning. In Stage 2, the cost depends on dimensions parameter r.

While FLoRA shows that Primed-LoRA can

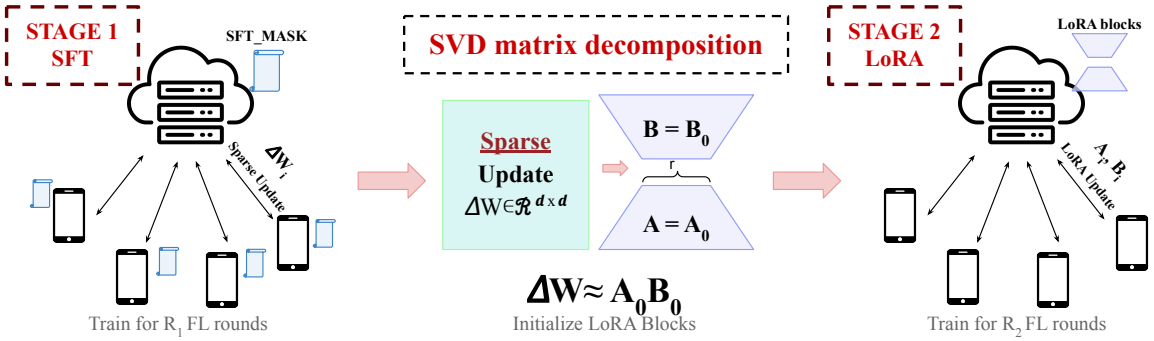


Figure 4: Overview of SLoRA; First server initializes a mask, and clients only update the parameters in the mask. Then using SVD, the updates are decomposed into LoRA blocks which will be used as an initialization for Stage 2.

meet our targets in terms of parameter efficiency, one important question is yet to be answered. How to preserve this performance but reduce the training costs? This is especially important during the training because, in cross-device federated learning, clients have a limited budget.

By looking at the cost of FLoRA, we can figure out that there are two parameters involved in this cost; the number of rounds and the communication/computation cost of each round. As a result, one way to decrease the cost in Stage 1 is to reduce the number of FFT rounds. However, as depicted in Fig. 5, the performance of the model at the end of Stage 1 directly impacts the performance of the final model.

Another way to make FLoRA more efficient is to reduce the data processing and update size transmitted from clients. Towards this goal, in the following subsection, we propose **Primed-LoRA with Sparse Fine-tuning** or SLoRA, where the clients only update a fraction of the parameters in Stage 1 instead of fully fine-tuning the parameters.

4.2 Sparse Fine-tuning for priming LoRA

Sparse Fine-Tuning (SFT) (Ansell et al., 2021; Zaken et al., 2021; Guo et al., 2020) aims to achieve parameter efficiency by sparsifying the updates. In other words, in $W_R = W_0 + \Delta W$, the update (ΔW) is a set of highly sparse matrices that can be stored and transmitted efficiently.

We opt to employ the approach of Ansell et al. (2021), but all the works follow similar ideas. In particular, the goal is to find a binary mask such that 1's indicates the position of weights that can change in each round. Ansell et al. (2021) proposal to generate such mask is to find the *top-K most important* weights based on their contribution in FFT. The weights that change the highest – from

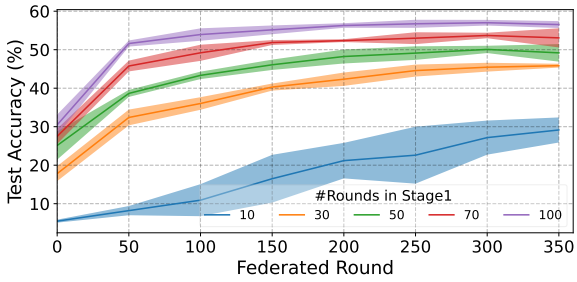


Figure 5: Impact of the number of federated rounds in stage 1 on the final performance of the model after stage 2 in FLoRA for 20News group dataset on Albert.

their original pre-trained values – in FFT are the ones we should keep training in SFT. As a result, the authors add a new warm-up stage where they fine-tune the weight for several rounds to detect such weights.

Sparse Fine-tuning in Stage 1. Ansell et al. (2021) is designed for the centralized setting where the data is located in the same location and can be utilized in finding the mask. But, the situation is different in the federated setting. The server cannot fine-tune the model on the current task because it does not have data. Clients can individually train the model and come up with personalized masks. However, the difference in clients' masks increases the density of the aggregated update and reduces parameter efficiency. Alternatively, if the clients want to find the important weights together, they require to do FFT, which is against our goal of reducing the cost and number of FFT communications.

To solve this problem, we propose the server so generate a random data-independent binary mask with uniform density for all layers at the beginning of training. Then, the clients only train the weights using this mask. Thus, the density of the update from the server and clients does not change, and

they can benefit from reduced communication. We want to point out that we choose the mask's density to be higher than the number of parameters in the LoRA blocks.

Primed-LoRA in Stage 2. Stage 2 in SLoRA follows the same procedure as discussed in Section 4.1. After the first Stage 1, we employ SVD to decompose ΔW into the two components $\mathbf{A}$ and $\mathbf{B}$ used in the LoRA algorithm. Algorithm 4 summarizes the different steps in SLoRA.

Algorithm 1 Overview of Primed LoRA

R_i: FL rounds in stage i, N: Total # clients, K: Total # participant per round, E: Local epoch, W_R: Model weights in round R, r: LoRA parameter, d_i: update density in stage i, $Algorithm$ choice between FLoRA and SLoRA
# Stage 1
if $Algorithm = SLoRA$ **then**
  $SFT\_Mask = generateMask(W_0, d_1)$
else
  $SFT\_Mask = 1$
end if
for $R = 1$ **to** R_1 **do**
  for $k = 1$ **to** K **do**
    $W^k_{R-1} = train(W_R, SFT\_Mask, E)$
  end for
  $W_R = aggregate(W^{0,..,k}_{R-1})$
end for
$\Delta W = W_R - W_0$
$[\mathbf{A}, \mathbf{B}]^0 = SVD(\Delta W, r)$
# Stage 2
for $R = 1$ **to** R_2 **do**
  for $k = 1$ **to** K **do**
    $[\mathbf{A}, \mathbf{B}]^k_R = trainLoRA([\mathbf{A}, \mathbf{B}]_{R-1}, E)$
  end for
  $[\mathbf{A}, \mathbf{B}]_R = aggregate([\mathbf{A}, \mathbf{B}]^{0,...,K}_{R-1})$
end for

5 Experiments

5.1 Setting

Models. Our experiments focus on the following two models: Albert Lan et al. (2019) and DistilBERT Sanh et al. (2019).

Datasets. We show our results for two datasets, News Category and 20News group Lang (1995). Since we rely on non-IID label distribution, we focus on classification tasks. The 20News group dataset includes 20 news topics with about 19K data points. We use a 60 % - 40 % split for train and test, respectively. The news category is another dataset about news, and it has 15 different labels. This dataset includes about 330K training data, but we only use a randomly sampled 10% for training. Clients only train for one epoch every round, and the batch size is always 32.

Federated Setting. The total number of clients (N) is 100 in all the experiments. The number of participants in every round (K) is 20 clients for pathological non-IID and 10 otherwise.

5.2 Metrics

We mainly focus on the accuracy of the final global model. Moreover, to compare the different costs of each algorithm, we report their communication cost and training time on our local GPU. We use a single NVIDIA-A100 GPU for each experiment. All the experiments are performed for 5 different seeds, and we report the average of the 3 best results. We used FedAvg (McMahan et al., 2017) as our aggregation method, where clients at round R receive W_R from the server. After fine-tuning, clients calculate the difference of the current weight with W_R and send it to the server. Finally, the server average all the updates from all the participants to compute W_{R+1}.

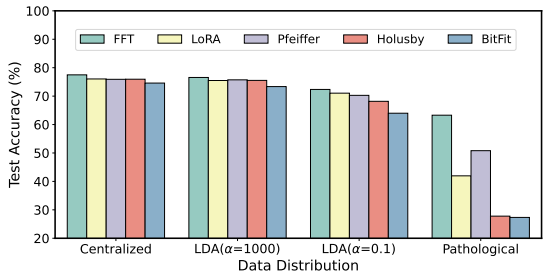
5.3 Baselines

In the following, we investigate LoRA, Holusby, Pfieffer, and BitFit to understand the impact of data heterogeneity and update size. For data heterogeneity, we used Latent Dirichlet Distribution (LDA) allocation (Reddi et al., 2020), which controls the data distribution of the dataset in each client with α parameter. Small α in this allocation means the data distribution is more heterogeneous. Also, to explore non-IID, we have added pathologically non-IID similar to McMahan et al. (2017).

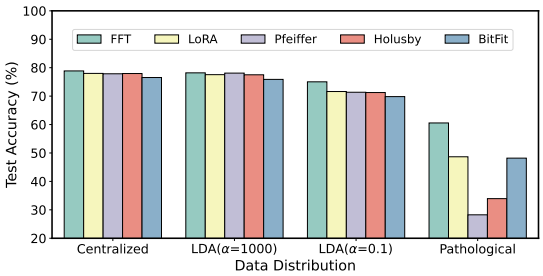
Table 1: r parameter for each setting. Using this parameter, we can calculate the update size of each method.

Method	LoRA	Holusby	Pfieffer
Small	20	38	76
Large	190	384	768

The update size in BitFit is fixed and equal to the total size of bias layers. For other algorithms, update size is controlled by parameter r, which indicates the size of the appended module and is summarized in Table 1. Generally, smaller r leads



(a) News Category dataset on Albert



(b) News Category dataset on DistilBERT

Figure 6: Performance of PEFT methods in the settings and data distributions for News Category dataset.

to smaller update sizes, and here we consider two different sizes. The density of the smaller update is approximately similar to BitFit's. To explore the impact of model size, we have included a larger update size with a higher density as well.

Also, for the LoRA algorithm, we can add the blocks in all the dense layers, but here we only select the 4 dense layers in the multi-head attention layer, similar to the original paper. LoRA has an extra α parameter which we set to be equal to r.

6 Evaluation

6.1 Data Heterogeneity

Fig. 3 and 6 show the impact of data heterogeneity on the performance of two models, DistilBERT and Albert, trained on 20News and News category datasets. The gap between the PEFT and FFT grows in all four settings by making the data more non-IID. This phenomenon indicates the necessity of a new PEFT algorithm tailored for FL.

6.2 Update Size

Table 2 shows the impact of update size for the 20News group dataset on the Albert model on the final performance of the global model. As expected, larger update sizes can help the performance and give a model with higher accuracy. But, the problem with this approach is that now the update size and communication cost of training would also increase, which is not desired.

6.3 Performance of SLoRA

We propose an algorithm called SLoRA which includes two stages. First, clients collaboratively update the model using SFT techniques. Then, the final update is used to generate the initialization for stage 2, which is LoRA. Therefore, we compare our method with *stage 1 without stage 2* and *stage 2 only*, which we call SFT and LoRA, respectively. Through this comparison, we aim to evaluate the extent of improvement achieved by SLoRA in contrast to the individual utilization of each stage.

Table 2: Impact of update density of different PEFT methods on the performance for the 20News group dataset on Albert.

Data Distribution	**Density** (%)	**LoRA** (%)	**Holusby** %	**Pfeiffer** (%)
Centralized	10	65.9	66	66.5
	1	65.5	66.3	66.2
$\alpha = 0.1$	10	61.4	53.1	58
	1	54.9	51.6	56.89
Pathological	10	40.2	38.8	21.1
	1	9	13.22	18.6

Here, we only consider pathologically non-IID data distribution as discussed earlier in this setting; PEFT causes a significant drop in its performance. For our proposed method SLoRA, we train the model in Stage 1 using an update sparsity of 10% to have a good model performance at Stage 1 that can be utilized at Stage 2, yet with a minimal cost. In Stage 2, for both SLoRA and FLoRA, we add the LoRA module by utilizing the SVD decomposed model update from Stage 1 with a higher target update sparsity, as discussed in Section 4.2. Specifically, the LoRA modules are added to each dense layer in the model, except for the embedding and classification layers. For the dense layers within the multi-head attention (MHA) block and the feed-forward block, we incorporate the parallel module with an assigned value of $r = 10$. Additionally, we use $r = 18$ for the pre-classification layer to achieve a better SVD decomposition in the last layer. The size of the added module is $0.14M$ parameters compared to $11.7M$ parameters of the original Albert model representing only

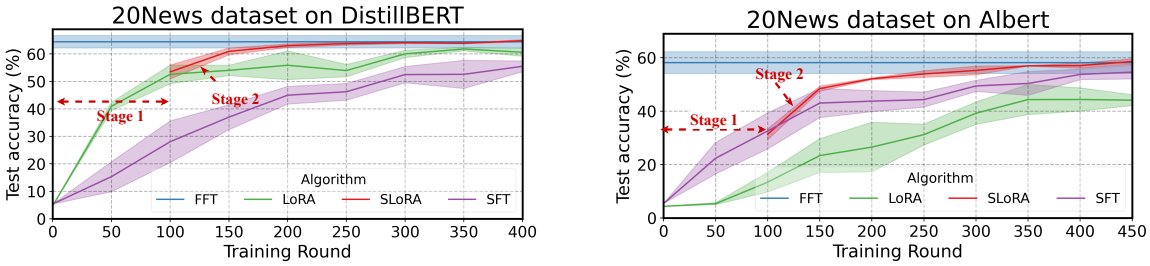


Figure 7: The performance of SLoRA using for 20News group dataset on Albert and DistilBERT.

Table 3: The training time (i.e., computation time) of different methods and the number of training rounds with the server for the Albert model on the 20news dataset.

Method	**FFT**	**LoRA**	**SFT**	**SLoRA**	**SVD-decomposition**
Computation time (sec/epoch)	0.39	0.43	2	Stage1: 2.1 - Stage2: 0.43	15.4 (one-time cost)
Total training rounds	250	1250	1250	350	—

1.3% of the original model size. For SFT, we train the model with a sparsity update of 1.3% to match the same target sparsity update of SLoRA.

Table 4: Performance and training cost of different algorithm for the 20News group dataset on Albert model.

	# Trainable Parameter	**Training Time (min)**	**Accuracy** (%)	**Communication (Gbits)**
FFT	$11.7M$	596.7	58.17 ± 4	174
LoRA	$\mathbf{0.14}M$	49.5	56.5 ± 1.2	**9.95**
SFT	$\mathbf{0.14}M$	78.4	57.6 ± 0.3	**9.95**
SLoRA	$\mathbf{0.14}M$	**40.4**	$\mathbf{58.6 \pm 1}$	**9.95**

We evaluate the performance of SLoRA for the 20News group dataset on Albert and DistilBERT in Fig. 7. As presented in the figure, SLoRA converges to better accuracy and requires smaller training rounds. This is particularly important for low-budget and resource-restricted settings of federated learning.

Regarding the model training time (e.g., computation time), we report the duration based on a single GPU. Table 3 summarizes the average training time per epoch over 10 distinct runs for various methods. We note that the time for SLoRA can be computed using the time for Stages 1 and 2. The SVD decomposition process is executed only once in Stage 2 and can be done efficiently by the server.

We analyze two aspects of the comparison. Firstly, we evaluate the performance of SLoRA while matching the communication budget across different baselines. In particular, in Stage 1, SLoRA communicates larger models compared to the other PEFT baselines (SFT and LoRA). To ensure a fair comparison, we allow the PEFT baselines to be trained for a longer duration in order to match the same communication budget as SLoRA. Table 3 summarizes the number of training rounds for the baselines. Despite the baselines having even higher communication rounds, Fig. 4 demonstrates that SLoRA still achieves comparable accuracy to the fully fine-tuned model, with a slight marginal improvement.

On the other hand, LoRA and SFT show a performance drop of -1.67 and -0.57 in their maximum accuracy, respectively. It's worth noting that this is achieved with a longer training time compared to SLoRA. Additionally, SLoRA exhibits higher stability than other baselines across different seeds.

In addition to the communication budget, the number of communication rounds is another crucial aspect of federated learning, especially when considering user availability and its impact on computational resource consumption. Therefore, we also compare the performance of SLoRA to the baselines while keeping the number of training rounds fixed, consistent with that of SLoRA. As depicted in Figure 7(b), we observe that the performance gap between SLoRA and the baselines (LoRA and SFT) increases to 4% and 14.2%, respectively.

7 Conclusion

In this work, we have investigated employing PEFT methods for fine-tuning language models in the FL setting to reduce communication and storage costs. We have found that different PEFT methods perform poorly compared to FFT as the diversity of the data increases. To overcome this limitation, we

have proposed a novel approach called SLoRA that can maintain the same performance as FFT with minimal communication, time, and storage cost.

Limitations

In this work, we assumed all the clients have similar resource limitations and power. However, in the real world, some users may have more power and can larger update sizes both for training and inference, while others may be more constrained. One interesting future work is extending this work to also consider resource heterogeneity.

Ethics Statement

We confirm that this work does not raise any ethical problems.

References

Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. 2022. Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction. *arXiv preprint arXiv:2212.01548*.

Alan Ansell, Edoardo Maria Ponti, Anna Korhonen, and Ivan Vulić. 2021. Composable sparse fine-tuning for cross-lingual transfer. *arXiv preprint arXiv:2110.07560*.

Sara Babakniya, Souvik Kundu, Saurav Prakash, Yue Niu, and Salman Avestimehr. 2022. Federated sparse training: Lottery aware model compression for resource constrained edge. *arXiv preprint arXiv:2208.13092*.

Sameer Bibikar, Haris Vikalo, Zhangyang Wang, and Xiaohan Chen. 2022. Federated dynamic sparse training: Computing less, communicating less, yet learning better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6080–6088.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846.

Enmao Diao, Jie Ding, and Vahid Tarokh. 2020. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot.

Mitchell A Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing bert: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*.

Demi Guo, Alexander M Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. 2021. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *Advances in Neural Information Processing Systems*, 34:12876–12889.

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*.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210.

Jiawen Kang, Zehui Xiong, Dusit Niyato, Yuze Zou, Yang Zhang, and Mohsen Guizani. 2020. Reliable federated learning for mobile networks. *IEEE Wireless Communications*, 27(2):72–80.

Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Ken Lang. 1995. Newsweeder: Learning to filter netnews. In *Machine learning proceedings 1995*, pages 331–339. Elsevier.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Ang Li, Jingwei Sun, Xiao Zeng, Mi Zhang, Hai Li, and Yiran Chen. 2021. Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, pages 42–55.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3):50–60.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

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.

Yue Niu, Saurav Prakash, Souvik Kundu, Sunwoo Lee, and Salman Avestimehr. 2022. Federated learning of large models at the edge via principal sub-model training. *arXiv preprint arXiv:2208.13141*.

Jonas Pfeiffer, Naman Goyal, Xi Victoria Lin, Xian Li, James Cross, Sebastian Riedel, and Mikel Artetxe. 2022. Lifting the curse of multilinguality by pre-training modular transformers. *arXiv preprint arXiv:2205.06266*.

Xinchi Qiu, Javier Fernandez-Marques, Pedro PB Gusmao, Yan Gao, Titouan Parcollet, and Nicholas Donald Lane. 2022. Zerofl: Efficient on-device training for federated learning with local sparsity. *arXiv preprint arXiv:2208.02507*.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. 2020. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Guangyu Sun, Matias Mendieta, Taojiannan Yang, and Chen Chen. 2022. Exploring parameter-efficient fine-tuning for improving communication efficiency in federated learning. *arXiv preprint arXiv:2210.01708*.

Yaqing Wang, Sahaj Agarwal, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022. Adamix: Mixture-of-adaptations for parameter-efficient model tuning. *arXiv preprint arXiv:2210.17451*.

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*.

Zhuo Zhang, Yuanhang Yang, Yong Dai, Lizhen Qu, and Zenglin Xu. 2022. When federated learning meets pre-trained language models' parameter-efficient tuning methods. *arXiv preprint arXiv:2212.10025*.

Han Zhou, Xingchen Wan, Ivan Vulić, and Anna Korhonen. 2023. Autopeft: Automatic configuration search for parameter-efficient fine-tuning. *arXiv preprint arXiv:2301.12132*.

A Priming LODA from FFT using SVD

Singular Value Decomposition is a matrix decomposition method that helps us to rewrite a $m \times n$ matrix M as a multiplication of three matrices, $M = \mathbf{U}\Sigma\mathbf{V}^T$, where $\mathbf{M} \in \mathbb{R}^{m\times n}$, $\mathbf{U} \in \mathbb{R}^{m\times m}$, $\Sigma \in \mathbb{R}^{m\times n}$, $\mathbf{V} \in \mathbb{R}^{n\times n}$ and Σ is a rectangular diagonal matrix with descending diagonal terms [1].

The accurate decomposition is not parameter efficient and generates matrices with large dimensions. Therefore, instead of the exact decomposition, we use an approximation that preserves most of the information in $\mathbf{M}$ (Our ΔW of interest throughout the paper). A common way to approximate a $m \times m$ matrix $\mathbf{M} \approx \tilde{\mathbf{U}}\tilde{\mathbf{V}}^T$ (where $\tilde{\mathbf{U}} \in \mathbb{R}^{m\times r}$, $\tilde{\mathbf{V}} \in \mathcal{R}^{m\times r}$ and $r \ll m$) is to take the first r columns of $\mathbf{U}$ and $\mathbf{V}$ which are associated with the largest singular values in Σ, and then constructing $\tilde{\mathbf{U}} = \mathbf{U}_{[1:m,1:r]}\Sigma_{[1:r,1:r]}$ and $\tilde{\mathbf{V}} = \mathbf{V}_{[1:m,1:r]}$. Note that increasing the r value makes this approximation more accurate as it approaches the true SVD but, at the same time, decreases the saving in the parameters.

B Performance and cost of 20News group on DistilBERT

Table 5 shows the training cost (time) of different baselines and components of SLoRA for the 20News group dataset on the DistilBERT model. As expected, the cost of Stage 1 is similar to SFT, and the cost of Stage 2 is equal to that in LoRA.

Table 6 summarizes the performance of different methods for the 20News group dataset on the DistilBERT model. As mentioned earlier, to have a fair comparison, we trained different methods for different rounds of federated learning and only fixed the communication cost. As shown in the

[1] SVD decomposition is not unique, but we are concerned with the decomposition where the singular values are organized in descending order

Table 5: The training time (i.e., computation time) of different methods and the number of training rounds with the server for the Distilbert model on the 20news dataset.

	FFT	LoRA	SFT	SLoRA	SVD-decomposition
Computation time (sec/epoch)	0.19	0.20	1.02	Stage 1: 1.1 Stage 2: 0.20	15.4 (one-time cost)
Total training rounds	250	1250	1250	300	—

table, SLoRA enjoys better performance compared to FFT and still has better training time.

Table 6: Performance and training cost of different algorithm for the 20News group dataset on DistilBERT model.

	# Trainable Parameter	Training Time (min)	Accuracy (%)	Communication (Gbits)
FFT	$67.0M$	3407	64.4 ± 2	997
LoRA	$\mathbf{0.7}M$	203	63.1 ± 0.5	**57.5**
SFT	$\mathbf{0.7}M$	220	62.3 ± 0.9	**57.5**
SLoRA	$\mathbf{0.7}M$	**186**	$\mathbf{64.8 \pm 0.4}$	**57.5**

The results for the News category dataset on Albert and DistilBERT are summarized in Table 7 and Table 8, respectively. As depicted in the tables, the same observations still hold for this dataset as well.

Table 7: Performance and training cost of different algorithms for the News category dataset on Albert model.

	# Trainable Parameter	Training Time (min)	Accuracy (%)	Communication (Gbits)
FFT	$11.7M$	559	65.2 ± 0.6	174
LoRA	$\mathbf{0.14}M$	**39.5**	56.8 ± 5	**9.95**
SFT	$\mathbf{0.14}M$	140	55.1 ± 0.6	**9.95**
SLoRA	$\mathbf{0.14}M$	41.5	$\mathbf{62.8 \pm 3}$	**9.95**

B.1 Impact of Update size.

In this section, we show the impact of update size for other settings as well. In all the settings, updates with higher density have better performance as expected, and the performance considerably drops by increasing the data heterogeneity.

Table 8: Performance and training cost of different algorithms for the News category dataset on DistilBERT model. We train SLoRA on Stage 2 for only 50 rounds.

	# Trainable Parameter	Training Time (min)	Accuracy (%)	Communication (Gbits)
FFT	$67.0M$	3406	61.6 ± 1.5	997
LoRA	$\mathbf{0.7}M$	161	50.2 ± 5	**45.5**
SFT	$\mathbf{0.7}M$	177	55.8 ± 0.1	**45.5**
SLoRA	$\mathbf{0.7}M$	**160**	$\mathbf{56.1 \pm 1}$	**45.5**

Table 9: Impact of update density of different PEFT methods on the performance for the 20News group dataset on DistilBERT.

Data Distribution	Density (%)	LoRA (%)	Holusby %	Pfeiffer (%)
Centralized	10	70.0	70.0	69.5
	1	69.8	69.6	68.7
$\alpha = 0.1$	10	66.3	65.3	65.3
	1	65.8	65.0	64.8
Pathological	10	58.6	55.1	55.7
	1	57.2	54.7	54.2

Table 10: Impact of update density of different PEFT methods on the performance for the News category dataset on Albert.

Data Distribution	Density (%)	LoRA (%)	Holusby %	Pfeiffer (%)
Centralized	10	76.0	76.0	76.0
	1	76.0	75.9	75.9
$\alpha = 0.1$	10	72.3	68.54	70.6
	1	71	68.17	70.27
Pathological	10	60.7	33.6	57.0
	1	41.96	27.78	50.8

Table 11: Impact of update density of different PEFT methods on the performance for the News category dataset on DistilBERT.

Data Distribution	Density (%)	LoRA (%)	Holusby %	Pfeiffer (%)
Centralized	10	78.4	78.0	78.15
	1	77.99	77.94	77.85
$\alpha = 0.1$	10	73.55	72.2	71.6
	1	71.6	71.6	71.362
Pathological	10	49.39	38.03	37.4
	1	33.9	33.9	28.2