SLoRA: Federated Parameter Efficient Fine-Tuning of Language Models

Sara Babakniya[*]
University of Southern California
`babakniy@usc.edu`

Ahmed Roushdy Elkordy[*]
University of Southern California
`aelkordy@usc.edu`

Yahya H. Ezzeldin
University of Southern California
`yessa@usc.edu`

Qingfeng Liu, Kee-Bong Song, Mostafa El-Khamy
SoC R&D Samsung Semiconductor Inc.
`qf.liu, keebong.s, mostafa.e@usc.edu`

Salman Avestimehr
University of Southern California
`avestime@usc.edu`

Abstract

Fine-tuning pre-trained models has gained significant success in delivering SOTA results across various NLP tasks. In the absence of centralized data, Federated Learning (FL) helps the model to benefit from clients' private data for fine-tuning. However, due to the limited communication, computation, and storage capabilities of edge devices and the huge sizes of popular pre-trained models, efficient fine-tuning is crucial. This work explores the opportunities and challenges of applying parameter-efficient fine-tuning (PEFT) methods in FL for language tasks. Specifically, our investigations reveal that with increasing data heterogeneity across users, the gap between fully fine-tuning the model and employing PEFT methods widens. To bridge this performance gap, we propose a method, 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 $\sim 1\%$ 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. To solve both problems we can benefit from Federated Learning (FL) [25, 18]; users collaboratively train a common model locally and only share their model update with a server that orchestrates the training process over multiple training rounds.

Although FL has already proven beneficial in various domains [15], 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 for clients with heterogeneous data distribution in pre-trained language models.

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

Large pre-trained language models have proven to perform very well even in the zero-shot setting [5]. However, as the tasks get more specialized, these models require fine-tuning to enhance their performance [14]. FL is a promising solution to provide privacy for fine-tuning, but asking the clients to fine-tune the models and communicate the update has downsides. In particular, fine-tuning can be computationally expensive, especially for larger models. 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 is also challenging, especially in memory-constrained scenarios (e.g., 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 overhead. 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) [14, 24, 21] 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.

In this work, we first explore the performance of the existing PEFT method in FL. We observe that the gap between Full Fine Tuning (FFT) and PEFT increases with clients' data heterogeneity. To this aim, we propose a new algorithm, SLoRA, designed for FL that can achieve first parameter efficiency, second, reduce training and communication cost, and finally close the gap between PEFT and FFT.

2 Related Work

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 within the original pre-trained model for each task [10]. 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 where the modules are inserted into the model. Adapter and its variants add bottleneck trainable modules serially to the model components [13, 27, 11]. Another approach is to add modules in parallel to model parameters such as LoRA [14] and prefix or prompt tuning added in parallel to the attention heads [24] or embeddings [21].

PEFT in FL. Recent studies [31, 33] have investigated the performance of various PEFT methods within the context of FL for vision and NLP tasks considering different aspects, 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.

Our study differs from the prior works in several key aspects. Firstly, we specifically study PEFT for 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 [7, 12, 1, 26, 16, 23, 22]. 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 [7, 12, 1, 26]. Other approaches utilize sparse training to optimize a model that is sparse both at the client and server sides [4, 3, 28, 22].

Given efficient learning and PEFT, both share a common goal of reducing the training complexity for the clients; they may seem very similar at first glance. However, PEFT also focuses on the unique aspect of storage load for 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 [8, 9, 6]. This can result in a huge storage penalty as sparsity patterns can differ across different tasks. On the other hand, PEFT retains the full pre-trained model but applies extremely sparse with ($\sim 1\%$) density updates [14, 13, 32, 27, 11] 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. 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, differing on where the module is added. Holusby [13] places a bottleneck module after the multi-head attention and feed-forward block in each Transformer layer. Pfieffer [27] places a bottleneck module only after the feed-forward block in each Transformer layer. In LoRA [14], the bottleneck module can be parallel to any dense layer in the model. Finally, BitFit [32] 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 challenges in FL is performance degradation caused by heterogeneous data distributions [15]. While this is a well-documented phenomenon reflected in FFT in FL, we observe that the performance penalty is even more substantial when using PEFT. 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.

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

Here, we present Low-Rank Adaptation (LoRA) [14] since it's the SOTA method for PEFT of large pre-trained language models, and our proposed algorithm adopts this method. The key idea of LoRA is that instead of fully fine-tuning the pre-trained weight matrix $\mathbf{W}_0 \in \mathbb{R}^{d\times d}$, its update is constrained with a low-rank decomposition $\mathbf{W}_0 + \Delta\mathbf{W} = \mathbf{W}_0 + \mathbf{BA}$, where $\mathbf{B} \in \mathbb{R}^{d\times r}$, $\mathbf{A} \in \mathbb{R}^{r\times d}$, and $r << d$, and only $\mathbf{B}$ and $\mathbf{A}$ are trained ($\mathbf{W}_0$ is frozen).

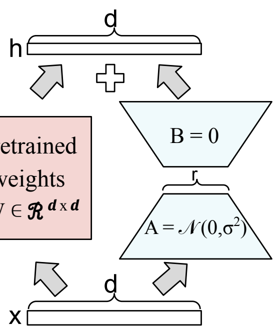


Figure 1: LoRA Block

Fig. 1 shows LoRA implementation, where a parallel module of a down projection matrix $\mathbf{B}$ followed by the up projection matrix $\mathbf{A}$ in parallel to the 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 (i.e., $\Delta\mathbf{W} = \mathbf{BA}$ is zero at the beginning of training). The modified forward pass after adding the LoRA module is given below where r is LoRA rank and β is a constant in r.

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

According to eq. 1, the output from the LoRA module is added coordinate-wise to the output of the original model. The scaling $\frac{\beta}{r}$ is used to reduce the need to re-tune hyper-parameters when varying r.

4 Our Proposed approach (Primed-LoRA)

In centralized learning, LoRA consistently shows promising performance in different tasks and closely follows the FFT accuracy. This still holds in FL with homogeneous data distribution (larger α) as shown in Fig. 2. However, in highly non-IID data distribution, LoRA fails to reach close to the FFT performance and suffers from a slower convergence rate compared to FFT. We hypothesize that the initialization of LoRA Blocks (Fig. 1, A with random noise and B with 0) can be one of the reasons behind this problem. While this initialization works in centralized settings where the data is ample and concentrated, it can potentially slow down the fine-tuning process in FL.

Data-driven Priming of LoRA. Based on our hypothesis, a better starting point for LoRA might improve its performance. Therefore, we propose a two-stage PEFT method, Primed-LoRA, based on the LoRA algorithm. In Stage 1, clients collaboratively find a mature starting point to prime the LoRA blocks. Then, in Stage 2, they use the LoRA algorithm with the learned initializers from Stage 1. In the remainder of the section, we discuss different ways of priming LoRA and their properties.

4.1 Full fine-tuning for priming LoRA

A straightforward approach for Stage 1 is to perform a full fine-tuning for a few rounds in Stage 1 of Primed-LoRA and then use SVD matrix decomposition to 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 $\Delta W \in \mathbb{R}^{d\times d}$. We use SVD to derive a low-rank approximation of $\Delta W = \mathbf{B} \times \mathbf{A}$, with $\mathbf{B} \in \mathbb{R}^{d\times r}$, $\mathbf{A} \in \mathbb{R}^{r\times d}$ that is used 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 stage 2, where clients only update the LoRA blocks and share those parameters with the server. As shown in Fig. 3, 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.

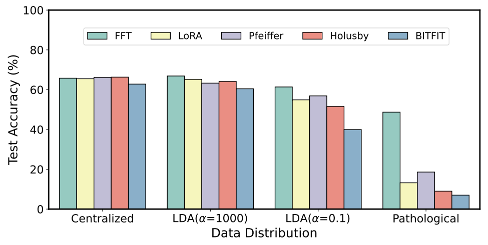


Figure 2: Performance of PEFT methods with different data distributions for 20News group dataset.

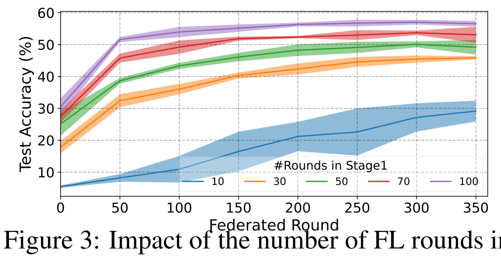


Figure 3: Impact of the number of FL rounds in stage 1 on the final performance in FLoRA for 20News group dataset on Albert.

Cost of FLoRA In FLoRA, adding stage 1 successfully enhances the performance while achieving parameter efficiency. However, the communication and computation cost of training at this stage is the same as full fine-tuning. Therefore, while FLoRA shows that Primed-LoRA can 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.

One can observe that there are two parameters involved in cost of stage 1; 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. 3, 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 clients' update size. 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 them.

4.2 Sparse Fine-tuning for priming LoRA

Sparse Fine-Tuning (SFT) [2, 32] aims to achieve parameter efficiency by sparsifying the updates. We opt to employ the approach of [2], but all the works follow similar ideas. [2] propose to generate sparse mask (binary mask such that 1's indicates the position of trainable weights in each round) is to find the *top-K most important* weights based on their contribution in FFT. The weights that change the highest – from the original pre-trained values – in FFT are the most important one in SFT. Clearly, to detect such weights, a new warm-up step is required to fine-tune the weight for several rounds.

Sparse Fine-tuning in Stage 1. [2] is designed for the centralized setting where the data can be utilized in finding the mask. However, in the federated setting, the server cannot fine-tune the model

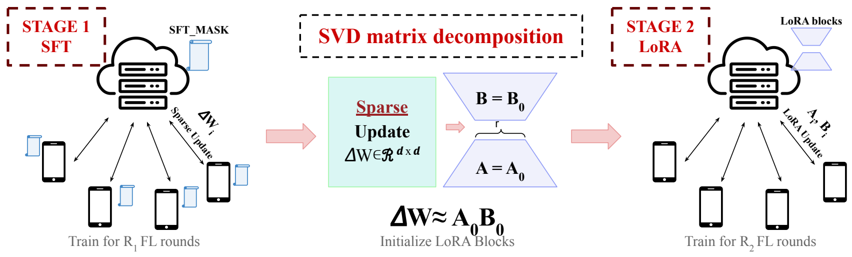


Figure 4: Overview of SLoRA; First server initializes a mask, and clients only update the parameters in the mask. Then, the updates are decomposed into LoRA blocks for the initialization in Stage 2.

because it does not have data. Clients can individually train the model and come up with personalized masks, but differences in clients' masks increase the density of the aggregated update and reduce parameter efficiency. Alternatively, for clients to find the important weights together, they need 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 generate a random data-independent binary mask with uniform density for all layers at the beginning. Then, the clients only train the weights using this mask. Thus, the update density does not change, and clients can benefit from reduced communication.

Primed-LoRA in Stage 2. Stage 2 in SLoRA follows the same procedure as discussed in Section 4.1. Algorithm 4 in the Appendix C summarizes the different steps in SLoRA.

5 Experiments

Settings. We focus on two models, Albert [19], and DistilBERT [30], and two datasets, News Category and 20News group [20]. Since we rely on non-IID label distribution, we focus on classification tasks. Due to lack of space, more details and results can be found in the Appendix B. The total number of clients is 100 in all the experiments. The number of participants in every round is 20 clients for News Category pathological non-IID and 10 otherwise. We used FedAvg [25] as our aggregation method. We mainly focus on the accuracy of the final global model. All the experiments are performed for 5 different seeds, and the average of the 3 best results are reported.

5.1 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 LDA [29], with parameter α (smaller α means more heterogeneity). Also, we added pathologically non-IID similar to [25].

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

Table 1: Parameter r, which we use to calculate the density.

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

6 Evaluation

Data Heterogeneity. Fig. 2 shows the impact of data heterogeneity on the performance 20New group on Albert. The gap between PEFT and FFT grows by making the data more non-IID, which indicates the necessity of a new PEFT algorithm tailored for FL. (More results in Appendix D).

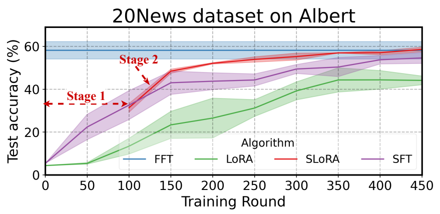


Figure 5: The performance of SLoRA.

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

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

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

Table 3: Performance and training cost for the 20News group on Albert.

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

Table 4: The training time and # training rounds for the 20news dataset on Albert.

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 improve performance with the cost of an increased communication budget. Other results are in Appendix E

6.1 Performance of SLoRA

To show the efficacy of the proposed stages, we compare our method with stage 1 only (SFT) and stage 2 only (LoRA). Here, we only consider pathologically non-IID as discussed earlier in this setting; PEFT experiences a significant drop in its performance. For 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 SLoRA, we add the LoRA module by utilizing the SVD decomposed model update from Stage 1 with a higher target update sparsity. Specifically, the LoRA modules are added to each dense layer in the model, except for the embedding and classification layers ($r = 10$ for the multi-head attention block and the feed-forward block and $r = 18$ for the pre-classification layer). The size of the added module is $0.14M$ parameters compared to $11.7M$ parameters of the original Albert model representing only 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. We evaluate the performance of SLoRA for the 20News group dataset on Albert and DistilBERT in Fig. 5 and Fig. 8. As shown 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 FL. Regarding the model training time (i.e., computation time), we report the duration based on a single GPU. Table 4 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 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 4 summarizes the number of training rounds for the baselines. Despite the baselines having even higher communication rounds, Table 3 demonstrates that SLoRA still achieves comparable accuracy to FFT, 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 FL rounds is another crucial aspect, especially when considering client availability. Therefore, we compare the performance of SLoRA to the baselines with a fixed number of training rounds. As depicted in Fig. 5, the performance gap between SLoRA and the baselines (SFT and LoRA) increases to 4% and 14.2%, respectively.

Comparison with a Concurrent Work. A concurrent work[17] addresses a similar problem. However, our study of PEFT under data heterogeneity is more comprehensive, covering two LLM with models and two different datasets with different data distributions (pathological and LDA) and the impact of update size compared to only LDA and one model. For the proposed methods, we adopt LoRA compared to the Houlsby adapter. The benefit of using LoRA is the reparametrization meaning that once the model is trained using PEFT, we can integrate the modules into the original parameters without increasing the model size during inference. However, this is not possible for the Holsuby adapters, as a nonlinear activation is used between the up and down projections. Also, [17] assumes that each user trains a hypernetwork locally in each training round, which puts additional costs on the training. In contrast, SLoRA does not have the extra computations, and the final model has the same size as the original model without performance degradation. Finally, we were unable to reproduce their results since the code and key aspects of the algorithms were missing.

7 Conclusion

We investigate employing PEFT methods for fine-tuning language models in the FL to reduce communication/storage costs. After observing the poor performance of PEFT compared to FFT for heterogeneous clients, we propose SLoRA to reduce the costs while maintaining the FFT performance.

Acknowledgement

This material is based upon work supported by Device Solutions Research America (DSRA-S.LSI), Samsung Semiconductor Inc. (SSI), USA.

References

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

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

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

[4] Sameer Bibikar, Haris Vikalo, Zhangyang Wang, and Xiaohan Chen. 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, 2022.

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

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

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

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

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

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

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

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

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

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

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

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

[17] Yeachan Kim, Junho Kim, Wing-Lam Mok, Jun-Hyung Park, and SangKeun Lee. Client-customized adaptation for parameter-efficient federated learning. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1159–1172, 2023.

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

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

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

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

[22] Ang Li, Jingwei Sun, Xiao Zeng, Mi Zhang, Hai Li, and Yiran Chen. 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, 2021.

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

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

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

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

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

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

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

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

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

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

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

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 [2].

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 Details of Datasets

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.

C Algorithm

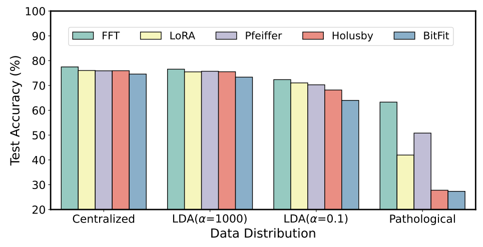
Algorithm 1 Overview of Primed LoRA

```
1: $R_i$: 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
2: # Stage 1
3: if $Algorithm = SLoRA$ then
4:     $SFT\_Mask = generateMask(W_0, d_1)$
5: else
6:     $SFT\_Mask = 1$
7: end if
8: for $R = 1$ to $R_1$ do
9:     for $k = 1$ to $K$ do
10:        $W^k_{R-1} = train(W_R, SFT\_Mask, E)$
11:    end for
12:    $W_R = aggregate(W^{0,..,k}_{R-1})$
13: end for
14: $\Delta W = W_R - W_0$
15: $[\mathbf{A}, \mathbf{B}]^0 = SVD(\Delta W, r)$
16: # Stage 2
17: for $R = 1$ to $R_2$ do
18:     for $k = 1$ to $K$ do
19:         $[\mathbf{A}, \mathbf{B}]^k_R = trainLoRA([\mathbf{A}, \mathbf{B}]_{R-1}, E)$
20:     end for
21:     $[\mathbf{A}, \mathbf{B}]_R = aggregate([\mathbf{A}, \mathbf{B}]^{0,...,K}_{R-1})$
22: end for
```

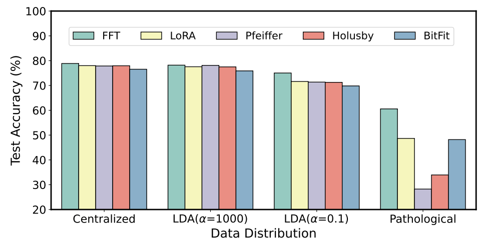
[2]SVD decomposition is not unique, but we are concerned with the decomposition where the singular values are organized in descending order

D Impact of Data Heterogeneity on PEFT Methods

Here we repeat the experiments in the evaluation section for XX and News Category datasets on Albert and DistilBERT models. A similar trend to Fig. 2 can be also observed in Fig. 6 and 7.

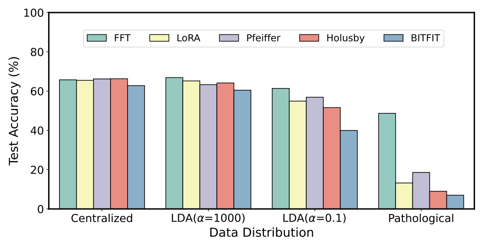


(a) News Category dataset on Albert

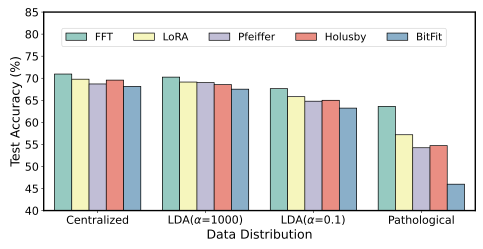


(b) News Category dataset on DistilBERT

Figure 6: Performance of PEFT methods for different data distributions for News Category dataset.



(a) 20News group dataset on Albert



(b) 20News group dataset on DistilBERT

Figure 7: Performance of PEFT methods for different data distributions for 20News group dataset.

E Impact of Update size.

In tables 5, 6, and 7, we show the impact of update size for different settings. In all the settings, as we expected, updates with higher density have better performance and the performance considerably drops by increasing the data heterogeneity.

Table 5: Impact of update density of PEFT methods 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

F Performance of SLoRA on DistilBERT

Fig. 8 shows the performance of SLoRA, LoRA and SFT for 20News group dataset on DistilBERT.

Table 6: Impact of update density of PEFT methods 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 7: Impact of update density of PEFT methods 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

G Performance and cost of 20News group on DistilBERT

Table 8 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 8: The training time (i.e., computation time) of different methods and # training rounds 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 9 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 table, SLoRA enjoys better performance compared to FFT and still has better training time.

The results for the News category dataset on Albert and DistilBERT are summarized in Table 10 and Table 11, respectively. As depicted in the tables, the same observations still hold for this dataset as well.

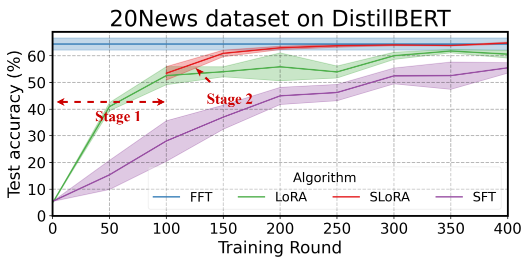


Figure 8: The performance of SLoRA using for 20News group dataset on DistilBERT.

Table 9: 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**

Table 10: 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**

Table 11: 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**