SparsyFed: SPARSE ADAPTIVE FEDERATED TRAINING

Adriano Guastella[*,1] **Lorenzo Sani**[*,2,3] **Alex Iacob**[2,3]

Alessio Mora[1] **Paolo Bellavista**[1] **Nicholas D. Lane**[2,3]

ABSTRACT

Sparse training is often adopted in cross-device federated learning (FL) environments where constrained devices collaboratively train a machine learning model on private data by exchanging pseudo-gradients across heterogeneous networks. Although sparse training methods can reduce communication overhead and computational burden in FL, they are often not used in practice for the following key reasons: (1) data heterogeneity makes it harder for clients to reach consensus on sparse models compared to dense ones, requiring longer training; (2) methods for obtaining sparse masks lack adaptivity to accommodate very heterogeneous data distributions, crucial in cross-device FL; and (3) additional hyperparameters are required, which are notably challenging to tune in FL. This paper presents *SparsyFed*, a practical federated sparse training method that critically addresses the problems above. Previous works have only solved one or two of these challenges at the expense of introducing new trade-offs, such as clients' consensus on masks versus sparsity pattern adaptivity. We show that *SparsyFed* simultaneously (1) can produce 95% sparse models, with negligible degradation in accuracy, while only needing a single hyperparameter, (2) achieves a per-round weight regrowth 200 times smaller than previous methods, and (3) allows the sparse masks to adapt to highly heterogeneous data distributions and outperform all baselines under such conditions.

1 INTRODUCTION

Federated Learning (McMahan et al., 2017) has become a standard technique for distributed training on private data (Yang et al., 2018; Ramaswamy et al., 2019; Pati et al., 2022; Wang et al., 2023; Huba et al., 2022; Bonawitz et al., 2019), particularly on edge devices. Given its application to constrained hardware, mitigating communication and computational overheads—significant in standard FL infrastructures (Kairouz et al., 2021; Bellavista et al., 2021)—remains a key field focus. Practical cross-device FL methods typically assume stateless clients with imbalanced, heterogeneous datasets and constrained, diverse hardware (Wang et al., 2021). Restricted client hardware and low communication bandwidth significantly increase training time compared to centralized methods, elongating hyperparameter tuning (Khodak et al., 2021). Additionally, unknown data distributions and dynamic client availability demand robust optimization methods that can handle these variations. When device availability is constrained, the federated orchestrator may struggle to sample a representative client subset (Eichner et al., 2019; Cho et al., 2020; Li et al., 2020b), inducing trade-offs between sampling ratio and efficiency (Charles et al., 2021b).

Sparse training methods improve computational and communication efficiency by reducing (a) memory footprint and FLOPs during training (Raihan & Aamodt, 2020), and (b) the communication costs (Bibikar et al., 2022). However, applying these methods in cross-device FL is challenging due to client availability and data heterogeneity, which can disrupt the binary mask structure across clients (Qiu et al., 2021). Such inconsistencies hinder consensus on the binary mask, lowering global model performance (Babakniya et al., 2023). Recent approaches address these issues using fixed sparse masks (Huang et al., 2022; Qiu et al., 2021) or dynamic methods involving mask

[*] Equal Contribution, correspondence to adriano.guastella2@unibo.it [1] Dipartimento di Informatica - Scienza e Ingegneria, Università di Bologna [2] Department of Computer Science and Technology, University of Cambridge [3] Flower Labs, UK

warmup and refreshing (Babakniya et al., 2023). However, fixed masks reduce the adaptability to unseen distributions, while dynamic methods require careful tuning of additional hyperparameters, like warmup duration and refresh interval. Fixed-mask methods also limit model plasticity—the ability to rewire and adapt to diverse distributions (Lyle et al., 2022; 2023). For example, it is known that in a multi-task or continual learning setting, neural networks can be iteratively pruned to build task-specific sub-networks (Mallya & Lazebnik, 2018), a lost ability when adopting a fixed-mask method. This lack of adaptability makes fixed-mask approaches unsuitable for cross-device FL, where unseen distributions frequently arise. Thus, we argue that sparse training methods for cross-device FL should (a) adopt dynamic masking and (b) remain agnostic to optimization and selection methods while minimizing hyperparameter complexity.

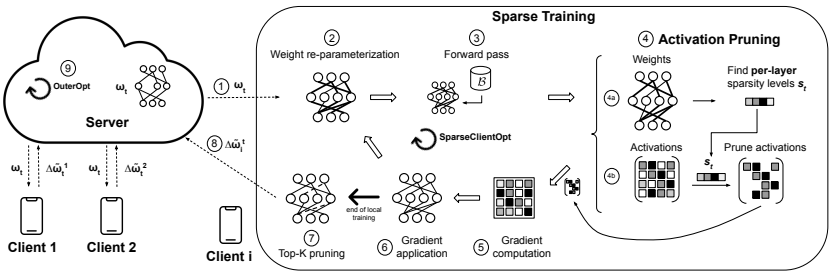

Figure 1: *SparsyFed* pipeline. (1) Server broadcasts the global model ω_t. (2) Client i re-parameterizes local weights. (3) Executes a forward pass on batch $\mathcal{B}$. (4a) Computes layer-wise sparsity s_t. (4b) Prunes activations using s_t and stores them. (5) Computes grads. (6) Applies grads. (7) Computes model updates and applies `Top-K` pruning. (8) Sends sparse updates $\Delta\tilde{\omega}_i^t$ back to the server. (9) Apply server optimizer to obtain the global model. Steps (2-6) repeat until convergence.

Our work addresses these challenges by introducing *SparsyFed*, a sparse training method for training global models in cross-device federated learning. During local training, as illustrated in Fig. 1, *SparsyFed* uses an easy-to-tune approach to prune (1) activations with an adaptive layer-wise method and (2) model weights before communication using the target sparsity. These strategies can reduce FLOPs and memory usage, with suitable hardware support (Raihan & Aamodt, 2020), and the communication costs by transmitting sparse updates. Our layer-wise approach prunes based on each layer's parameter proportion, unlike previous methods (Babakniya et al., 2023; Qiu et al., 2021), which applied fixed global sparsity across layers. This approach removes more capacity from dense layers while preserving parameter-efficient ones. *SparsyFed* also prunes models at the end of each round, allowing the complete flexibility of the local optimization. *SparsyFed* provides robust initialization for crucial layers, such as embeddings, without excluding layers to preserve performance as done in past works (Qiu et al., 2021; Raihan & Aamodt, 2020). Additionally, *SparsyFed* employs a sparsity-inducing weight re-parameterization (Schwarz et al., 2021) based on a single parameter, enhancing the model's resilience to sparsity and improving adaptivity, which enables changing the global mask for new clients with diverse data distributions. *SparsyFed* is agnostic to the choice of the outer optimizer, treating sparse model updates as pseudo-gradients. It is also compatible with biased client selection policies, allowing training masks to adapt to the training data utilized. Our work's contributions are:

1. We introduce *SparsyFed*, a method which accelerates on-device training in FL. *SparsyFed* achieves high sparsity (up to 95%) without sacrificing accuracy through a novel approach combining hyperparameter-free activation pruning and weight re-parameterization.
2. We compare *SparsyFed* against the latest state-of-the-art techniques, demonstrating superior accuracy over sparse training baselines (including those using fixed sparse masks) and, in some cases, surpassing non-pruned baselines at extreme sparsity levels.
3. *SparsyFed* quickly achieves consensus on client sparse model masks, enabling faster global convergence and significantly reducing downlink communication costs (up to $19.29\times$).
4. We evaluate *SparsyFed* with ablation studies on typical cross-device FL datasets, including CIFAR-10/100 (Krizhevsky, 2012) and Speech Commands (Warden, 2018), under various data heterogeneity conditions.

We provide the developed code publicly available in this repository to facilitate result reproducibility and for the community of researchers in the field.

2 BACKGROUND

In the following, we describe typical sparse training and weight re-parametrization techniques, which are key components of our work, and discuss their relevance to cross-device FL.

2.1 CROSS-DEVICE FEDERATED LEARNING

Cross-device FL (Kairouz et al., 2021) involves the distributed training of a machine learning (ML) model across a population of edge devices exchanging model updates with a central server through heterogeneous networks (McMahan et al., 2017; Li et al., 2020a). Clients in these settings usually possess minimal data samples and very heterogeneous data distributions Kairouz et al. (2021). Given the constraints of edge devices (e.g., limited processing power and battery life) and the heterogeneity of networks (e.g., diverse bandwidth), computational- and communication-efficient training methods are crucial for practical FL (Bonawitz et al., 2019). The research community has proposed means to optimize FL for communication efficiency (Sattler et al., 2019; Jiang et al., 2023) and computational efficiency (Horvath et al., 2021; Niu et al., 2022; Mei et al., 2022). Our approach aims to optimize both by leveraging sparse training.

2.2 SPARSE TRAINING

In centralized settings, sparse training tries to learn a sparse model during training to achieve model compression, lower computational demands during training, or faster inference. A typical sparse training pipeline tends to start with a random sparse network and follow a cycle of regular training, pruning, and regrowth (Mocanu et al., 2018; Mostafa & Wang, 2019; Dettmers & Zettlemoyer, 2019; Liu et al., 2020), acting only on model parameters. A more advanced technique relevant to our work, Sparse Weight Activation Training (SWAT) (Raihan & Aamodt, 2020), tailors the forward and backward passes, acting on both activations and model parameters, to induce a sparse weight topology and reduce the computational burden. In each forward iteration, SWAT selects the `Top-K` weights based on magnitude, using only these as the *active* weights. Only the highly activated neurons associated with `Top-K` pruned activations are considered for backpropagation during the backward pass. Notably, full gradients are still applied in the backward pass, allowing updates to both *active* and *inactive* weights. This mechanism enables the dynamic exploration of different network topologies throughout the training process.

2.3 WEIGHT RE-PARAMETRIZATION

Weight re-parametrization (Salimans & Kingma, 2016; Li et al., 2019; Gunasekar et al., 2017; Miyato et al., 2018; Vaskevicius et al., 2019; Kusupati et al., 2020; Schwarz et al., 2021; Zhao et al., 2022) in neural network training involves restructuring how weights are represented to improve training dynamics, optimize convergence, or introduce specific properties such as sparsity, without changing the network's architecture. Particularly relevant to our work, Schwarz et al. (2021) propose a sparsity-inducing weight re-parametrization that aims to shift the weight distribution towards higher density near zero, aiding in pruning low-magnitude weights. This simplification emerges by raising the model parameters to the power of $\beta > 1$ during the forward pass while preserving their sign. The re-parametrized weight vector component w is computed as $w = \text{sign}(v) \cdot |v|^{\beta-1}$, where v represents the original weight vector component, and β is a scalar value. Due to the chain rule, small-valued parameters receive smaller gradient updates, while large-valued parameters receive more significant updates, reinforcing a "rich get richer" dynamic.

3 SPARSE ADAPTIVE FEDERATED TRAINING

In the following, we present our *SparsyFed* method for sparse training in cross-device FL settings. Our method introduces a novel approach based on activation pruning and weight parametrization, applicable to any cross-device FL setting for obtaining a sparse global model. *SparsyFed* reduces the computational and communication overhead of highly heterogeneous FL environments, adapting the sparse mask of the global model. The procedure is outlined in Algorithms 1 and 2.

Algorithm 1 Sparse federated training pipeline of *SparsyFed*.

Require: ω_0: initial model's weights, β: weight re-parametrization exponent, $\hat{s}$: target sparsity
Require: T: number of federated rounds, E: number of client local epochs per round
Require: P: clients population, $\eta_t = \eta(t)$: learning rate scheduler as function of round t
Require: $\{\mathcal{D}_i\}_{i \in P}$: clients' datasets, B: local batch size

1: **procedure** *SparsyFed*
2: **for** $t = 0, \ldots, T-1$ **do**
3: Server samples a subset of clients $S_t \subseteq P$
4: **for** each client $i \in S_t$ **in parallel do**
5: $\omega_{i,0} \leftarrow \omega_t$
6: **for** $k = 0, \ldots, E-1$ **do**
7: $\omega_{i,k+1} \leftarrow \texttt{SparseClientOpt}(\omega_{i,k}, \mathcal{D}_i, B, \beta, \eta_t)$ ▷ See Alg.2
8: $\Delta\omega_i^t \leftarrow \omega_{i,E} - \omega_t$ ▷ Compute pseudo-gradient
9: $\Delta\tilde{\omega}_i^t \leftarrow \texttt{Top-K}(\Delta\omega_i^t, \hat{s})$ ▷ Prune $\Delta\omega_i^t$ w/ global unstructured `Top-K` using target $\hat{s}$
10: $\omega_{t+1} \leftarrow \texttt{OuterOPT}(\omega_t, \{\Delta\tilde{\omega}_i^t\}_{i \in S_t})$ ▷ Server optimization, e.g., Reddi et al. (2021)
11: **return** ω_T

Assumptions on the FL setting. As in any cross-device FL setting, the training is orchestrated by a parameter server (McMahan et al., 2017) that is in charge of initializing the global model, sampling a subset of clients every federated round, aggregating the pseudo-gradients after clients have trained on their local datasets. By following practical considerations (as extensively discussed in Bonawitz et al. (2019); Wang et al. (2021)), *SparsyFed* does not require any particular assumption on the client selection policy, nor on the server optimizer, nor the client optimizer. Thus, our algorithm's design allows it to benefit from any present or future federated optimizer practitioners, and researchers may develop without losing its properties. In particular, *SparsyFed* only requires the addition of one hyperparameter compared to standard dense training, whose sensitivity is discussed in Appendix E.2, making it most suitable for cross-device FL where the hyperparameter optimization (HPO) is challenging (Khodak et al., 2021). We also present a hyperparameter-free alternative in Appendix E.2.2. After initialization, the parameter server iteratively samples clients, broadcasts the latest version of the global model, collects the pseudo-gradients from clients, and aggregates the updates for obtaining the new global model, as described in Algorithm 1. As such, we assume agnosticism w.r.t. the server optimizer (line 10 in Algorithm 1) to allow practitioners to use their preferred one, e.g., `ServerOpt` in Reddi et al. (2021).

Algorithm 2 Sparse Client Optimization of *SparsyFed*

Require: ω_0: initial model's weights, β: weight re-parametrization exponent, η: learning rate
Require: $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1,\ldots,N}$: dataset composed of N samples with inputs $\mathbf{x}_i$ and outputs y_i
Require: $\mathcal{B}$: batch of data samples, B: batch size, $T = \lceil \frac{N}{B} \rceil$: number of batches, F: cost function
Require: $s(\theta_{t,l})$: function computing the sparsity of the layer $\theta_{t,l}$ at time t
Require: *size*(θ): computes the number of layers in the model θ
Require: $\texttt{GetLayer}(a_t, l)$: function extracts the weights and activations for the current layer.
Require: $\texttt{SetLayer}(\tilde{a}_{t,l}, l)$: function updates the sparse weights and activations in the model.

procedure SparseClientOpt($\omega_0, \mathcal{D}, B, \beta, \eta$)
for each step $t = 0, \ldots, T-1$ **do**
$\mathcal{B}_t \leftarrow \texttt{GetNextMiniBatch}(\mathcal{D}, B, t)$
$\theta_t \leftarrow \text{sign}(\omega_t) \cdot |\omega_t|^\beta$ ▷ Point-wise weights re-parametrization (Schwarz et al., 2021)
$a_t \leftarrow \texttt{Forward}(\theta_t, \mathcal{B}_t)$ ▷ Forward pass to compute activations
for each layer $l = 0, \ldots, \textit{size}(\theta_t) - 1$ **do**
$\theta_{t,l}, a_{t,l} \leftarrow \texttt{GetLayer}(\theta_t, l), \texttt{GetLayer}(a_t, l)$
$s_{t,l} = s(\theta_{t,l})$
$\tilde{a}_{t,l} \leftarrow \texttt{Top-K}(a_{t,l}, s_{t,l})$ ▷ Prune $a_{t,l}$ w/ unstructured `Top-K` using target $s_{t,l}$
$\tilde{a}_t \leftarrow \texttt{SetLayer}(\tilde{a}_{t,l}, l)$
$g_t \leftarrow \nabla F(\theta_{t,l}, \tilde{a}_t, \mathcal{B}_t)$ ▷ Compute gradients using sparse activations
$\omega_{t+1} = \texttt{ClientOpt}(\omega_t, g_t, \eta, t)$ ▷ Apply client optimizer, e.g., Reddi et al. (2021)
return ω_T

Sparsity-Inducing Weights Re-parametrization. Before the local forward pass at step t, we apply a sparsity-inducing re-parametrization to the local model weights ω_t, producing θ_t (line 4, Algorithm 2). As discussed in Section 2.3, re-parametrization techniques have been widely studied for various purposes. In our case, we aim to enhance pruning effectiveness and efficiency by inducing sparsity directly through optimization and adapting to input data. These outcomes are particularly beneficial in cross-device FL settings, where datasets are highly imbalanced, and some clients may only have a few samples (Section 2.1). From the available sparsity-inducing methods, we adopt Powerpropagation (Schwarz et al., 2021) due to its simplicity (introducing only one hyperparameter) and its ability to preserve the neural network's functional relationships during training. Furthermore, it avoids introducing non-uniform biases across layers, which improves its compatibility with pruning. Every parameter weight $w \in \omega$ is transformed into $v = \text{sign}(w) \cdot |w|^{\beta}$, where $\text{sign}(w)$ is the sign of w, $|w|$ is its L1 norm, and β is the Powerpropagation parameter. This re-parametrization enhances global training by promoting client consensus since the dynamics introduced naturally guide training toward the subset of non-zero weights in the model. Weights transitioning from zero to non-zero during training typically have smaller magnitudes, limiting their impact on updates. This ensures that clients focus on a shared subset of weights, facilitating aggregation.

Pruning Activations During Local Training. The local training procedure of *SparsyFed* is outlined in Algorithm 2 and relies on two main pillars. First, it ensures that the edge devices benefit from the model's sparsity by reducing memory footprint and FLOPs (depending on the underlying implementation, see Appendix H). Second, it guarantees the retention of as much information as possible during the training. We follow Raihan & Aamodt (2020) in three aspects. First, we use dense activation vectors (a_t, line 5) to retain all learned information during the forward pass. Second, we prune activations before the backward pass by aligning their sparsity with the weight vectors (lines 7–10). Specifically, activations are pruned layer-wise using the `Top-K` method with a target per-layer sparsity level ($s_{t,l}$, line 8), determined by the corresponding per-layer weight sparsity (line 9). Such pruning involves retrieving the weights and activations per layer (line 7) and updating the pruned activations before computing gradients. The weight parametrization preserves high sparsity throughout training (Fig. 12, in the appendix), ensuring consistent patterns between weights and activations and seamless integration into the model's sparse structure. Third, we keep gradient vectors (g_t, line 11) dense to avoid losing crucial information during updates. Pruning activations before the backward pass reduces computational cost while maintaining dense gradients for robust updates. Notably, initial model weights remain unpruned to allow meaningful training initialization, as clients train a dense model in the first round. With these principles, the local training procedure is designed to be optimizer-agnostic, e.g., compatible with `ClientOpt` (Reddi et al., 2021).

Pruning model parameters before communication. The data-driven and hyperparameter-less pruning procedure described above requires a further step to ensure compliance with the communication requirements. Clients receive a model parameters target sparsity, $\hat{s}$, from the server, which must be met before communicating the pseudo-gradient updates. Thus, the client applies a global unstructured pruning step based on `Top-K` using the target value $\hat{s}$ for the output sparsity, guaranteeing to save communication costs using a single parameter. Notably, this allows for non-uniform sparsity across layers, which has been proved to help maintain performance (Kusupati et al., 2020).

4 Experimental Design

Datasets and tasks. We selected three datasets to assess *SparsyFed*'s performance: CIFAR-10/100 (Krizhevsky, 2012), and Speech Commands (Warden, 2018). CIFAR-10 and CIFAR-100 datasets contain 32×32 color images of 10 and 100 classes, respectively. The Speech Commands dataset includes audio samples of 35 predefined spoken words and is used for a speech recognition task. Since these datasets' samples can be easily distributed across a pre-defined number of clients, they are common for simulating the heterogeneous data distributions of federated learning settings. On all datasets, we trained models for multi-label classification tasks.

Data partitioning and sampling. The datasets above are distributed among 100 clients and partitioned using the method in Hsu et al. (2019), simulating various degrees of data heterogeneity. The distribution of labels across clients is controlled via a concentration parameter α that rules a Latent Dirichlet Allocation (LDA), where a low α value translates to non-IID distribution and a high value to the IID distribution of labels. Specifically, we refer to data distributions as IID for $\alpha = 10^3$ and

non-IID for $\alpha = 1.0$ and $\alpha = 0.1$. To ensure reproducibility, we fixed the seed to 1337 for the LDA partitioning process. The federated orchestrator randomly sampled 10 clients out of the 100 clients in the population every round.

Model and training implementation. We employed a ResNet-18 (He et al., 2016) backbone for all experiments, adapting the classification layer to each specific task depending on the number of classes. ResNet-18 was chosen for its size, its popularity in the area, and the scalability of the ResNet family. While our training pipeline is implemented with `PyTorch` (Paszke et al., 2019), we designed custom layers and functions for some of *SparsyFed*'s components, such as layer-wise activation pruning and weight parametrization. We used `Flower` (Beutel et al., 2022) to simulate the federated learning setting. All models were trained from scratch without relying on any pre-trained weights.

Sparsity ratios. In our experiments, we targeted different values for the sparsity ratio in the set $\{0.9, 0.95, 0.99, 0.995, 0.999\}$. We chose 0.9 as the minimum value because it has been shown to bring effective gains (Frankle & Carbin, 2019) for both memory footprints and FLOPs (Appendix H). Our investigation spans to the extreme value of 0.999 to fairly present the downsides of these sparsity ratios. We applied the same target sparsity for all devices in the federation. For completeness, we show in Appendix E.6 how *SparsyFed* performs when adopting heterogeneous sparsity targets among devices in the federation.

Communication costs. Measuring the communication costs of *SparsyFed* is crucial for understanding its practical implications on cross-device FL settings. To make our analysis agnostic to any compression technique implementation for sparse unstructured models, we report the costs as the number of non-zero parameters effectively exchanged during the FL. Therefore, the communication cost is derived from the effective sparsity of the transmitted model, considering both the downlink (server-to-clients) and uplink (clients-to-server) communication steps. For clarity, we calculate the communication cost as if only one client participated, assuming all communication occurs in parallel. This ensures that our measurements reflect the total communication load without temporal delays between clients and avoids any bias introduced by the varying sampling proportion of clients in each round.

Reproducibility. Three different seeds were used for client sampling (5378, 9421, and 2035), while other stochastic processes were seeded with 1337 for reproducibility purposes.

5 Evaluation

This section discusses the evaluation of *SparsyFed* for adaptive sparse training in FL cross-device. The experimental results shown here aim to answer the following research questions.

1. Can *SparsyFed* mitigate the expected accuracy degradation at high and very high sparsity levels? We compare against the baselines, i.e., `Top-K`, ZeroFL (Qiu et al., 2021), and FLASH (Babakniya et al., 2023), for both the accuracy and the communication costs (Sections 5.1 and 5.2, respectively).
2. How does our adaptive sparsity pattern interact with the heterogeneous data distributions compared to other sparse training methods? Similarly to Babakniya et al. (2023), we analyze the consensus across clients on the sparse pattern (Section 5.3).
3. How do the main components of *SparsyFed* contribute to maintaining the accuracy at different sparsity levels? We ablate the activation pruning step (Section 5.5) and vary the weight re-parameterization (Section 5.4) to answer this question.

To ensure a fair and comprehensive comparison, we reimplemented state-of-the-art methods in our experiments, including ZeroFL, FLASH, and `Top-K` pruning. A detailed description of these baseline methodologies and their comparisons against *SparsyFed* is presented in Appendix G.

5.1 Accuracy Degradation

The analysis discussed in this paragraph demonstrates *SparsyFed*'s resilience to the performance degradation typically associated with pruning, as *SparsyFed* achieves competitive results even at high sparsity levels. Compared to other competitive methods under the same sparsity constraints, Table 1 shows that *SparsyFed* exhibits the lowest accuracy drop across all settings and target sparsity

Dataset	Sparsity	$\alpha = 1.0$				$\alpha = 0.1$			
		ResNet-18	ZeroFL	FLASH	***SparsyFed***	ResNet-18	ZeroFL	FLASH	***SparsyFed***
CIFAR-10	dense	83.70 ± 1.70	-	-	-	73.81 ± 4.84	-	-	-
	0.9	80.56 ± 1.90	76.16 ± 1.30	81.15 ± 1.03	$\mathbf{82.13 \pm 1.53}$	69.79 ± 3.78	67.40 ± 4.11	71.87 ± 2.63	$\mathbf{75.00 \pm 2.78}$
	0.95	74.71 ± 3.29	75.53 ± 2.27	79.36 ± 1.03	$\mathbf{82.60 \pm 1.58}$	60.00 ± 4.66	61.55 ± 4.18	72.08 ± 2.09	$\mathbf{75.95 \pm 3.39}$
	0.99	66.27 ± 5.08	70.71 ± 0.15	73.45 ± 1.37	$\mathbf{77.71 \pm 1.69}$	43.96 ± 11.99	51.71 ± 3.54	56.91 ± 3.55	$\mathbf{63.69 \pm 3.90}$
	0.995	63.82 ± 2.41	56.02 ± 3.95	69.15 ± 1.60	$\mathbf{70.01 \pm 0.43}$	19.02 ± 10.77	41.33 ± 3.64	52.15 ± 3.87	$\mathbf{56.79 \pm 3.97}$
	0.999	31.79 ± 19.10	17.66 ± 8.34	36.07 ± 7.49	$\mathbf{51.39 \pm 3.19}$	11.50 ± 4.49	18.76 ± 4.28	29.31 ± 6.75	$\mathbf{43.68 \pm 7.61}$
CIFAR-100	dense	52.29 ± 1.14	-	-	-	48.34 ± 2.71	-	-	-
	0.9	46.57 ± 1.71	40.70 ± 4.72	51.99 ± 0.21	$\mathbf{53.08 \pm 0.90}$	41.96 ± 2.16	31.92 ± 7.65	45.59 ± 0.75	$\mathbf{48.37 \pm 1.73}$
	0.95	28.07 ± 23.27	38.82 ± 1.75	47.19 ± 1.88	$\mathbf{52.81 \pm 1.72}$	11.48 ± 17.51	34.21 ± 7.65	44.31 ± 2.14	$\mathbf{48.27 \pm 2.70}$
	0.99	19.65 ± 16.30	18.97 ± 2.08	42.76 ± 4.08	$\mathbf{46.64 \pm 1.59}$	0.14 ± 0.72	13.07 ± 2.26	34.75 ± 3.38	$\mathbf{41.03 \pm 2.14}$
	0.995	9.51 ± 14.81	6.01 ± 4.74	36.43 ± 4.97	$\mathbf{42.21 \pm 1.03}$	0.14 ± 0.72	7.04 ± 5.25	26.44 ± 17.35	$\mathbf{35.72 \pm 2.01}$
	0.999	3.81 ± 2.18	1.96 ± 0.66	5.80 ± 2.86	$\mathbf{15.96 \pm 0.64}$	0.14 ± 0.72	1.66 ± 0.97	3.56 ± 2.07	$\mathbf{13.84 \pm 3.69}$
Speech Commands	dense	91.49 ± 0.94	-	-	-	80.15 ± 2.69	-	-	-
	0.9	84.28 ± 0.88	87.79 ± 1.40	88.68 ± 1.72	$\mathbf{92.32 \pm 1.59}$	65.44 ± 0.97	70.35 ± 2.65	77.15 ± 0.77	$\mathbf{79.67 \pm 2.78}$
	0.95	78.58 ± 0.44	84.29 ± 1.50	84.89 ± 0.49	$\mathbf{89.14 \pm 1.15}$	57.39 ± 1.04	65.90 ± 1.88	71.28 ± 1.75	$\mathbf{75.46 \pm 2.24}$
	0.99	65.01 ± 0.84	57.79 ± 0.82	69.22 ± 1.59	$\mathbf{75.82 \pm 3.72}$	50.42 ± 6.26	41.42 ± 1.60	53.55 ± 2.00	$\mathbf{56.69 \pm 4.56}$
	0.995	56.73 ± 1.00	37.16 ± 2.71	58.23 ± 1.84	$\mathbf{68.02 \pm 3.14}$	34.20 ± 1.43	22.61 ± 3.45	43.16 ± 3.47	$\mathbf{48.30 \pm 5.39}$
	0.999	21.56 ± 12.79	10.10 ± 4.01	17.70 ± 2.58	$\mathbf{47.43 \pm 1.66}$	19.25 ± 6.01	8.85 ± 3.76	17.14 ± 2.97	$\mathbf{29.24 \pm 2.34}$

Table 1: Aggregated results for CIFAR-10, CIFAR-100, and Speech Command datasets, with ResNet-18, ZeroFL, FLASH, and *SparsyFed* implementations.

levels. A noticeable drop in performance compared to the dense model was observed only at 99% sparsity. This advantage arises from using weight re-parameterization, which, in some cases, can even enhance the performance of the dense model. Additionally, the minimal performance drop at lower sparsity levels ($90 - 95\%$) allows *SparsyFed* to outperform the dense model in specific scenarios. To stress more our method capabilities, we increased the target sparsity to the point (99.9%) where *SparsyFed* is no longer able to retain sufficient accuracy. Our results show that all the baselines struggle to train effectively under such conditions. The adaptivity of *SparsyFed*'s sparsity patterns promotes consistent performance across clients, even in highly sparse settings, leading to a more synchronized and globally pruned model.

5.2 Communication costs

The promising accuracy achieved at very high sparsity ratios makes *SparsyFed* particularly suitable for cross-device FL settings, where communication costs are a critical concern. As illustrated in Fig. 2 (left), *SparsyFed* significantly outperforms the baselines regarding both communication savings ($19.29\times$ less communication costs compared to the dense model and $1.66\times$ compared to ZeroFL) and preserving required accuracy (consistently above 45%). FLASH has comparable communications costs, $0.97\times$ ours, but results in lower accuracy. Notably, *SparsyFed* consistently achieves higher accuracy for a given communication cost than the baselines, as shown in Fig. 2 (left). This advantage arises from *SparsyFed*'s ability to prune weights during client training and maintain a close-to-target sparsity ratio during server aggregation, effectively reducing uplink and downlink communication costs. Importantly, FLASH does not see an increase in model density after aggregation due to its fixed-mask local training regime, which prevents weight regrowth. This characteristic results in stable communication costs. In contrast, ZeroFL experiences a substantial increase in model density after aggregation, leading to a systematic and significant increase in downlink communication costs.

5.3 Consensus on the sparse masks across clients

Achieving consensus on the sparse masks of the clients' updates after each training round is crucial for achieving high accuracy. This consensus dynamic can be interpreted as the global model stabilizing around the target sparsity level as federated rounds progress. As shown in Fig. 2 (right), *SparsyFed* demonstrates minimal deviation from the target sparsity (90%), whereas ZeroFL and `Top-K` drop below 47% and 83%, respectively, during the initial training stage. *SparsyFed*'s consistency allows clients to effectively collaborate in training the same subset of parameters, which reduces the need for excessive pruning after local training and helps retain more helpful information. Such consistency appears despite our clients being allowed to dynamically change their mask during training, unlike fixed-mask approaches. For all methods, local updates maintain consistent sparsity due to post-training pruning; therefore, any increase in the global model's density indicates that local weight regrowth during training alters the local masks. Once these altered updates are ag-

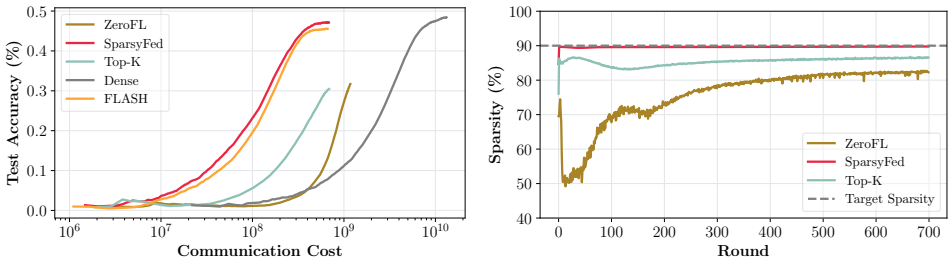

Figure 2: **(left)** The plot on the left compares accuracy versus communication cost for four implementations: ZeroFL, `Top-K`, FLASH, and *SparsyFed*, with the dense model as a reference. The test is conducted on CIFAR-100 partitioned with LDA($\alpha = 0.1$) and 95% sparsity. *SparsyFed* outperforms the baselines, achieving high accuracy and communicating less. **(right)** The plot on the right shows the global model sparsity level, measured on the server after aggregating local updates (CIFAR-100, $\alpha = 0.1$). The density gain reflects mismatches between client updates, causing the aggregated model to regain density, which can degrade performance and increase downlink communication. **Note**: FLASH maintains target sparsity after the first round with a fixed mask.

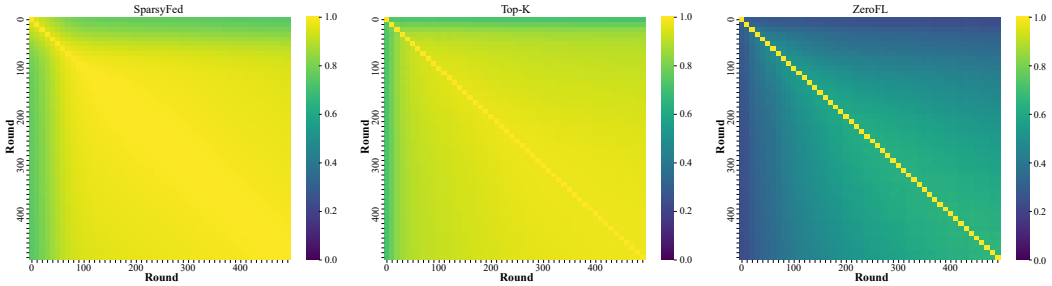

Figure 3: Intersection over Union (IoU) of global model binary masks between training rounds for *SparsyFed*, `Top-K`, and ZeroFL (CIFAR-100, $\alpha = 0.1$, 95% target sparsity). The IoU is calculated between each mask and all other masks across rounds to show changes over time. The x and y axes represent training rounds indices–the diagonal indicates the identity. Higher IoU values (close to 1.0) signify stronger similarity between masks, while lower values indicate significant changes. *SparsyFed* shows consistent mask movement with minimal variation, suggesting strong consensus on weight usage among clients. ZeroFL struggles to find mask consensus, with masks continuing to shift even in later rounds. **Note:** FLASH is absent since the global mask is fixed.

gregated, the result is a denser global model on the server. This misalignment among clients' masks can negatively impact both communication efficiency and accuracy. Global training appears to benefit from this consensus in terms of convergence, as shown in Fig. 2 (left), where both *SparsyFed* and FLASH outperform `Top-K` despite similar overall communication costs.

5.3.1 Sparsity pattern dynamics

We analyze the global model's sparse mask dynamics during training to shed light on *SparsyFed*'s ability to maintain a stable sparse pattern driven by two key factors: (1) focused weight utilization and (2) sparse mask adaptivity. First, the weight re-parameterization method targets a critical subset of weights, concentrating information where it is most impactful. This targeted approach improves training efficiency, enhances client collaboration, and reduces communication costs, as shown in Fig. 2. Second, unlike fixed-mask methods like FLASH, *SparsyFed* promotes dynamic mask adaptation, achieving robustness to heterogeneous data distributions throughout the training process. This allows a natural warm-up phase during which the sparsity pattern emerges organically, as shown in Fig. 3 (left). Clients rapidly converge on a stable shared mask, ensuring consistent performance. In contrast, on Fig. 3 (center), `Top-K` never fully settles on a stable mask, resulting in continuous variation across rounds while being faster in its initial rounds. In Fig. 3 (right), ZeroFL,

though more robust initially, struggles to maintain performance as it forces weight regrowth to adjust the mask, leading to instability in later rounds.

We do not consider FLASH in this comparison since it fixes its mask after the first round, eliminating any changes in the sparse mask pattern but reducing mask adaptivity entirely. While this strategy ensures that the model adheres to a fixed sparse structure, it introduces potential drawbacks. Specifically, since the mask is determined based on the data distribution observed in the first round, FLASH becomes highly dependent on the initial client data. This lack of adaptivity contrasts with our method, which allows continuous changes. Thus, it is more effective at handling shifts in data distribution across rounds.

5.4 Ablation on Weight re-parameterization

In this ablation study, we evaluated various weight re-parameterization techniques for their ability to sustain a sparsity-driven model while preserving dense-like performance. Since the sparse activations during the backward pass rely on a sparse weight model, the model must maintain high sparsity levels throughout training while minimizing accuracy loss.

We evaluated three approaches: fixed-mask training, spectral re-parameterization (Miyato et al., 2018), and Powerpropagation (Schwarz et al., 2021). We provide more details on the spectral re-parameterization in Appendix D.3. Each technique was applied to a ResNet-18 model trained with sparse activations for the backward pass and `Top-K` unstructured pruning at the end of each training round. This setup enabled direct comparison with a baseline model that lacked any re-parameterization. Among the methods, Powerpropagation proved to be most effective for our use case (Fig. 4), demonstrating superior resilience to preserve the accuracy with minimal degradation. By leveraging a "rich get richer" dynamic, Powerpropagation naturally induces sparsity during training. This enables the model to remain sparse during training, complementing our sparse activation backward pass and improving the model's overall efficiency and effectiveness.

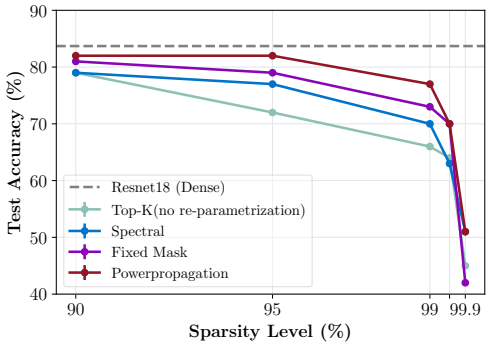

Figure 4: We report the test accuracy of different re-parameterization methods with sparse activations during backpropagation. We deployed a ResNet-18 trained on the CIFAR-10 dataset using LDA($\alpha = 1$). This plot illustrates the methods' performance under different sparsity levels. Powerpropagation exhibited superior robustness to the applied sparsity levels, achieving the best overall performance among these methods.

5.5 Ablation on Activation Pruning

The activation pruning step in *SparsyFed* (lines 6–10 in Algorithm 2) is designed to reduce computational costs of local training, which is particularly sensitive when executing on edge devices. This decision is motivated by prior studies Section 2 and Appendix H. As part of our ablation studies, we analyzed the impact of this pruning step on the accuracy and overall performance of the model. Since pruning activations during the backward pass do not directly affect weight density, we focus explicitly on its influence on test accuracy.

In Table 2, we compare *SparsyFed* with and without activation pruning during the backward pass. Our results show that activation pruning minimally impacts test accuracy at higher density levels. Significant performance degradation occurred only under extreme sparsity, where excessive pruning in specific layers substantially reduced activations, leading to diminished overall performance. Ultimately, the computational speedup achieved through activation pruning validates its inclusion in *SparsyFed*, as it balances efficiency with model accuracy.

Sparsity	$\alpha = 10^3$ (IID)		$\alpha = 1.0$ (non-IID)		$\alpha = 0.1$ (non-IID)	
	no Act. Pr.	Act. Pr.	no Act. Pr.	Act. Pr.	no Act. Pr.	Act. Pr.
0.90	83.92 ± 1.58	$\mathbf{84.31 \pm 0.86}$	$\mathbf{82.27 \pm 2.21}$	82.13 ± 1.23	$\mathbf{76.60 \pm 1.54}$	75.00 ± 2.78
0.95	83.80 ± 0.90	$\mathbf{84.25 \pm 1.38}$	81.53 ± 2.10	$\mathbf{82.6 \pm 1.58}$	75.29 ± 2.64	$\mathbf{75.95 \pm 3.39}$
0.99	$\mathbf{77.54 \pm 1.98}$	77.16 ± 0.85	75.76 ± 1.78	$\mathbf{77.71 \pm 1.69}$	$\mathbf{63.79 \pm 3.96}$	63.69 ± 3.90
0.995	$\mathbf{74.6 \pm 1.01}$	72.71 ± 0.65	$\mathbf{70.89 \pm 2.22}$	70.01 ± 0.43	$\mathbf{59.15 \pm 2.49}$	56.79 ± 3.97
0.999	$\mathbf{62.12 \pm 1.74}$	55.24 ± 2.09	$\mathbf{62.67 \pm 2.19}$	51.39 ± 3.191	$\mathbf{49.43 \pm 2.45}$	43.68 ± 7.61

Table 2: Accuracy comparison between *SparsyFed* with and without the pruning of the activation, on CIFAR-10 with LDA $\alpha = 10^3$, $\alpha = 1.0$, and $\alpha = 0.1$.

6 Related Work

Sparse training in centralized settings. Methods to enforce sparsity in neural networks can be grouped into two main categories: (1) dense-to-sparse methods that train a dense model and achieves a sparse model after training (Molchanov et al., 2017; Louizos et al., 2017), (2) sparse-to-sparse methods where pruning happens during training (Louizos et al., 2018; Dettmers & Zettlemoyer, 2019; Evci et al., 2020; Jayakumar et al., 2021; Raihan & Aamodt, 2020), thus theoretically reducing computational requirements (Bengio et al., 2015) and speeding up training. Our work introduces a novel method to implement sparse training directly within FL clients, inspired by sparse-to-sparse approaches in centralized settings such as Raihan & Aamodt (2020).

Pruning model updates in FL. After-training pruning, where clients first train dense models and then prune updates, is a common approach in FL (Sattler et al., 2019; Wu et al., 2020; Malekijoo et al., 2021). Sparse Ternary Compression (Sattler et al., 2019) combines `Top-K` pruning with ternary quantization to compress client updates, while FedZip (Malekijoo et al., 2021) uses layer-wise pruning, and FedSCR (Wu et al., 2020) employs patterns in client updates for more aggressive compression. However, these methods primarily improve communication efficiency without reducing computational overhead, as they still train dense models. In contrast, our method trains sparse models from the start, resulting in sparse updates and maintaining sparsity even after server aggregation, ensuring efficient upstream and downstream communication.

Sparse training in FL. Several studies have explored sparse learning in federated settings (Bibikar et al., 2022; Huang et al., 2022; Jiang et al., 2023; Qiu et al., 2021; Babakniya et al., 2023), but each has limitations. FedDST (Bibikar et al., 2022) applies RigL (Evci et al., 2020) to train sparse models, focusing on heterogeneous data without addressing extreme sparsity levels. FedSpa (Huang et al., 2022) uses a fixed sparse mask throughout training without a clear rationale behind it. PruneFL (Jiang et al., 2023) computes a sparse mask using biased client data and requires full gradient uploads, increasing communication costs. ZeroFL (Qiu et al., 2021) integrates SWAT (Raihan & Aamodt, 2020) for local training but struggles with weight regrowth, requiring pruning after each round, which can cause information loss. FLASH (Babakniya et al., 2023) introduces a fixed mask after a warm-up phase, but it depends heavily on the chosen clients during the first sampling and does not adapt to concept drift. In contrast, our method uses a dynamic sparse mask, offering more flexibility and better performance in highly non-IID FL settings.

7 Conclusions

This work presents *SparsyFed*, an adaptive sparse training method tailored for cross-device federated learning (FL). We show that *SparsyFed* can achieve impressive sparsity levels while minimizing the accuracy drop due to the compression. *SparsyFed* outperforms in accuracy three federated sparse training baselines, `Top-K`, ZeroFL, and FLASH, using adaptive and fixed sparsity for three typical datasets used in cross-device FL. We were able to ensure a limited drop in accuracy at sparsity levels of up to 95%, achieving up to a $19.29\times$ reduction in communication costs compared to dense baselines. The results presented in this work make our proposal particularly suitable for cross-device FL settings, which may require extreme communication cost reductions and the capability to adapt to heterogeneous distributions across federated rounds.

ACKNOWLEDGMENTS

This research was supported by the following entities: The Royal Academy of Engineering via DANTE (a RAEng Chair); the European Research Council, specifically the REDIAL project; Google through a Google Academic Research Award; in addition to both IMEC and the Ministry of Education of Romania (through the Credit and Scholarship Agency); and partly supported by the European Union under the NRRP partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART") and by the National PRIN JOULE project. Furthermore, we thank Javier Fernandez-Marques, Stefanos Laskaridis, Xinchi Qiu, and Francesco Corti for their invaluable feedback and collaborative spirit throughout this research project and the ICLR reviewers for helping improve our paper.

REFERENCES

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*, 2023.

Paolo Bellavista, Luca Foschini, and Alessio Mora. Decentralised Learning in Federated Deployment Environments: A System-Level Survey. *ACM Computing Surveys (CSUR)*, 54(1):1–38, 2021.

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2022.

Sameer Bibikar, Haris Vikalo, Zhangyang Wang, and Xiaohan Chen. Federated dynamic sparse training: Computing less, communicating less, yet learning better. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pp. 6080–6088. AAAI Press, 2022. doi: 10.1609/AAAI.V36I6.20555.

Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In Ameet Talwalkar, Virginia Smith, and Matei Zaharia (eds.), *Proceedings of the Second Conference on Machine Learning and Systems, SysML 2019, Stanford, CA, USA, March 31 - April 2, 2019*. mlsys.org, 2019.

Zachary Charles, Zachary Garrett, Zhouyuan Huo, Sergei Shmulyian, and Virginia Smith. On large-cohort training for federated learning, 2021a.

Zachary Charles, Zachary Garrett, Zhouyuan Huo, Sergei Shmulyian, and Virginia Smith. On large-cohort training for federated learning. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 20461–20475, 2021b.

Yae Jee Cho, Jianyu Wang, and Gauri Joshi. Client selection in federated learning: Convergence analysis and power-of-choice selection strategies. *CoRR*, abs/2010.01243, 2020.

Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples, 2017.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

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

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, 2021. URL https://arxiv.org/abs/2010.11929.

Hubert Eichner, Tomer Koren, Brendan McMahan, Nathan Srebro, and Kunal Talwar. Semi-cyclic stochastic gradient descent. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1764–1773. PMLR, 2019.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International conference on machine learning*, pp. 2943–2952. PMLR, 2020.

Farzan Farnia, Jesse M. Zhang, and David Tse. Generalizable adversarial training via spectral normalization, 2018.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks, 2019.

Scott Gray, Alec Radford, and Diederik P. Kingma. Gpu kernels for block-sparse weights. 2017. URL https://cdn.openai.com/blocksparse/blocksparsepaper.pdf.

Suriya Gunasekar, Blake Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nathan Srebro. Implicit regularization in matrix factorization, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

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.

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.

Erdong Hu, Yuxin Tang, Anastasios Kyrillidis, and Chris Jermaine. Federated learning over images: Vertical decompositions and pre-trained backbones are difficult to beat, 2023. URL https://arxiv.org/abs/2309.03237.

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

Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, Kaikai Wang, Anthony Shoumikhin, Jesik Min, and Mani Malek. PAPAYA: practical, private, and scalable federated learning. In Diana Marculescu, Yuejie Chi, and Carole-Jean Wu (eds.), *Proceedings of the Fifth Conference on Machine Learning and Systems, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*. mlsys.org, 2022.

Siddhant M. Jayakumar, Razvan Pascanu, Jack W. Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training, 2021.

Geonhwa Jeong, Po-An Tsai, Abhimanyu Rajeshkumar Bambhaniya, Stephen W. Keckler, and Tushar Krishna. Abstracting sparse DNN acceleration via structured sparse tensor decomposition. *CoRR*, abs/2403.07953, 2024.

Yuang Jiang, Shiqiang Wang, Víctor Valls, Bong Jun Ko, Wei-Han Lee, Kin K. Leung, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices. *IEEE Trans. Neural Networks Learn. Syst.*, 34(12):10374–10386, 2023. doi: 10.1109/TNNLS.2022.3166101.

Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Found. Trends Mach. Learn.*, 14(1-2):1–210, 2021. doi: 10.1561/2200000083.

Mikhail Khodak, Renbo Tu, Tian Li, Liam Li, Maria-Florina Balcan, Virginia Smith, and Ameet Talwalkar. Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 19184–19197, 2021.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, pp. 5544–5555. PMLR, 2020.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze (eds.), *Proceedings of the Third Conference on Machine Learning and Systems, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org, 2020a.

Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020b.

Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations, 2019.

Zi Lin, Jeremiah Zhe Liu, Zi Yang, Nan Hua, and Dan Roth. Pruning redundant mappings in transformer models via spectral-normalized identity prior, 2020.

Junjie Liu, Zhe Xu, Runbin Shi, Ray C. C. Cheung, and Hayden Kwok-Hay So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 3288–3298, 2017.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Ávila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 23190–23211. PMLR, 2023.

Amirhossein Malekijoo, Mohammad Javad Fadaeieslam, Hanieh Malekijou, Morteza Homayounfar, Farshid Alizadeh-Shabdiz, and Reza Rawassizadeh. Fedzip: A compression framework for communication-efficient federated learning. *arXiv preprint arXiv:2102.01593*, 2021.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *CVPR*, pp. 7765–7773. Computer Vision Foundation / IEEE Computer Society, 2018.

Brendan McMahan et al. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Yiqun Mei, Pengfei Guo, Mo Zhou, and Vishal Patel. Resource-adaptive federated learning with all-in-one neural composition. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry P. Vetrov. Variational dropout sparsifies deep neural networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2498–2507. PMLR, 2017.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pp. 4646–4655. PMLR, 2019.

Yue Niu, Saurav Prakash, Souvik Kundu, Sunwoo Lee, and Salman Avestimehr. Federated learning of large models at the edge via principal sub-model training. *CoRR*, abs/2208.13141, 2022. doi: 10.48550/ARXIV.2208.13141.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 8024–8035, 2019.

Sarthak Pati, Ujjwal Baid, Brandon Edwards, Micah Sheller, Shih-Han Wang, G Anthony Reina, Patrick Foley, Alexey Gruzdev, Deepthi Karkada, Christos Davatzikos, et al. Federated learning enables big data for rare cancer boundary detection. *Nature communications*, 13(1):7346, 2022.

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.

Md Aamir Raihan and Tor M Aamodt. Sparse weight activation training. In *Advances in Neural Information Processing Systems*, December 2020.

Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated learning for emoji prediction in a mobile keyboard. *CoRR*, abs/1906.04329, 2019.

Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*, 2019.

Jonathan Schwarz et al. Powerpropagation: A sparsity inducing weight reparameterisation. In *35th Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Esha Singh, Shane Bergsma, Nolan Simran Dey, Joel Hestness, and Gavia Gray. Empirical upper bounds for unstructured sparsity in compute-efficient language modeling. In *Workshop on Machine Learning and Compression, NeurIPS 2024*, 2024. URL `https://openreview.net/forum?id=qOnKSqiGtR`.

Jure Sokolic, Raja Giryes, Guillermo Sapiro, and Miguel R. D. Rodrigues. Generalization error of invariant classifiers, 2017.

Tomas Vaskevicius, Varun Kanade, and Patrick Rebeschini. Implicit regularization for optimal sparse recovery. *CoRR*, abs/1909.05122, 2019.

Ewen Wang, Boyi Chen, Mosharaf Chowdhury, Ajay Kannan, and Franco Liang. FLINT: A platform for federated learning integration. In Dawn Song, Michael Carbin, and Tianqi Chen (eds.), *Proceedings of the Sixth Conference on Machine Learning and Systems, MLSys 2023, Miami, FL, USA, June 4-8, 2023*. mlsys.org, 2023.

Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H. Brendan McMahan, Blaise Agüera y Arcas, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, Suhas N. Diggavi, Hubert Eichner, Advait Gadhikar, Zachary Garrett, Antonious M. Girgis, Filip Hanzely, Andrew Hard, Chaoyang He, Samuel Horváth, Zhouyuan Huo, Alex Ingerman, Martin Jaggi, Tara Javidi, Peter Kairouz, Satyen Kale, Sai Praneeth Karimireddy, Jakub Konečný, Sanmi Koyejo, Tian Li, Luyang Liu, Mehryar Mohri, Hang Qi, Sashank J. Reddi, Peter Richtárik, Karan Singhal, Virginia Smith, Mahdi Soltanolkotabi, Weikang Song, Ananda Theertha Suresh, Sebastian U. Stich, Ameet Talwalkar, Hongyi Wang, Blake E. Woodworth, Shanshan Wu, Felix X. Yu, Honglin Yuan, Manzil Zaheer, Mi Zhang, Tong Zhang, Chunxiang Zheng, Chen Zhu, and Wennan Zhu. A field guide to federated optimization. *CoRR*, abs/2107.06917, 2021.

Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

Peter Welinder, Steve Branson, Takayuki Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. *Technical Report CNS-TR-2010-001, California Institute of Technology*, 2010. URL `http://www.vision.caltech.edu/visipedia/CUB-200.html`.

Xueyu Wu, Xin Yao, and Cho-Li Wang. Fedscr: Structure-based communication reduction for federated learning. *IEEE Transactions on Parallel and Distributed Systems*, 32(7):1565–1577, 2020.

Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *CoRR*, abs/1812.02903, 2018.

Junbo Zhao, Yoon Kim, Kelly Zhang, Alexander Rush, and Yann LeCun. Adversarially regularized autoencoders. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5902–5911. PMLR, 10–15 Jul 2018.

Peng Zhao, Yun Yang, and Qiao-Chu He. High-dimensional linear regression via implicit regularization. *Biometrika*, 109(4):1033–1046, February 2022. ISSN 1464-3510. doi: 10.1093/biomet/asac010.

A IMPLEMENTATION

We implemented all experiments using a ResNet-18 backbone (He et al., 2016), which was adjusted to accommodate the number of classes for each task. The training pipeline was developed using `PyTorch` (Paszke et al., 2019), and specific components of the various methods, such as layer-wise activation pruning and weight re-parameterization, were implemented through custom layers and functions.

To simulate the federated learning environment, we used the `Flower` framework (Beutel et al., 2022). All models were trained from scratch, without pre-trained weights, to ensure a fair evaluation of *SparsyFed*'s performance under different sparsity settings. We only performed fine-tuning of a pre-trained model for the experiments on Visual Transformer (ViT) models.

We utilized a template from CaMLSys Lab to structure the code and ensure easy reproducibility. The complete code for the implementation is available in this repository.

B RELATED WORK

Sparse training in centralized settings. Methods to enforce sparsity in neural networks can be grouped into two main categories: (1) dense-to-sparse methods that train a dense model and achieves a sparse model after training (Molchanov et al., 2017; Louizos et al., 2017), (2) sparse-to-sparse methods where pruning happens during training (Louizos et al., 2018; Dettmers & Zettlemoyer, 2019; Evci et al., 2020; Jayakumar et al., 2021; Raihan & Aamodt, 2020), thus theoretically reducing computational requirements (Bengio et al., 2015) and speeding up training. Our work introduces a novel method inspired by sparse-to-sparse approaches in centralized settings such as Raihan & Aamodt (2020) to implement sparse training directly within FL clients.

Pruning model updates in FL. After-training model pruning, where clients regularly train their dense model and then apply pruning to the resulting updates, has been widely explored in the FL literature (Sattler et al., 2019; Wu et al., 2020; Malekijoo et al., 2021). Sattler et al. (2019) introduced Sparse Ternary Compression, which combines `Top-K` pruning with ternary quantization on client updates. Similarly, FedZip (Malekijoo et al., 2021) applies a layer-wise pruning approach, while FedSCR (Wu et al., 2020) leverages patterns in client updates for more aggressive compression. However, these methods primarily focus on enhancing communication efficiency without tackling the issue of reducing computational overhead during training, as they conduct regular training on a dense model before pruning weight updates. Our approach simultaneously addresses both concerns, as training a sparse model naturally yields sparse updates. Furthermore, existing after-training pruning methods typically focus solely on upstream communications or experience performance degradation when applied downstream. In contrast, our method effectively maintains sparsity after server-side aggregation, as clients rapidly converge on a shared sparsity mask, ensuring that server-to-client payloads remain sparse.

Sparse training in FL. Few studies have investigated the benefits of sparse learning in federated settings (Bibikar et al., 2022; Huang et al., 2022; Jiang et al., 2023; Qiu et al., 2021; Babakniya et al., 2023). Specifically, FedDST (Bibikar et al., 2022) used RigL (Evci et al., 2020) to train sparse models on clients but mainly focused on highly heterogeneous data distributions without considering extreme sparsity levels. FedSpa (Huang et al., 2022) employed a randomly initialized sparse mask that remained fixed throughout training, providing no clear justification for this approach. PruneFL (Jiang et al., 2023) computes the sparse mask during the initial round on a designated client using its potentially biased data. Qiu et al. (2021) empirically found that, after local training, the `Top-K` weights differ across clients, particularly at higher sparsity levels, leading to divergent sparse masks. This divergence makes aggregation inefficient and results in downstream dense models. In response, Qiu et al. (2021) proposed ZeroFL, which integrates unstructured SWAT during local client training. However, this alone does not guarantee achieving the desired sparsity level, as SWAT often leads to weight regrowth with each optimizer step. To address this, ZeroFL applies `Top-K` pruning before sending the model back to the server, ensuring the model meets the targeted sparsity. It is essential to highlight that ZeroFL applies the same sparsity level uniformly across all model layers, regardless of their sensitivity. The recent work in Babakniya et al. (2023) introduces FLASH, which employs a fixed binary mask throughout the training process. This mask is established during a warm-up

phase where a subset of clients ($[10, 20]$) train their model for a non-negligible number of epochs ($[10, 20, 40]$), and compute their per-layer sensitivity. The server aggregates the client's sensitivities to determine the mask. This mask's fixed nature means no weight regrowth is allowed, and no further pruning is required after the warm-up phase.

Cross-device federated learning. Federated Learning (FL) has emerged as a paradigm shift from traditional machine learning approaches, improving data privacy and moving the computation load to the network's edge. FL enables collaborative model training across decentralized devices while keeping data localized, thereby mitigating risks associated with centralized data aggregation (McMahan et al., 2017). In FL, client devices participate in model training via iterative rounds, aggregating local updates to build a global model. This distributed approach is advantageous for scenarios involving edge devices with limited computational resources and intermittent connectivity (Kairouz et al., 2021). Cross-device Federated Learning (FL) is particularly challenging due to the heterogeneous and resource-constrained nature of client devices, such as smartphones and IoT devices. The variability in hardware capabilities and data distributions across devices necessitates specialized techniques to optimize computation and communication techniques, such as quantization and model pruning, which have shown promise in reducing the amount of data that must be transmitted during each communication round. However, these methods often face challenges in maintaining model performance and achieving consensus on the sparsity patterns among clients.

Model pruning. Sattler et al. (2019) propose Sparse Ternary Compression (STC), a lossy compression scheme able to reduce the per-round communication cost of FL iterations significantly. STC first applies `Top-K` pruning after unaltered on-device training and then further compresses the weight updates using a ternary quantization. Other types of work have followed the idea proposed in Rigging the Lottery Ticket (Evci et al., 2020), where an initial sparse mask is fixed at the beginning of the training and remains primarily unchanged throughout. This approach allows clients to train only the initially fixed weights (Jiang et al., 2023; Babakniya et al., 2023).

Weight parametrization. Re-parameterization of weights in machine learning refers to the process of altering the re-parameterization of a model's weights to achieve various goals or facilitate specific training strategies (Li et al., 2019; Gunasekar et al., 2017; Zhao et al., 2022; Vaskevicius et al., 2019). This approach can be beneficial for enhancing the model's robustness to the application of sparsity. This work considers a re-parameterization based on Weight Spectral Normalization (Miyato et al., 2018) and Powerpropagation (Schwarz et al., 2021). The former re-parameterizes the weights based on the proportion of each weight relative to the one with the highest magnitude. At the same time, the latter applies an alpha power to the weights, inducing a "rich get richer" mechanism.

$$W_{\text{refactor}} = W \cdot \frac{W}{\sigma(W)}$$

C ADDITIONAL EXPLANATIONS ON *SparsyFed* COMPONENTS

Model preparation for pruning. The first step is to prepare the model for pruning by applying a re-parameterization to the weights at the layer level, as proposed in Schwarz et al. (2021). This approach leverages the information already present in the model by applying power to a specific value, β, to the network weights. This re-parameterization aims to induce a sparse representation of the weight of the network.

Following the original approach, each weight w_i is replaced by w_i^β. This transformation only applies to the neural network weights, leaving other parameters unchanged. Given the re-parameterized loss function $L(\mathbf{w}^\beta)$, the gradient with respect to $\mathbf{w}$ becomes:

$$\nabla_{\mathbf{w}} L(\mathbf{w}^\beta) = \nabla_{\mathbf{w}^\beta} L(\mathbf{w}^\beta) \cdot \beta \mathbf{w}^{\beta-1}$$

Here, $\nabla_{\mathbf{w}^\beta} L(\mathbf{w}^\beta)$ is the gradient concerning the re-parameterized weights. This gradient is scaled element-wise by $\beta \mathbf{w}^{\beta-1}$, which adjusts the step size proportionally to the magnitude of each weight. This update is distinct from simply scaling the gradients in the original re-parameterization, as it

directly modifies the re-parameterized weights, not the original weights. This modified gradient step enables a "rich-get-richer" dynamic where the gap between high and low-magnitude weights increases, creating a natural separation between them. This makes the model relatively insensitive to pruning.

Weight pruning. Our first concern was to reduce the model's size during communication rounds to speed up FL training. In this context, a compression mechanism is typically used to reduce the payload size that has to be exchanged. To achieve this, reducing the number of parameters in the network is necessary, inducing a high level of sparsity. We used a `Top-K` pruning method to remove all low-magnitude weights from the network. This pruning operation is implemented at the end of each local training on the client. This decision is based on several observations: (a) since the parametrization affects gradient descent, pruning before training would not be beneficial, (b) the server is not aware of the features of the data on the clients, and initializing the sparsity mask on the server side could cause a significant drop in performance.

We also decided to induce sparsity from the very first round. The clients would only use a small portion of the weights during training, so they must start training with the restricted fraction to maximize their effectiveness. Sparsity is induced from the first round of local training on each device, allowing the local updates to be sparse from the beginning of the federated training.

During the following training rounds, a second quality of Powerpropagation comes into play, drastically reducing the regrowth of new weights during training. This results in minimal shifts of the model's sparsity mask from the global one received from the server. Thus, the sparsity mask of the new model update sent to the server will differ very little from the global one. In other words, using Powerpropagation, we are forcing all clients to converge towards a shared sparsity mask, similar to Evci et al. (2020), without inhibiting the regrowth of new paths. The clients decide the mask in the first round of training based on their local data. This means that the global model will remain highly sparse even after aggregating new local updates from the clients. This result allows for significant compression during the downlink communication from the server to the clients in subsequent training rounds.

Activation pruning. To further reduce the computation footprint during the training, we implemented a layer-wise pruning similar to the one proposed in Raihan & Aamodt (2020). The original proposal was to speed up inference and training by reducing the number of computational operations, inducing a fixed level of sparsity on the weights of each layer during both the forward and backward passes. However, this could cause a significant drop in performance with high sparsity levels, as not all layers retain the same level of information. Applying the same sparsity to all layers could negatively impact those that naturally retain more information. In our implementation, we heavily modified the approach, retaining only key concepts. Since we keep the model sparse during almost all training rounds, except for the first, pruning the weights during the forward pass is not necessary.

For the pruning of activations during the backward pass, instead of applying the same level of sparsity to all layers of the model, the global sparsity mask is used to determine the pruning sensitivity of each layer. The level of pruning applied to the activations is directly proportional to the level of the sparsity of the weights in the same layer, allowing layers that retain more information to maintain denser activations while drastically reducing the activations in layers that reach a high level of sparsity.

D Additional Background

D.1 Powerpropagation

Powerpropagation is a weight re-parameterization technique that induces sparsity in neural networks. Essentially, it causes gradient descent to update the weights in proportion to their magnitude, leading to a "rich get richer" dynamic where small-valued parameters remain largely unaffected by learning. As a result, models trained with Powerpropagation exhibit a weight distribution with significantly higher density at zero, allowing more parameters to be pruned without compromising

performance. Powerpropagation involves raising model parameters to the power of β (where $\beta > 1$) in the forward pass while preserving their sign. This re-parameterization can be expressed as

$$w = v \cdot |v|^{\beta-1}$$

where w represents the weights, and v are the re-parameterized parameters. Due to the chain rule, this re-parameterization causes the magnitude of the parameters (raised to $\beta-1$) to appear in the gradient computation. Consequently, small-valued parameters receive smaller gradient updates, while large-valued parameters receive larger updates, thus amplifying the "rich get richer" dynamic. Powerpropagation leads to intrinsically sparse networks, meaning that a significant portion of the weights converge to values near zero during training. This property allows for the pruning (removal) of many weights without significantly compromising model performance. Powerpropagation can also be easily integrated with existing sparsity algorithms to enhance performance further. Studies show the benefits of combining Powerpropagation with popular methods such as Iterative Pruning and TopKAST (Gale et al., 2019; Jayakumar et al., 2021).

D.2 Fixed mask sparse training (FLASH)

Training a fixed subset of weights can be a practical approach for sparse model training in Federated Learning (FL), offering several significant benefits. Limiting the number of active (non-zero) weights reduces the computational and memory demands compared to dense models, which is particularly advantageous given the resource constraints often found in client devices within FL. In addition to these resource savings, sparse models enhance communication efficiency between clients and servers. Since only the active weights need to be transmitted, the message size is reduced, leading to faster training and lower bandwidth usage.

Another advantage of this method is its potential to uncover "winning tickets" within neural networks Frankle & Carbin (2019). Research indicates that dense, randomly initialized networks often contain sparse sub-networks, known as "winning tickets", which, when trained independently, can achieve performance similar to the full model. Training a fixed subset of weights promotes this sparsity and encourages the model to learn more efficient representations, potentially revealing these sub-networks.

Despite these benefits, specific challenges arise when using a fixed subset of weights in FL. If the mask is not initialized correctly or fails to adapt during training, the model's performance may be compromised. To mitigate these issues, strategies such as sensitivity-based pruning and selective mask updates are crucial for fully leveraging the advantages of sparse learning in FL Babakniya et al. (2023). The training process for sparse models can be formalized as:

$$W_{\text{sparse}} = M \odot W$$

where W_{sparse} represents the sparse weights, M is the binary mask (with values 0 or 1), and $\odot$ denotes element-wise multiplication.

D.3 Spectral Normalization

Spectral Normalization (SN) is a weight normalization technique that aims to limit the most significant singular value of each weight matrix, thereby controlling the Lipschitz constant of the function represented by the network. This is achieved by constraining the spectral norm of each layer $g : h_{in} \to h_{out}$. Formally, given a weight matrix W, its spectral norm is defined as the most significant singular value $\sigma(W)$, i.e.,

$$\sigma(W) = \max_{h \neq 0} \frac{\|Wh\|_2}{\|h\|_2}$$

SN normalizes the weight matrix W by dividing it by its spectral norm:

$$\bar{W} = \frac{W}{\sigma(W)}$$

This ensures that the Lipschitz constant of each layer remains bounded, promoting stability during training, particularly in scenarios where gradients may explode. Originally proposed as a regularization technique to stabilize discriminator training in Generative Adversarial Networks (GANs) Miyato et al. (2018), SN has since found broader applications, including improvements in generative neural networks Zhao et al. (2018), and has been theoretically linked to enhanced generalization and adversarial robustness Farnia et al. (2018); Sokolic et al. (2017); Cisse et al. (2017).

In the context of federated learning and sparse training, SN can significantly improve the resilience of models to pruning. SN regularizes the model mappings by enforcing the Lipschitz constraint, reducing sensitivity to high sparsity levels. This effect has been observed in prior work on pruning, where SN was used to prune redundant mappings and enforce a spectral-normalized identity prior Lin et al. (2020).

However, by strictly constraining the Lipschitz constant, SN may reduce the model's flexibility during training, potentially affecting convergence in some cases. To address this, following an approach similar to those proposed in other re-parameterization works Vaskevicius et al. (2019), we propose a slightly different method where the spectral norm is modified to enhance the distribution of the weights. Specifically, we redefine the weight update rule as:

$$w = w \cdot \frac{|w|}{\sigma(W)}$$

This approach allows us to maintain most of the model's performance while improving its resilience to pruning.

E Additional Experimental Results

E.1 Naive Powerpropagation Federated Adaptation

In the original Powerpropagation paper, pruning was applied at the end of centralized training after training a dense model. To create a baseline for comparison, we implemented a naive federated version of Powerpropagation that follows this methodology. In this version, pruning is not applied during local training at the client level. Instead, clients train with the full model, sending and receiving full model updates. At the end of the federated training (after the final round), pruning is applied to the global model using the `Top-K` method.

We compare this naive version of federated Powerpropagation to our proposed method, *SparsyFed*, across various sparsity levels and different levels of data heterogeneity, in Table 3. As shown in the results, *SparsyFed* significantly outperforms the naive federated Powerpropagation in terms of performance and stability, even in sparsity.

Sparsity	$\alpha = 10^3$ (IID)		$\alpha = 1.0$ (non-IID)		$\alpha = 0.1$ (non-IID)	
	Naive PP	***SparsyFed***	Naive PP	***SparsyFed***	Naive PP	***SparsyFed***
0.000	84.69 ± 1.57	-	84.31 ± 1.01	-	74.86 ± 2.28	-
0.900	71.41 ± 7.01	**84.31 ± 0.86**	66.70 ± 3.76	**82.13 ± 1.53**	45.82 ± 6.32	**75.00 ± 2.78**
0.950	35.02 ± 6.37	**84.25 ± 1.38**	33.84 ± 13.71	**82.60 ± 1.58**	24.28 ± 8.79	**75.95 ± 3.39**
0.990	12.40 ± 4.17	**77.16 ± 0.85**	12.30 ± 3.68	**77.71 ± 1.69**	9.28 ± 2.52	**63.69 ± 3.90**
0.995	10.33 ± 0.57	**72.71 ± 0.65**	10.03 ± 0.05	**70.01 ± 0.43**	9.74 ± 3.24	**56.79 ± 3.97**
0.999	9.86 ± 0.25	**55.24 ± 2.09**	10.01 ± 0.02	**51.39 ± 3.19**	11.67 ± 6.66	**43.68 ± 7.61**

Table 3: Accuracy comparison for Naive Powerpropagation and *SparsyFed* on CIFAR-10 with different LDA settings ($\alpha = 10^3$, $\alpha = 1.0$, and $\alpha = 0.1$).

E.2 Powerpropagation Exponent in *SparsyFed*

Applying Powerpropagation in *SparsyFed* introduces a new hyperparameter that must be tuned alongside others. To address this, we explored different approaches. The first follows the methodology of the original paper, where a series of fixed exponents were proposed for re-parameterization. In the latter approach, we propose a novel method for determining the exponent dynamically, making it dependent on the network's weights rather than being predefined. This follows a strategy similar to spectral normalization.

E.2.1 Sensitivity Analysis of Fixed Powerpropagation Exponent

Following the approach of the original paper, we evaluated different values of β to assess their impact on model performance. Preliminary tests suggest that the sensitivity of this hyperparameter is not as critical as initially expected. We tested several values proposed in the original paper to evaluate their effectiveness in sparse training within a federated learning setting. As shown in Fig. 5, any β value between 1 and 2 significantly improves performance in dynamic sparse training compared to the baseline without re-parameterization.

E.2.2 Hyperparameter-Free Powerpropagation Exponent

To eliminate the need for an additional hyperparameter, we propose an alternative version of *SparsyFed* where β is computed at runtime. Instead of a fixed β for Powerpropagation, a tailored exponent is derived based on the layer-wise weight magnitude distribution, leveraging concepts from the spectral norm.

We denote the spectral norm of a weight matrix $\mathbf{W}$ as $\text{spectral_norm}(\mathbf{W})$, which computes the maximum magnitude of the elements in $\mathbf{W}$ and normalizes it by dividing $\mathbf{W}$ by its maximum value $\mathbf{W}_{\max}$. Thus, $\text{spectral_norm}(\mathbf{W})$ results in a matrix of positive values ranging between 0 and 1. Using $\text{spectral_norm}(\mathbf{W})$, we construct a tailored exponent matrix for the network's weights by implementing custom convolutional and linear layers, where the matrix is computed at the beginning of the forward pass. The exponentiation process raises each weight w in a layer to the power of

$1 + \text{spectral_norm}(w)$, ensuring that each weight is scaled by a factor between 1 and 2, proportional to its relative magnitude compared to the largest weight in the network.

To reduce computational overhead, the exponent matrix is calculated only during the first forward pass and stored for subsequent iterations. Additionally, to minimize memory usage, we avoid storing the full exponent matrix and instead compute and store only the average value per layer, reducing memory requirements to a single scalar per layer. This averaging approach results in higher values for denser layers with more nonzero weights, while highly sparse layers tend to have significantly smaller values. The sparsest layers, often found toward the end of the network, tend to be larger and more sparsely populated, amplifying this effect.

The overall computational cost remains marginal and comparable to the standard Powerpropagation implementation, as the computation is only performed during the first forward pass. Performance analysis in Fig. 5 shows that this method outperforms naive `Top-K` pruning without reparameterization and matches or surpasses most fixed alpha values. However, it still falls short of the best-performing fixed β, suggesting that further refinements would be necessary to bridge this performance gap. While more extensive experiments are needed, this represents a useful insight into potential enhancements for this approach.

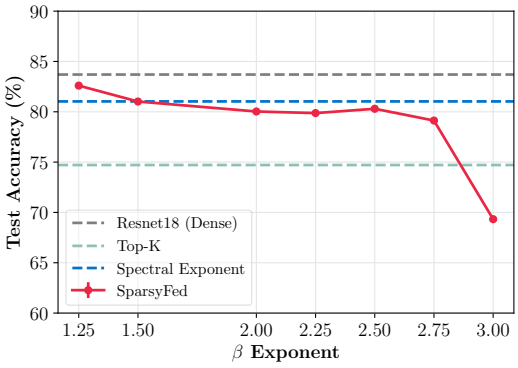

Figure 5: Test Accuracy with different β values, with 95% sparsity on CIFAR-10 LDA $\alpha = 1.0$. The accuracy of the dense model (gray), the hyperparameter-free Spectral Exponent version, and the `Top-K` method are also reported for reference.

E.3 Weight Movement Metrics in Federated Learning

To better understand the aftermath of *SparsyFed* on the weight of the model during training, a comprehensive evaluation of the weight evolution throughout training has been conducted by comparing the initial global model, the local updates sent to the server at the end of each round, and the subsequently obtained aggregated global model. This allows for gated metrics that capture the overall training progression and the round-specific dynamics. We focus on two primary metrics: the L2 norm between weight matrices and cosine similarity. The former (L2) quantifies the magnitude of weight changes. It is measured as follows: (i) the cumulative L2 norm between the initial model and the aggregated model at each round for long-term evolution, and (ii) the L2 norm between consecutive global models to assess round-level variation. In contrast, cosine similarity captures functional consistency in the updates. It is measured as (i) the similarity between client updates to assess alignment across non-IID data distributions and (ii) between consecutive global models to evaluate directional stability in weight updates.

Experiments were conducted using ResNet18 on CIFAR-10, CIFAR-100 and Speech Commands, with data partitioned using LDA ($\alpha = 0.1$) to simulate non-IID distributions. The model was trained for 200 rounds using *SparsyFed* and `Top-K` under sparsity levels of 95%. The results reveal distinct training behaviors across methods.

As shown in Fig. 6, the global L2 distance—measuring the deviation from the initial model—smoothly increases in *SparsyFed*, whereas in other implementations, it rises sharply before stabilizing after several rounds, indicating excessive drift in the early stages. In contrast, *SparsyFed*

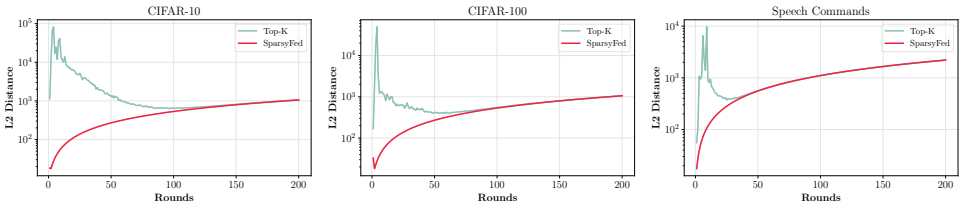

Figure 6: Global L2 Norm – This metric tracks how far the global model moves from its initial state throughout training. A smoother increase indicates more stable updates, while sharp rises suggest rapid drift.

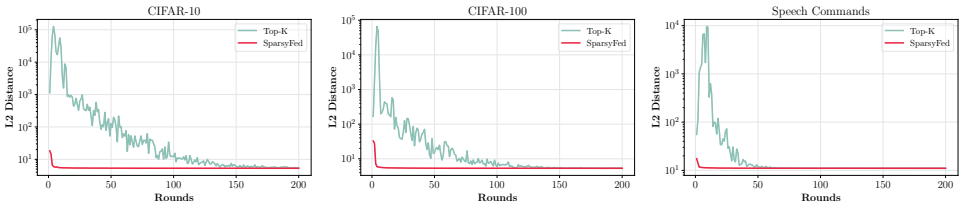

Figure 7: Round L2 Norm – By measuring the L2 norm between consecutive global models, this metric highlights the magnitude of updates applied at each training round, revealing how steadily or abruptly the model evolves.

exhibits a more gradual trajectory, suggesting stable and incremental updates. A similar trend is observed when examining round-by-round evolution, highlighting the steps taken by the model after each aggregation. Fig. 7 illustrates the high variance in updates for `Top-K`, whereas *SparsyFed* maintains stable and smaller updates.

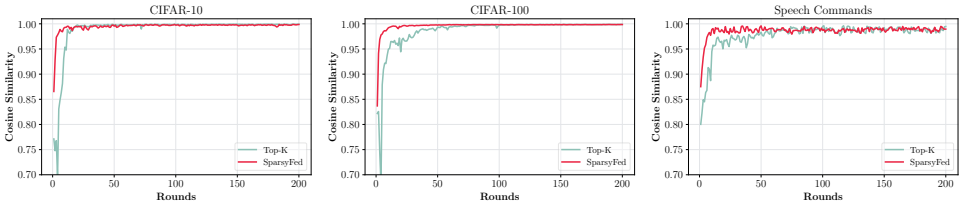

Figure 8: Round Cosine Similarity – This captures the consistency of global model updates across rounds. A high and stable similarity suggests smooth training dynamics, while fluctuations may indicate instability in model aggregation.

Cosine similarity metrics provide complementary insights. The round-wise cosine similarity (Fig. 8) represents the cosine similarity between the global model before local training and after aggregation. This measurement shows how much the model is modified by a training round. The results highlight that *SparsyFed* quickly attains high alignment and maintains stability throughout training, while `Top-K` exhibits more significant fluctuations in the initial rounds, suggesting inconsistent updates across rounds.

Figure 9 shows client round-wise cosine similarity, which measures the similarity among updates sent to the server by clients at the end of each training round. The results indicate that *SparsyFed* tends to have a faster and smoother trajectory. In both cases—client and round— all methods eventually achieve high similarity, though at different stages of training. Notably, *SparsyFed* demonstrates the fastest convergence and smoother dynamics. A slightly lower final similarity value in the CIFAR-10 experiment is not concerning, provided it remains consistent and stable throughout training, as this may indicate better handling of data heterogeneity, allowing clients to adapt more effectively to their respective data distributions.

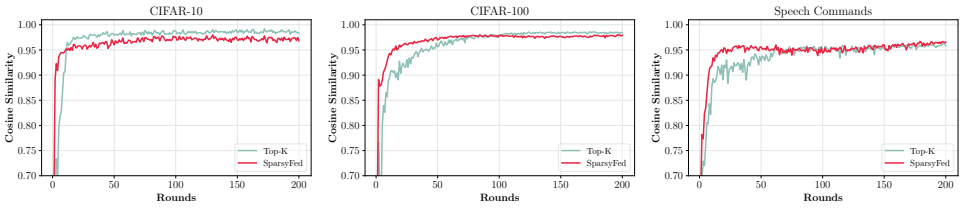

Figure 9: Client Cosine Similarity – This measure assesses how similar client updates are. Higher similarity suggests greater alignment across different clients, while lower values indicate more diverse local updates due to data heterogeneity.

E.4 Global models sparsity levels

The global model's sparsity fluctuates during training due to the mismatch in client updates. As a result, the global model's sparsity is not always fixed and can fluctuate significantly throughout the training process. The following figures show that the sparsity target directly influences the sparsity level. Higher sparsity targets tend to lead to more significant weight regrowth during training, as seen in Fig. 12, which results in a more considerable mismatch on the server, leading to a denser model. This effect is particularly noticeable at the beginning of training, when the model is more susceptible to significant changes in shape, as illustrated in Fig. 3. Following this, we present a plot of the sparsity measure on the global model for different sparsity targets during training.

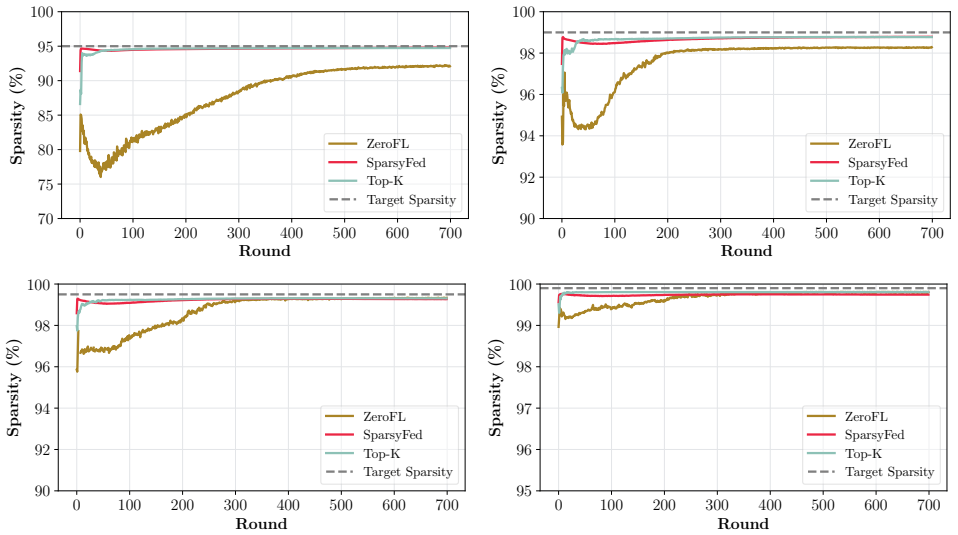

Figure 10: **(a)** From the top left to the bottom right, we present the plots of the sparsity levels for ZeroFL, *SparsyFed*, and `Top-K` at sparsity levels of 95%, 99%, 99.5%, and 99.9%. ZeroFL struggles to reach the target sparsity in all cases, partially due to its aggregation method, which only aggregates non-zero weights. This leads to large magnitude weights, even if they are present in only a fraction of the clients. *SparsyFed* and `Top-K` tend to reach the target more quickly, with *SparsyFed* showing a small fluctuation in the initial training phase due to the movement of the mask, as shown in Fig. 3.

E.5 Distribution of the sparsity through the layers.

Sparsity distribution is crucial as the sparsity achieved in the weights of each layer is used to determine the sparsity level applied to the activations during the backward pass. Each layer is pruned with a distinct sparsity level based on the information it contains, leveraging layer sensitivity to implement an effective pruning strategy for activations. This ensures that the sensitivity observed in the weights is reflected in the activation pruning, allowing for dynamic sparsity for both weights

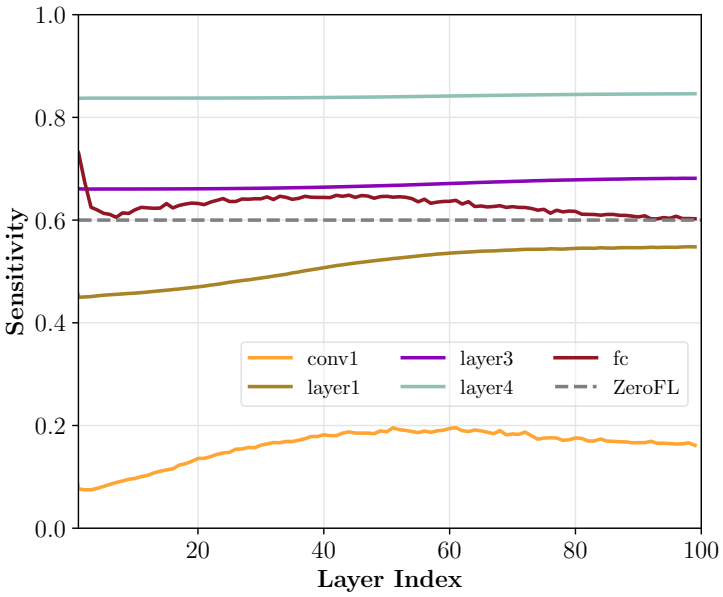

Figure 11: Different sparsity ration of some relevant layer, ZeroFL (red) vs *SparsyFed* with Powerpropagation and sparsity ratio of 60%. The first layers are not shown as they are kept dense in ZeroFL. Empirical observation regarding the nature of the pruning procedure (constant across layers for ZeroFL, variable unstructured across layers for *SparsyFed*). We could use this in the main paper only if we make the case that our method performs better because of this. We consider this appendix material for now.

and activations based on the natural sensitivity of each layer. Empirical evidence, shown in Fig. 11, supports this approach, showing that in ZeroFL implementations, the sparsity of the layers remains uniform across all layers of the model, while in others, the sparsity levels vary significantly between layers. The first layers tend to be nearly fully dense, while the deeper layers exceed the global sparsity target, indicating that the first layers retain more information than the deeper layers.

A key factor in this behavior is the limited weight regrowth observed in *SparsyFed*, where only a small number of weights transition from zero to non-zero values after each training round. As depicted in Fig. 12, *SparsyFed* exhibits minimal regrowth, stabilizing quickly over a few rounds. This is directly attributable to using Powerpropagation during training, drastically reducing the impact of smaller weights. Although this behavior was not highlighted in the original paper, it represents empirical evidence supporting the effectiveness of an inherited sparse model training procedure.

As illustrated in Fig. 3, our implementation shows that the global model's mask shifts slightly during the initial training rounds. This suggests that overly rigid approaches, such as those proposed in FLASH, may negatively impact performance by failing to accommodate necessary flexibility. On the other hand, *SparsyFed*'s approach maintains flexibility while still converging toward a consistent global mask.

E.6 Heterogeneous Sparsity Experiments

In this experimental setup, we aim to evaluate heterogeneous sparsity by using two distinct sets of model sparsity:

1. $[\mathbf{0.9}, \mathbf{0.85}, \mathbf{0.8}]$: These sparsity levels are based on the settings proposed by FLASH in their heterogeneous setup. These values represent a moderate range of sparsity, which does not significantly impact the model's performance in this task.
2. $[\mathbf{0.99}, \mathbf{0.95}, \mathbf{0.9}]$: These denser models previously tested in other experiments offer a more challenging setup. *SparsyFed* outperforms FLASH in these settings due to its ability to adapt efficiently to varying sparsity levels across clients.

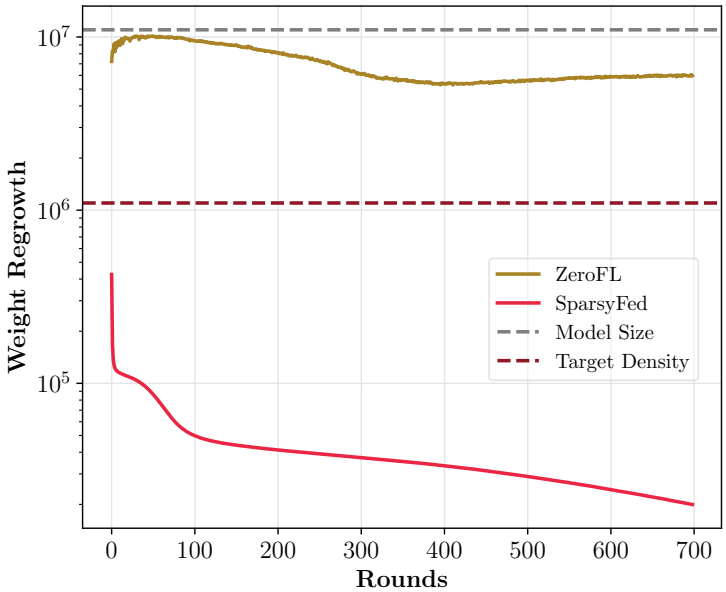

Figure 12: Number of weights regrown (weights switching from zero to non-zero) with one local epoch, compared between *SparsyFed* and ZeroFL. ZeroFL performs the worst of all tested implementations, with 50-80% of weights regrowing after local training. In contrast, *SparsyFed* shows near-zero regrowth (200× lower than ZeroFL). This ensures that the movement of the mask focuses on important weights rather than being distributed across many. **Note**: FLASH does not allow weight regrowth due to its fixed-mask training approach, so it is excluded from this graph.

The clients are partitioned into three groups containing $[40, 30, 30]$ clients. Each group trains on a different level of sparsity in the model. They receive a model matching their capability and send back an update of the same dimensions. For clarity, a client in group 0 in the first setting will receive a model with 0.9, train on it, and then send back a sparse update with 0.9. As shown in Table 4, we achieve high performance in all the settings, with a clear margin in the more extreme setting

		Lower Sparsity		Higher Sparsity	
Setting	LDA	FLASH	*SparsyFed*	FLASH	*SparsyFed*
Moderate	$\alpha = 10^3$	83.15 ± 0.96	$\mathbf{83.28 \pm 0.44}$	$\mathbf{83.77 \pm 0.84}$	83.27 ± 0.44
	$\alpha = 0.1$	70.9 ± 0.92	$\mathbf{74.97 \pm 2.39}$	74.65 ± 0.96	$\mathbf{74.98 \pm 2.39}$
Extreme	$\alpha = 10^3$	74.72 ± 0.49	$\mathbf{81.04 \pm 0.26}$	74.72 ± 0.67	$\mathbf{81.20 \pm 0.39}$
	$\alpha = 0.1$	57.63 ± 4.83	$\mathbf{69.74 \pm 0.28}$	59.82 ± 2.99	$\mathbf{69.96 \pm 0.19}$

Table 4: Performance comparison between FLASH and *SparsyFed* across different heterogeneous sparsity settings and LDA values. The model has been evaluated on all the sparsity-level trained. For simplicity, we show the test accuracy for the denser model (which reaches a density equal to the target density trained, 0.8 and 0.95 for moderate and extreme settings) and the less dense models (0.9 and 0.99 for moderate and extreme). While in the moderate setting, the performances are similar and in line with the dense model's performance (see Table 1), in the extreme setting, *SparsyFed* demonstrated more versatility, achieving high performance even with the sparsest model. It is important to note that in such a setting, the model's performance would align with the sparsest model trained, especially considering it is trained from a larger (though not majority) group of clients compared to the others.

E.7 WIDE FEDERATED CIFAR10

In this experiment, we aimed to address the challenge of training a model in a setting where the number of samples per client is extremely low. To do this, following an approach similar to the one proposed in Charles et al. (2021a), we partitioned the CIFAR-10 dataset into 1000 clients following an LDA distribution with $\alpha = 0.1$. As shown in Table 5, *SparsyFed* significantly outperformed the alternative in this setting, where clients retain minimal information. Notably, both sparse methods outperformed the dense one in this setting, likely due to the dilution of information in the dense model when such a small amount of data is used at each round.

Method	Accuracy (%)
Dense	47.11 ± 2.77
FLASH	51.96 ± 2.84
SparsyFed	$\mathbf{54.37} \pm \mathbf{1.28}$

Table 5: Performance comparison of the dense model, FLASH, and *SparsyFed*. Both FLASH and *SparsyFed* have been trained with a sparsity of 0.95. The experiments were conducted on CIFAR-10, partitioned across 1000 clients following LDA with $\alpha = 0.1$, and a participation rate of 0.1%.

E.8 VISION TRANSFORMER

To extend our experimental evaluation, we designed an experiment using a Vision Transformer (ViT) (Dosovitskiy et al., 2021) on the CUB-200-2011 dataset (Welinder et al., 2010), following the setup proposed in (Hu et al., 2023). Specifically, we use a pre-trained ViT-Base model on ImageNet-21k (Deng et al., 2009) and fine-tuned on CIFAR-100.[1] The data is partitioned using LDA with $\alpha = 1000.0$ among 100 clients, each receiving approximately 60 samples. The participation rate for training and evaluation is set to 10%. The experiments were conducted using the AdamW optimizer with various settings. While hyperparameter tuning can be challenging and resource-intensive in this context, initial results indicate a noticeable advantage of using *SparsyFed* over a naive `Top-K` approach in federated learning. The following results compare four different combinations of learning rate and the number of local epochs. In the first case, only one local epoch is performed per round, while in the other cases, three local epochs are used, with the first two serving as a warm-up phase. Additionally, two different learning rate strategies are evaluated: one with a fixed learning rate of 0.01 and another with a server-side decay schedule that reduces the learning rate from 0.01 to 0.001. The global sparsity is set to 50%, and the *SparsyFed* and `Top-K` implementations are compared. Overall, *SparsyFed* leads to a consistent improvement in performance, with more evident benefits in some instances. In all settings, *SparsyFed* can remarkably maintain a sparsity level close to the target throughout training. In contrast, the `Top-K` implementation tends to drift away from the desired sparsity as training progresses. Our method thus reduces communication overhead during training while effectively achieving the target sparsity level in the final model.

[1] The pre-trained model was sourced on the Hugginface Hub from this repository

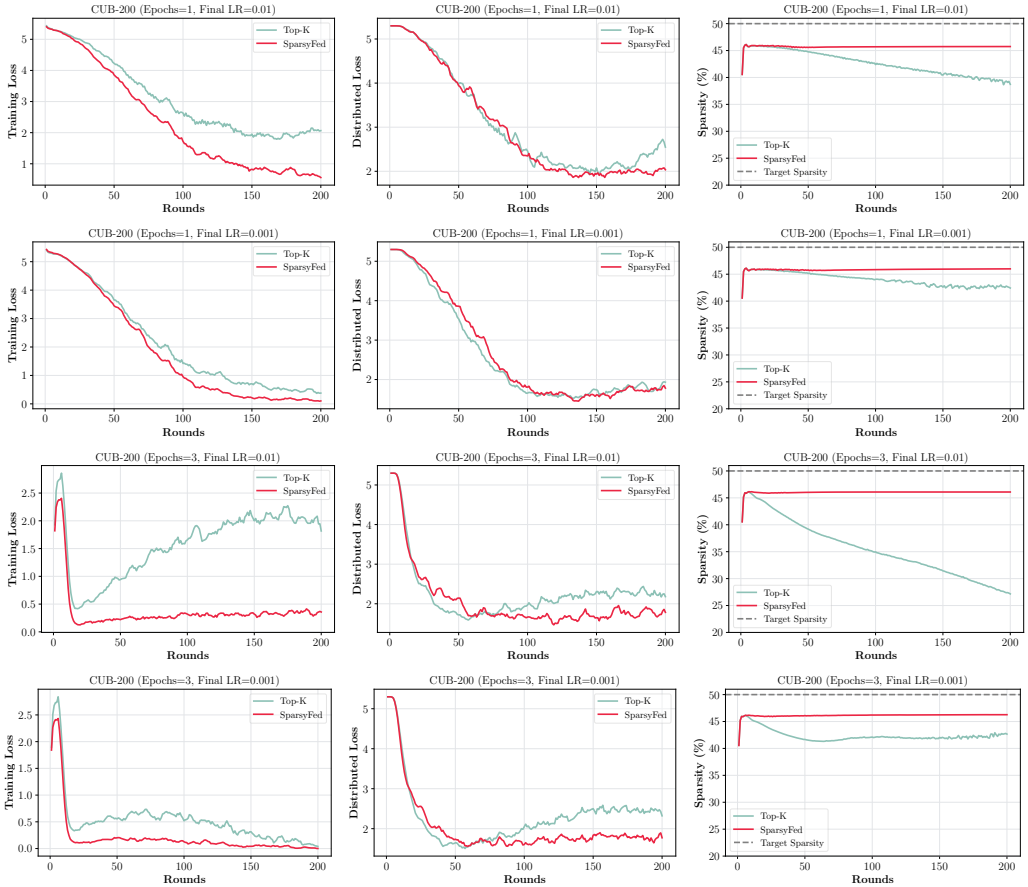

Figure 13: Experiment using a Vision Transformer (ViT) on the CUB-200-2011 dataset with LDA $\alpha = 1000.0$, 100 clients, and 10% partial participation rate for training and evaluation. For each experimental setting (rows), we show the training loss, distributed evaluation loss, and sparsity level on the left, central, and right columns, respectively. The training loss is measured during training. The distributed evaluation loss is collected during the client data evaluation. The sparsity level refers to the global model after aggregation on the server. Different settings are shown, from top to bottom: (first row) one local epoch with a fixed learning rate, (second row) one local epoch with server-side learning rate decay from 0.01 to 0.001, (third row) three local epochs with a fixed learning rate, and (fourth row) three local epochs with server-side learning rate decay from 0.01 to 0.001. The number of local epochs and the final learning rate are reported at the top of each plot.

E.9 FULL TABLES ACCURACY RESULTS

Dataset	Sparsity	Resnet18	ZeroFL	FLASH	*SparsyFed*
CIFAR10 ($\alpha = 10^3$)	dense	85.14 ± 1.18	-	-	-
	0.9	82.16 ± 0.80	78.67 ± 1.52	82.57 ± 2.05	**84.31 ± 0.86**
	0.95	77.92 ± 0.97	76.16 ± 1.28	82.22 ± 0.14	**84.25 ± 1.38**
	0.99	68.11 ± 3.50	72.40 ± 1.08	**77.48 ± 2.86**	77.16 ± 0.85
	0.995	54.49 ± 8.70	60.31 ± 4.11	71.85 ± 1.63	**72.71 ± 0.65**
	0.999	17.87 ± 6.22	21.91 ± 1.11	46.43 ± 16.73	**55.24 ± 2.09**
CIFAR100 ($\alpha = 10^3$)	dense	53.79 ± 1.63	-	-	-
	0.9	46.47 ± 1.74	44.11 ± 0.93	53.06 ± 1.10	**54.44 ± 1.33**
	0.95	13.45 ± 21.45	33.67 ± 3.34	49.25 ± 1.63	**54.33 ± 1.48**
	0.99	1.54 ± 1.22	10.59 ± 2.48	44.40 ± 1.77	**47.62 ± 1.67**
	0.995	0.97 ± 0.64	4.04 ± 3.43	36.82 ± 1.72	**42.05 ± 1.21**
	0.999	0.97 ± 0.64	0.87 ± 0.21	9.61 ± 3.61	**13.85 ± 1.01**
Speech Commands ($\alpha = 10^3$)	dense	92.98 ± 0.67	-	-	-
	0.9	85.53 ± 0.84	89.12 ± 1.11	89.89 ± 0.74	**91.34 ± 0.52**
	0.95	80.19 ± 1.92	85.63 ± 1.12	88.16 ± 1.37	**89.79 ± 0.21**
	0.99	67.67 ± 2.32	60.79 ± 2.44	76.41 ± 1.31	**77.00 ± 0.73**
	0.995	36.69 ± 1.45	41.16 ± 3.10	67.54 ± 1.15	**70.03 ± 1.14**
	0.999	63.63 ± 3.24	16.77 ± 6.23	31.83 ± 2.18	**49.27 ± 0.50**

Table 6: Aggregated results for CIFAR10, CIFAR100, and Google Speech Command datasets.

Dataset	Sparsity	Resnet18	ZeroFL	FLASH	*SparsyFed*
CIFAR10 ($\alpha = 1.0$)	dense	83.70 ± 1.70	-	-	-
	0.9	80.56 ± 1.90	76.16 ± 1.30	81.15 ± 1.03	**82.13 ± 1.53**
	0.95	74.71 ± 3.29	75.53 ± 2.27	79.36 ± 1.03	**82.60 ± 1.58**
	0.99	66.27 ± 5.08	70.71 ± 0.15	73.45 ± 1.37	**77.71 ± 1.69**
	0.995	63.82 ± 2.41	56.02 ± 3.95	69.15 ± 1.60	**70.01 ± 0.43**
	0.999	31.79 ± 19.10	17.66 ± 8.34	36.07 ± 7.49	**51.39 ± 3.19**
CIFAR100 ($\alpha = 1.0$)	dense	52.29 ± 1.14	-	-	-
	0.9	46.57 ± 1.71	40.70 ± 4.72	51.99 ± 0.21	**53.08 ± 0.90**
	0.95	28.07 ± 23.27	38.82 ± 1.75	47.19 ± 1.88	**52.81 ± 1.72**
	0.99	19.65 ± 16.30	18.97 ± 2.08	42.76 ± 4.08	**46.64 ± 1.59**
	0.995	9.51 ± 14.81	6.01 ± 4.74	36.43 ± 4.97	**42.21 ± 1.03**
	0.999	3.81 ± 2.18	1.96 ± 0.66	5.80 ± 2.86	**15.96 ± 0.64**
Speech Commands ($\alpha = 1.0$)	dense	91.49 ± 0.94	-	-	-
	0.9	84.28 ± 0.88	87.79 ± 1.40	88.68 ± 1.72	**92.32 ± 1.59**
	0.95	78.58 ± 0.44	84.29 ± 1.50	84.89 ± 0.49	**89.14 ± 1.15**
	0.99	65.01 ± 0.84	57.79 ± 0.82	69.22 ± 1.59	**75.82 ± 3.72**
	0.995	56.73 ± 1.00	37.16 ± 2.71	58.23 ± 1.84	**68.02 ± 3.14**
	0.999	21.56 ± 12.79	10.10 ± 4.01	17.70 ± 2.58	**47.43 ± 1.66**

Table 7: Aggregated results for CIFAR10, CIFAR100, and Google Speech Command datasets.

Dataset	Sparsity	Resnet18	ZeroFL	FLASH	*SparsyFed*
CIFAR 10 ($\alpha = 0.1$)	dense	73.81 ± 4.84	-	-	-
	0.9	69.79 ± 3.78	67.40 ± 4.11	71.87 ± 2.63	**75.00** $\pm$ **2.78**
	0.95	60.00 ± 4.66	61.55 ± 4.18	72.08 ± 2.09	**75.95** $\pm$ **3.39**
	0.99	43.96 ± 11.99	51.71 ± 3.54	56.91 ± 3.55	**63.69** $\pm$ **3.90**
	0.995	19.02 ± 10.77	41.33 ± 3.64	52.15 ± 3.87	**56.79** $\pm$ **3.97**
	0.999	11.5 ± 4.494	18.76 ± 4.28	29.31 ± 6.75	**43.68** $\pm$ **7.61**
CIFAR100 ($\alpha = 0.1$)	dense	48.34 ± 2.71	-	-	-
	0.9	41.96 ± 2.16	31.92 ± 7.65	45.59 ± 0.75	**48.37** $\pm$ **1.73**
	0.95	11.48 ± 17.51	34.21 ± 7.65	44.31 ± 2.14	**48.27** $\pm$ **2.70**
	0.99	0.14 ± 0.72	13.07 ± 2.26	34.75 ± 3.38	**41.03** $\pm$ **2.14**
	0.995	0.14 ± 0.72	7.04 ± 5.25	26.44 ± 17.35	**35.72** $\pm$ **2.01**
	0.999	0.14 ± 0.72	1.66 ± 0.97	3.56 ± 2.07	**13.84** $\pm$ **3.69**
Speech Commands ($\alpha = 0.1$)	dense	80.15 ± 2.69	-	-	-
	0.9	65.44 ± 0.97	70.35 ± 2.65	77.15 ± 0.77	**79.67** $\pm$ **2.78**
	0.95	57.39 ± 1.04	65.90 ± 1.88	71.28 ± 1.75	**75.46** $\pm$ **2.24**
	0.99	50.42 ± 6.26	41.42 ± 1.60	53.55 ± 2.00	**56.69** $\pm$ **4.56**
	0.995	34.20 ± 1.43	22.61 ± 3.45	43.16 ± 3.47	**48.30** $\pm$ **5.39**
	0.999	19.25 ± 6.01	8.85 ± 3.76	17.14 ± 2.97	**29.24** $\pm$ **2.34**

Table 8: Aggregated results for CIFAR10, CIFAR100, and Google Speech Command datasets.

F Additional Experimental Configuration Details

Learning rate scheduler. The learning rate follows a scheduled pattern defined by the function:

$$\eta_t = \eta_{\text{start}} \exp\left(\frac{t}{T} \ln\left(\frac{\eta_{\text{end}}}{\eta_{\text{start}}}\right)\right) \tag{1}$$

Reproducibility. Seeds were used for client sampling, while others were fixed for reproducibility purposes. All simulations were conducted using three different client sampling seeds: 5378, 9421, and 2035.

Experimental Setting. Each round consisted of one local epoch with a local batch size of 16 samples. The initial learning rate was set to 0.5, gradually decreasing to a final value of 0.01 following Eq. (1). For *SparsyFed*, the exponent for re-parameterization was set to $\beta = 1.25$ for the CIFAR-10/100 experiments and $\beta = 1.15$ for the Speech Commands experiment. The CIFAR experiments were run for 700 rounds, while the Speech Commands experiment was run for 500 rounds.

G Comparative analysis on algorithms and baselines

Here is a brief description of the implementation used during the experiments for `Top-K`, ZeroFL Qiu et al. (2021), FLASH Babakniya et al. (2023), and *SparsyFed*:

G.1 Top-K

1. **Pruning**: The model is pruned per round at clients after executing the (local) training on their own data, i.e., just before sending the updated model to the server. The pruning method used is global unstructured `Top-K`, which prunes all the model parameters except for the k largest values.
2. **Aggregation strategy**: FedAvg. Alternatively, other aggregation strategies acting on the pseudo-gradients can also be applied straightforwardly.

G.2 ZEROFL

1. **Pruning**: Pruning is performed during the local training at clients and before sending the model update back to the central server. During training, the SWAT unstructured (per-layer, in contrast to standard global unstructured `Top-K`) approach is used to prune the weights before the forward pass and the activations during the forward pass. This alone doesn't ensure obtaining a model with the targeted degree of sparsity because SWAT often results in weight regrowth per optimizer step. To achieve the target sparsity, the model is pruned again before sending it back to the server, using the same approach as `Top-K`, i.e., global unstructured. In the original work, three levels of masks ($[0.0, 0.1, 0.2]$) have been proposed to increase the density of the model before the uplink communication. For our experiments, we used the smaller one (0.0) since it is more in line with
2. **Aggregation strategy**: A slight variation of FedAvg is used, where the averaging is executed among the non-zero weights to avoid excessive dilution of information in the presence of highly sparse models. Adapting the aggregation function to act only on non-zero weights supports alternative aggregation strategies acting on the pseudo-gradients.

G.3 FLASH - SPDST (SENSITIVITY-DRIVEN PRE-DEFINED SPARSE TRAINING)

(THE ONE USED IN THE EXPERIMENTS)

1. **Pruning**: Pruning is performed at the end of the first round of training (similar to `Top-K`), which, in the original paper, is referred to as a warm-up phase. Thus, the first round of training is executed using the dense model, producing the initial mask. The binary mask obtained during the warm-up phase is fixed for the subsequent federated rounds, and only the non-zero weights are trained. This means there is no need for further pruning of the weights in subsequent rounds since no regrowth is allowed (in this version of Flash).
2. **Aggregation strategy**: The aggregation is performed only among the non-zero weights, similar to ZeroFL. During the first aggregation, at the end of the initial training round, the model is further pruned on a per-layer basis to counter the regained density caused by the mismatch in local masks. The pruning uses the average sparsity level all clients achieve for each layer. For example, if the average sparsity of layer l is d_l, adjusted by a factor r, then layer l of the pruned global model will have sparsity d_l. The factor r helps to maintain the target global sparsity by ensuring that the sum of individual layer sparsity levels meets the overall goal. The resulting binary mask becomes the final one, preserved for all subsequent training rounds. This process, defined in the original paper as sensitivity analysis, is applied only in the first round, as fixed mask training prevents further mask modification in later rounds.

G.4 FLASH - JMWST (JOINT MASK WEIGHT SPARSE TRAINING)

1. **Pruning**: A normal training procedure is applied (NO FIXED MASKS). Pruning is performed at the end of each local training session.
2. **Aggregation strategy**: The server aggregates and then prunes the model, applying the same sensitivity analysis introduced in SPDST to address the regained density. This is done every r round, as the training method allows for the regrowth of the clients. The original paper proposed two values for r: $r = 1$ and $r = 5$.

G.5 *SparsyFed*

1. **Pruning**: Powerpropagation is applied to re-parameterize the weights during the forward pass executed at clients. Pruning during training is applied only to the activations during the backward pass. At the end of the local training, the model is pruned and returned to the server. The first round of training executes using a full-size model.
2. **Aggregation strategy**: FedAvg. Alternatively, other aggregation strategies acting on the pseudo-gradients can also be applied straightforwardly.

H FLOPs REDUCTIONS FOR UNSTRUCTURED SPARSE TRAINING.

Unstructured sparsity can theoretically reduce FLOPs linearly in the percentage of zero weights (Dettmers & Zettlemoyer, 2019; Singh et al., 2024) if an optimal sparse algorithm exists for a given operation. However, such gains are fully realized only when the hardware can efficiently skip zero-valued operations. In contrast, structured sparsity methods—such as block-sparsity pruning (Gray et al., 2017)—yield more predictable speed-ups on standard GPUs due to optimized kernels despite typically achieving lower overall sparsity than unstructured sparsity. Emerging accelerators and libraries increasingly support unstructured sparsity (Jeong et al., 2024), bridging the gap between theoretical FLOP reductions and actual runtime improvements.