FLASC: Federated LoRA with Sparse Communication

Anonymous authors
Paper under double-blind review

Abstract

Low-rank adaptation (LoRA) is a promising method for finetuning models in communication-constrained settings such as cross-device federated learning (FL). Prior work has explored ways to improve the efficiency of LoRA in federated settings by imposing additional sparsity constraints. However, as we show, existing methods for sparse LoRA not only harm accuracy but can in fact *increase* overall communication costs. We instead propose **FLASC**, a simple approach with two key components: First, **FLASC** combines LoRA with *sparse communication*, which outperforms baselines such as using a lower LoRA rank or pruning LoRA weights. Second, **FLASC-Search** efficiently searches the space of sparsity-and-rank configurations by iteratively comparing pairs of configurations and increasing either the rank or density. Across four FL datasets, we demonstrate that **FLASC** outperforms existing sparse LoRA methods with up to 20% higher accuracy or 10× less communication. Our work highlights the importance of considering the constraints of existing efficient finetuning methods and provides a simple and competitive baseline for future work in federated finetuning.

1 Introduction

As pretrained models continue to advance state-of-the-art performance in a variety of domains, it is critical to develop methods to efficiently finetune models in low-resource settings. In this work, we consider the cross-device federated learning (FL) setting which seeks to train models across a network of decentralized and heterogeneous clients (McMahan et al., 2017a). A major bottleneck in FL is the cost of *uploading model updates* to the server, which can significantly slow down finetuning (Konečný et al., 2017).

Recently, parameter-efficient finetuning (PEFT) has emerged as an effective way to reduce costs in both centralized and federated settings (Houlsby et al., 2019; Hu et al., 2021; Zhang et al., 2023c). We focus on *low-rank adaptation* (LoRA), a popular method that injects trainable low-rank adapters into a model and freezes the pretrained backbone. To further improve LoRA, prior works in centralized ML apply sparsity by pruning individual entries or groups of weights during training (Wu & Chen, 2022; He et al., 2022; Zhang et al., 2022; 2023a). Similar works in the FL setting also consider sparsity, but focus on improving LoRA in heterogeneous and private FL settings (Cho et al., 2023; Babakniya et al., 2023a; Sun et al., 2024). Unfortunately, prior works fail to address the key challenge of *configuring LoRA*, as they must still tune a rank hyperparameter and additional hyperparameters related to sparsity—costs that can quickly overshadow the savings attained from sparsity. Further, even with optimal hyperparameters, these methods can perform worse than simply using LoRA with an equal communication budget (see Fig. 1).

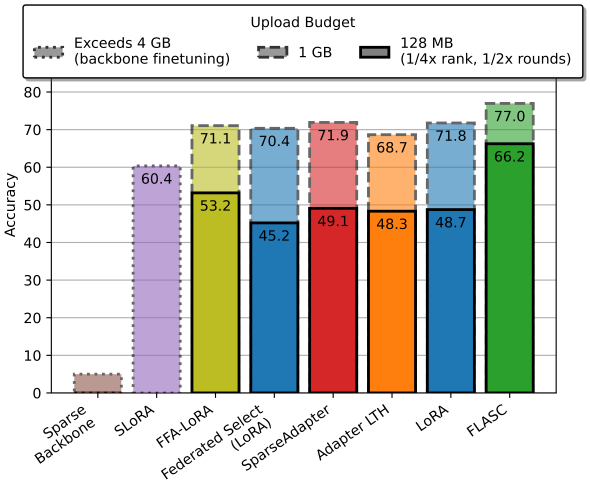

Figure 1: We train FL models on 20NewsGroups while applying sparsity and/or LoRA. LoRA itself can be more efficient than existing methods, while **FLASC** outperforms LoRA in constrained settings.

To address these issues, we present a seemingly overlooked baseline in communication-efficient federated finetuning: **FLASC** (**F**ederated **L**ow-Rank **A**daptation with **S**parse **C**ommunication). Our method has two key contributions: First, **FLASC** computes dense local updates while only communicating sparse updates, which we show greatly improves performance over approaches that reduce the LoRA rank or freeze LoRA weights. Second, we present a hyperparameter search method **FLASC-S** (**FLASC**-**S**earch) that makes it practical to realize the benefits of sparsity. This method relies on a key tradeoff observed when applying sparsity via **FLASC**: reducing the rank is more helpful for reducing redundant communication under a large budget, while sparsity is more helpful for improving utility under a small budget. In terms of both a single training run and the total cost of tuning, our method significantly improves the communication-efficiency of federated LoRA. Furthermore, we find that **FLASC** is equally robust to LoRA in terms of other common FL concerns such as heterogeneity and privacy. Overall, our work makes the following contributions:

1. We propose **FLASC**, a simple method which reduces the communication cost of LoRA by applying unstructured sparsity to upload communication. **FLASC** can reduce communication of LoRA by up to 10× without harming accuracy, or improve accuracy by up to 20% under the same communication budget. Despite its simplicity, **FLASC** has been overlooked as a baseline by prior work in efficient FL.

2. We study the interaction between LoRA rank and sparse communication in **FLASC**. We find that sparsity is more important at small communication budgets, while reducing the rank is more important at large budgets. Based on this observation, we propose **FLASC-S**, an efficient tuning method which sequentially tunes the density followed by rank, allowing **FLASC** to save communication even when considering the cost of hyperparameter tuning.

3. We conduct extensive experiments which show that both LoRA and **FLASC** are effective when deployed in settings where other common challenges in FL are present, such as data/systems heterogeneity or differential privacy. In fact, we find that **FLASC** outperforms other sparsity and freezing-based methods that were designed specifically for these concerns.

2 Related Work

Communication-efficient federated learning. Upload communication is a key bottleneck in cross-device FL (Konečný et al., 2017). While large pretrained models can significantly improve utility in FL applications (Radford et al., 2018; Nguyen et al., 2022), these models present new challenges with communication and finetuning at the edge. Many types of methods have been explored to reduce FL communication costs, including quantization (Reisizadeh et al., 2020; Ozkara et al., 2021), sparsity (Horvath et al., 2021; Stripelis et al., 2022; Isik et al., 2022; Huang et al., 2022), parameter-efficient finetuning (Chen et al., 2023), and their combinations (Caldas et al., 2018b; Ro et al., 2022; Babakniya et al., 2023a).

Sparsity and pruning. In centralized ML, *pruning* is a popular area of research that aims to reduce compute and storage costs by sparsifying and freezing weights (Frankle & Carbin, 2018; Dettmers & Zettlemoyer, 2019; Sung et al., 2021). Early studies of sparsity in distributed and federated learning focus on the use of *sparse communication* only (Basu et al., 2019; Gao et al., 2021; Mitra et al., 2021), while more recent FL methods utilize pruning by sending a sparse model to clients and finetuning in a sparse-to-sparse manner (Shah & Lau, 2021; Qiu et al., 2021; Bibikar et al., 2022; Babakniya et al., 2023b). In this work, we study *unstructured* (weight-level) as opposed to structured (feature-level) sparsity, due to the limited utility of structured sparsity in both centralized (Liu et al., 2018) and FL settings (Caldas et al., 2018b; Cheng et al., 2022).

Parameter-efficient finetuning (PEFT). PEFT reduces the cost of finetuning by training a small number of parameters and freezing the rest of the model (Ding et al., 2022). In this work, we focus on *low-rank adaptation* (LoRA), a popular reparameterization-based method which has two advantages: First, LoRA parameters can be merged with the backbone after training in order to maintain the same inference costs (Houlsby et al., 2019; Hu et al., 2021). Second, LoRA tends to outperform PEFT methods which finetune the backbone (Guo et al., 2021; Zaken et al., 2022; Sung et al., 2021; Gong et al., 2022).

Efficient LoRA. Our work is most similar to recent works in the centralized setting that improve LoRA's efficiency via *unstructured pruning* (Wu & Chen, 2022; He et al., 2022). Other works consider improving LoRA with structured sparsity (Ding et al., 2023; Liu et al., 2024), quantization (Xu et al., 2023; Dettmers et al., 2024), and flexibly adjusting the rank (Zhang et al., 2022; 2023a). LoRA itself can also be used to efficiently update or guide pruning of the backbone parameters (Zhao et al., 2023; Zhang et al., 2023b; Zhao et al., 2024; Ribeiro et al., 2024). In contrast to these prior works, we focus on how to make LoRA more communication-efficient in the context of FL.

Federated LoRA. Many works have observed that LoRA can reduce the communication cost of FL finetuning (Sun et al., 2022; Malaviya et al., 2023; Zhang et al., 2023d; Nguyen et al., 2024). However, as we show, using LoRA alone is a rigid approach for communication reduction—potentially leaving a substantial amount of cost savings on the table. Our work takes an approach similar in spirit to ComPEFT, which considers merging compressed LoRA adapters. However, they target the application of merging adapters from multiple sources in a one-shot fashion and use more complex merging procedures (Yadav et al., 2023). In contrast, we study how to reduce both upload and download communication over multiple rounds of FL training and analyze these modifications in the context of standard FL optimization methods such as FedAvg and FedAdam (McMahan et al., 2017a; Reddi et al., 2020). Beyond communication, prior works have studied the use of LoRA in FL with respect to other concerns, such as data heterogeneity (Kim et al., 2023b; Yi et al., 2023; Lu et al., 2024; Jiang et al., 2024; Babakniya et al., 2023a), systems heterogeneity (Cho et al., 2023; Bai et al., 2024), and differential privacy (Sun et al., 2024). Overall, we show that our use of sparse uploads can reduce LoRA communication costs without harming robustness to heterogeneity and privacy—resulting in performance that matches or even exceeds the performance of these specialized methods.

3 FLASC: Federated Low-Rank Adaptation with Sparse Communication

We first introduce several sparsity baselines and then discuss benefits of **FLASC** relative to these methods.

Federated LoRA. LoRA is a PEFT method that freezes the pretrained backbone $W \in \mathbb{R}^{d\times k}$ and constrains its update $\Delta W \in \mathbb{R}^{d\times k}$ to be a product BA where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$ are newly inserted low-rank parameters (Hu et al., 2021). To apply LoRA to FL, we set (A, B) as the trainable weights for FedAdam. *See Appendix A for details on FedAdam and all other methods introduced in this section.*

Sparse LoRA Baselines. While the communication cost of (A, B) can be reduced by initializing (A, B) with a smaller rank r, **"sparse LoRA"** methods use *unstructured (weight-level) pruning* for more expressive updates. To the best of our knowledge, there are two such baselines: Adapter LTH (Wu & Chen, 2022) and SparseAdapter (He et al., 2022). While these methods consider pruning LoRA in the context of centralized learning, adapting them to FL is straightforward. For each matrix of trainable parameters P, we maintain a binary mask M which may be adjusted between FL rounds. *Pruning* refers to applying M to all operations throughout FL, namely communication, inference ($P \odot M$), and gradient computation ($\nabla_P \odot M$). After applying a mask with $d\%$ density, the new cost of communicating P is slightly more than $d\%$ of the original cost. The small additional cost comes from communicating M, which we discuss in Appendix A.

Adapter LTH (Wu & Chen, 2022) applies iterative magnitude pruning (IMP) (Frankle & Carbin, 2018), which alternates between pruning a small fraction of the lowest magnitude weights and re-training the remaining weights. We use an efficient version of IMP which skips weight re-initialization after pruning (Renda et al., 2019); this keeps communication costs competitive with the non-pruned baseline.

SparseAdapter (He et al., 2022) applies pruning-at-initialization (PaI) (Lee et al., 2018) prior to training. However, standard magnitude-based metrics for determining the parameters to prune are unsuitable for LoRA. Since the B matrix in LoRA is initialized to all zeros, magnitude-based sensitivity prioritizes pruning the B weights. If the density is low enough, this entirely prunes B and leaves A unable to train. To fairly evaluate SparseAdapter, we perform one initial round of FL before applying magnitude pruning.

Sparse Backbone Baselines. In addition to sparse LoRA methods, we evaluate **"sparse backbone"** methods which prune the backbone W (i.e. do not consider LoRA) and are proposed specifically for FL. FedSparsify-Global (Stripelis et al., 2022) applies IMP to FL; it prunes the smallest global parameters after every few rounds. SPDST (Babakniya et al., 2023b) applies PaI to FL; it prunes random global parameters in each layer with a density proportional to that layer's sensitivity. We also evaluate two methods where the server maintains a dense global model: Federated Select applies temporary magnitude pruning before the model is downloaded. FedSpa is similar, but applies a personalized local mask at each client.

Method	**Description**	**Upload (↓)**	**Acc (↑)**
Sparse backbone: Train W and reduce communication by 25 or 50%			
Full Finetune	-	100GB	78.0
FedSparsify (Stripelis et al., 2022)	Prune a fraction of remaining parameters after model aggregation	75GB 50GB	65.3 5.0
SPDST (Babakniya et al., 2023b)	Prune random parameters before FL; amount based on layer-wise sensitivity	75GB 50GB	75.6 5.0
Federated Select (Charles et al., 2022)	Learn dense global model; apply a global mask before download	75GB 50GB	76.8 5.0
FedSpa (Huang et al., 2022)	Learn dense global model and a personalized mask for each client	75GB 50GB	5.0 5.0
Sparse LoRA: Train A, B and reduce communication by 75%			
LoRA ($r = 16$) (Hu et al., 2021)	-	4GB	78.2
Adapter LTH (Wu & Chen, 2022)	Prune a fraction of remaining LoRA parameters after model aggregation	1GB	68.7
SparseAdapter (He et al., 2022)	Prune parameters before FL; based on individual parameter sensitivity	1GB	71.9
FLASC (ours)	Sparsify only communication	**1GB**	**78.1**

Table 1: We finetune GPT2 on 20NewsGroups for 200 rounds and reduce communication with sparsity, LoRA, or both. LoRA is much more efficient than "Sparse backbone" methods. "Sparse LoRA" methods (Adapter LTH, SparseAdapter) can further reduce the communication of LoRA, but harm accuracy. **FLASC** only sparsifies communication and is able to match the accuracy of LoRA with 4× less communication.

Performance limitations. Table 1 shows that **sparse backbone** approaches are much less efficient than LoRA. When we prune W by 50%, all sparse backbone methods drop to 5% accuracy, indicating that pretrained weights are important for finetuning. Meanwhile, simply applying LoRA reduces communication by 90% without any drop in accuracy. On top of this, the two **sparse LoRA** baselines can further reduce the communication of LoRA by 75% while outperforming the sparse backbone methods. Since sparse backbone methods are unable to compete with LoRA, we only consider LoRA and sparse LoRA methods as baselines throughout this work.

FLASC. When comparing sparse LoRA methods, our key observation in Table 1 is that pruning severely hurts accuracy. Pruning applies sparsity to both communication and local training, but this is excessively restrictive. Since our goal is to reduce communication, **FLASC** relaxes the finetuning constraint and only applies sparsity to communication. This simple change enables **FLASC** to match the performance of LoRA with 4× less communication. We describe **FLASC** in Algorithm 1 and Figure 2. The key difference between **FLASC** and pruning methods lies in the finetuning step; we finetune all entries of $P \odot M$ (L11) and only sparsify the update ΔP during upload (L17-18). This allows for much more expressive local updates compared to only finetuning the sparse non-zero entries of $P \odot M$. Furthermore, while other baselines propose complex methods for selecting the mask M, **FLASC** uses a simple TopK operation with ℓ_1 criterion.

Algorithm 1: FLASC

```
1  Require: $r$, $d_{\text{down}}, d_{\text{up}}, \eta_{\text{s}}, \eta_{\text{c}}, S_{\text{server}}, S_{\text{client}}$
2  $P \leftarrow$ Initialize LoRA with rank $r$
3  optim ← torch.nn.optim.Adam($P, \eta_{\text{s}}$)
4  for $s_{server} = 1, \ldots, S_{server}$ do
5      $M_{\text{down}} \leftarrow$ TopKMask($|P|, d_{\text{down}}$)
6      Sample clients $c_1, \ldots, c_n$ uniformly
7      at random without replacement
8      # communication and local training
9      for $i = 1, \ldots, n$ in parallel do
10         $P_i = P \odot M_{\text{down}}$
11         $P'_i \leftarrow P_i$
12         for $s_{client} = 1, \ldots, S_{client}$ do
13             $(x, y) \leftarrow$ sample data from $c_i$
14             $\nabla_{P'_i} = \nabla_1 \mathcal{L}(P'_i, x, y)$
15             $P'_i \leftarrow P'_i - \eta_{\text{client}} \nabla_{P'_i}$
16         $\Delta P_i \leftarrow P_i - P'_i$
17         $M_{\text{up},i} \leftarrow$ TopKMask($|\Delta P_i|, d_{\text{up}}$)
18         $\Delta P_i \leftarrow \Delta P_i \odot M_{\text{up},i}$
19     # server takes one step of FedAdam
20     optim.grad ← $\frac{1}{n} \sum_{i=1}^{n} \Delta P_i$
21     optim.step()
```

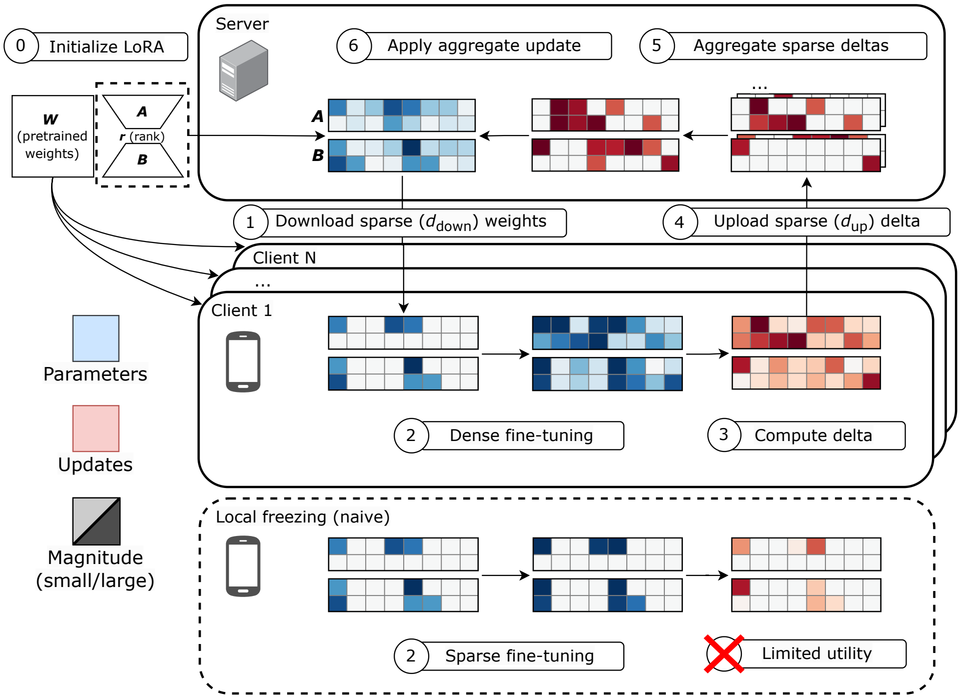

Figure 2: A step-by-step overview of **FLASC** with LoRA rank = 2 and density = 1/4. Step 0 is executed prior to FL. Each round of FL repeats steps 1-6. Blue/red squares indicate the magnitude of weights/updates respectively. **Darker** squares indicate a **larger magnitude**, which is the criterion (ℓ_1) used for sparsity.

FLASC is as compute-efficient as LoRA. Since **FLASC** finetunes all LoRA parameters, it has the same computational costs as LoRA. However, this is not a significant disadvantage compared to the other sparse LoRA baselines, because the benefits of pruning (communication, compute, and storage) are largely limited to communication when pruning LoRA. This is mainly because most compute and memory costs come from the weights and activations of the backbone, not the LoRA adapters (Cai et al., 2020; Kim et al., 2023a). Additionally, LoRA can be merged with the backbone after training, which removes its additional inference costs (Luo et al., 2023). Finally, the already marginal compute benefits of sparse LoRA can only be realized in specific settings with a very high sparsity or specific sparsity patterns (Muralidharan, 2023). Due to these limitations, we instead focus on the more significant benefit of reducing communication in FL.

4 Results

In this section, we test **FLASC** in a variety of FL settings and show that it is effective at handling concerns of communication efficiency (4.1), hyperparameter tuning (4.2), heterogeneity, (4.3, 4.4), and privacy (4.5). Overall, we show that LoRA with sufficient hyperparameter tuning is robust to these concerns and that **FLASC** is able to match the performance of LoRA with up to 10× less communication.

Datasets. We present experiments on CIFAR10, 20NewsGroups, Reddit, and FLAIR (Krizhevsky, 2009; Lang, 1995; Caldas et al., 2018a; Song et al., 2022) (Table 2). CIFAR10 and FLAIR are image datasets, and 20NewsGroups and Reddit are text datasets. For the image datasets, the images are resized to 224 × 224 to match ImageNet (the ViT pretraining dataset). We train with standard data augmentations (random crops and flips). For the text datasets, each example (an email or Reddit comment) is encoded and truncated to a maximum of length of 256 tokens. We use Reddit for two different tasks: the first is a next-token prediction task (Reddit), and the second is a topic classification task (Subreddits). We partition CIFAR10 and 20NewsGroups using a synthetic Dirichlet distribution over the set of labels (Hsu et al., 2019). Reddit and FLAIR are obtained from social media sites (Reddit and Flickr) and are naturally partitioned by user.

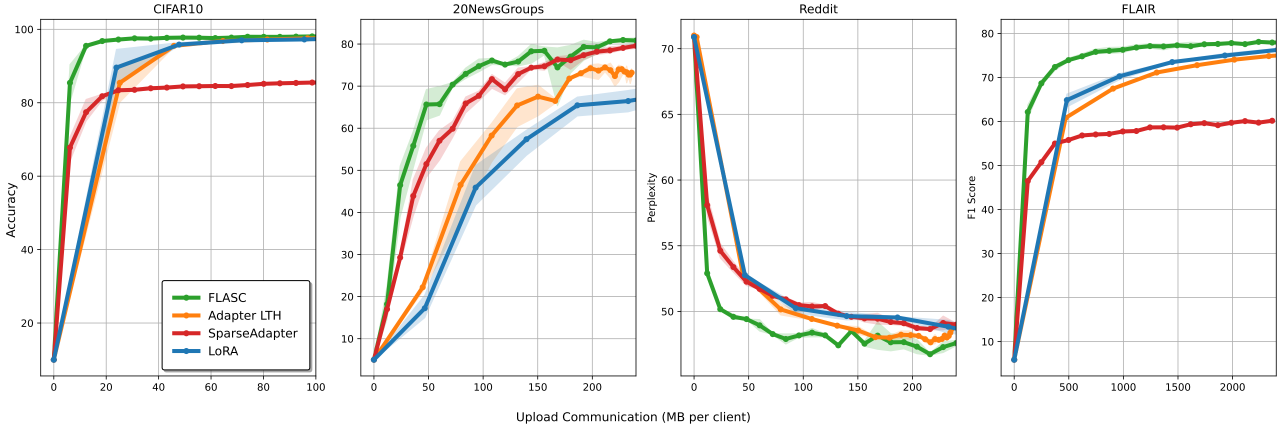

Figure 3: We compare utility vs. communication while training LoRA (rank 16) and reducing communication with sparsity. **FLASC** is the most efficient method, while Adapter LTH (Wu & Chen, 2022) is inefficient early in training and SparseAdapter (He et al., 2022) may converge to a significantly lower utility than LoRA. Shaded bands show the min/mean/max over 3 random seeds.

Model Details. We used ViT-B-16 (85M params) and GPT2-Small (124M params) as the backbone for image and text tasks respectively (Dosovitskiy et al., 2021; Radford et al., 2019). For the image datasets, we finetune a classification head in addition to the LoRA weights. For the text classification tasks (Subreddits and 20NewsGroups), we format the data as a text-to-text task where the model is trained to output the label text (Raffel et al., 2020). For all text datasets, we only finetune the LoRA weights and do not finetune the language modeling head. For all datasets, we use a local batch size of 16. For FLAIR, we sample 200 clients per round, finetune for 2 local epochs, and communicate for up to 5000 rounds. For the other datasets, we sample 10 clients each round, finetune for 1 local epoch, and communicate for up to 400 rounds. Full details on preprocessing and hyperparameters are provided in Appendix B.4.

4.1 Communication Efficiency

First, we show each method's utility as a function of communication used during a single training run. In Figure 3, we compare LoRA and the three sparse LoRA methods introduced in Table 1: Adapter LTH (Wu & Chen, 2022), SparseAdapter (He et al., 2022), and **FLASC**. We use rank 16 for all methods. The sparsity ratio of Adapter LTH (i.e. fraction of parameters removed per round) is tuned separately on each dataset to either 1% or 2%. We use a density of 25% for SparseAdapter and **FLASC**. While we use 25% density across all datasets in this experiment, we show how other methods are unable to handle lower densities in Fig. 5. Across all tasks, **FLASC** matches or exeeds the performance of LoRA while using 3–10× less communication. In contrast, the other two methods cannot reliably match the performance of LoRA. For three out of the four datasets, SparseAdapter converges to a significantly lower accuracy than LoRA. Adapter LTH has limited benefits due to its iterative nature; in early rounds, it uses similar communication as LoRA, while in later rounds, the adapters are too sparse to continue training and performance plateaus.

Dataset	#Clients	#Examples	#Classes	Metric
CIFAR10	500	50K	10	Accuracy (↑)
20NewsGroups	350	20K	20	Accuracy (↑)
Reddit (next-token prediction)	20K	2M	50257	Perplexity (↓)
Subreddits (topic classification)	100	35K	50	Accuracy (↑)
FLAIR	41K	345K	17	F1 Score (↑)

Table 2: Training partition statistics of the datasets.

Communication savings. In Figure 4, we measure the upload communication that each method needs to reach two target perplexity (PPL) values on Reddit: 53 PPL and 50 PPL. SparseAdapter and Adapter LTH reduce per-round communication but take significantly more rounds to reach the same perplexity as LoRA, which leads to limited overall benefits. For example, in the 50 PPL setting, SparseAdapter ends up using the same communication as LoRA, since although it reduces per-round communication by 4×, it also requires 4× as many rounds to match the performance of LoRA. For **FLASC**, we test two upload density values of 1/4 and 1/16. **FLASC** with 1/4 density performs well and does not require any additional rounds to match the utility of LoRA, which results in exactly 1/4 the communication of LoRA. When we further decrease the density to 1/16, **FLASC** requires additional rounds to train, but saves even more communication than 1/4 density. For example, in the 50 PPL setting, **FLASC** with 1/16 density requires $16 * 62/600 \approx 1.65\times$ more rounds than LoRA, but uses $600/62 \approx 10\times$ less upload communication overall.

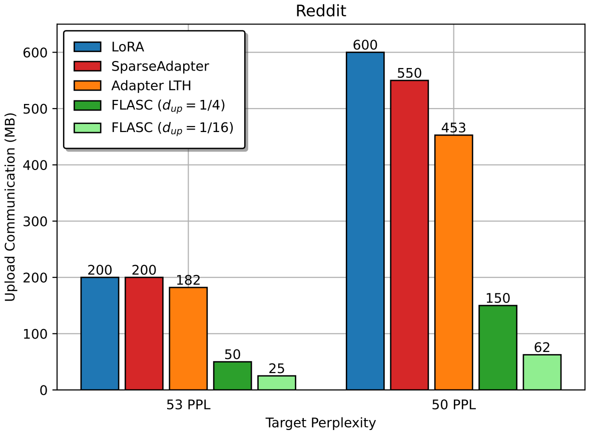

Figure 4: We measure upload communication (↓) needed to reach a given perplexity (53 or 50 PPL) on Reddit. **FLASC** (with either 1/4 or 1/16 density) saves significantly more communication than other sparse LoRA baselines.

Sparsity without freezing. In Figure 5, we compare **FLASC** to SparseAdapter and Federated Select, which are both introduced in Section 3. These two methods respectively correspond to two simple adjustments to **FLASC**: *server-level* and *client-level* freezing. SparseAdapter is an example of server-level freezing. After pruning the model, the zeroed weights are globally frozen; neither the server or clients adjust the mask. Federated Select (Charles et al., 2022) is an example of client-level freezing. The server temporarily prunes the model and freezes the zeroed weights during the current round. In other words, clients only download and finetune the non-zero weights. However, unlike SparseAdapter, Federated Select adjusts the mask across rounds and can potentially discover more useful weights. While Federated Select is proposed in the context of backbone finetuning (and is evaluated as such in Table 1), we apply it to LoRA in Fig. 5. Finally, **FLASC** does not freeze at all; it allows clients to finetune the entire LoRA module and upload sparse updates with varying masks.

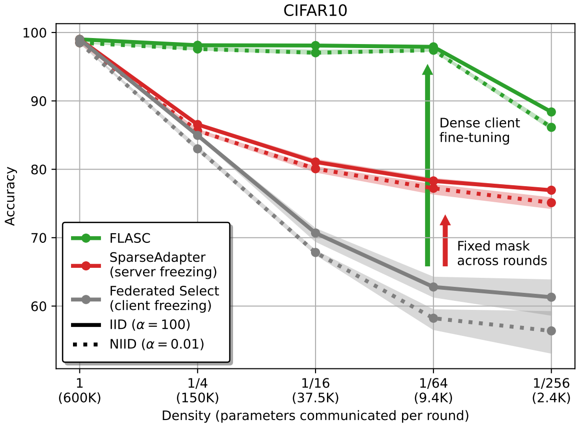

Figure 5: We compare **FLASC** to two ways of freezing weights. All methods use the same amount of communication. SparseAdapter and Federated Select freeze weights, which significantly harms accuracy. **FLASC** does not freeze weights and only sparsifies communication.

Although freezing generally harms utility, server-level freezing is a simple yet competitive baseline which can outperform more complex methods which dynamically adjust the sparse mask between rounds (Babakniya et al., 2023b). Our results in Figure 5 support this claim, and shows that SparseAdapter works better than Federated Select across all density values despite employing a relatively simpler method. Finally, **FLASC** greatly improves utility over both methods by only considering sparse communication without any freezing at all. As discussed in Section 3, freezing LoRA parameters has relatively small compute savings, which motivates us to leverage the utility of dense local updates and then reduce commmunication afterwards.

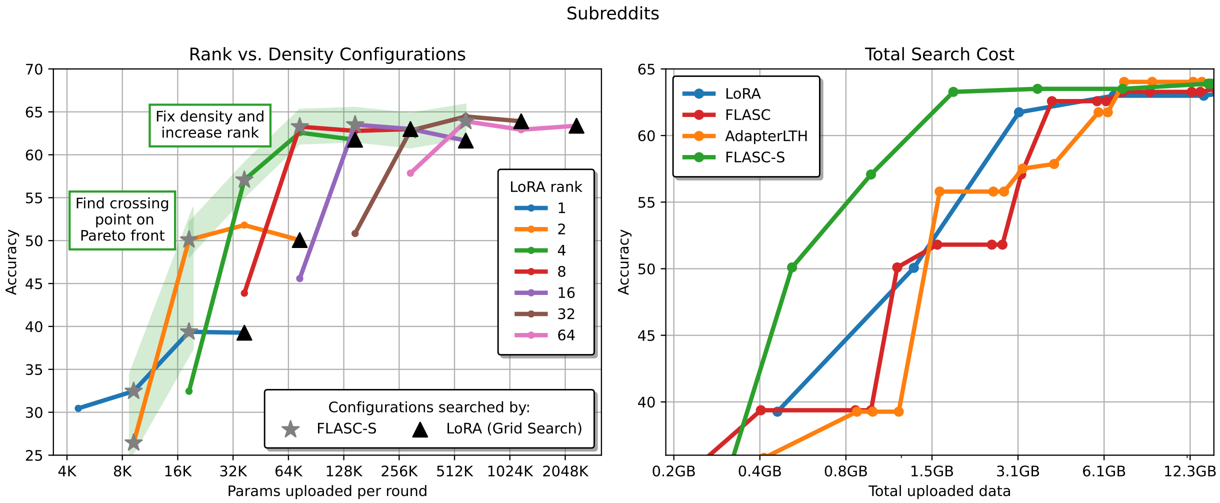

Figure 6: We search over a grid of ranks (1 to 64) and densities (1/8 to 1) on the Subreddits task. **Left:** Each curve shows a fixed LoRA rank while varying density. **FLASC-S** efficiently tunes the density on low rank configurations, followed by fixing the density and increasing the rank. **Right:** Sparse LoRA methods (red, orange) introduce additional hyperparameters which are more expensive to tune than the LoRA rank alone (blue). **FLASC-S** (green) efficiently configures **FLASC** by separately tuning density and rank.

4.2 Tuning rank and sparsity with FLASC-S

FLASC, like other sparse LoRA methods, requires tuning the rank and sparsity. Generally, a new dataset requires a new search for optimal hyperparameters, and FLASC is no exception. While **FLASC** can train a single model using less communication than LoRA, the additional cost of tuning the sparsity can make **FLASC** more expensive than only tuning the rank of LoRA and therefore negate the benefits of applying sparsity in the first place. To address this issue, we present **FLASC-S** (**FLASC-S**earch), a method for efficiently tuning **FLASC**.

Algorithm 2: FLASC-S

```
Require: Rank grid $R = \{r_1, ...r_M\}$
and density grid $D = \{d_1, ..., d_{N-1}, 1\}$.
for $i$=1..$N$ do
    $y_1 = \textbf{FLASC}(r_1, d_i * r_2/r_1)$
    $y_2 = \textbf{FLASC}(r_2, d_i)$
    if $y_1 < y_2$ then
        break
for $j$=$i$..$M$ do
    $y_j = \textbf{FLASC}(r_3, d_i * r_1/r_2)$
```

The left plot of Figure 6 provides a high-level illustration of how **FLASC-S** works. We sweep over rank and density values increasing by a factor of 2 and highlight the configurations that are evaluated by **FLASC-S**. A critical observation is that points on the Pareto front apply a similar density $(0.25 - 0.5\times)$, while a lower density $(0.125\times)$ performs worse than lower rank $(8\times)$ configurations of equal cost. Furthermore, too low of a rank harms performance, while too high of a rank results in redundant communication.

Based on these observations, our tuning method **FLASC-S** (Alg. 2.) tunes the sparsity using the lowest-rank configurations, fixes the sparsity, and then increases the rank. Given two equal-cost (rank, density) configurations (r_2, d) and $(r_1, dr_2/r_1)$ where r_1 and r_2 are the two lowest ranks in the search space, we expect r_2 to perform worse when d is extremely small. However, we also expect r_2 to improve more quickly than r_1 as d is increased. This allows us to sweep over values of the density d; once we encounter a value where r_2 performs better than r_1, we know (r_2, d) lies on the rank-density Pareto front. After finding this configuration, we fix d and increase the rank to find other configurations which lie close to the Pareto front.

The right plot of Figure 6 shows the communication cost of using **FLASC-S** versus running grid search on sparse LoRA baselines. The cost (x) is measured in terms of the accumulated communication from testing multiple (rank, sparsity) configurations, and the performance (y) is given by the best configuration found so far. Using grid search to tune sparse LoRA baselines (including **FLASC**) uses more communication than regular LoRA due to the cost of tuning extra sparsity hyperparameters. By tuning density on the smallest ranks before scaling up the rank, **FLASC-S** provides cheaper tuning costs than LoRA.

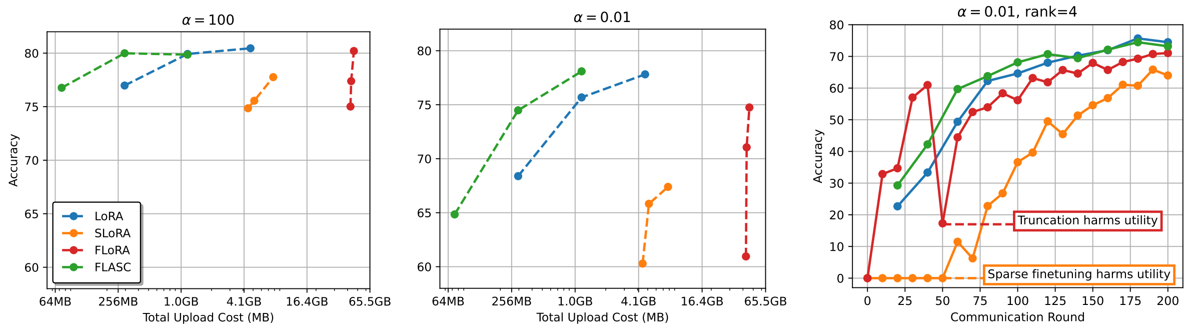

Figure 7: We compare **FLASC** to LoRA and SLoRA on 20NewsGroups. **Left, middle**: We test two settings of data heterogeneity α. Each curve varies the LoRA rank from 1 to 16. **FLASC** with 0.25 density improves over LoRA in all settings, while FLoRA (SLoRA) has large overhead costs due to an initial 50 rounds of "Stage 1" full-rank finetuning. **Right**: We compare the per-round accuracy of methods. FLoRA performs best during Stage 1, but degrades significantly after low-rank truncation. SLoRA applies a 10% density to Stage 1, which saves communication but harms overall accuracy.

4.3 Data Heterogeneity

Data heterogeneity is a commonly studied issue which can harm FL optimization (Li et al., 2020). In the context of LoRA, Babakniya et al. (2023a) finds that LoRA suffers more from data heterogeneity compared to full-rank finetuning and proposes FLoRA (SLoRA), a method that learns a full-rank (and sparse) update and then initializes LoRA with its truncated SVD (see Appendix A).

Our experiment in this section compares **FLASC** to LoRA, FLoRA, and SLoRA under varying levels of data heterogeneity α. Following standard practice in FL literature, we partition the data using a Dirichlet distribution over the set of labels (Hsu et al., 2019). Each client draws a vector from a Dirichlet(α) distribution, where the vector dimension is equal to the number of labels. We then normalize each dimension by that label's global frequency, and the resulting vectors correspond to the clients' individual label frequencies. We show two separate plots for the heterogeneity parameter $\alpha \in \{100, 0.01\}$. At $\alpha = 100$, clients have an approximately uniform number of examples per label, while at $\alpha = 0.01$, over 90% of most clients' examples belong to a single label.

In Figure 7, each curve (left and middle plots) shows LoRA with rank varying from $r \in \{1, 4, 16\}$. A 4× larger rank costs 4× more total upload (x-axis). When the data is identically distributed ($\alpha = 100$), LoRA rank 1 performs well and achieves only 1-2% less accuracy than larger ranks. When the data is heterogeneous ($\alpha = 0.01$), all ranks perform worse, but the impact on lower ranks is much more significant; the accuracy gap between LoRA rank 1 versus rank 16 grows from 2 to 12%. As long as we select an adequately large rank, **FLASC** is a strong baseline in both i.i.d. and non-i.i.d. settings. **FLASC** can use a density of 0.25× with minor impact on accuracy, resulting in $2 - 8\%$ higher accuracy compared to LoRA with a 4× smaller rank.

Finally, we evaluate FLoRA, which runs an initial 50 rounds of full-rank finetuning called "Stage 1" (see Appendix A). Unfortunately, full-rank finetuning costs over 200× the communication of LoRA and dominates the overall cost of finetuning (left and middle plots). A modification to reduce this cost called SLoRA applies sparsity to Stage 1, but this severely limits model utility (right plot). Furthermore, SVD truncation after Stage 1 severely harms accuracy and is worse than simply using LoRA throughout. Therefore, while FLoRA improves quickly and performs best on a per-round basis during Stage 1, these benefits cannot be fully realized due to (1) high communication cost (up to 200× the cost of LoRA) and (2) performance drop after SVD truncation.

4.4 Systems Heterogeneity

In addition to data heterogeneity, FL settings also face issues of systems heterogeneity where clients with limited system resources can slow down or harm training. In recent work, Cho et al. (2023) develop HetLoRA,

Method (Description)	**Accuracy**			
	$\alpha = 1$		$\alpha = 0.01$	
	3 tiers	5 tiers	3 tiers	5 tiers
All lowest tier (rank 1) (Restrict all clients to lowest communication tier)	77.4		68.4	
Highest tier only (rank 16) (Drop lower-tier stragglers from training)	74.6	72.0	56.2	45.1
HetLoRA (Clients train LoRA with varying rank)	77.8	76.8	64.3	40.0
FLASC (Clients upload sparse LoRA updates)	**80.7**	**80.4**	**75.8**	**78.1**

Table 3: We compare methods for handling systems heterogeneity on 20NewsGroups. We vary both data and systems heterogeneity in terms of the label frequency (α) and tiers of upload budget (tiers) respectively. Data heterogeneity exacerbates the issues of systems heterogeneity. **FLASC** is more robust than other baselines to both data and systems heterogeneity.

which aims to address systems heterogeneity by training LoRA modules with different ranks across clients. During training, client c with assigned rank r_c will download the uppermost r_c rows of A and leftmost r_c columns of B from the global LoRA weights (with rank r_s) and then use the weights to initialize a local LoRA module (with rank r_c). For a fair comparison with other methods in terms of hyperparameter tuning, we do not use the self-pruning method proposed by HetLoRA (i.e. we set $\gamma = 1, \lambda = 0$).

Our following experiment on systems heterogeneity constrains the upload budget of each client. We consider two heterogeneity settings of low ($b_s = 3, r_{\text{base}} = 4$) and high ($b_s = 5, r_{\text{base}} = 2$), where b_s is the number of **tiers (unique upload budgets)** and r_{base} is the **growth factor between budget tiers**. Each client c is assigned to one of these upload budgets $b_c \in \{1, 2, ..., b_s\}$ uniformly at random. To satisfy the upload budget, HetLoRA assigns clients a local rank $r_c = r_{base}^{b_c-1}$. **FLASC** uses the same upload communication as HetLoRA by finetuning a rank r_s module and applying an upload density of $d = r_c/r_s = r_{\text{base}}^{(b_c-b_s)}$. For both methods, the server initializes a LoRA adapter with rank $r_s = r_{\text{base}}^{b_s-1} = 16$. In addition, we evaluate two baselines **"All lowest tier"** and **"Highest tier only"**. To address settings where clients have significantly different upload speeds, "All lowest tier" restricts all clients to LoRA rank 1 in order to accommodate the clients with the slowest upload speeds. On the other hand, "Highest tier only" drops the clients with slow upload and only uses model updates from clients which are assigned the highest budget (rank 16).

In Table 3, we show that **FLASC** outperforms HetLoRA as well as the two naïve baselines. Generally, performance is better when systems are less (3 tiers) as opposed to more (5 tiers) heterogeneous. Additionally, we vary the data heterogeneity between $\alpha = 1$ or 0.01 (see Section 4.3). Heterogeneous data ($\alpha = 0.01$) can significantly exacerbate issues of systems heterogeneity; this is because data heterogeneity affects the accuracy of LoRA at lower ranks, while systems heterogeneity affects which clients are assigned to those lower ranks. When $\alpha = 0.01$, "Highest tier only" performs extremely poorly because it can only utilize a small set of skewed training data. Surprisingly, HetLoRA performs worse than "Highest tier only" when $\alpha = 0.01$, tiers = 5. In this setting, HetLoRA performed better in earlier rounds (due to leveraging data from more clients), but converged to a worse solution (due to heterogeneous updates of varying rank).

4.5 Privacy

Finally, we consider the performance of **FLASC** when used in conjunction with differential privacy. FL model updates are susceptible to privacy leakage and adversarial attacks (Geiping et al., 2020), which necessitates additional techniques such as differential privacy (DP). DP is a popular framework that adds randomness to an algorithm in order to mask example-level contributions to the algorithm's output (Abadi et al., 2016; McMahan et al., 2017b). Full finetuning scales poorly to strict DP budgets due to DP noise overwhelming the signal in the model updates, but PEFT (e.g. LoRA) is a promising solution to this problem (Luo et al.,

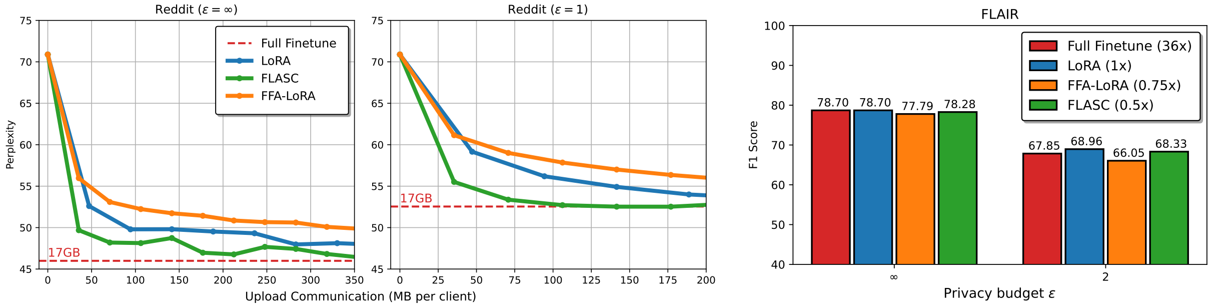

Figure 8: We compare finetuning methods with DP-FedAdam. While full finetuning performs best in non-private settings, LoRA is much cheaper and can even outperform full finetuning in private settings. FFA-LoRA harms utility and is less efficient than LoRA, while **FLASC** improves the efficiency of LoRA.

2021; Yu et al., 2022). In the FL space, Sun et al. (2024) suggest that combining FL and LoRA can amplify noise from DP and proposes FFA-LoRA, a method which freezes the A matrix of LoRA.

Our following experiments evaluate FL methods in non-private ($\epsilon = \infty$) and private ($\epsilon < \infty$) settings. ϵ is a privacy budget parameter which bounds the sensitivity of the algorithm's output with respect to any individual client in the training data. We compare full finetuning, LoRA, FFA-LoRA (reduce comm. by 25%), and **FLASC** (reduce comm. by 75% on Reddit, 50% on FLAIR). In Figure 8, we show the communication efficiency (left) and final utility (right) of each method. Similar to prior works in DP, we find that LoRA is more robust to DP than full finetuning. While FFA-LoRA reduces communication, it also significantly harms utility and leads to worse efficiency. In Figure 9, we further study the effect of parameter count by varying the rank of LoRA from 1 to 64. We find that a larger rank does better in non-private settings, while the optimal rank is smaller in private settings. Overall, using **FLASC** with a sufficiently small rank is both more accurate and efficient than applying FFA-LoRA to any rank.

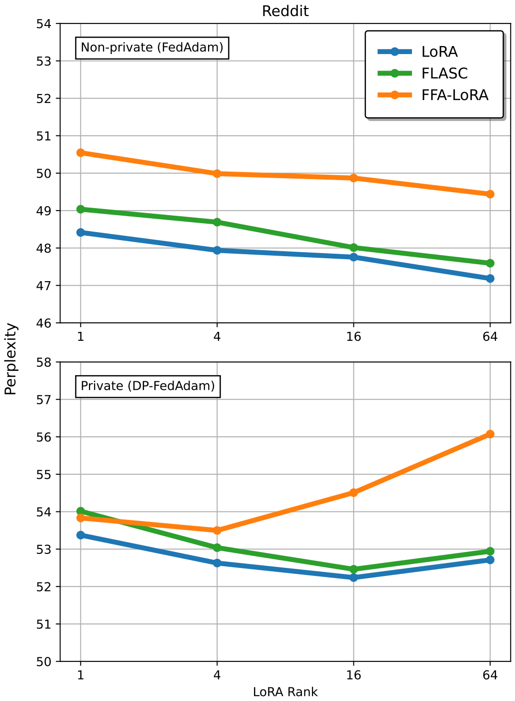

Figure 9: We vary the rank of LoRA in private vs non-private settings. Higher rank is generally better in non-private settings, while the optimal rank is lower in private settings.

5 Conclusion

In this paper, we introduce **FLASC**, an efficient FL method that significantly reduces the communication cost of LoRA. Our method provides both higher utility and substantial communication savings relative to existing pruning-based methods. Furthermore, we show that methods which introduce sparsity to LoRA can generally increase communication costs once accounting for hyperparameter tuning, due to the extra hyperparameters these methods introduce. To address this issue, we propose **FLASC-S**, a method which sequentially tunes the sparsity of **FLASC** followed by the rank. Finally, we find that **FLASC** is competitive with specific solutions for other FL concerns of heterogeneity and privacy while achieving superior communication efficiency. Overall, our results indicate that **FLASC** can serve as a strong baseline for future works in federated fine-tuning. Still, many important questions remain on how to make LoRA even more efficient in resource-constrained federated networks. For instance, LoRA itself may be insufficient when scaling to even larger models. In the future, we aim to investigate such questions and design methods to make high-quality models more accessible to low-resource users.

References

Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 308–318, 2016.

Sara Babakniya, Ahmed Elkordy, Yahya Ezzeldin, Qingfeng Liu, Kee-Bong Song, MOSTAFA EL-Khamy, and Salman Avestimehr. SLoRA: Federated parameter efficient fine-tuning of language models. In *International Workshop on Federated Learning in the Age of Foundation Models in Conjunction with NeurIPS 2023*, 2023a. URL `https://openreview.net/forum?id=06quMTmtRV`.

Sara Babakniya, Souvik Kundu, Saurav Prakash, Yue Niu, and Salman Avestimehr. Revisiting sparsity hunting in federated learning: Why does sparsity consensus matter? *Transactions on Machine Learning Research*, 2023b.

Jiamu Bai, Daoyuan Chen, Bingchen Qian, Liuyi Yao, and Yaliang Li. Federated fine-tuning of large language models under heterogeneous language tasks and client resources. *arXiv preprint arXiv:2402.11505*, 2024.

Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. Qsparse-local-sgd: Distributed sgd with quantization, sparsification and local computations. *Advances in Neural Information Processing Systems*, 32, 2019.

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, pp. 6080–6088, 2022.

Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tinytl: Reduce memory, not parameters for efficient on-device learning. *Advances in Neural Information Processing Systems*, 33:11285–11297, 2020.

Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018a.

Sebastian Caldas, Jakub Konečny, H Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210*, 2018b.

Zachary Charles, Zachary Garrett, Zhouyuan Huo, Sergei Shmulyian, and Virginia Smith. On large-cohort training for federated learning. *Advances in neural information processing systems*, 34:20461–20475, 2021.

Zachary Charles, Kallista Bonawitz, Stanislav Chiknavaryan, Brendan McMahan, et al. Federated select: A primitive for communication-and memory-efficient federated learning. *arXiv preprint arXiv:2208.09432*, 2022.

Chaochao Chen, Xiaohua Feng, Jun Zhou, Jianwei Yin, and Xiaolin Zheng. Federated large language model: A position paper. *arXiv preprint arXiv:2307.08925*, 2023.

Gary Cheng, Zachary Charles, Zachary Garrett, and Keith Rush. Does federated dropout actually work? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3387–3395, 2022.

Yae Jee Cho, Luyang Liu, Zheng Xu, Aldi Fahrezi, Matt Barnes, and Gauri Joshi. 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*, 2023.

Soham De, Leonard Berrada, Jamie Hayes, Samuel L Smith, and Borja Balle. Unlocking high-accuracy differentially private image classification through scale. *arXiv preprint arXiv:2204.13650*, 2022.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*, 2022.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. Sparse low-rank adaptation of pre-trained language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4133–4145, 2023.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=YicbFdNTTy`.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

Hongchang Gao, An Xu, and Heng Huang. On the convergence of communication-efficient local sgd for federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7510–7518, 2021.

Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients-how easy is it to break privacy in federated learning? *Advances in neural information processing systems*, 33:16937–16947, 2020.

Zhuocheng Gong, Di He, Yelong Shen, Tie-Yan Liu, Weizhu Chen, Dongyan Zhao, Ji-Rong Wen, and Rui Yan. Finding the dominant winning ticket in pre-trained language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 1459–1472, 2022.

Demi Guo, Alexander M Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4884–4896, 2021.

Shwai He, Liang Ding, Daize Dong, Jeremy Zhang, and Dacheng Tao. Sparseadapter: An easy approach for improving the parameter-efficiency of adapters. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 2184–2190, 2022.

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.

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*, pp. 2790–2799. PMLR, 2019.

Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

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.

Tiansheng Huang, Shiwei Liu, Li Shen, Fengxiang He, Weiwei Lin, and Dacheng Tao. Achieving personalized federated learning with sparse local models. *arXiv preprint arXiv:2201.11380*, 2022.

Berivan Isik, Francesco Pase, Deniz Gunduz, Tsachy Weissman, and Michele Zorzi. Sparse random networks for communication-efficient federated learning. *arXiv preprint arXiv:2209.15328*, 2022.

Feibo Jiang, Li Dong, Siwei Tu, Yubo Peng, Kezhi Wang, Kun Yang, Cunhua Pan, and Dusit Niyato. Personalized wireless federated learning for large language models. *arXiv preprint arXiv:2404.13238*, 2024.

Jeonghoon Kim, Jung Hyun Lee, Sungdong Kim, Joonsuk Park, Kang Min Yoo, Se Jung Kwon, and Dongsoo Lee. Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. *Advances in Neural Information Processing Systems*, 36:36187–36207, 2023a.

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*, pp. 1159–1172, 2023b.

Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency, 2017.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *Master's thesis, University of Toronto*, 2009.

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

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2018.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

Wei Liu, Ying Qin, Zhiyuan Peng, and Tan Lee. Sparsely shared lora on whisper for child speech recognition. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 11751–11755. IEEE, 2024.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2018.

Qikai Lu, Di Niu, Mohammadamin Samadi Khoshkho, and Baochun Li. Hyperflora: Federated learning with instantaneous personalization. In *Proceedings of the 2024 SIAM International Conference on Data Mining (SDM)*, pp. 824–832. SIAM, 2024.

Gen Luo, Minglang Huang, Yiyi Zhou, Xiaoshuai Sun, Guannan Jiang, Zhiyu Wang, and Rongrong Ji. Towards efficient visual adaption via structural re-parameterization. *arXiv preprint arXiv:2302.08106*, 2023.

Zelun Luo, Daniel J Wu, Ehsan Adeli, and Li Fei-Fei. Scalable differential privacy with sparse network finetuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5059–5068, 2021.

Shubham Malaviya, Manish Shukla, and Sachin Lodha. Reducing communication overhead in federated learning for pre-trained language models using parameter-efficient finetuning. In *Conference on Lifelong Learning Agents*, pp. 456–469. PMLR, 2023.

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*, pp. 1273–1282. PMLR, 2017a.

H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963*, 2017b.

Aritra Mitra, Rayana Jaafar, George J Pappas, and Hamed Hassani. Linear convergence in federated learning: Tackling client heterogeneity and sparse gradients. *Advances in Neural Information Processing Systems*, 34: 14606–14619, 2021.

Saurav Muralidharan. Uniform sparsity in deep neural networks. *Proceedings of Machine Learning and Systems*, 5, 2023.

Duy Phuong Nguyen, J Pablo Munoz, and Ali Jannesari. Flora: Enhancing vision-language models with parameter-efficient federated learning. *arXiv preprint arXiv:2404.15182*, 2024.

John Nguyen, Jianyu Wang, Kshitiz Malik, Maziar Sanjabi, and Michael Rabbat. Where to begin? on the impact of pre-training and initialization in federated learning. In *Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022)*, 2022.

Kaan Ozkara, Navjot Singh, Deepesh Data, and Suhas Diggavi. Quped: Quantized personalization via distillation with applications to federated learning. *Advances in Neural Information Processing Systems*, 34:3622–3634, 2021.

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. In *International Conference on Learning Representations*, 2021.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Sashank J Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2020.

Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*, pp. 2021–2031. PMLR, 2020.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations*, 2019.

Lucas Grativol Ribeiro, Mathieu Leonardon, Guillaume Muller, Virginie Fresse, and Matthieu Arzel. Flocora: Federated learning compression with low-rank adaptation. *arXiv preprint arXiv:2406.14082*, 2024.

Jae Hun Ro, Theresa Breiner, Lara McConnaughey, Mingqing Chen, Ananda Theertha Suresh, Shankar Kumar, and Rajiv Mathews. Scaling language model size in cross-device federated learning. *arXiv preprint arXiv:2204.09715*, 2022.

Suhail Mohmad Shah and Vincent KN Lau. Model compression for communication efficient federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Congzheng Song, Filip Granqvist, and Kunal Talwar. Flair: Federated learning annotated image repository. *Advances in Neural Information Processing Systems*, 35:37792–37805, 2022.

Dimitris Stripelis, Umang Gupta, Greg Ver Steeg, and Jose Luis Ambite. Federated progressive sparsification (purge-merge-tune)+. In *Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022)*, 2022.

Guangyu Sun, Matias Mendieta, Taojiannan Yang, and Chen Chen. Conquering the communication constraints to enable large pre-trained models in federated learning. *arXiv preprint arXiv:2210.01708*, 2022.

Youbang Sun, Zitao Li, Yaliang Li, and Bolin Ding. Improving loRA in privacy-preserving federated learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=NLPzL6HWNl`.

Yi-Lin Sung, Varun Nair, and Colin A Raffel. Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205, 2021.

Jiarun Wu and Qingliang Chen. Pruning adapters with lottery ticket. *Algorithms*, 15(2):63, 2022.

Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, XIAOPENG ZHANG, and Qi Tian. Qa-lora: Quantization-aware low-rank adaptation of large language models. In *The Twelfth International Conference on Learning Representations*, 2023.

Prateek Yadav, Leshem Choshen, Colin Raffel, and Mohit Bansal. Compeft: Compression for communicating parameter efficient updates via sparsification and quantization. *arXiv preprint arXiv:2311.13171*, 2023.

Liping Yi, Han Yu, Gang Wang, and Xiaoguang Liu. Fedlora: Model-heterogeneous personalized federated learning with lora tuning. *arXiv preprint arXiv:2310.13283*, 2023.

Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, Sergey Yekhanin, and Huishuai Zhang. Differentially private fine-tuning of language models. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=Q42f0dfjECO`.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–9, 2022.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. Increlora: Incremental parameter allocation method for parameter-efficient fine-tuning. *arXiv preprint arXiv:2308.12043*, 2023a.

Mingyang Zhang, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, Bohan Zhuang, et al. Pruning meets low-rank parameter-efficient fine-tuning. *arXiv preprint arXiv:2305.18403*, 2023b.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2022.

Xuechen Zhang, Mingchen Li, Xiangyu Chang, Jiasi Chen, Amit K Roy-Chowdhury, Ananda Theertha Suresh, and Samet Oymak. Fedyolo: Augmenting federated learning with pretrained transformers. *arXiv preprint arXiv:2307.04905*, 2023c.

Zhuo Zhang, Yuanhang Yang, Yong Dai, Qifan Wang, Yue Yu, Lizhen Qu, and Zenglin Xu. Fedpetuning: When federated learning meets the parameter-efficient tuning methods of pre-trained language models. In *Annual Meeting of the Association of Computational Linguistics 2023*, pp. 9963–9977. Association for Computational Linguistics (ACL), 2023d.

Bowen Zhao, Hannaneh Hajishirzi, and Qingqing Cao. Apt: Adaptive pruning and tuning pretrained language models for efficient training and inference. *arXiv preprint arXiv:2401.12200*, 2024.

Weilin Zhao, Yuxiang Huang, Xu Han, Zhiyuan Liu, Zhengyan Zhang, and Maosong Sun. Cpet: Effective parameter-efficient tuning for compressed large language models. *arXiv preprint arXiv:2307.07705*, 2023.

A Methods

FedAdam. FedAdam is an optimization method that follows the FedOpt framework (Reddi et al., 2020). At every round, each participating client i downloads a copy of the trainable weights P, finetunes P to obtain updated weights P'_i, and uploads $\Delta P_i = P - P'_i$ to the server. The server then computes an average update $\Delta P = \frac{1}{n}\sum_{i=1}^{n} \Delta P_i$, where n is the number of clients sampled per round. The average may optionally be weighted by each client's dataset size. ΔP can be interpreted as a global pseudo-gradient; for example, the update rule for FedAvg is to set $P \leftarrow P - \Delta P$ for the next round (McMahan et al., 2017a). In the case of FedAdam, a stateful Adam optimizer (Kingma, 2014) takes ΔW as input and outputs an adapted global update at each round. In Algorithm 3, we show how Adam and FedAdam differ only in how ΔP is computed.

We use a single parameter vector P to refer to the trainable parameters of the model. In the context of LoRA, P is a flattened and concatenated vector of LoRA weights $\{A_l, B_l\}_{l=1}^{L}$ where L is the number of layers LoRA is applied to. Applying `TopKMask` to P naturally results in varying sparsity per layer. An alternative approach is to uniformly sparsify each layer (A_l, B_l) in before concatenation, but the former tended to perform better.

Algorithm 3: General functions for central and federated training with pruning

```
# FL-specific lines are colored red.
Require:
    $P$ (trainable parameters)
    $\mathcal{L}$ (loss function)
    $S_{\text{server}}$ (central training steps or FL rounds)
    $\eta_{\text{server}}$ (learning rate)
    $M$ (sparse mask)
Require for FL:
    $S_{\text{client}}$ (num. local steps)
    $\eta_{\text{client}}$ (client learning rate)
    $n$ (clients per round)
Function FLRound($P$, $M$, $S_{\text{client}}, \eta_{\text{client}}, n$):
    Sample client datasets $c_1, ..., c_n$ uniformly at random w/o replacement
    for $i = 1, ..., n$ in parallel do
        $P_i \leftarrow P$
        for $s = 1, ..., S_l$ do
            Sample batch of data $(x, y)$ from $c_i$
            $\nabla_{P_i} = \nabla_1 \mathcal{L}(P_i, x, y) \odot M$ # sparse gradient
            $P_i \leftarrow P_i - \eta_{\text{client}} \nabla_{P_i}$
        $\Delta P_i \leftarrow P_i - P$
    return $\frac{1}{n}\sum_{i=1}^{n} \Delta P_i$
Function TrainStep($P$,$M$):
    if FedAdam then
        return FLRound($P$, $M$, $S_{client}, \eta_{client}, n$)
    else if Adam then
        Sample batch of data $(x, y)$ from central dataset
        return $\nabla_1 \mathcal{L}(P, x, y) \odot M$ # sparse gradient
```

Algorithm 4: Standard LoRA training

```
$P \leftarrow$ Initialize LoRA with rank $r$
$M \leftarrow 1$ # all-ones matrix
optim $\leftarrow$ torch.nn.optim.Adam($P, \eta_{\text{server}}$)
for $s_{\text{server}} = 1, ..., S_{\text{server}}$ do
    $P$.grad $\leftarrow$ TrainStep($P$,$M$)
    optim.step()
```

Sparse LoRA and Sparse FL baselines. In addition to showing how Adam and FedAdam both fit in the same general training framework, Algorithm 3 shows how centralized pruning methods can be adapted to FL. We use this framework to succinctly describe our baselines i.e. LoRA in Alg. 4, iterative pruning methods Adapter LTH and FedSparsify-Global in Alg. 5, and fixed density methods SparseAdapter, SPDST, and Federated Select in Alg. 6.

Communication cost of the mask. For the purposes of communication, the mask is stored as a binary vector with p entries where p is the number of trainable parameters in the model. The cost of communicating this mask is (1/32)*(4 bits per parameter)*p, which is independent of the actual amount of sparsity in the mask. We use this format for simplicity and because FLASC (like other model sparsity methods) requires relatively large density values. If communication is sufficiently sparse, it would be beneficial to use a sparse matrix format which scales with sparsity. Such a matrix format (e.g. CSR) stores only the non-zero indices but each index (e.g. uint8) requires multiple bits.

Iterative Magnitude Pruning. In our work, we consider using iterative magnitude pruning (IMP) without weight rewinding. In the original Lottery Ticket Hypothesis paper by Frankle & Carbin (2018), the authors propose using IMP to find sparse subnetworks that can be trained starting from their randomly initialized weights. However, the procedure of identifying this subnetwork is extremely expensive, as it requires $10-30$ iterations of (1) training the model to convergence, (2) removing the $\%p$ smallest magnitude weights, and (3) rewind (i.e. reset) the remaining weights to their initial values. Therefore, this method is unsuitable if the goal is simply to efficiently train a single model. Therefore, Renda et al. (2019) proposes a more efficient alternative which is to skip the rewinding step and simply continue training from the pruned state. Overall, they show that "IMP without rewinding" is also able to identify trainable sparse subnetworks without the large overhead cost of rewinding.

Algorithm 5: Iterative Magnitude Pruning (IMP) without rewinding

```
Require: $p$ (prune ratio), $q$ (prune frequency)
if Adapter LTH then
    $P \leftarrow$ Initialize LoRA with rank $r$
else if FedSparsify-Global then
    $P \leftarrow$ Initialize model backbone weights
$d \leftarrow 1$
optim ← torch.nn.optim.Adam($P, \eta_{\text{server}}$)
for $s_{\text{server}} = 1, ..., S_{\text{server}}$ do
    if $0 \equiv s_{server} \mod q$ then
        $M \leftarrow$ TopKMask($|P|, d$)
        $P \leftarrow P \odot M$ # permanently remove entries in $P$
        $d \leftarrow d(1-p)$
    $P$.grad ← TrainStep($P$,$M$)
    optim.step()
```

Pruning-at-initialization. Pruning-at-initialization (PaI) is a class of methods which prune the model to a fixed density at the start of training. While IMP (as its name suggests) generally uses magnitude to rank parameters, pruning-at-initialization (PaI) methods use various different sensitivity metrics which determine the parameters to prune. In our work, we consider SparseAdapter, which applies PaI to LoRA in centralized learning, and SPDST, which applies PaI to full finetuning in federated learning.

SparseAdapter. As discussed in Section 3, SparseAdapter as proposed is unsuitable for pruning of LoRA. In general, any senstivity metric which depends on parameter magnitude is unsuitable for LoRA because the B matrix in LoRA is initialized to all zeros. Therefore, all B weights will be pruned before any weights in A. For matrices $B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}$, pruning at any density below $d/(d+k)$ will entirely prune the B matrix, which leaves us unable to train the model. Therefore, instead of using the SNIP metric, we use a modified version of SPDST where the metric is based on finetuning the model for a limited number of iterations.

SPDST. SPDST locally runs a pruning method called SNFS (Dettmers & Zettlemoyer, 2019) and produces a sparse model on each client. It then measures the average density of each layer across clients to determine which layers are more or less sensitive. Finally, it randomly prunes each layer of the initial model proportionally to its sensitivity. We found that SPDST has limited utility due to (1) producing sparse models with SNFS and (2) randomly selecting the final sparse parameters. Therefore, we make two adjustments of (1) regular (dense) finetuning instead of SNFS and (2) deterministically pruning the least sensitive weights based on parameter magnitude. In other words, we apply IMP for a single iteration, after which the model is immediately pruned to the target dnesity. This applies the same principle as SPDST, which is to directly finetune the model in order to determine which parameters should be pruned.

Federated Select. Federated Select is a method which falls under the broader category of fixed-density pruning. Like the two PaI methods, Federated Select prunes the model to a given density and freezes the zeroed weights during client finetuning. However, unlike PaI methods, Federated Select retains the original dense weights at the server; after aggregation, it can adjust the mask based on the updated weights and have clients download a model with an updated sparsity pattern in the following round.

Algorithm 6: Fixed-density pruning

```
Require: $d$ (prune density)
if SparseAdapter then
    $P \leftarrow$ Initialize LoRA with rank $r$
else
    # We use Federated Select to prune the backbone in Table 1 and to prune LoRA in Sec. 4.2.
    $P \leftarrow$ Initialize model backbone weights
# Compute importance scores; SparseAdapter proposes using SNIP (gradient-magnitude)
  sensitivity, but this is unsuitable for LoRA. (see Sec. 3). Instead, we use "SPDST" sensitivity.
if SPDST then
    # We use a modified version of SPDST which finetunes the model and computes TopK weights.
    $M \leftarrow 1$ # matrix of all ones
    $P' \leftarrow$ TrainStep($P$,$M$)
    $S \leftarrow |P'|$
    $M \leftarrow$ TopKMask($S, d$)
if not Federated Select then
    $P \leftarrow P \odot M$ # permanently remove entries in $P$
optim $\leftarrow$ torch.nn.optim.Adam($P, \eta_{\text{server}}$)
for $s_{\text{server}} = 1, ..., S_{\text{server}}$ do
    if Federated Select then
        $M \leftarrow$ TopKMask($|P|, d$)
        $P' \leftarrow P \odot M$ # download a temporarily sparse $P'$
    else
        $P' = P$ # $P$ is already sparse
    $P$.grad $\leftarrow$ TrainStep($P'$,$M$)
    optim.step()
```

FedSpa. FedSpa is a sparse backbone method proposed by Huang et al. (2022). Similar to Federated Select, the server maintains a dense global model while clients download a sparse model and only finetune the non-zero parameters of these sparse models. FedSpa additionally learns a personalized mask at each client. After local finetuning, FedSpa runs a local "mask searching" algorithm which adjusts the mask in advance of the next round the client is selected for training. Unfortunately, this introduces a limitation in that the personalized masks must be statefully maintained across rounds, which is expensive when a large number of clients participate in FL. We found that FedSpa performed worse than other sparse backbone methods under a limited budget of 200 rounds, though it can achieve non-trivial performance with a larger budget of rounds.

SLoRA. FLoRA and SLoRA (Babakniya et al., 2023a) are two-stage methods that run multiple rounds of full-rank finetuning (Stage 1), apply SVD decomposition and truncation to the model update, and then train LoRA using the truncated weights as an initialization (Stage 2). The motivation is that when data in

Method / Round	200	400	600
FedSpa (density=0.75)	5.00	49.48	5.00
FedSpa (density=0.5)	5.00	16.67	40.17

Table 4: Evaluation of FedSpa on 20NewsGroups.

FL is more heterogeneous, a higher effective rank is beneficial for capturing the diversity in client updates. Unfortunately, full-rank finetuning (as used in FLoRA) can be expensive and dominates communication costs (see Sec. 4.3). To reduce this cost, SLoRA is a modification that applies weight-level sparsity i.e. 90% of the full-rank parameters are frozen at the start of Stage 1.

FFA-LoRA. FFA-LoRA (Sun et al., 2024) is a method which freezes the A matrix of LoRA. The communication savings of FFA-LoRA depends on the relative size of the A, B matrices, which varies across layers and model architectures. In our experiments, we apply LoRA to the K,V (CIFAR10 / FLAIR) or K,V,Q (20NewsGroups, Reddit) projections in the attention layers. For these two settings, FFA-LoRA uses 2/3 and 3/4 of the original communication respectively. Due to the relatively small savings compared to the 1/4 density we apply to **FLASC**, we train FFA-LoRA with a proportionally fewer number of rounds (by a factor of 3/8 and 1/3 respectively).

We present an additional experiment which shows that the optimization method can affect the performance of FFA-LoRA vs. LoRA. The original FFA-LoRA work uses FedAvg, while our work uses FedAdam. We chose FedAdam because it is a state-of-the-art FL optimizer which is commonly used in practice and generally outperforms FedAvg. With FedAvg, FFA-LoRA slightly improves over LoRA when using a large batch size and significantly improves when using a small batch size. We hypothesize that LoRA can fail to train in a small-batch setting due to noise from stochastic gradients compounding with the inexact aggregation of LoRA. However, FedAdam appears to mitigate these issues. When using FedAdam with a small batch size (16), both LoRA and FFA-LoRA outperform their FedAvg counterparts and LoRA performs better than FFA-LoRA.

Method	FedAvg, Batch=200	FedAvg, Batch=16	FedAdam, Batch=16
LoRA	91.45	50.00	**93.65**
FFA-LoRA	**91.74**	**90.77**	92.68

Table 5: Comparison of FFA-LoRA vs. LoRA on a 3-client IID partition of QNLI. FFA-LoRA performs better with FedAvg, but LoRA performs better with FedAdam. Additionally, FedAdam generally outperforms FedAvg and can tolerate a smaller batch size.

FFA-LoRA is designed for the stronger notion of *local* (as opposed to *global*) differential privacy. In local DP, clients locally run DP-SGD, which is only feasible in *cross-silo* FL settings where each client has a large number of local examples. In contrast, we fo cus on cross-device settings which assume a large widespread pool of clients with few examples per client.

Differentially Private (DP)-FedAdam. To apply global DP to FedAdam, clients upload updates computed by non-private SGD. The server clips these updates, aggregates them, normalizes by the clipping norm, and then adds Gaussian noise with scale σ (De et al., 2022). This protects client privacy at a coarse-grained level where the "neighboring datasets" definition of DP applies to the addition or removal of one client's local dataset rather than a single example (McMahan et al., 2017b).

To obtain reasonable privacy guarantees when using DP-FedAdam, we must bound the sensitivity of the aggregate update with respect to any individual client. The most obvious way to achieve this is to sample a large cohort of clients (Charles et al., 2021). However, when running experiments with private FL, this can make training costs prohibitively expensive. To make simulation feasible in terms of wall-clock time, a common trick is to select a large ('simulated') client cohort size, compute the noise scale according to the privacy constraints, and then linearly scale it down according to a smaller cohort size actually used for experiments. For instance, Song et al. (2022) (Sec. 5.1, p.7) uses "200 users sampled per round to simulate the noise-level with a cohort size of 5,000". We follow this simulation setup for our experiments on FLAIR. For Reddit, we sample 10 users per round and simulate the noise-level with a cohort size of 1,000. Simulating a larger cohort size has two effects: (1) our models are less private than the reported privacy budget, but (2) provide a *lower bound* on the accuracy we would obtain from training with the true cohort size / privacy budget.

B Experiment Details

Dataset	Task	Partition	#Clients	#Examples	#Classes
CIFAR10	Image Classification	Dirichlet	500	50K	10
20NewsGroups	Sequence Classification	Dirichlet	350	20K	20
Reddit	Next Token Prediction	Natural	32K	1.1M	50257
Subreddits	Sequence Classification	Natural	100	35K	50
FLAIR	Object Detection	Natural	41K	345K	17 (multilabel)

Table 6: More detailed information on the datasets used in the experiments.

B.1 Artifacts

We provide the code in the supplementary material of our submission.

B.2 Compute Resources

We simulate FL training on a single NVIDIA L40S GPU and parallelize trials over multiple GPUs.

B.3 Training Details

We apply LoRA to the attention layers only, specifically the K,V mappings for ViT, and K,V,Q mappings for GPT2. For CIFAR10 and FLAIR, we train a classification layer and the LoRA parameters. On 20NewsGroups and Subreddits, we do not train a classification layer. Instead, we format the dataset such that the model outputs the label in text form. We freeze the language modeling (embedding) head and only finetune LoRA parameters. Reddit is a next-token prediction task so we do not perform any additional modification to the model or data.

B.4 Hyperparameters

FedAdam hyperparameters:

- Learning rates $\eta_{\text{server}}, \eta_{\text{client}}$:
  - CIFAR10: 5e-3, 1e-3
  - 20NewsGroups: 1e-2, 1e-3
  - Reddit: 1e-2, 1e-3
  - FLAIR: 1e-3, 1e-2
- Betas: $\beta_1 = 0.9, \beta_2 = 0.999$
- Clients per round: 10 (CIFAR10, 20Newsgroups, Reddit), 200 (FLAIR)
- Training rounds:
  - CIFAR10: 200
  - 20NewsGroups: 200 (Fig. 1, 7, Tab. 3), 400 (Fig. 3)

  - Reddit: 200 (Fig. **??**), 400 (Fig. 4, 8, 9)
  - FLAIR: 5000

- Client optimizer: SGD (batch size= 16, momentum= 0.9)

Figure 1:

1. 1 GB budget:
   - LoRA rank r: 4
   - **FLASC**, Adapter LTH, SparseAdapter rank r: 16
   - **FLASC**, SparseAdapter density: 0.25
   - Adapter LTH density: 0.98
   - Training rounds: 200
2. 128 MB budget:
   (a) LoRA rank r: 1
   (b) **FLASC**, Adapter LTH, SparseAdapter rank r: 4
   (c) **FLASC**, SparseAdapter density: 0.25
   (d) Adapter LTH density: 0.96
   (e) Training rounds: 100

Figure 3:

- LoRA rank r: 16
- Adapter LTH density: $d_{\text{LTH}} \in [0.97, 0.98, 0.99]$ (based on num. of rounds $\in [100, 200, 400]$ respectively) We prune each round for all datasets besides FLAIR, and prune once every 25 rounds for FLAIR with density 0.98.
- SparseAdapter density d: 1/4
- **FLASC** density: $p_{\text{down}} = d_{\text{up}} \in \mathbf{1/4}$. For 20NewsGroups and Reddit, we set $d_{\text{down}} = 1$.
- Label heterogeneity α: CIFAR10: 0.1, 20NewsGroups: 0.01

Figure 4 (Reddit):

- Adapter LTH density: $p_{\text{LTH}} = 0.99$
- SparseAdapter density: $p = 1/4$
- **FLASC** density: $p_{\text{down}} = 1/4, p_{\text{up}} \in [1/16, 1/4]$

Figure 5 (CIFAR10):

- LoRA rank: $r = 16$
- Density (all methods): $d \in [1, 1/4, 1/16, 1/64, 1/256]$

Figure 7 (20NewsGroups):

- SLoRA Stage 1 LR: 1e-3
- Stage 1 rounds: 50 (25% of 200 total rounds)

Table 3 (20NewsGroups): Details in text of Sec. 4.4.

Figure 8 (FLAIR):

- Noise multiplier: $\sigma \in [0, 0.34]$
- Clipping norm: $C = 5 * 10^{-3}$

Figure 8,9 (Reddit):

- Client learning rate: $\eta_{\text{client}} = 5 * 10^{-4}$
- Noise multiplier: $\sigma \in [0, 0.013, 0.072, 0.58]$
- Clipping norm: $C = 10^{-4}$