Adaptive Channel Sparsity for Federated Learning under System Heterogeneity

Dongping Liao*
State Key Lab of IoTSC, CIS Dept,
University of Macau, Macau SAR, China.
yb97428@um.edu.mo

Xitong Gao*
Shenzhen Institutes of Advanced Technology,
Chinese Academy of Sciences, Shenzhen, China.
xt.gao@siat.ac.cn

Yiren Zhao
Imperial College London,
London, UK.
a.zhao@imperial.ac.uk

Chengzhong Xu†
State Key Lab of IoTSC, CIS Dept,
University of Macau, Macau SAR, China.
czxu@um.edu.mo

Abstract

Owing to the non-i.i.d. nature of client data, channel neurons in federated-learned models may specialize to distinct features for different clients. Yet, existing channel-sparse federated learning (FL) algorithms prescribe fixed sparsity strategies for client models, and may thus prevent clients from training channel neurons collaboratively. To minimize the impact of sparsity on FL convergence, we propose Flado to improve the alignment of client model update trajectories by tailoring the sparsities of individual neurons in each client. Empirical results show that while other sparse methods are surprisingly impactful to convergence, Flado can not only attain the highest task accuracies with unlimited budget across a range of datasets, but also significantly reduce the amount of floating-point operations (FLOPs) required for training more than by $10\times$ *under the same communications budget, and push the Pareto frontier of communication/computation trade-off notably further than competing FL algorithms.*

1. Introduction

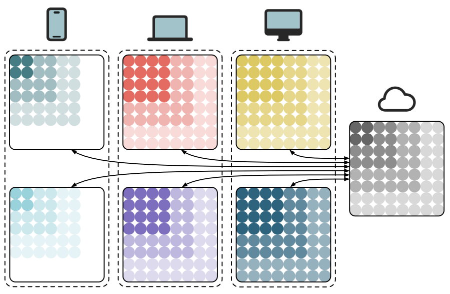
In the light of the importance of personal data and recent privacy regulations, *e.g.* the General Data Protection Regulation (GDPR) of the European Union [28,31,34], there is now a great amount of risk, responsibility [7, 10] and technical challenges for securing private data centrally [30]; it is often impractical to upload, store and use data on central servers. To this end, *federated learning* (FL) [21,24] enables multiple edge devices to learn a global shared model collaboratively in a communication-efficient way without collecting their local training data. *Federated averaging* (FedAvg) [24] and subsequent FL algorithms [16, 22] can notably reduce the burden of data transmission.

These FL algorithms, however, neglected that the clients exhibit a high degree of *system heterogeneity*. That is, the clients may occupy a wide spectrum of different hardware training capabilities [12]. Yet, dropping stragglers (*i.e.*, the slowest clients) inherently increases statistical heterogeneity, which causes a negative impact on convergence [22]. Motivated by this, recent FL methods, *e.g.*, Federated Dropout [5], and FjORD [13], thus prescribe fixed channel sparsity strategies for each client, depending on their corresponding computational capabilities (Figure 1a).

Yet, owing to the non-i.i.d. nature of client data, a fixed sparsity strategy is suboptimal, as neurons may specialize to distinct features [3, 37] for different clients (Figure 2). Intuitively, clients may waste computational effort on neurons that lead to conflicting update trajectories "canceling out" each other. Since neuron training is heavily dependent on the clients' data, it presents us an opportunity: *can we adapt neuron sparsities such that clients can focus their computational efforts to collaborate more effectively?* Inspired by the observation above, an optimal model sparsity strategy should thus adapt to clients' model training, and make clients collaborate on similar model update trajectories by making such neurons denser, while sparsifying neurons that conflict in update directions.

Adaptive channel sparsity for FL clients is, however, a nontrivial endeavor. First, naïvely pruning channel neurons *i.e.*, setting to 0, would cause them to make no contribution in training after pruning. It is thus difficult to decide if and when a pruned channel should be recovered. Second, as neurons tend to extract distinct features from data, data heterogeneous clients thus specialize to training different neurons. Third, prescribing sparsities to channels is suboptimal, it may be desirable to allow certain clients to collaborate

*These authors contributed equally to this work.
†Corresponding author.


(a) FjORD prescribes fixed sparsity.

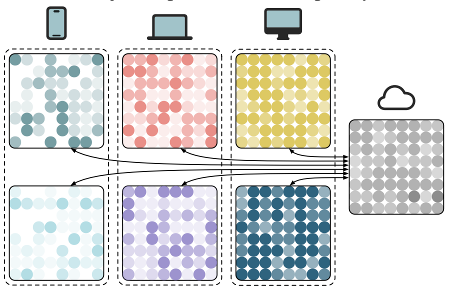

(b) Adaptive sparsity with Flado.

Figure 1. Comparing existing sparse FL algorithms (*e.g.*, FjORD [13]) with Flado. **(a) FjORD trains models with fixed channel sparsities for clients with different capabilities. (b) Flado adapts channel sparsities with underlying training trajectories and capabilities for each training round.** As an example we use 6 clients with different computational capabilities and a convolutional layer with 8 input and 8 output channels. Each colored bullet "●" denotes a filter computation, and lighter color ones are more likely to skip, and "⟷" arrows indicate communications.

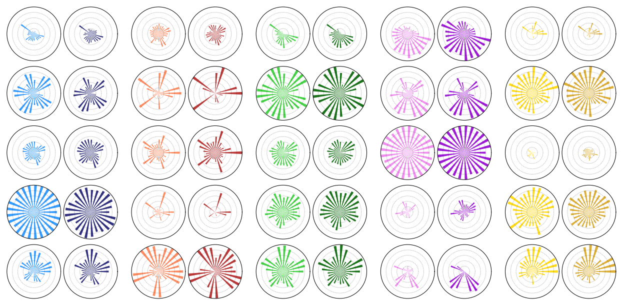


Figure 2. **Clients sharing similar data distribution also share similar trained parameters. Simply aggregating the conflicting update trajectories from multiple clients can result in wasteful computations as the trajectories are mostly orthogonal to each other.** As an example, we trained the same LeNet-5 model using 10 clients with each pair of clients receiving only same-class images from Fashion-MNIST to simulate concept disparity. Each circle denotes a channel neuron of the 1th layer, and it contains the update magnitudes for all its parameters (arranged radially) after 1 round of training. The 5 rows represent the first 5 channel neurons, and each column is a different client (grouped in pairs).

on particular neurons, but the neurons may not exist in the sparse models. In other words, the optimal sparsification strategy must adapt different sparsities for each neuron in each client, depending on the training trajectories.

To this end, this paper introduces Flado, a method that optimizes channel activation probabilities to sparsify client models with trajectory alignment towards the global trajectory. The advantage of Flado is two-fold. First, coarse-grained channel dropouts can be easily implemented and leveraged by existing models and hardware devices to accelerate client training. Second, a light-weight trajectory-alignment algorithm optimizes the sparsity of each channel in each client with very low overhead for the clients, and it can reap immense computational benefits. We summarize the contributions of this paper as follows:

- In contrast to existing fixed sparsity strategies for FL, Flado propose to further optimize channel activation probabilities to accelerate sparse training for each channel in each client.
- As evinced by our experiments, it can drastically reduce the amount of floating-point operations (FLOPs) required for training by more than 10× under the same communications budget, and enjoys a much improved communication/computation Pareto frontiers than competing FL approaches when training under data and system heterogeneity.
- Flado widens its lead in convergence rate when high degrees of heterogeneity are present in both data distributions and system capabilities. Furthermore, it scales well to larger models and fractional client participation.

2. Related Work

Federated learning. Distributed machine learning has a long history of progress and success [20, 27], yet it mainly focuses on training with i.i.d. data. The Federated Learning (FL) paradigm and the Federated Averaging algorithm (FedAvg) initially introduced by [24] allow clients to train collaboratively without sharing the private data in a communication-efficient manner. As a result of privacy limitations and personal preferences, the FL clients naturally collect data that are non-i.i.d., *i.e.*, the data on training clients may display varying levels of differences and imbalance in their distribution. This presents a notable challenge for FL algorithms to converge efficiently. We refer readers to [21, 33] for a more comprehensive literature review. While being effective at reducing communication, most of the state-of-the-art FL methods neglected the computational costs associated with the training process.

Computation *vs.* communication during federated learning. There are a few precursory methods that focus on the joint optimization of computation and communication costs during training. Caldas *et al.* [5] introduced federated dropout, which prunes parameters following a uniform random distribution, whereas PruneFL [15] proposes a greedy gradient-magnitude-based unstructured pruning. In each FL round, both methods produce a shared pruned model with fine-grained sparsity for all clients. Unstructured sparsity is, however, difficult to accelerate on hardware, and a shared global model cannot exploit the data distribution of individual clients. HeteroFL [9] propose nested models to tackle the problem of system heterogeneity in FL, where sub-models are derived from a large base model by masking the last channel neurons of each layer, and let clients with lower computational budgets to train on smaller sub-models. On top of this, FjORD [13] introduces ordered dropout by allowing clients to randomly choose sub-models to train during each local training step. As shown by our experiments however, they overlooked the side effect of introducing a fixed sparsity scheme that is oblivious to client training trajectories induced by non-i.i.d. data distributions. It is noteworthy that Flado differs from them as it infers individual channel selection probabilities for all clients, in order to minimize its impact on convergence.

Model Pruning and Sparse Training. Model pruning methods [2, 11, 19, 23] introduce sparsity by removing the neurons or parameters of deep learning models to accelerate inference. Sparse training [36], which draws inspiration from the progress made in model pruning, has been gaining significant traction in the deep learning community recently, owing to the growing cost of training [4]. To tackle this challenge, Yuan *et al*. [36] propose to progressively grow a neural network with continuous relaxation of network structure to save computational costs. Zhou *et al*. [38] introduce an efficient sparse training method that identifies important neurons with a variance reduced policy gradient estimator to achieve practical training acceleration. Mohtashami *et al*. [25] propose to iteratively optimize an over-parameterized neural network and its compute-efficient subnetwork. Flado differs from these approaches as it tackles the challenge of optimal gradient alignment introduced by collaborative sparse training of FL clients with non-i.i.d. data, which necessitate a simultaneous minimization of compute and communication costs. In contrast, all above related works only consider centralized sparse training.

3. The Flado Method

Flado augments FL by allowing clients to adopt sparsity. Client training sparsifies models by sandwiching convolutional layers with channel-wise channel selection layers. Before each round of client training, the server compresses the parameter update trajectory with a random embedding, such that freshly-joined clients receive global model parameters, and an embedding of the global trajectory. Each client then aligns the training direction along the global trajectory for each channel neuron in the client's model (Section 3.3). This process also takes into consideration the FLOPs budget constraints to encourage client models to sparsify (Section 3.2). Finally, following conventional FedAvg, the server then broadcasts the weighted-average model parameters to all training clients in the new round, and also sends the embedded trajectory to the respective clients to commence the next round of training.

3.1. Preliminaries and Definitions

We assume the training loss function of a client $c \in \mathbb{C}$ to be $\ell_c(\boldsymbol{\theta}_c)$, where $\boldsymbol{\theta}_c$ comprises the parameters of all layers in the model of client c. We illustrate a FL training round for a set of sampled clients $c \in \mathbf{C} \subseteq \mathbb{C}$. In each FL training round t, clients train on the loss function, with initial parameters $\boldsymbol{\theta}^{(t)}$ received from the server for this round:

$$\boldsymbol{\theta}_c^{(t+1)} = \mathsf{SGD}_c\big(\ell_c, \boldsymbol{\theta}^{(t)}, \eta, E\big). \tag{1}$$

Here, SGD_c indicates that client c carries out stochastic gradient descent (SGD) on $\ell_c(\boldsymbol{\theta}_c^{(t)})$ locally, and it uses a learning rate η for E epochs. The FedAvg server then aggregates client model parameters after the t^{th} training round, by taking the weighted average of them:

$$\boldsymbol{\theta}^{(t+1)} = \textstyle\sum_{c\in\mathbf{C}} \lambda_c \boldsymbol{\theta}_c^{(t+1)}, \tag{2}$$

where λ_c is the weight of client c and is proportional to the size of its training set $|\mathbb{D}_c|$ with $\sum_{c\in\mathbf{C}} \lambda_c = 1$. Finally, the $(t+1)^{\text{th}}$ training round starts by repeating the above procedure.

3.2. Sparsity Enforcement

To stochastically induce sparsity in model, for each convolutional layer in a client c, we introduce a vector of channel activation probabilities $\mathbf{p}_c$. Each channel is thus randomly activated by sampling an independent Bernoulli distribution of probability $\mathbf{p}_c^n$ during model training. By doing so, the sparse model drops not only the output channels of each convolutional layer, but also the input channels from their preceding layers with sparse activations.

To enforce sparsity for each client c, we constrain the feasible set of $\mathbf{p}_c$ to be within $g_c(r_c, \mathbf{p}_c) \geq 0$, where $r_c \in (0, 1]$ dials the FLOPs budget permitted by the client c, with $r_c = 1$ reducing to the full model. The function $g_c(r_c, \mathbf{p}_c)$ thus evaluates the following FLOPs budget constraint:

$$g_c(r_c, \mathbf{p}_c) = r_c - \mathrm{flops}(\ell_c, \mathbf{p}_c) \big/ \mathrm{flops}(\ell_c, \mathbf{1}), \tag{3}$$

and the terms $\mathrm{flops}(\hat{\ell}_c, \mathbf{p}_c)$ and $\mathrm{flops}(\ell_c, \mathbf{1})$ respectively denote the FLOPs of a sparse model (combined with its loss evaluation) with channel activation probabilities $\mathbf{p}_c$, and the full FLOPs of the corresponding dense model. In Appendix B, we explain how one can compute the FLOPs of a model.

3.3. Sparsity-driven Trajectory Alignment

As client training trajectories may not be well aligned with the global trajectory, we propose to adaptively adjust the clients' channel activation probabilities before client training. Specifically, it aligns the client's gradient direction with the global model trajectory, by maximizing the following objective for a new set of channel activation probabilities $\mathbf{p}_c$ at the start of t^{th} training round:

$$\begin{aligned}&\max_{\mathbf{p}_c} \mathbb{E}_{\mathbf{b}_c \sim \mathcal{B}(\mathbf{p}_c)} \\ &\quad \mathsf{cossim}\big(J\big(\Delta\boldsymbol{\theta}^{(t)}\big), J\big(\nabla_{\boldsymbol{\theta}^{(t)}} \ell_c\big(\mathbf{b}_c \circ \boldsymbol{\theta}^{(t)}\big)\big)\big), \\ &\quad \textit{s.t. } g_c(r_c, \mathbf{p}_c) \geq 0.\end{aligned} \tag{4}$$

Here, $\Delta\boldsymbol{\theta}^{(t)} \triangleq \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t-1)}$ is the global model update trajectory from round t to $t+1$, and $\nabla_{\boldsymbol{\theta}^{(t)}} \ell_c\big(\mathbf{b}_c \circ \boldsymbol{\theta}^{(t)}\big)$ is the stochastic gradient of the local client model under sampled sparsity $\mathbf{b}_c \sim \mathcal{B}(\mathbf{p}_c)$. The function $\mathsf{cossim}(\mathbf{a}, \mathbf{b})$ evaluates the cosine similarity between $\mathbf{a}$ and $\mathbf{b}$, and the function J performs a differentiable fast Johnson-Lindenstrauss transform (FJLT) [1] to compress the representation of its input with a random embedding by dimension reduction.

We define the FJLT function $J : \mathbb{R}^d \to \mathbb{R}^k$ with $k \ll d$ as the following random embedding:

$$J(\mathbf{z}) = \mathbf{PHDz}, \tag{5}$$

where $\mathbf{P}$ is a $k \times d$ sparse matrix and its entries are 0 with probability $1-q$ and remaining are drawn randomly with $P_{ij} \sim \mathcal{N}(0, q^{-1})$. $\mathbf{H}$ is a $d \times d$ Hadamard matrix, and $\mathbf{D}$ is a $d \times d$ diagonal matrix with entries drawn uniformly from $\{-1, 1\}$. With high probability, the FJLT of any vector $\mathbf{z} \in \mathbb{R}^d$ can be evaluated in $O(d \log d + qd\epsilon^{-2})$ time, where ϵ denotes the approximation gap. The FJLT can effectively preserve the geometry of the compressed input, and thus the cosine similarity $\mathsf{cossim}(J(\mathbf{a}), J(\mathbf{b}))$ closely matches $\mathsf{cossim}(\mathbf{a}, \mathbf{b})$ with a very high probability. Appendix C provides theoretical and empirical evidences for the geometry preserving properties of FJLT, and its performance.

The rationale for employing cosine similarity over the ℓ_2 distance is that $\Delta\boldsymbol{\theta}^{(t)}$ accumulates the average of parameters updates after multiple client training steps, whereas $\ell_c(\mathbf{b} \circ \boldsymbol{\theta}_c)$ is only one-step of the client's local gradient update. Aligning them by cosine similarity thus encourages local gradients to follow the same direction of global convergence trajectory.

We also note that the optimization of (4) can be trained with stochastic gradient descent on the channel activation probabilities $\mathbf{p}_c$, and this process incurs minimal overhead. First, if the client did not participate in the previous round of training (this happens under fractional device participation), it requires training clients to receive a very small $J(\Delta\boldsymbol{\theta}^{(t)})$ along with the global model parameters. Otherwise, it can evaluate $J(\Delta\boldsymbol{\theta}^{(t)})$ efficiently. Second, it only takes a few steps of local training to converge well, where we set the number of steps to be 1% of the client's local training steps.

3.4. The Overall Algorithm

Algorithm 1 provides an algorithmic overview of Flado. It uses the client model loss functions $\hat{\ell}_c$, client weights λ_c, and the FLOPS constraints functions $g_c : [0,1]^{\mathbb{C}\times\mathbb{N}} \to \mathbb{R}$ for each client $c \in \mathbb{C}$. It further takes a local learning rate η, the number of local epochs per round E the number of FL rounds T, and a client sub-sampling ratio $\mathcal{R}$. It returns the optimized model parameters $\boldsymbol{\theta}^{(T+1)}$ upon completion.

The algorithm starts by initializing model parameters $\boldsymbol{\theta}^{(0)}$ to be shared across all clients, and assigns a uniform keep probability $\mathbf{p}$ to all channels, which satisfies the FLOPs constraint $g(\mathbf{p}) = 0$ (lines 3).

It then optimizes the trajectory similarity of the training client locally, as described on line 12. Line 16 then performs conventional stochastic gradient descent on the model, but with updated sparsity. Finally, the server performs weighted model averaging and randomly embeds the trajectory with FJLT for the next round of clients.

Algorithm 1 The Flado algorithm.

```
1:  function FLADO($\{(g_c, \hat{\ell}_c, \lambda_c) : c \in \mathbb{C}\}, \eta, E, T, \mathcal{R}$)
2:      // Initialization.
3:      initialize($\boldsymbol{\theta}^{(0)}, \mathbf{p}$)
4:      // For each round t...
5:      for $t \leftarrow 1, 2, \dots T$ do
6:          // Sample a fast JL transform with seed t.
7:          $J \leftarrow \text{fjlt\_sample}(t)$
8:          // Sample a subset of clients.
9:          $\mathbf{C} \leftarrow \text{subsample}(\mathbb{C}, \mathcal{R})$
10:         // For each sampled client in parallel...
11:         for $c \in \mathbf{C}$ in parallel do
12:             if $t > 1$ then
13:                 // After the 1st round of training, optimize
14:                 // client channel selection probabilities.
15:                 $\mathbf{p}_c \leftarrow \text{argmax}_{\mathbf{q}_c \textit{ s.t. } g_c(\mathbf{q}_c) \geq 0}$
16:                     $\mathsf{cossim}\Big(\overline{\Delta\boldsymbol{\theta}^{(t)}}, J\big(\nabla_{\boldsymbol{\theta}^{(t)}} \ell_c\big(\mathbf{b}_c \circ \boldsymbol{\theta}^{(t)}\big)\big)\Big)$
17:             end if
18:             // Client training with sparsity.
19:             $\boldsymbol{\theta}_c^{(t+1)} \leftarrow \mathsf{SGD}_c\big(\hat{\ell}_c, \boldsymbol{\theta}^{(t)}, \mathbf{p}_c, \eta, E\big)$
20:         end for
21:         // Server aggregation with FedAvg.
22:         $\boldsymbol{\theta}^{(t+1)} \leftarrow \sum_{c \in \mathbf{C}} \lambda_c \boldsymbol{\theta}_c^{(t+1)}$
23:         // Global trajectory embedding.
24:         $\overline{\Delta\boldsymbol{\theta}}^{(t+1)} \leftarrow J(\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)})$
25:     end for
26:     return $\boldsymbol{\theta}^{(T+1)}$
27: end function
```

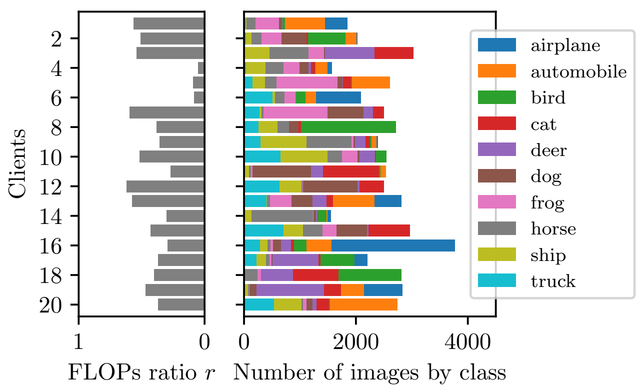


Figure 3. An example data and system heterogeneity for 20 clients on CIFAR-10. It shows for all clients the uneven class distributions of images under $\alpha = 0.5$ and their disparate FLOPs budgets.

4. Evaluation of Sparse FL Algorithms

4.1. Experiment Setup

In this section, we present a comprehensive evaluation of both the communication and computation costs of Flado and compare it against other sparse FL algorithm baselines. Figure 3 exemplifies the data and system heterogeneous properties of the default experimental setup. We provide an explanation of the training setups below.

Dataset configurations. We conduct experiments on three popular vision datasets CIFAR-10 [17], Fashion-MNIST [35] and SVHN [26]. Unless specified otherwise, the experiment adopts a common baseline, which uses 20 clients for CIFAR-10, and 100 clients for both Fashion-MNIST and SVHN, defined as follows. Similar to [14, 32], we simulate data heterogeneity using a Dirichlet distribution $\mathcal{D}ir_{|\mathbb{C}|}(\alpha)$ to split the training dataset class-wise among clients, where α controls data heterogeneity, which we fix at 0.5 by default. Here, $\alpha \to \infty$ gives uniform class distribution across clients, and $\to 0$ denotes extreme heterogeneity. Although we split the datasets for non-i.i.d. characteristics, we evaluate global model accuracies with unified test datasets.

System configurations. By default, we use a VGG-9, a 9-layer VGG-style architecture [29] for CIFAR-10, whereas Fashion-MNIST and SVHN use LeNet-5 [18] of different widths. Refer to Appendix A in the appendix for more on the model architectures. To simulate system heterogeneity across training clients, we draw FLOPs budgets $r_c \sim \mathcal{U}(0.04, 0.64)$ for all clients $c \in \mathbb{C}$, *i.e.*, the densest clients allow 64% FLOPs of the base model, whereas the sparsiest ones use 4% — effectively enabling clients with $16\times$ difference in computational capabilities to train collaboratively. Note that the data and computational budgets are independent sampled to make the task even more challenging for sparse FL methods.

Training configurations. We use the same hyperparameters (batch size $B = 32$, learning rate $\eta = 0.1$, and local training epochs $E = 1$) and train all models for up to $T = 2000$ rounds by default. We note that existing sparse FL algorithms provides different fixed channel keep probabilities instead of respecting the actual FLOPs budget of each client. To compare more fairly, we thus solve the quadratic Eq. (3), for an average p given a client's FLOPs budget r_c, and reassign the probabilities for all channels in each layer according to their respective methods, while maintaining the mean within the same layer to be p to keep a constant FLOPs consumption by the sparse model. The comparing sparse FL algorithms are as follows, and please refer to the supplementary material for detailed explanations:

- *UniProb*. This is a simple sparse baseline, where all neurons share an activation probability of p, and p is chosen w.r.t. the client's respective capability.
- *HeteroFL* [9]. In a training round, each client receives and trains a p-reduced model, which removes the last $1-p$ of all output channels in each layer, where p is chosen w.r.t. its capability. For the next round, the server then receives and averages each channel parameters across clients that trains it.
- *FjORD* [13]. Each device receives a common base model. In each local training step, it samples the model density from a set of p values uniformly, and forms a p-reduced model for training.
- *eFD* (an extension of federated dropout [5] by [13]). Before each training round, federated dropout [5] simply pre-samples a sub-model with a shared probability p of enabling channel neurons and zeroing the rest for each client. As it shares a global dropout probability for all devices, it does not natively support heterogeneous client capabilities. [13] thus proposes eFD, which extends it by allowing each client to adapt its channel selection probability p to its computational capability.
- *Flado*. This is the method proposed in this paper, which uses Algorithm 1 to optimize the channel activation probabilities under the FLOPs constraints of each client during each round of server aggregation.

4.2. Main Evaluation

Flado attains higher converged accuracies. Table 1 shows the highest accuracies achieved by the sparse FL algorithms with 2000 rounds of training, and the total numbers of FLOPs and communicated parameters consumed by each method to reach their respective best accuracies. Flado consistently trains the models to the *highest* accuracies compared to other sparse FL algorithms, while maintaining a low consumption of computational and communication resources. We highlight that with 2000 rounds of training,

Table 1. Comparing the sparse FL algorithms on the converged accuracies (%) using the default training setup, sorted by their competitiveness. For each row, it also evaluates the number of **additional FLOPs and communicated parameters ("Comm. Params")** required by the method in the current row to **match the best accuracy of the previous row**, and negative values indicate saving. Note that these values are consecutive comparisons and cannot be accumulated by row. We repeat experiments 3 times for statistical bounds with different random seeds for initialized parameters, and the sampling of system and data heterogeneity. "—" denotes failure to satisfy the accuracy budget.

Method	**CIFAR-10**			Permitting −5% accuracy budget from Flado			Permitting −10% accuracy budget from Flado		
	Accuracy	Δ FLOPs	Δ Comm. Params	Rounds	FLOPs	CommParams	Rounds	FLOPs	Comm. Params
HeteroFL	53.83% ±3.66%	2.13 P ±14.67 T	18.08 G ±124.45 M	—	—	—	—	—	—
UniProb	80.91% ±0.61%	−2.90 P ±2.12 P	−14.01 G ±18.03 G	—	—	—	1329.5 ±2.9	6.21 P ±14.48 T	75.59 G ±176.28 M
eFD	81.82% ±0.31%	−113.08 T ±91.82 T	−34.02 G ±1.09 G	—	—	—	1287.5 ±5.2	6.01 P ±25.31 T	51.56 G ±218.57 M
FjORD	84.38% ±0.18%	−6.59 P ±76.82 T	−47.06 G ±657.26 M	562.5 ±5.76	2.63 P ±14.48 T	31.98 G ±176.28 M	314.5 ±4.6	1.47 P ±22.77 T	17.88 G ±277.20 M
Flado	**87.24**% ±0.17%	−7.21 P ±6.18 T	−87.71 G ±75.17 M	**330.5** ±3.45	**1.54** P ±8.99 T	**18.79** G ±108.93 M	**215.5** ±2.3	**1.01** P ±11.71 T	**12.25** G ±142.62 M
Method	**SVHN**			Permitting −2% accuracy budget from Flado			Permitting −5% accuracy budget from Flado		
	Accuracy	Δ FLOPs	Δ Comm. Params	Rounds	FLOPs	Comm. Params	Rounds	FLOPs	Comm. Params
HeteroFL	89.07% ±0.23%	390.35 T ±1.02 T	38.63 G ±103.75 M	—	—	—	163.5 ±1.7	55.28 T ±647.74 G	5.47 G ±64.11 M
UniProb	90.39% ±0.07%	−299.57 T ±83.02 T	−22.48 G ±8.22 G	—	—	—	426.5 ±2.9	139.25 T ±1012.26 G	18.07 G ±131.38 M
eFD	91.11% ±0.06%	−226.29 T ±40.39 T	−39.24 G ±5.24 G	1540.5 ±2.3	502.85 T ±804.65 G	51.35 G ±83.41 M	430.5 ±5.2	140.55 T ±1.75 T	14.35 G ±182.26 M
FjORD	92.36% ±0.04%	−399.73 T ±10.61 T	−34.31 G ±1.08 G	667.5 ±3.5	217.86 T ±1.18 T	28.29 G ±156.45 M	253.5 ±1.7	82.75 T ±625.16 G	10.74 G ±81.18 M
Flado	**92.90**% ±0.04%	−354.03 T ±1.46 T	−45.98 G ±194.05 M	**442.5** ±2.9	**144.48** T ±1012.26 G	**18.75** G ±131.38 M	**199.5** ±2.9	**65.14** T ±1012.26 G	**8.45** G ±131.38 M
Method	**Fashion-MNIST**			Permitting −5% accuracy budget from Flado			Permitting −10% accuracy budget from Flado		
	Accuracy	Δ FLOPs	Δ Comm. Params	Rounds	FLOPs	Comm. Params	Rounds	FLOPs	Comm. Params
UniProb	83.00% ±0.11%	1.83 P ±2.84 T	44.38 G ±69.06 M	—	—	—	698.5 ±1.7	656.28 T ±1.76 T	15.56 G ±42.67 M
eFD	84.94% ±0.09%	−1.18 P ±3.66 T	−33.68 G ±88.83 M	—	—	—	410.5 ±2.3	386.11 T ±2.29 T	6.24 G ±38.14 M
FjORD	85.54% ±0.06%	−253.89 T ±51.17 T	+7.49 G ±847.86 M	—	—	—	366.5 ±1.1	344.15 T ±1.21 T	8.16 G ±29.45 M
HeteroFL	87.29% ±0.17%	−1.35 P ±3.66 T	−36.75 G ±88.83 M	498.5 ±4.6	491.32 T ±4.69 T	7.66 G ±74.94 M	122.5 ±5.8	115.10 T ±5.56 T	2.73 G ±134.95 M
Flado	**90.58**% ±0.09%	−1.05 P ±147.24 T	−11.56 G ±2.30 G	**354.5** ±4.0	**333.07** T ±3.93 T	**7.90** G ±95.42 M	**81.5** ±1.7	**80.33** T ±1.84 T	**1.25** G ±29.45 M

most algorithms have already converged, or converging so slowly that expending a large amount of additional resources are unlikely to notably increase the best accuracies. Figure 4 illustrates this convergence behavior.

4.3. Elasticity under Heterogeneity

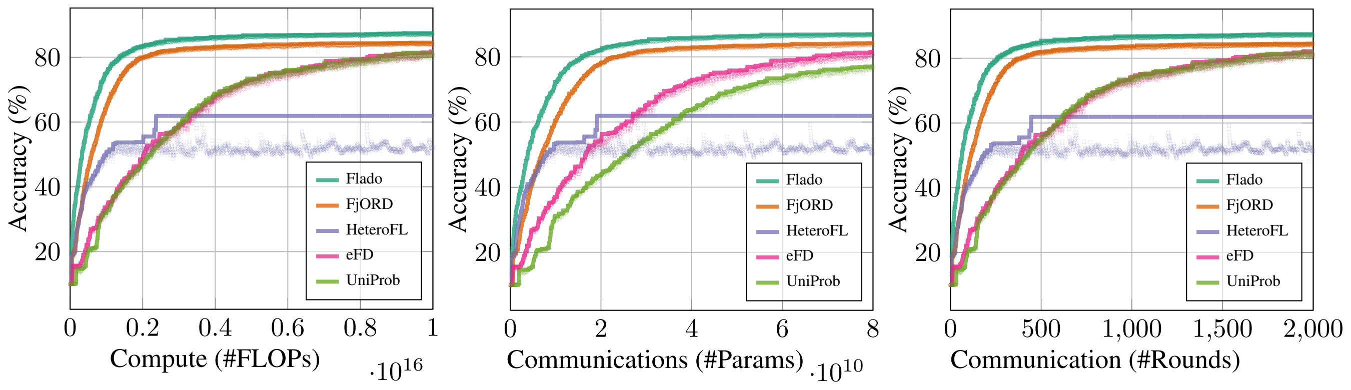
Flado tolerates aggressive data heterogeneity. To investigate how different data distributions affect the performance of sparse FL algorithms, in addition to the $\alpha = 0.5$ baseline, we examine $\alpha \in \{0.05, 5.0, \infty\}$ in the data sampling distribution $\mathcal{D}ir_{|\mathbb{C}|}(\alpha)$ for CIFAR-10 to simulate varying degrees of imbalanced data. We provide the convergence results in Table 2. Note that with an aggressive degree of data imbalance, unlike other sparse algorithms, Flado can still maintain much higher converged accuracies, and gain a notable advantage over the competing methods in terms of FLOPs and communication savings.

Flado is highly elastic under system heterogeneity. In addition to data heterogeneity, we perform sensitivity analyses of sparse FL training by adopting varying levels of performance disparities. Table 3 progressive increases the system heterogeneity from a constant FLOPs budget across all clients, to the highest system disparity, *i.e.*, we draw client FLOPs budget from $\mathcal{U}(\underline{r}, 0.64)$, with increasing values of disparity: $\underline{r} \in \{0.64, 0.32, 0.16, 0.08\}$. Note that $\underline{r} = 0.64$ corresponds to the case when all clients models consume the same number of FLOPs. Again, the result reveals that Flado typically leads the convergence in comparison to other sparse FL methods under even the heaviest system heterogeneity.

To summarize, we observe a remarkable gap between Flado and the other sparse FL algorithms, with a widening of the client performance differences. This also verifies the effectiveness of our optimization scheme.

Table 2. Following the evaluation of Table 1, we further compare the sparse FL algorithms for client data distributions under increasingly larger data heterogeneity on CIFAR-10.

$\alpha = \infty$	Accuracy	Δ FLOPs	Δ Comm. Params
HeteroFL	83.20% ±0.42%	2.49 P ±14.67 T	21.13 G ±124.45 M
UniProb	85.38% ±0.22%	+0.80 P ± 2.02 P	+31.49 G ± 17.14 G
eFD	85.86% ±0.21%	−0.87 P ±98.72 T	−40.32 G ± 1.17 G
FjORD	87.58% ±0.09%	−6.43 P ±36.67 T	−44.91 G ±313.70 M
Flado	**89.16**% ±0.08%	−7.13 P ±58.68 T	−86.80 G ±714.29 M
$\alpha = 5$	Accuracy	Δ FLOPs	Δ Comm. Params
HeteroFL	82.51% ±0.34%	5.17 P ±14.67 T	43.86 G ±124.45 M
UniProb	84.82% ±0.17%	+1.69 P ±14.67 T	+39.67 G ±124.45 M
eFD	85.69% ±0.25%	−1.75 P ±14.48 T	−48.29 G ±176.28 M
FjORD	86.92% ±0.17%	−5.38 P ±14.47 T	−32.17 G ±123.96 M
Flado	**88.85**% ±0.10%	−6.97 P ±14.48 T	−84.81 G ±176.28 M
$\alpha = 0.05$	Accuracy	Δ FLOPs	Δ Comm. Params
HeteroFL	28.06% ±5.04%	2.27 P ±14.67 T	19.29 G ±124.45 M
UniProb	63.05% ±1.14%	+0.60 P ±14.67 T	+15.51 G ±124.45 M
eFD	62.84% ±1.20%	−0.34 P ±13.10 T	−49.88 G ±159.45 M
FjORD	77.64% ±0.91%	−10.54 P ±14.44 T	−82.39 G ±124.11 M
Flado	**79.14**% ±1.12%	−5.92 P ±14.48 T	−72.07 G ±176.28 M



(a) Accuracy *vs*. FLOPs. (b) Accuracy *vs*. Communicated parameters. (c) Accuracy *vs*. Communication rounds.

Figure 4. Visualization of convergence curves for CIFAR-10. Flado leads a large gap against other sparse FL methods, in both computational and communication savings w.r.t. trained model accuracy.

Table 3. We further compare the sparse FL algorithms under increasingly level of system heterogeneity on CIFAR-10, following the evaluation of Table 1.

$\mathcal{U}(0.64, 0.64)$	Accuracy	Δ FLOPs	Δ Comm. Params
FjORD	87.01%±0.11%	15.26 P±23.83 T	112.89 G±176.28 M
HeteroFL	87.43%±0.09%	−4.72 P±24.22 T	−47.32 G±150.50 M
UniProb	87.44%±0.13%	−3.11 P±23.79 T	−4.81 G±176.28 M
eFD	88.26%±0.09%	−2.18 P±23.82 T	−29.42 G±149.96 M
Flado	**88.82**%±0.14%	−11.88 P±23.83 T	−25.67 G±176.28 M
$\mathcal{U}(0.32, 0.64)$	Accuracy	Δ FLOPs	Δ Comm. Params
HeteroFL	58.91%±4.23%	2.96 P±19.11 T	21.40 G±138.38 M
eFD	85.55%±0.15%	+227.50 T±18.92 T	+1.89 G±138.11 M
UniProb	86.08%±0.31%	−2.81 P±18.84 T	−1.70 G±176.28 M
FjORD	86.43%±0.14%	−102.70 T±18.84 T	−960.70 M±176.28 M
Flado	**87.88**%±0.13%	−8.98 P±18.84 T	+3.04 G±176.28 M
$\mathcal{U}(0.16, 0.64)$	Accuracy	Δ FLOPs	Δ Comm. Params
HeteroFL	57.64%±4.65%	2.52 P±16.53 T	20.02 G±131.09 M
eFD	83.82%±0.28%	+842.98 T±16.44 T	+6.78 G±131.05 M
UniProb	83.89%±0.43%	−331.42 T±16.35 T	+25.80 G±176.28 M
FjORD	85.84%±0.19%	−6.83 P±16.35 T	−73.60 G±176.28 M
Flado	**86.91**%±0.16%	−5.09 P±16.34 T	−54.90 G±176.28 M
$\mathcal{U}(0.08, 0.64)$	Accuracy	Δ FLOPs	Δ Comm. Params
HeteroFL	57.39%±4.20%	2.35 P±15.36 T	19.39 G±126.96 M
eFD	81.45%±0.59%	+999.93 T±15.02 T	+8.46 G±126.61 M
UniProb	81.70%±0.52%	−186.77 T±15.10 T	+29.60 G±176.28 M
FjORD	84.58%±0.19%	−6.62 P±15.10 T	−77.24 G±176.28 M
Flado	**86.98**%±0.11%	−5.93 P±15.08 T	−69.17 G±176.28 M

4.4. Scalability Experiments

Fractional Device Participation. To measure the performance of Flado in the more practical distributed scenario, we scale the number of devices by a factor of 10, while reducing the fraction of participation rate $\mathcal{R}$ per round to 10%, yielding results in Table 4. Flado outperforms other sparse training for FL under fractional device participation by a notable margin. Not only does it increase the converged accuracies, it can also drastically reduce the FLOPs and communication cost.

Table 4. Comparing the accuracies of sparse FL algorithms, considering 10× the original device count and a 10% fractional device participation ratio. We remind that each row also reports the number of additional FLOPs and communicated parameters ("Comm. Params") required by the current method to match the best accuracies of the previous row, and negative values indicate saving.

Method	**CIFAR-10**		
	Accuracy	Δ FLOPs	Δ Comm. Params
UniProb	25.98%±1.07%	829.11 T±1.26 T	111.25 G±176.28 M
eFD	38.59%±0.88%	−88.17 T±1.28 T	−44.27 G±118.47 M
HeteroFL	60.91%±1.54%	−660.16 T±1.26 T	−59.79 G±117.45 M
FjORD	73.74%±1.19%	+45.57 T±1.28 T	+41.06 G±176.28 M
Flado	**78.71**%±0.84%	−15.91 T±1.41 T	−20.04 G±176.28 M
Method	**SVHN**		
	Accuracy	Δ FLOPs	Δ Comm. Params
UniProb	86.77%±0.12%	88.76 T±61.88 G	84.69 G±56.02 M
eFD	87.13%±0.15%	−6.02 T±138.27 G	−22.08 G±104.31 M
FjORD	89.37%±0.06%	−31.10 T±137.76 G	−12.38 G±131.38 M
HeteroFL	90.60%±0.06%	−25.97 T±137.38 G	−24.77 G±131.38 M
Flado	**91.12**%±0.08%	−19.96 T±142.25 G	−33.98 G±104.55 M
Method	**Fashion-MNIST**		
	Accuracy	Δ FLOPs	Δ Comm. Params
eFD	77.58%±0.21%	203.20 T±318.92 G	30.40 G±47.98 M
UniProb	78.60%±0.36%	−45.20 T±321.37 G	+3.70 G±69.06 M
FjORD	81.04%±0.26%	−60.82 T±317.30 G	−12.91 G±69.06 M
HeteroFL	86.32%±0.21%	−131.26 T±317.15 G	−28.68 G±69.06 M
Flado	**86.86**%±0.18%	−27.55 T±331.94 G	−18.56 G±48.15 M

Scaling to larger models. Table 5 scales the competition to a larger ResNet-18 model, which employs batch normalization (BN) layers. Under $E = 1$ local epochs per round, HeteroFL and eFD failed to produce models even with 2000 rounds of training. It reveals that frequent communications (*i.e.*, $E = 1$) are surprisingly harmful to sparse FL algorithms that allow clients to train only sub-models. The experiments thus use a search of $E \in \{1, 2, 4, 8, 16, 32, 64\}$, and

Table 5. Following the evaluation of Table 1, we further compare the sparse FL algorithms with a much larger ResNet-18 model than the baseline VGG-9 on CIFAR-10.

ResNet-18	Accuracy	Δ FLOPs	Δ Comm. Params
eFD	$19.35\%_{\pm 1.16\%}$	$3.58\,\mathrm{P}_{\pm 239.47\,\mathrm{T}}$	$11.67\,\mathrm{G}_{\pm 778.04\,\mathrm{M}}$
HeteroFL	$24.51\%_{\pm 4.26\%}$	$+1.75\,\mathrm{P}_{\pm 476.43\,\mathrm{T}}$	$+2.88\,\mathrm{G}_{\pm 778.22\,\mathrm{M}}$
UniProb	$73.87\%_{\pm 0.37\%}$	$+1.66\,\mathrm{P}_{\pm 953.86\,\mathrm{T}}$	$+3.61\,\mathrm{G}_{\pm 1.26\,\mathrm{G}}$
FjORD	$78.36\%_{\pm 2.08\%}$	$-7.23\,\mathrm{P}_{\pm 953.86\,\mathrm{T}}$	$-9.79\,\mathrm{G}_{\pm 1.26\,\mathrm{G}}$
Flado	$\mathbf{92.04}\%_{\pm 0.50\%}$	$-6.90\,\mathrm{P}_{\pm 1.20\,\mathrm{P}}$	$-36.06\,\mathrm{G}_{\pm 1.26\,\mathrm{G}}$

report the highest accuracies of the competing algorithms, to give them a better chance. As noted by [9], there are privacy concerns related to frequent uploads of running BN statistics, we thus follow [9] to query clients to update BN statistics after all training rounds have been completed. Our results show that under both data and system heterogeneity, Flado demonstrates the best trained accuracies scaling to a larger model.

Flado efficiently trades off communication and computational costs. To bolster the trade-off between communication and computation, we additionally conducted a hyperparameter exploration to navigate the Pareto frontiers of this trade-off relationship. The hyperparameters include the number of local epochs per round $E \in \{1, 2, 4, 8, 16, 32, 64\}$ and the batch size $B \in \{16, 32, 64\}$. Figure 5 shows the FLOPs/communication trade-off of the FL methods under both data and system heterogeneity. Here, we allow a 5% accuracy degradation from the best accuracy attained by all algorithms (87.24%). It shows that Flado Pareto-dominates all competing algorithms in terms of FLOPs *vs*. communication trade-offs. It is notable that with decreasing FLOPs budgets, Flado expends minimal communications, whereas other methods consume significantly higher communication costs (more than $10\times$) to reach the same target accuracy.

In summary, **Flado substantially outperforms all competing sparse FL methods**, showing that aligning trajectories across training clients is a powerful technique to improve convergence under sparsity for FL. In contrast, existing sparse FL algorithms prescribes fixed channel sparsities. They are unable to adapt their sparsity strategies to disparate model training trajectories incurred by non-i.i.d. data distributions and a wide spectrum of client capabilities. It also demonstrates high versatility to scaling, and good adaptability to hyperparameter settings.

Additional observations. In Appendix D, we provide ablation and sensitivity analyses of components in Flado, we also show in Appendix C that the FJLT provides an efficient and effective approximation of the cosine-similarity metric.

5. Conclusion

Recent federated learning (FL) algorithms focus on the reduction of communication costs, while neglecting the expensive local compute required by edge clients. This could be further exacerbated as the models we employ may increase in size over time to attain a higher task performance. We presented Flado, a novel FL technique that uses channel sparsities for all neurons in clients with adaptive sparse probabilities to concentrate the clients' training effort on neurons that they specialize well, while making the models sparse to reduce computational costs. Experiments show that Flado can push the Pareto frontiers of communication/computation trade-off of FL scenarios notably further than existing algorithms. We believe this paves the way for future work that allow FL algorithms to scale up models being trained considerably, and consequently improve their performance on more challenging training tasks.

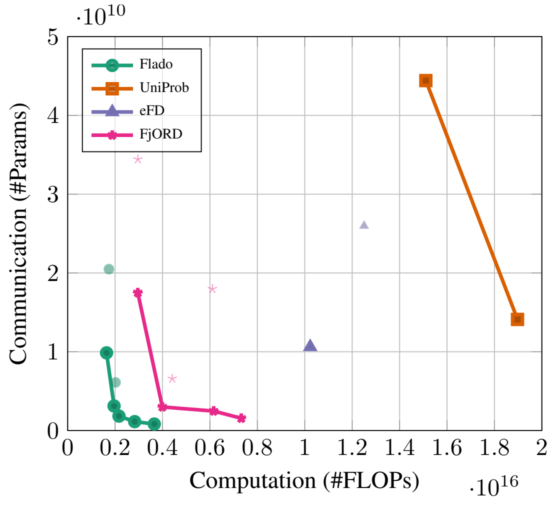


Figure 5. Comparing the FLOPs *vs*. communicated parameters trade-off across different FL methods reaching a target accuracy under both data and system heterogeneity. Here, we permit an accuracy degradation of 5% on CIFAR-10 from the best possible baseline accuracy (87.24%) and report the FLOPs and communication costs minimally required by the methods under a wide parameter grid search to reach the target accuracy. We highlight the Pareto frontier of each algorithm with solid lines. Note that some algorithms report fewer or no results (*e.g.*, HeteroFL) as they may fail to train to the target accuracy under the given parameter setups.

Acknowledgements

This work is supported in part by Science and Technology Development Fund of Macao S.A.R (FDCT) under Nos. 0015/2019/AKP, 0123/2022/AFJ, and 0081/2022/A2, Guang-Dong Basic and Applied Basic Research Foundation (No. 2020B1515130004), and Shenzhen Science and Technology Innovation Commission (No. JCYJ20190812160003719). This work was carried out in part at SICC, which is supported by SKL-IOTSC, University of Macau.

References

[1] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 557–563, 2006.

[2] Amjad Almahairi, Nicolas Ballas, Tim Cooijmans, Yin Zheng, Hugo Larochelle, and Aaron Courville. Dynamic capacity networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning (ICML)*, pages 2549–2558, 2016.

[3] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Agata Lapedriza, Bolei Zhou, and Antonio Torralba. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 2020.

[4] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[5] Sebastian Caldas, Jakub Konečny, H Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210*, arXiv:1812.07210, 2018. https://arxiv.org/abs/1812.07210.

[6] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konecný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. LEAF: A benchmark for federated settings. *ArXiv*, abs/1812.01097, 2018. http://arxiv.org/abs/1812.01097.

[7] Mary J Culnan and Cynthia Clark Williams. How ethics can enhance organizational privacy: lessons from the choicepoint and TJX data breaches. *Mis Quarterly*, pages 673–687, 2009.

[8] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Struct. Algorithms*, 22(1):6065, jan 2003.

[9] Enmao Diao, Jie Ding, and Vahid Tarokh. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. In *International Conference on Learning Representations*, 2021.

[10] Benjamin Edwards, Steven Hofmeyr, and Stephanie Forrest. Hype and heavy tails: A closer look at data breaches. *Journal of Cybersecurity*, 2(1):3–14, 2016.

[11] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng-Zhong Xu. Dynamic channel pruning: Feature boosting and suppression. In *International Conference on Learning Representations*, 2019.

[12] Geekbench. Latest geekbench ML inference results. https://browser.geekbench.com/ml/v0/inference?page=1.

[13] Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos I. Venieris, and Nicholas D. Lane. FjORD: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *CoRR*, abs/2102.13451, 2021.

[14] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *NeurIPS Workshop on Federated Learning for Data Privacy and Confidentiality*, abs/1909.06335, 2019.

[15] Yuang Jiang, Shiqiang Wang, Bong-Jun Ko, Wei-Han Lee, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices. *CoRR*, abs/1909.12326, 2019.

[16] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic controlled averaging for federated learning. In Hal Daum III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5132–5143. PMLR, 13–18 Jul 2020.

[17] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The CIFAR-10 and CIFAR-100 datasets. *http://www.cs.toronto.edu/ kriz/cifar.html*, 2014.

[18] Yann Lecun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[19] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems (NIPS)*, pages 598–605. 1990.

[20] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems*, 27:19–27, 2014.

[21] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[22] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 429–450, 2020.

[23] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning ef-

ficient convolutional networks through network slimming. In *International Conference on Computer Vision (ICCV)*, 2017.

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

[25] Amirkeivan Mohtashami, Martin Jaggi, and Sebastian Stich. Masked training of neural networks with partial gradients. pages 5876–5890, 2022.

[26] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[27] Diego Peteiro-Barral and Bertha Guijarro-Berdiñas. A survey of methods for distributed machine learning. *Progress in Artificial Intelligence*, 2(1):1–11, 2013.

[28] Eugenia Politou, Efthimios Alepis, and Constantinos Patsakis. Forgetting personal data and revoking consent under the GDPR: Challenges and proposed solutions. *Journal of Cybersecurity*, 4(1), 2018.

[29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[30] Yunchuan Sun, Junsheng Zhang, Yongping Xiong, and Guangyu Zhu. Data security and privacy in cloud computing. *International Journal of Distributed Sensor Networks*, 10(7):190903, 2014.

[31] Paul Voigt and Axel Von dem Bussche. The EU general data protection regulation (GDPR). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10:3152676, 2017.

[32] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *International Conference on Learning Representations*, 2020.

[33] Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H Brendan McMahan, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, et al. A field guide to federated optimization. *arXiv preprint arXiv:2107.06917*, 2021.

[34] PTJ Wolters. The security of personal data under the GDPR: a harmonized duty or a shared responsibility? *International Data Privacy Law*, 7(3):165–178, 2017.

[35] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. 2017. https://github.com/zalandoresearch/fashion-mnist.

[36] Xin Yuan, Pedro Henrique Pamplona Savarese, and Michael Maire. Growing efficient deep networks by structured continuous sparsification. In *International Conference on Learning Representations*, 2020.

[37] Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba. Interpreting deep visual representations via network dissection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(9):2131–2145, 2018.

[38] Xiao Zhou, Weizhong Zhang, Zonghao Chen, Shizhe Diao, and Tong Zhang. Efficient neural network training via forward and backward propagation sparsification. In *Advances in Neural Information Processing Systems*, 2021.