PAPER *Special Section on Forefront Computing*

Towards Superior Pruning Performance in Federated Learning with Discriminative Data

Yinan YANG[†a), ***Member***

SUMMARY Federated Learning (FL) facilitates deep learning model training across distributed networks while ensuring data privacy. When deployed on edge devices, network pruning becomes essential due to the constraints of computational resources. However, traditional FL pruning methods face bias issues arising from the varied distribution of local data, which poses a significant challenge. To address this, we propose DDPruneFL, an innovative FL pruning framework that utilizes Discriminative Data (DD). Specifically, we utilize minimally pre-trained local models, allowing each client to extract semantic concepts as DD, which then inform an iterative pruning process. As a result, DDPruneFL significantly outperforms existing methods on four benchmark datasets, adeptly handling both IID and non-IID distributions and Client Selection scenarios. This model achieves state-of-the-art (SOTA) performance in this field. Moreover, our studies comprehensively validate the effectiveness of DD. Furthermore, a detailed computational complexity analysis focused on Floating-point Operations (FLOPs) is also conducted. The FLOPs analysis reveals that DDPruneFL significantly improves performance during inference while only marginally increasing training costs. Additionally, it exhibits a cost advantage in inference when compared to other pruning FL methods of the same type, further emphasizing its cost-effectiveness and practicality.

key words: *federated learning, network pruning, network compression, edge-device, deep learning*

1. Introduction

Deep Neural Networks (DNNs) have achieved remarkable advancements, largely dependent on extensive training data [1]. However, with the ever-increasing volume of data being collected and produced at edge devices, achieving a better model performance under the traditional centralized approach often necessitates compromising user privacy by transmitting private data to a central server [2]–[4]. In response to these privacy concerns, Federated Learning (FL) has emerged as an innovative solution. It enables the training of DNNs across distributed networks without the need for data centralization, thereby preserving privacy [5], [6]. This approach not only maintains user privacy but also ensures high model performance [7]–[9].

With the increasing complexity and size of DNNs, resource constraints for on-device computation emerge as a significant limitation. Network pruning, which aims to reduce model size while preserving performance [10], [11], has become a critical solution for these resource limitations in FL models [12]–[15]. However, in FL frameworks, limited access to global data can lead to significantly biased subnetwork during pruning, as the effectiveness of most pruning methods is influenced by data distribution [16]–[18]. Specifically, the widely recognized Adaptive method [13], known for effectively combining FL's privacy protection with efficient pruning [19], suffers from pruning bias due to its overreliance on the initial pruning on a single client [16]. Furthermore, this pruning bias is influenced by both heterogeneous (non-IID) and homogeneous (IID) local data distributions [16], presenting a significant challenge in maintaining the performance of pruning FL models.

To address the issue of pruning bias arising from the limitation of using local data, we propose DDPruneFL, a novel FL pruning method using Discriminative Data. The core idea involves extracting semantic concepts as Discriminative Data from each client, which are then for collaborative pruning across multiple clients. This method consists of three key steps, as illustrated in Fig. 1. First, clients engage in local pre-training using their own datasets. Second, inspired by [17], we apply the Automatic Concept-based Explanations (ACE) method [20] to identify superpixel segments important for classification on each client. These identified segments are termed Discriminative Data (DD), which are different across clients. Third, the pruning score for each connection (e.g., connection sensitivity [17], [21]–[23]) is calculated on each client after DD is passed through their networks. These scores are then uploaded to the server for aggregating analysis to create a global pruning mask. This global mask is then sent back to the clients for rounds of FL fine-tuning, marking the completion of one full iterative loop. Importantly, by retaining DD locally and only sending the pruning scores to the server, our model effectively maintains the privacy of FL.

The underlying intuition of our method to mitigate pruning bias lies in the utilization of DD from each client. As demonstrated in [17], DD has been proven to be more effective for pruning, enhancing the generalization capabilities of models. In our method, despite starting with identical models from the server, clients generate diverse DD due to their varying data distributions. Unlike traditional methods such as Adaptive [13], which are limited by local data, our method indirectly involves all clients in the pruning process by smartly utilizing distributed DD.

As a result, our model achieves SOTA performance across four benchmark datasets, demonstrating its robustness in pruning FL models. Specifically, we conduct extensive benchmark tests across four diverse datasets

Manuscript received January 3, 2024.
Manuscript revised May 11, 2024.
Manuscript publicized July 11, 2024.

[†]Graduate School of Information Science and Engineering, Ritsumeikan University, Kusatsu-shi, 525–8577 Japan.
a) E-mail: gr0528xf@ed.ritsumei.ac.jp

DOI: 10.1587/transinf.2024PAP0002

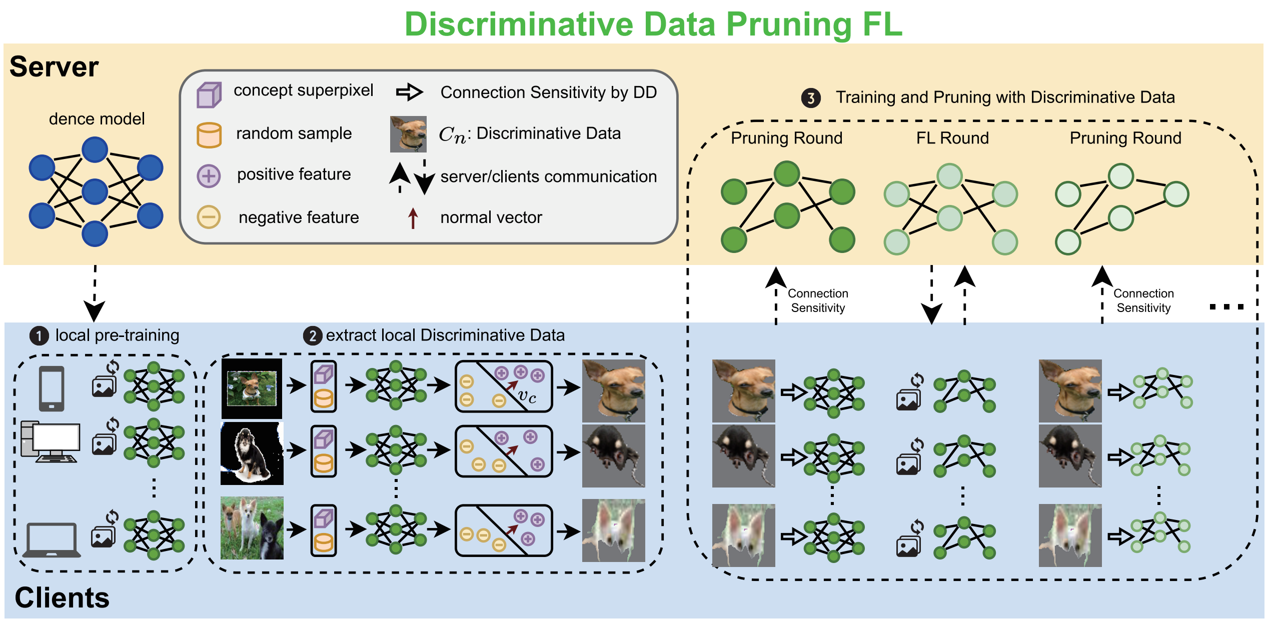

Fig. 1 Overview of DDPruneFL Workflow. DDPruneFL has three steps: (1) Pre-training the model using local data on clients for several rounds; (2) Extract DD: Identifying task-positive features in the Euclidean space of the model, trained with concept superpixels segmented from images, and storing these as each client's Discriminative Data; (3) Utilizing DD in further pruning and FL fine-tuning processes.

(CIFAR-10 [24], FEMNIST [25], CelebA [26], ImageNet-100 [27]) using four distinct models (two Convolutional Networks [28], VGG [29], ResNet [30]), respectively. Our dataset selection covers both IID and non-IID [31] FL data distributions, and we also address FL Client Selection training mode [32]. In all benchmark comparisons, our method consistently outperformed other similar methods at high sparsity levels. Moreover, although FedTiny [16] addresses pruning bias with client-side batch normalization, DDPruneFL outperforms it in the same settings.

Our ablation study confirms that pruning bias in the Adaptive method could be caused by initial pruning on a single client. Eliminating initial pruning significantly enhances the pruning performance at the high-sparsity level. Ablation studies also demonstrate our method's stability across various pruning schedules. Moreover, we investigate the impact of initial pre-training rounds on pruning performance, finding a balance between performance and stability. Additionally, we demonstrate that data distribution affects the loss trend, supporting our approach of modifying input data to improve effectiveness.

We also conduct a comprehensive computational cost evaluation using FLOPs (Floating-point Operation) analysis [33]. Demonstrated by CIFAR-10 experiments, our findings reveal that DDPruneFL leads to a mere 0.45% increase in computational training cost per client compared to standard dense model training. Furthermore, its inference cost is significantly lower than that of other similar FL pruning methods, highlighting its efficiency and practicality.

Our contributions are summarized as follows.

(1) The proposed DDPruneFL establishes a new SOTA standard for pruning within the FL framework across various benchmark tests. It significantly boosts the overall network efficiency and performance of the pruned FL model.

(2) Our experiments suggest that pruning bias within FL can stem from data dependency in the pruning process. Additionally, our comprehensive ablations confirm the effectiveness of Discriminative Data.

(3) Our research offers a detailed examination of the computational complexity associated with implementing DDPruneFL on client devices. We perform a quantitative evaluation of the single-client FLOPs on the CIFAR-10 dataset. The findings indicate that DDPruneFL substantially enhances performance during inference with a minimal rise in training costs. It also exhibits a cost advantage in inference when compared to other similar FL pruning methods, underscoring its efficiency and practicality.

2. Related Work

2.1 Network Pruning

Network Pruning, evolving since the 1980s [22], [34], [35], gained significant momentum with Han *et al.*'s research [10], applying it to DNNs. This marked a notable integration of magnitude-based iterative pruning in complex neural architectures. The subsequent Lottery Ticket Hypothesis (LTH) [36] illuminates the potential of efficient, sparse subnetworks within larger models. However, LTH faces limitations due to high resource demands [11], [37]. On the other hand, one-shot pruning methods at initialization, exemplified by gradient-informed connection sensitivity (SNIP) [21] and GraSP [38], have been developed as efficient pruning methods with significantly lower computational costs. Specifically, SNIP operates by assessing the connection sensitivity in the network at the initialization stage to identify redundancies, whereas GraSP utilizes the Hessian matrix to preserve gradient flow during pruning. Synflow [39] introduces a data-independent, iterative pruning approach at initialization. Recent work by Yang *et al.* [17] explicitly demonstrates the data dependency of One-shot Pruning at Initialization. However, their analysis is based on models that had already undergone full training, and they do not consider iterative pruning methods.

Contrasting with traditional network pruning that relies on centralized data, our approach operates within the FL framework, characterized by distributed data. In this framework, each client accesses only a part of the full dataset, diverging from models that assume complete data availability. Our research contributes by investigating how limited data access impacts pruning effectiveness within FL.

2.2 Efficient Federated Learning

In recent years, Federated Learning has attracted extensive attention as a viable solution to data privacy challenges in collaborative machine learning. Federated Averaging (FedAvg) [5], a pioneering FL method, uses model updates on local devices rather than raw data transfer, allowing for private knowledge sharing. Inspired by pruning at initialization, Xu *et al.* [12] propose server-side pruning of the original full-size model, followed by local data fine-tuning on devices. Additionally, pruning methods like SNIP, GraSP, and SynFlow, originally designed for pruning at initialization, can be adapted for server-side pruning in FL [13], [16]. However, these methods often overlook how data distribution across clients impacts the pruning effectiveness.

To address the distributed nature of data in FL, recent studies have shifted some pruning operations to devices in a federated setup. FedPrune [14] allocates full dense local training to select devices, guiding pruning based on updated activation patterns. LotteryFL [15] iteratively prunes the full-size model on devices at a fixed rate to discover personalized local subnetworks, which are then integrated using server-side mask fusion. FedDST [40] employs device-side mask adjustments, with the server generating a new global model through sparse aggregation and magnitude pruning. Despite increased computational costs and potential delays in FL, these methods involve local devices in computation [13]. Benchmark studies, however, show that compared to similar pruning methods like SNIP, they do not exhibit superior performance at high sparsity levels [16], [41]. Adaptive [13] initiates pruning coarsely on a single client, followed by more refined global pruning and fine-tuning. Their method explores the balance between model sparsity and time efficiency in FL. However, our experiments indicate that such single-client coarse pruning introduces significant bias, which adversely affects the model's performance.

Recent research, FedTiny [16], focuses on mitigating pruning bias on the client side by employing an adaptive batch normalization selection module. This approach effectively counteracts biases inherent in local data, achieving SOTA results. However, when compared with their reported outcomes, our experiments indicate that our Discriminative Data-based FL pruning method is more effective in reducing bias. It yields superior experimental results and sets a new SOTA performance standard.

3. Proposed Method

Overview: Section 3.1 introduces some definitions and notations in DDPruneFL. Section 3.2 provides details on extracting DD and the pruning process within an FL framework.

3.1 Notations Related to Pruning and Mask

A FL framework includes a center server S and clients N. Each client $n \in [N] = \{1, 2, \ldots, N\}$ follows a local empirical risk $F_n(\mathbf{w}) := (1/D_n) \sum_{i \in \mathcal{D}_n} f_i(\mathbf{w})$, trained on its local dataset $\mathcal{D}_n$ with $D_n := |\mathcal{D}_n|$, where $\mathbf{w}$ denotes the model parameter vector. Here, $f_i(\mathbf{w})$ represents the loss function, quantifying the difference between the predicted and actual outputs for the i-th data sample.

The overarching goal of the FL framework is to find an optimal parameter vector $\mathbf{w}$ that minimizes the global empirical risk. This objective is formulated as:

$$\min_{\mathbf{w}} F(\mathbf{w}) := \sum_{n \in [N]} p_n F_n(\mathbf{w}) \tag{1}$$

where $p_n > 0$ are the client weights that satisfy the condition $\sum_{n \in [N]} p_n = 1$.

We refer to the multiple local iterations and a subsequent fusion step as a round, denoted by r. To alleviate delays from waiting for all clients in FL, we also consider Client Selection (C.S.) [32], [42] scenario, where each round may involve only a subset of clients.

The objective of network pruning is to extract a high-performing and edge-device-friendly subnetwork from the dense architecture of the original network. This process involves identifying and retaining critical parameters while pruning redundant ones. For the network parameter vector w, a binary mask $m \in \{0, 1\}$ is applied. In this mask, a value of 1 signifies that the parameter is to be retained, while a value of 0 indicates that it is to be pruned. Hence, the pruned global empirical risk function is formalized as follows,

$$\min_{\mathbf{w},\mathbf{m}} F(\mathbf{w} \odot \mathbf{m}) := \sum_{n \in [N]} p_n F_n(\mathbf{w} \odot \mathbf{m}), \tag{2}$$

where $\mathbf{m}$ is the mask vector and $\odot$ denotes the Hadamard product.

In our approach, we adopt Connection Sensitivity [21] to serve as the criterion for pruning. Following [21], the importance of each parameter is determined by the product of the absolute value of its gradient and its magnitude, as defined by:

$$s(w) = \left| \frac{\partial F(\mathbf{w})}{\partial w} \odot w \right|. \tag{3}$$

Algorithm 1 DDPruneFL

```
1: // Server-side
2: Initial $S$, $\mathbf{m} = \mathbb{1}$, $n \in [N] = \{1, 2, \dots, N\}$    ▷ Initialization
3: Server push $\mathbf{w}$ and $\mathbf{m}$ to Clients
4: for $r = 0, \dots, R-1$ do
5:     // Clients-side    ▷ Client update parameters
6:     $F_n(\mathbf{w} \odot \mathbf{m}) := (1/D_n) \sum_{i \in \mathcal{D}_n} f_i(\mathbf{w} \odot \mathbf{m})$
7:     Clients push $\mathbf{w}$ to Server
8:     // Server-side    ▷ Parameters aggregation
9:     $\min_{\mathbf{w}} F(\mathbf{w}) := \sum_{n \in [N]} p_n F_n(\mathbf{w})$
10:    if $r == R_{\text{pruning}}$ then
11:        // Clients-side
12:        for $n = 0, \dots, N-1$ do
13:            if $C_n == \emptyset$ then    ▷ Extract local DD
14:                ▷ SLIC segmentation
15:                $\mathcal{D}_n \to \mathrm{SP}_n \in \mathbb{R}^{\gamma \times w \times h}$
16:                ▷ Cluster
17:                $[C_n^1, \dots, C_n^K] = \mathrm{Cluster}(l : \mathrm{SP}_n \to \mathbb{R}^{\Gamma})$
18:                ▷ Score and Filter DD
19:                $C_n = \{\mathrm{Score}([C_n^1, \dots, C_n^K]) \geq \varphi\}$
20:            end if
21:            ▷ FL pruning with DD
22:            Calculate Connection Sensitivity in Formula.3
23:            Client Push Connection Sensitivity to Server
24:        end for
25:        // Server-side
26:        $s(w) = \mathrm{AVG}(w_n\text{'s Connection Sensitivity})$
27:        $m(w) = \text{Top-}\kappa(s(w))$
28:        Server push $\mathbf{m}$ to Clients
29:    end if
30:    // Server-side
31:    if $r == R_{\text{pretrain}}$ then    ▷ Local pre-training
32:        Continue
33:    else if $r != R_{\text{pretrain}}$ then    ▷ Distribute
34:        Server push $\mathbf{w}$ to clients
35:    end if
36: end for
```

Then, we rank the parameters by their importance scores and set the mask values m to 1 for the top-κ parameters, where κ is determined by the targeted sparsity level. Parameters that do not rank within the top-κ are pruned, setting their mask values to 0.

3.2 Pruning in FL with Discriminative Data

DDPruneFL is a novel FL pruning method that indirectly leverages the data of multiple clients for collaborative network pruning. In DDPruneFL, masks are aggregated at the server level to form a global mask, ensuring that data from diverse clients are appropriately utilized in the pruning process. The workflow is illustrated in Fig. 1. Moreover, the comprehensive algorithmic procedure is detailed in Algorithms 1, with the following specific steps.

(1) Local Pre-training

After initializing a neural network, the server distributes it to each client. During the pre-training round, denoted by R_{pretrain}, client n independently trains its local network using its local dataset $\mathcal{D}_n \in \mathbb{R}^{\gamma \times w \times h}$, where γ represents the number of input channels, w and h denote the width and height of the input, respectively. Clients update the server with their gradients, but the server aggregates these without updating the clients' parameters. This approach offers two benefits: it reduces network communication overhead during pre-training; it also enhances training set accuracy on the client's data, which is crucial for identifying relevant image features for the task.

(2) Extract Local DD

This section outlines the utilization of the ACE (Automatic Concept-based Explanation) method [17], [20] within clients to extract informative data, which, in our study, are referred to as Discriminative Data (DD). These data serve as the foundational material for subsequent pruning steps. Moreover, the ACE method effectively identifies critical image concepts crucial for the classification task. The extraction of DD via ACE entails segment clustering for concept recognition, followed by scoring and filtering these concepts.

SLIC and cluster. In this step, we aim to cluster image segments in the Euclidean space using clients' pre-trained model, in order to identify clusters with the same semantic meaning. Specifically, after completing R_{pretrain} rounds, we set a bottleneck layer l in the network to serve as the Euclidean activation space for clustering segments to recognize image concepts. $\mathcal{D}_n$ used in pre-training in each client is segmented into concept superpixels by employing the SLIC superpixel segmentation method [43], $\mathcal{D}_n \to \mathrm{SP}_n \in \mathbb{R}^{\gamma \times w \times h}$, where SP_n are the superpixels resized to align with the network's input dimensions. We pick the SP_n into the network and select the activations of these concept superpixels in layer l, mapped as $l : \mathbb{R}^{\gamma \times w \times h} \to \mathbb{R}^{\Gamma}$. Then those concept's activations are grouped in Euclidean space via the k-means algorithm into semantically coherent clusters, $[C_n^1, C_n^2, \dots, C_n^K]$.

Score. We utilize TCAV (Testing with Concept Activation Vectors) [44] to measure the degree of classification relevance for each semantic cluster, which is represented by scores. Kim et al. [44] define the sensitivity of a particular image concept C for a specific classification k in any given input x within a neural network as follows:

$$S_{C,k,l}(\boldsymbol{x}) = \nabla h_{l,k}\left(f_l(\boldsymbol{x})\right) \cdot \boldsymbol{v}_C^l, \tag{4}$$

where $f_l(\boldsymbol{x})$ is the activation in layer l, $\nabla h_{l,k}(f_l(\boldsymbol{x}))$ represents the logistic gradient of the activation $f_l(\boldsymbol{x})$ at layer l for input x, and $\boldsymbol{v}_C^l$ is the Concept Activation Vector (CAV) [20], [44]. The CAV serves as the vector normal that distinguishes random samples from the concept in the activation space. In other words, this sensitivity reflects the similarity between the intermediate layer activation gradients of the sample and the CAV. A positive and higher sensitivity indicates a greater contribution of the image concept to the classification.

We measure the score of each semantic cluster by calculating the statistical percentage of effective concepts within the same semantic cluster. Specifically, we compute a binary linear classification plane for each concept semantic cluster and the activation of random samples. Then, we measure the similarity between the normal vector of this plane and the gradients of the concept for class k at layer l as sensitivity, as shown in Eq. (4). The importance score is the percentage of concepts within the same semantic cluster with sensitivity greater than zero. Thus, a higher importance score indicates that the concept represented by the semantic cluster is more significant for this classification

Filter DD. We select the concept superpixels with an importance score above the threshold φ as the DD in each client n, denoted by C_n, as illustrated in Fig. 1. In our experiments, we typically set the factor $\varphi = 0.1$, which balances the efficacy of DD with the requirements for data collection. Notably, C_n is stored on each client for subsequent iterative pruning processes, thereby maintaining data privacy in the FL framework.

(3) FL Training and Pruning with DD

Client pruning with DD and upload. After collecting the specific C_n from each client, if the current round is identified as a pruning round, C_n is passed through the client n's network to collect the gradient of parameters. We adopt the SNIP criterion for pruning, where the pruning score is determined by the absolute value of the product of the parameter magnitude and its gradient, as defined in Eq. (3). The clients subsequently upload pruning scores for each parameter of the model to the server.

Server pruning aggregation. The server aggregates the pruning scores (Connection Sensitivity) from all clients using an averaging strategy and computes the aggregated pruning scores for active parameters within the model. The server then identifies the parameters with the top-κ scores, from which the pruning mask is generated. This mask, along with updated model parameters, is distributed to each client, marking the completion of the pruning round R_{pruning}.

We employ iterative pruning, where each pruning iteration removes $(1-\kappa)\%$ of the surviving parameters. In our setup, we achieve the target sparsity through 20 pruning iterations, following [10], [15].

FL upload and distribute. In rounds without pruning, the algorithm follows the conventional FedAvg training process for both clients and the server. Specifically, clients collect gradients for each round and upload them to the server. The server then employs FedAvg to aggregate these gradients, computes the new model parameters, and distributes them to each client, thus completing the round.

4. Experiments

Overview: Sections 4.1 and 4.2 provide detailed settings and results of the benchmark comparisons. Section 4.3 examines the effects of removing the initial pruning step. The ablation experiments in Sects. 4.4 and 4.5 demonstrate the effectiveness of DD and the positive impact of pre-training rounds. Section 4.6 investigates the impact of data distribution on the loss trend.

4.1 Datasets, Models Setting, and Baseline Methods

In this study, we conduct four separate experiments, each using a well-known benchmark dataset: FEMNIST [25], CIFAR-10 [24], CelebA [26], and ImageNet-100 [13], [27]. FEMNIST, specifically designed for FL, comprises handwritten characters across 62 classes from diverse participants, exemplifying the non-IID data challenge in real-world settings. CIFAR-10, a standard IID dataset in FL, includes 60,000 images across 10 classes. CelebA, employed here for binary classification of the smile attribute, offers a varied collection of non-IID celebrity images. Finally, ImageNet-100, the first hundred classes of the larger ImageNet-1k, serves as a comprehensive and complex dataset for evaluating FL performance.

The four datasets are evaluated using four standard FL models, respectively: Convolutional Networks [25], [28] (Conv-2, Conv-4), VGG-11 [29], and ResNet-18 [30], with experiments detailed provided in Table 1. The performance of our model is compared against five methods: conventional dense FL (baseline) [5], Adaptive [13], Iterative [10], SNIP [21] and FedTiny [16]. The baseline represents the standard, non-pruning FL framework. Adaptive introduces single-client coarse pruning using gradient square information for iterative pruning. Iterative refers to the magnitude-based pruning method proposed by Han *et al.* [10]. SNIP is a one-shot connection sensitivity pruning method.

Due to the specific nature of FedTiny, our comparisons are limited to CIFAR-10 without C.S. and ImageNet-100. All methods, except for the initial client-level pruning step

Table 1 Experiments Setting on Different Datasets

Dataset	FEMNIST	CIFAR-10	CelebA	ImageNet-100
SGD params in round r	LR = 0.25[1]	LR = $0.1 \cdot 0.5 \frac{r}{10000}$	LR = 0.2	LR = $0.05 \cdot 0.5 \left\lfloor \frac{r}{1000} \right\rfloor \cdot 0.1$
Number of clients (Non-C.S., C.S.) [2]	10, -	10, 100	10, -	10, -
Mini-batch size, local iterations in each round	20,5	20,5	20,5	20,5
Total number of FL rounds	10,000	10,000	3,000	20,000
Data Distribute	non-IID	IID	non-IID	IID
Model	Conv-2	VGG-11	Conv-4	ResNet-18
Local Pre-training Rounds	40	40	40	50
Rounds between Pruning	400	400	100	400
SLIC segments	[3,5,10]	[10,15,20]	[10,25,50]	[15,50,80]

[1] LR Stands for Learning Rate.
[2] C.S. stands for Client Selection; Non-C.S. stands for non-Client Selection.

Table 2 Top-1 Test Accuracy with CIFAR-10 on VGG-11, No C.S.

baseline	86.05							
sparsity	**DDPruneFL**	Iterative	Adaptive	SNIP	FedTiny	SynFlow [16]	FedDST [16]	LotteryFL [16]
60.0%	**86.27±0.37**	85.39±0.15	81.52±2.74	85.51±0.22	86.26±0.19	-	-	-
80.0%	**85.71±0.15**	84.69±0.26	80.37±3.38	82.84±0.33	85.68±0.49	-	-	-
90.0%	**85.63±0.27**	83.20±0.13	79.94±3.28	74.59±0.97	85.30±0.42	-	-	-
95.0%	**84.88±0.27**	79.87±0.71	80.42±2.02	25.79±22.33	84.19±0.32	-	-	-
98.0%	**83.23±0.28**	71.84±0.50	77.04±3.41	10.00±0.00	81.66±0.91	-	-	-
99.0%	**81.07±0.18**	65.90±0.88	73.48±3.36	10.00±0.00	78.12±0.69	58.03	60.67	61.83
99.5%	**78.58±0.50**	59.49±1.01	68.04±4.09	10.00±0.00	74.82±0.99	43.76	42.92	43.76

in Adaptive, use server-side aggregation for model pruning.

4.2 Performance Comparison on Different Sparsity

In this section, we compare the performances of our method across four datasets, with using four distinct models, against other methods. All our experiments are conducted in three independent trials. The figures in this section feature solid lines representing the average results of these trials, with shaded areas around these lines indicating upper and lower boundaries. The results in the tables are also displayed as mean and variance.

(1) CIFAR-10

Figure 2 displays a comparison of top-1 test accuracy and loss throughout the complete training process for the VGG-11 model on the CIFAR-10 dataset at 98% global sparsity without C.S. Table 2 details the top-1 test accuracy results at various sparsity levels, ranging from 60% to an extreme of 99.5%.

In Fig. 2, the Adaptive method, indicated by the red line, shows significant accuracy drops. These drops could potentially be attributed to pruning bias arising from the local data distribution. It is noteworthy that SNIP performs poorly at such high sparsity levels, consistent with observations by [18], [39], [45], [46]. This is attributed to the phenomenon of gradient explosion and vanishing experienced by SNIP at extremely high target sparsity, leading to the near-complete pruning of intermediate layer parameters, thus impacting model performance [39], [46]. In contrast, as shown in Table 2, DDPruneFL consistently outperforms other methods across all sparsity levels, achieving high accuracy even at extremely high sparsity.

Additionally, the results of SynFlow [47], FedDST [40], and LotteryFL [15] on the CIFAR-10 dataset are included for comparison in Table 2. When compared to these methods at higher sparsity levels, DDPruneFL's performance is notably superior, indicating that it may have established a new SOTA in this field.

(2) FEMNIST

Table 3 and Fig. 3 present the experimental results of the FEMNIST dataset using the CONV-2 model without C.S. In these experiments, DDPruneFL exhibits stable performance, sustaining high accuracy even at increased sparsity levels.

While the SNIP and Iterative methods show better performance at lower sparsity levels, DDPruneFL distinguishes itself in higher sparsity cases, showcasing its stability under stringent pruning conditions. In Fig. 3, the Adaptive method (indicated by the red line) shows a decline in effectiveness at higher pruning levels, likely due to pruning bias. The consistent effectiveness of DDPruneFL across various sparsity levels further affirms its stability in FL frameworks.

Additionally, our DDPruneFL achieves peak test top-1 accuracy during its training phase before full pruning and then displays a pattern of decline followed by an increase. A similar trend is observed with the iterative method, suggesting that pruned networks require extended training to reach optimal performance [10]. We detail a longer training experiment in Appendix B, Fig. A· 1, where networks pruned using DDPruneFL eventually exceed their initial peak accu-

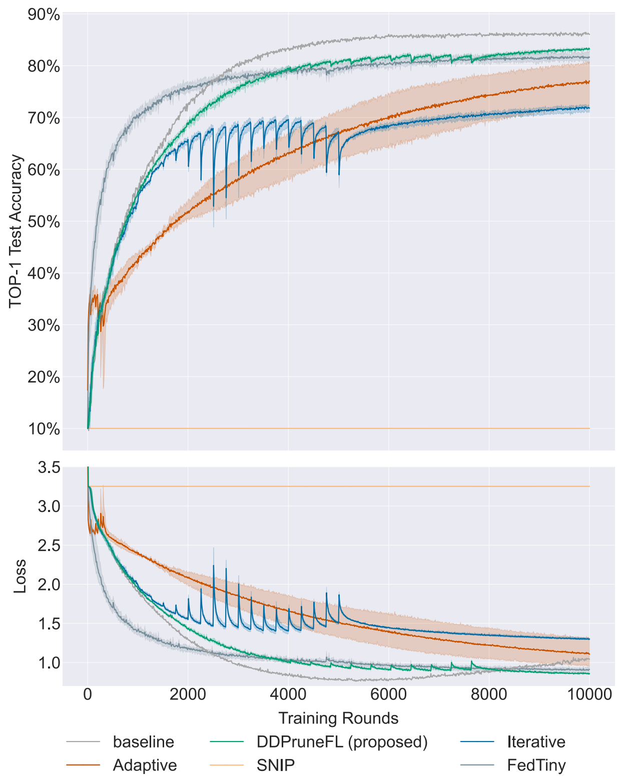

Fig. 2 Test Accuracy and Loss Value with CIFAR-10 on VGG-11 at 98% Sparsity, without Client Selection (C.S.). The solid line is the average of three independent experiments, and the light-colored areas are the up and down boundary. Among these, our DDPruneFL method shows the best outcomes. Notably, Adaptive demonstrates high early-stage performance due to initial pruning from a single client, but it underperforms on the complete training set.

Table 3 Top-1 Test Accuracy with FEMNIST on CONV-2, No C.S.

baseline	84.29			
sparsity	**DDPruneFL**	Iterative	Adaptive	SNIP
60.0%	84.32±0.26	84.28±0.13	83.73±0.19	**84.55±0.22**
80.0%	84.09±0.29	**84.17±0.26**	83.65±0.02	83.96±0.42
90.0%	**84.28±0.11**	83.19±0.13	83.99±0.15	83.76±0.29
95.0%	**83.84±0.16**	82.21±0.22	82.80±0.25	83.33±0.03
98.0%	**83.18±0.48**	79.96±0.08	81.82±0.72	82.69±0.49
99.0%	**81.98±0.38**	77.75±0.44	79.98±0.32	80.72±0.24
99.5%	**80.09±0.34**	73.67±0.46	77.87±0.19	75.22±1.51

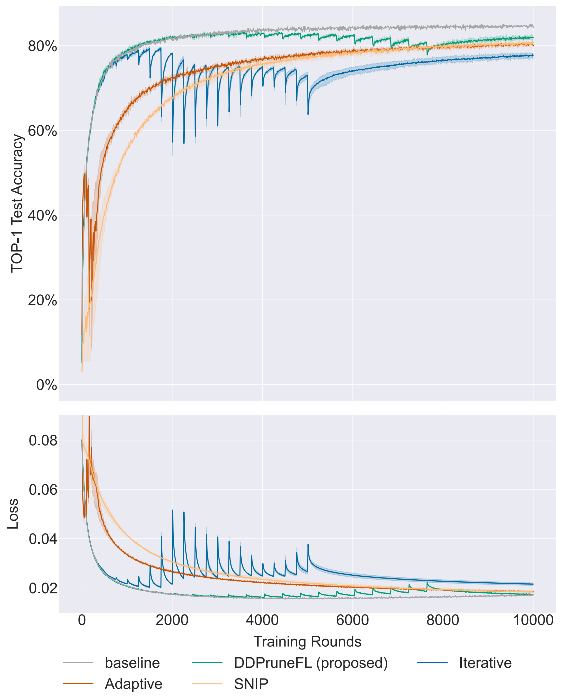

Fig. 3 Test Accuracy and Loss Value with FEMNIST on Conv-2 at 99% Sparsity, No C.S.

racy.

(3) CelebA

Figure 4 and Table 4 show the results for the CelebA dataset in a four-layer convolutional network. The Iterative method outperforms others at sparsity levels below 90%. However, DDPruneFL shows competitive results and excels particularly at higher sparsity levels. Its performance advantage becomes more pronounced as sparsity increases. In Fig. 4, the Adaptive (red line) initially shows high accuracy with its coarse, single-client dataset-based pruning. However, this accuracy declines over subsequent training rounds.

(4) ImageNet-100

Figure 5 and Table 5 present the results on the ImageNet-100 dataset using a ResNet-18 model, focusing on final top-1 accuracy values at model sparsity levels above 90%. DDPruneFL consistently outperforms other methods across these sparsity levels, with its advantage becoming more pronounced at higher levels of sparsity.

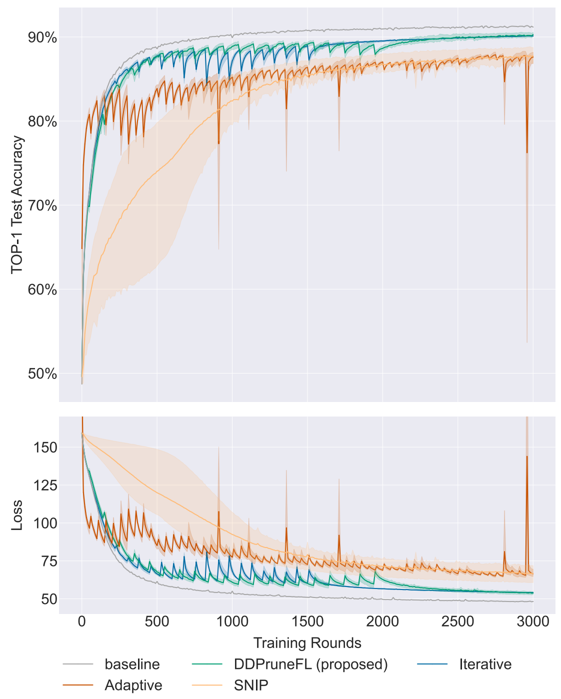

Fig. 4 Test Accuracy and Loss Value with CelebA on Conv-4 at 95% Sparsity, No C.S.

Table 4 Top-1 Test Accuracy with CelebA on CONV-4, No C.S.

baseline	91.17			
sparsity	**DDPruneFL**	Iterative	Adaptive	SNIP
60.0%	91.00±0.13	**91.04±0.07**	90.01±0.08	90.90±0.10
80.0%	90.79±0.09	**90.89±0.05**	89.80±0.12	90.61±0.11
90.0%	90.46±0.04	**90.72±0.08**	88.59±0.23	89.79±0.06
95.0%	**90.28±0.21**	90.12±0.03	87.60±0.58	87.75±0.76
98.0%	**89.10±0.26**	88.64±0.14	87.00±0.19	51.29±0.00
99.0%	**87.76±0.17**	83.55±0.80	85.76±0.39	51.29±0.00
99.5%	**82.83±2.31**	79.76±2.93	76.39±9.06	51.29±0.00

(5) CIFAR-10, with Client Selection

We also compared FL cases with C.S., using the CIFAR-10 dataset with up to 100 clients. Each client contains 500 training images, 1% of the training set. During each training round, the framework randomly selects 10 clients for model training. The results are shown in Fig. 6 and Table 6.

Similar to the trends observed without C.S., our DDPruneFL method consistently outperforms others in all compared sparsity-level cases. At lower sparsity levels, our approach even surpasses the baseline performance, demonstrating the stability and superiority of our method in accuracy. DDPruneFL consistently surpasses other methods across all sparsity levels tested. Notably, at lower sparsity levels, our method even exceeds baseline performance [36], underscoring the stability and effectiveness of DDPruneFL in maintaining accuracy.

Table 5 Top-1 Test Accuracy with ImageNet100 on ResNet-18, No C.S.

baseline	71.34				
sparsity	**DDPruneFL**	Iterative	Adaptive	SNIP	FedTiny
90.0%	**71.11±0.19**	69.12±0.32	69.31±0.27	69.00±0.17	67.65±0.51
95.0%	**69.29±0.26**	67.61±0.38	64.43±0.19	66.64±0.35	63.51±0.48
98.0%	**65.34±0.96**	62.94±0.29	54.70±0.54	60.64±0.24	56.80±0.53
99.0%	**62.14±0.40**	58.19±0.78	50.80±0.18	54.71±0.18	52.50±0.17
99.5%	**57.16±0.42**	51.98±0.18	49.17±0.36	48.21±0.62	48.02±0.25

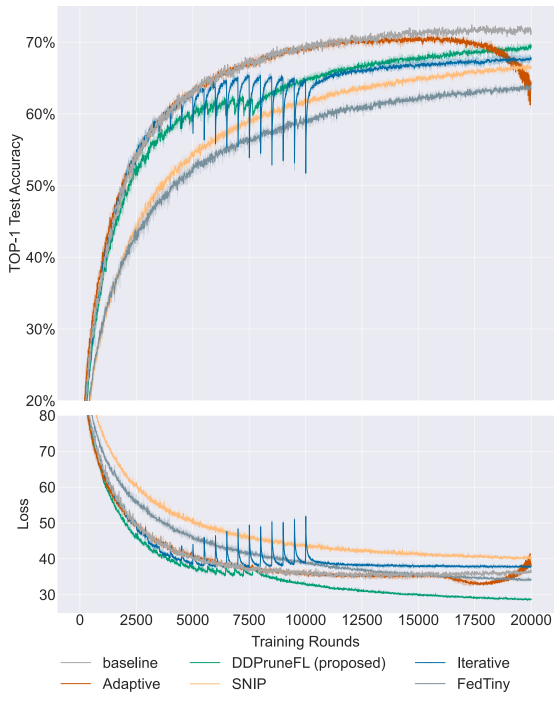

Fig. 5 Test Accuracy and Loss Value with ImageNet100 on ResNet-18 at 95% Sparsity, No C.S.

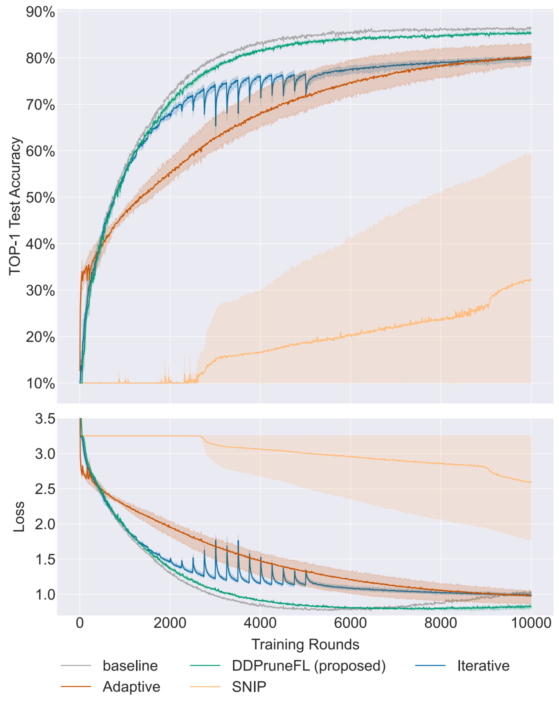

Fig. 6 Test Accuracy and Loss Value with CIFAR-10 on VGG-11 at 95% Sparsity, C.S.

Table 6 Top-1 Test Accuracy with CIFAR-10 on VGG-11, C.S.

baseline	86.44			
sparsity	**DDPruneFL**	Iterative	Adaptive	SNIP
60.0%	**86.60±0.21**	85.41±0.14	82.03±2.14	85.40±0.14
80.0%	**86.27±0.16**	84.86±0.19	81.31±3.15	82.83±0.47
90.0%	**85.74±0.23**	83.16±0.28	81.53±2.35	74.11±1.66
95.0%	**85.36±0.32**	79.90±0.43	80.28±1.96	32.42±20.46
98.0%	**84.19±0.33**	72.28±0.57	77.17±1.43	10.00±0.00
99.0%	**82.86±0.28**	65.97±0.66	74.86±1.68	10.00±0.00
99.5%	**80.56±0.27**	59.41±1.72	69.43±0.91	10.00±0.00

Table 7 Impact of Pruning Bias in CIFAR-10

sparsity	98%	98.4239%	99%
DDPruneFL	83.23±0.28	82.8	81.07±0.18
Adaptive (no init pruning)	-	80.04	-
Adaptive (with init pruning)	77.04±3.41	-	73.48±3.36

4.3 Ablation Experiments: Mitigate Bias by Removing Initial Pruning

We conduct a comparative analysis of our method against the Adaptive method in scenarios without an initial pruning strategy. This comparison aimed to ascertain if removing the coarse pruning could eliminate bias and enhance performance at high sparsity levels. The results are detailed in Table 7. It is crucial to note that due to its systematic settings, the Adaptive method, utilizing threshold-based pruning methods, cannot directly assign a precise final model sparsity when initial pruning is removed. To ensure a fair comparison in our study, we matched our method's final sparsity level to that achieved by the Adaptive method, 98.4239%.

At a global sparsity of 98.4239%, DDPruneFL attained a top-1 accuracy of 82.8%, while the Adaptive method without initial pruning achieved 80.04%. The performance of Adaptive improves at very high sparsity levels when initial pruning is removed. This suggests that at such high sparsity levels, eliminating coarse pruning, which often relies on a single client, can enhance pruning outcomes.

In Fig. 7, we further analyze the progression of sparsity with training rounds and top-1 test accuracy between DDPruneFL and the Adaptive method without initial pruning, targeting a final global sparsity of 98.4239%. In this comparison, the dotted green line representing DDPruneFL consistently exhibits higher accuracy than the Adaptive method's red line. Notably, Adaptive maintains higher spar-

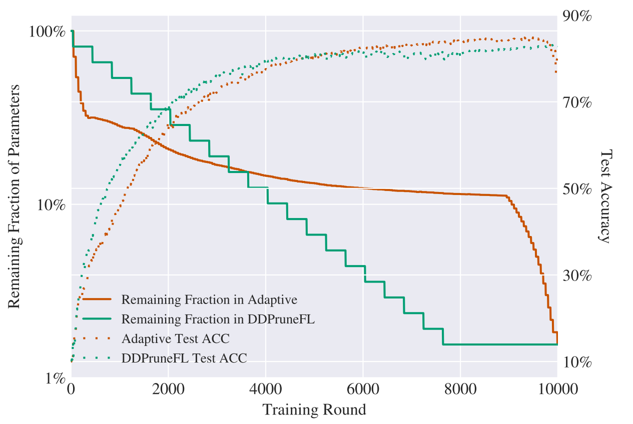

Fig. 7 Comparison of Model Sparsity and Test Top-1 ACC during Training at a Target Sparsity of 98.4239%. Adaptive without initial pruning maintained a higher sparsity for a longer duration compared to the proposed DDPruneFL, yet it did not achieve superior model performance. While Adaptive without initial pruning mitigates pruning bias to some extent, DDPruneFL demonstrates a more rational and effective design in its pruning schedule.

Table 8 Top-1 Test ACC Different Data Source at 90% Sparsity with CIFAR-10 on VGG-11

Methods	ACC
SNIP (one-shot)	74.59±0.97
DDPruneFL(one-shot)	79.65
Adaptive (iterative pruning)	79.94±3.28
DDPruneFL (iterative pruning)	**85.63±0.27**

sity for an extended period compared to DDPruneFL. The Adaptive approach resembles a post-training pruning strategy, heavily relying on intensive pruning during the final stages to reach the predetermined sparsity level. These results illustrate that our DDPruneFL method is more efficient in attaining the desired fixed sparsity in the final model, showcasing its superior pruning efficacy.

4.4 Ablation Experiments: Effectiveness of DD

In this ablation experiment with CIFAR-10 on VGG-11, we demonstrate that DD, even when extracted in significantly fewer rounds than standard, proves to be highly effective in FL pruning. This high efficiency is evident in both one-shot and iterative pruning methods.

In line with the pruning criteria of SNIP, we implement a one-shot pruning strategy. For this, we extract DD from a locally pre-trained model after just pre-training 67 rounds, at which point the training data accuracy had reached near-perfect levels (99.9%). This early extraction requires significantly fewer rounds than the standard 10,000 typically needed for full training [17].

As detailed in Table 8, the results demonstrate that our model, utilizing DD, surpasses the original SNIP in performance. Remarkably, even in a one-shot scenario, our method closely matches the performance levels of the Adaptive method, which is an iterative method. Furthermore, when iterative pruning is incorporated into our method, there is a notable enhancement in model performance. These outcomes highlight the considerable potential of using DD in FL pruning methods.

Table 9 Ablation Experiments on Pre-training Rounds with CIFAR-10 on VGG-11 at 99% Sparsity

Pre-train Rounds	10	20	30	40	50
Final Shot Test ACC	77.04	77.28	76.76	77.39	77.62
Final Shot Test Loss	1.202	1.172	1.212	1.165	1.161
Final Test ACC	80.72	80.77	81	81.33	81.68

4.5 Ablation Experiments: Pre-Training Rounds

In this ablation study, we show that increasing the number of pre-training rounds positively impacts DDPruneFL's performance. We vary the pre-training rounds within $[10, 20, 30, 40, 50]$ on CIFAR-10 at a 99% global sparsity. More than 50 rounds often lead to near-perfect training data accuracy, making additional rounds redundant. Too few pre-training rounds can lead to insufficient learning in the local model, hindering effective concept activation extraction.

We assess the impact of pre-training rounds on three metrics: test accuracy at the final shot, test loss at the final shot, and the final test accuracy of the model, with results presented in Table 9. It shows that the final accuracy exhibited an increasing total trend with more pretraining rounds. It can also be observed that our choice of 40 rounds for pre-training in the primary experiments represents a well-considered balance between cost and efficacy. This also aligns with our objective to keep pre-training limited to under one epoch (all the training data is used at once).

Furthermore, this ablation study shows a positive impact of pre-training rounds on pruning performance, thereby reinforcing the findings in Sect. 4.4. It confirms that DD performs well even when extracted from far fewer rounds than typically required, supporting our assertions of its effectiveness.

4.6 Convergence Analysis on non-IID Data

To analyze the convergence of our DDPruneFL method on non-IID data, we conducted ablation experiments using the CelebA dataset, selected for its non-IID distribution, to investigate loss trends under different conditions including data imbalance and varying numbers of clients. We introduced the Imbalance Ratio (IR), a metric reflecting the proportion of the majority class at each client. For instance, IR = 0.5 indicates an equal distribution of samples for both labels at each client, while IR = 1 signifies that all data at a client belong to a single label, either ‘smiling’ or ‘not smiling’. Experiments are performed with IR values of [0.5, 0.8, 1], corresponding to balanced, moderately imbalanced, and completely imbalanced data distributions, respectively.

Additionally, the impact of varying the number of clients [10, 20, 30] is assessed, with results indicating that while data imbalance significantly affects loss trends, changes in the number of clients minimally influence model

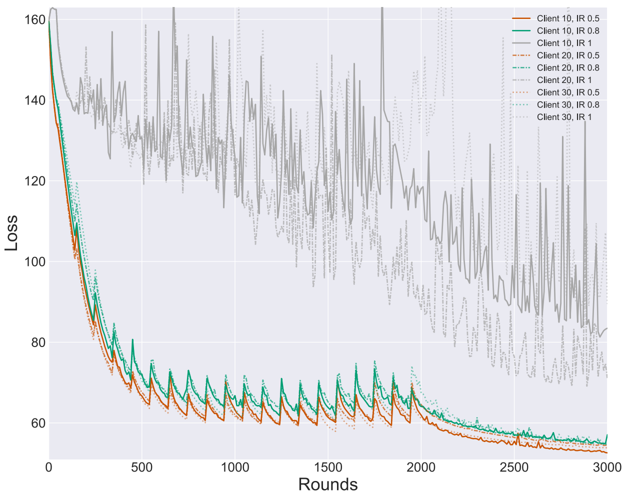

Fig. 8 The loss values on CelebA at 95% global sparsity with varying numbers of clients and different Imbalance Ratios (IR). As we utilize only two label classes in CelebA, 'smiling' and 'not smiling'. We define IR as the percentage of samples in the predominant class within the current client. IR of [0.5, 0.8, 1] are displayed in orange, green, and grey lines. The number of clients [10, 20, 30] on training are represented by solid, dashed, and dotted lines.

performance. Optimal model fitting is observed at an IR of 0.5, showing that balanced data distribution enhances the effectiveness of DDPruneFL. These results demonstrate that data distribution plays a crucial role in the performance of our method, as illustrated in Fig. 8.

At low sparsity, the network retains enough parameters to mitigate the shortcomings of traditional FL pruning methods, all performing near baseline levels without significant advantages on non-IID datasets due to the efficiency of existing criteria. At high sparsity, traditional methods often face performance issues like pruning bias. Our method diverges by utilizing DD to improve data aggregation, effectively extracting DD from client datasets, thereby addressing the prevalent issues in high-sparsity FL pruning on non-IID data.

5. Computational Cost Analysis

In this section, we analyze the resource cost of our algorithm on the client side using the Floating-point Operations (FLOPs) method [33], [48]. Section 5.1 provides an analysis of the total computational training cost, and Sect. 5.2 presents a comparison of the costs during inference. For the FLOPs count for each step of the training process computation, we detail in Appendix A.1–A.4.

FLOP analysis, being independent of specific hardware and software configurations, provides a direct and unbiased measure for comparing different models and assessing computational efficiency [49]. We use the training of a VGG-11 model on the CIFAR-10 dataset as an example to fully demonstrate the resource consumption of our algorithm. Specifically, in the CIFAR-10 FL experiment, there are 10 clients, each possessing one-tenth of the training set, which equates to 5,000 images. In FLOP analysis, each addition, multiplication, or comparison operation is counted as one FLOP. We analyze the FLOPs required for each step in our algorithm, particularly focusing on the extraction of DD, which is the primary cost part.

5.1 Total FLOPs for Extracting DD

Our method for extracting DD involves the following four steps: (1) Generating superpixels using the SLIC algorithm; (2) Propagating data through the local neural network and capturing gradients at the bottleneck layer; (3) Clustering activations with k-means; (4) Applying a linear classifier to distinguish between random sample activations and concept activations, and subsequently calculating CAV scores.

By aggregating all of the algorithm steps, we conclude that for a single category of DD extraction in a CIFAR-10 dataset within a VGG-11 FL framework, a total of approximately 0.377 TFLOPs is required per client. If all clients extract DD for all categories, the total computational cost would amount to 3.77 TFLOPs. Notably, in reality, since the initial dataset distribution varies across clients, not all can extract DD for every category. As a result, the actual FLOPs required are significantly less than this worst-case estimate.

In our calculations, we omit the smaller-scale computational processes within the algorithm, such as the T-test procedure, which typically only accounts for tens of thousands of FLOPs. Based on the formula in [13], we estimate the training FLOPs to be approximately three times that of the inference FLOPs. In this worst-case scenario, additional computational cost (cost for extracting DD) amounts to merely 0.45% of the total FLOPs required for local training.

5.2 Compare Model Inference FLOPs

In this section, we compare the inference FLOPs required by client-side models for different methods when fine-tuned to the same extent.

For the computation of FLOPs, we adhere to the calculation method described in [13], [50]. After complete fine-tuning, models achieve 83% test top-1 accuracy, corresponding to approximately 86.05% of the baseline performance. We visualized the FLOPs, as illustrated in Fig. 9. In the figure, the transition from dark to light colors represents the different FLOPs required from the initial convolutional layer to the final fully connected layer of the model. The percentages next to the method names on the left show the model's sparsity. Importantly, DDPruneFL requires the fewest inference FLOPs while maintaining performance comparable to the baseline. This indicates that with a model at 98% sparsity, we can achieve performance that is only 3% below the baseline. This indicates that models pruned with DDPruneFL, when deployed on edge devices, can achieve enhanced performance at a lower inference cost.

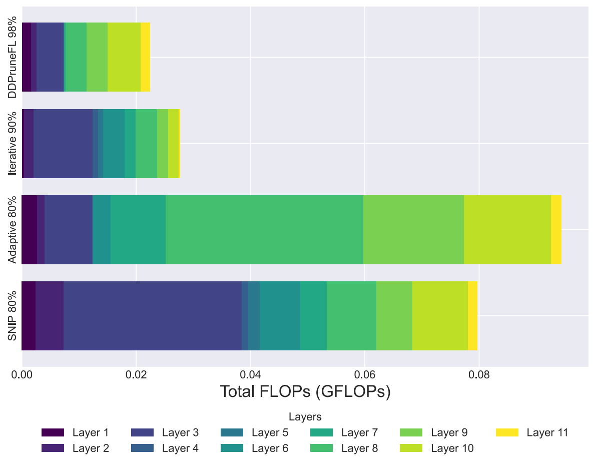

Fig. 9 Comparison of VGG-11 Model Inference FLOPs when the Fully Fine-tuned Model Attains a Test Accuracy within 3% below the Baseline

6. Conclusion

In summary, our research contributes significantly to the field of FL and Network Pruning. We demonstrated that single-client-based pruning results in bias, and our multi-client DD approach effectively counter this issue. This method sets a new SOTA in the FL pruning framework, enhancing network performance and efficiency. Empirical evaluations across diverse networks and datasets validate our method's superiority over existing techniques. Notably, it shows robustness against variations in pruning schedules and maintains efficiency, as evidenced by a marginal increase in computational costs. Overall, our work advances the understanding of network pruning in FL, offering a viable solution for efficient, privacy-preserving machine learning in resource-limited environments.

References

[1] S. Wang, C. Li, R. Wang, Z. Liu, M. Wang, H. Tan, Y. Wu, X. Liu, H. Sun, R. Yang, X. Liu, J. Chen, H. Zhou, I. Ben Ayed, and H. Zheng, "Annotation-efficient deep learning for automatic medical image segmentation," Nature communications, vol.12, no.1, p.5915, 2021.

[2] J. Wang, S. Guo, X. Xie, and H. Qi, "Protect privacy from gradient leakage attack in federated learning," IEEE INFOCOM 2022 - IEEE Conf. Computer Communications, vol.00, pp.580–589, 2022.

[3] D.C. Nguyen, Q.-V. Pham, P.N. Pathirana, M. Ding, A. Seneviratne, Z. Lin, O. Dobre, and W.-J. Hwang, "Federated learning for smart healthcare: A survey," ACM Computing Surveys (CSUR), vol.55, no.3, pp.1–37, 2022.

[4] R.C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," 2018.

[5] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B.A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," Artificial intelligence and statistics, pp.1273–1282, PMLR, 2017.

[6] V. Mothukuri, R.M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," Future Generation Computer Systems, vol.115, pp.619–640, 2021.

[7] N. Rieke, J. Hancox, W. Li, F. Milletarì, H.R. Roth, S. Albarqouni, S. Bakas, M.N. Galtier, B.A. Landman, K. Maier-Hein, S. Ourselin, M. Sheller, R.M. Summers, A. Trask, D. Xu, M. Baust, and M.J. Cardoso, "The future of digital health with federated learning," NPJ digital medicine, vol.3, no.1, p.119, 2020.

[8] N. Onoszko, G. Karlsson, O. Mogren, and E.L. Zec, "Decentralized federated learning of deep neural networks on non-iid data," 2021.

[9] N. Shibahara, M. Koibuchi, and H. Matsutani, "Performance improvement of federated learning server using smart nic," Proc. 11th International Symposium on Computing and Networking (CANDAR' 23) Workshops, The 6th Sustainable Computing Systems Workshop (SUSCW' 23), p.xxx–xxx, 2023.

[10] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," Advances in neural information processing systems, vol.28, 2015.

[11] H. Wang, C. Qin, Y. Bai, Y. Zhang, and Y. Fu, "Recent advances on neural network pruning at initialization," Proc. Thirty-First Int. Joint Conf. Artificial Intelligence, IJCAI-22, ed. L.D. Raedt, pp.5638–5645, Int. Joint Conf. Artificial Intelligence Organization, 7 2022. Survey Track.

[12] W. Xu, W. Fang, Y. Ding, M. Zou, and N. Xiong, "Accelerating federated learning for iot in big data analytics with pruning, quantization and selective updating," IEEE Access, vol.9, pp.38457–38466, 2021.

[13] Y. Jiang, S. Wang, V. Valls, B.J. Ko, W.H. Lee, K.K. Leung, L. Tassiulas, and Y. Jiang, "Model Pruning Enables Efficient Federated Learning on Edge Devices," IEEE Trans. Neural Networks and Learning Systems, vol.PP, no.99, pp.1–13, 2022.

[14] M.T. Munir, M.M. Saeed, M. Ali, Z.A. Qazi, and I.A. Qazi, "Fedprune: Towards inclusive federated learning," 2021.

[15] A. Li, J. Sun, B. Wang, L. Duan, S. Li, Y. Chen, and H. Li, "Lotteryfl: Empower edge intelligence with personalized and communication-efficient federated learning," 2021 IEEE/ACM Symposium on Edge Computing (SEC), pp.68–79, IEEE, 2021.

[16] H. Huang, L. Zhang, C. Sun, R. Fang, X. Yuan, and D. Wu, "Distributed Pruning Towards Tiny Neural Networks in Federated Learning," 2023 IEEE 43rd Int. Conf. Distributed Computing Systems (ICDCS), vol.00, pp.190–201, 2023.

[17] Y. Yang, Y. Wang, Y. Ji, H. Qi, and J. Kato, "One-shot network pruning at initialization with discriminative image patches," 33rd British Machine Vision Conference 2022, BMVC 2022, London, UK, November 21-24, 2022, BMVA Press, 2022.

[18] J. Frankle, G.K. Dziugaite, D. Roy, and M. Carbin, "Pruning neural networks at initialization: Why are we missing the mark?," Int. Conf. Learning Representations, 2021.

[19] A. Imteaj, U. Thakker, S. Wang, J. Li, and M.H. Amini, "A survey on federated learning for resource-constrained iot devices," IEEE Internet of Things Journal, vol.9, no.1, pp.1–24, 2022.

[20] A. Ghorbani, J. Wexler, J.Y. Zou, and B. Kim, "Towards automatic concept-based explanations," Advances in neural information processing systems, vol.32, 2019.

[21] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single-shot Network Pruning based on Connection Sensitivity," Int. Conf. Learning Representations, 2019.

[22] M.C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," Advances in neural information processing systems, pp.107–115, 1989.

[23] P. de Jorge, A. Sanyal, H. Behl, P. Torr, G. Rogez, and P.K. Dokania, "Progressive skeletonization: Trimming more fat from a network at initialization," Int. Conf. Learning Representations, 2021.

[24] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," Master's thesis, University of Toronto, Toronto, Canada, 2009.

[25] S. Caldas, S.M.K. Duddu, P. Wu, T. Li, J. Konečný, H.B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," 2019.

[26] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," Proc. IEEE Int. Conf. Computer Vision (ICCV), De-

cember 2015.
[27] J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," 2009 IEEE Conf. Computer Vision and Pattern Recognition, pp.248–255, 2009.
[28] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "Imagenet classification with deep convolutional neural networks," Advances in neural information processing systems, vol.25, 2012.
[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 3rd Int. Conf. Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, ed. Y. Bengio and Y. LeCun, 2015.
[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp.770–778, 2016.
[31] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," CoRR, vol.abs/1806.00582, 2018.
[32] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," Proc. Machine Learning and Systems, ed. A. Talwalkar, V. Smith, and M. Zaharia, pp.374–388, 2019.
[33] I.S. Dhillon and D.S. Modha, "A data-clustering algorithm on distributed memory multiprocessors," in Large-scale parallel data mining, pp.245–260, Springer, 2002.
[34] R. Reed, "Pruning algorithms-a survey," IEEE Trans. Neural Networks, vol.4, no.5, p.740–747, 1993.
[35] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," Advances in Neural Information Processing Systems, ed. D. Touretzky, Morgan-Kaufmann, 1989.
[36] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," Int. Conf. Learning Representations, 2019.
[37] T. Chen, J. Frankle, S. Chang, S. Liu, Y. Zhang, M. Carbin, and Z. Wang, "The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models," Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR), pp.16306–16316, June 2021.
[38] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," Int. Conf. Learning Representations, 2020.
[39] H. Tanaka, D. Kunin, D.L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," Advances in neural information processing systems, vol.33, pp.6377–6389, 2020.
[40] S. Bibikar, H. Vikalo, Z. Wang, and X. Chen, "Federated dynamic sparse training: Computing less, communicating less, yet learning better," Proc. AAAI Conf. Artificial Intelligence, vol.36, no.6, pp.6080–6088, Jun. 2022.
[41] D. Stripelis, U. Gupta, G.V. Steeg, and J.L. Ambite, "Federated progressive sparsification (purge-merge-tune)+," Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022), 2022.
[42] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," Int. Conf. Learning Representations, 2020.
[43] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels compared to state-of-the-art superpixel methods," IEEE Trans. Pattern Analysis and Machine Intelligence, vol.34, no.11, pp.2274–2282, 2012.
[44] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, and R. sayres, "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV)," Proc. 35th Int. Conf. Machine Learning, ed. J. Dy and A. Krause, Proc. Machine Learning Research, vol.80, pp.2668–2677, PMLR, 10–15 Jul 2018.
[45] J. Su, Y. Chen, T. Cai, T. Wu, R. Gao, L. Wang, and J.D. Lee, "Sanity-checking pruning methods: Random tickets can win the jackpot," Advances in neural information processing systems, vol.33, pp.20390–20401, 2020.
[46] S. Hayou, J.F. Ton, A. Doucet, and Y.W. Teh, "Robust pruning at initialization," Int. Conf. Learning Representations, 2021.
[47] H. Tanaka, D. Kunin, D.L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," Advances in neural information processing systems, vol.33, pp.6377–6389, 2020.
[48] R. Tang, A. Adhikari, and J. Lin, "Flops as a direct optimization objective for learning sparse neural networks," 2018.
[49] Y. Zhou, X. Lin, X. Zhang, M. Wang, G. Jiang, H. Lu, Y. Wu, K. Zhang, Z. Yang, K. Wang, Y. Sui, F. Jia, Z. Tang, Y. Zhao, H. Zhang, T. Yang, W. Chen, Y. Mao, Y. Li, D. Bao, Y. Li, H. Liao, T. Liu, J. Liu, J. Guo, X. Zhao, Y. WEI, H. Qian, Q. Liu, X. Wang, W. Kin, Chan, C. Li, Y. Li, S. Yang, J. Yan, C. Mou, S. Han, W. Jin, G. Zhang, and X. Zeng, "On the opportunities of green computing: A survey," 2023.
[50] L. Zhu, "Thop: Pytorch-opcounter." https://github.com/Lyken17/pytorch-OpCounter.
[51] X. Wang, G. Li, A. Plaza, and Y. He, "Revisiting slic: Fast superpixel segmentation of marine sar images using density features," IEEE Trans. Geoscience and Remote Sensing, vol.60, p.1–18, 2022.
[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol.12, pp.2825–2830, 2011.
[53] J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos, "Compute trends across three eras of machine learning," 2022 International Joint Conf. Neural Networks (IJCNN), pp.1–8, 2022.
[54] H. Dong, B. Yu, W. Wu, and C. He, "Enhanced lightweight end-to-end semantic segmentation for high-resolution remote sensing images," IEEE Access, vol.10, pp.70947–70954, 2022.
[55] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu, "An efficient k-means clustering algorithm: analysis and implementation," IEEE Trans. Pattern Analysis and Machine Intelligence, vol.24, no.7, pp.881–892, 2002.
[56] A. Sayar and F.T.Y. Vural, "Image annotation with semi-supervised clustering," 2009 24th International Symposium on Computer and Information Sciences, pp.12–17, IEEE, 2009.
[57] J. Tian, L. Zhu, S. Zhang, and L. Liu, "Improvement and parallelism of k-means clustering algorithm," Tsinghua Science and Technology, vol.10, no.3, pp.277–281, 2005.
[58] H. Sabit, A. Al-Anbuky, and H. Gholamhosseini, "Data stream mining for wireless sensor networks environment: energy efficient fuzzy clustering algorithm," International Journal of Autonomous and Adaptive Communications Systems, vol.4, no.4, pp.383–397, 2011.
[59] H. Sabit, Distributed incremental data stream mining for wireless sensor network, Ph.D. thesis, Auckland University of Technology, 2012.
[60] T. Kucukyilmaz, U. of Turkish Aeronautical Association, *et al.*, "Parallel k-means algorithm for shared memory multiprocessors," Journal of computer and communications, vol.2, no.11, p.15, 2014.

Appendix A: The Computational Cost in Extracting Discriminative Data

In the following sections, we analyze the FLOP count for each step and then provide an estimate of the total computational cost involved.

A.1 SLIC for Superpixel Generation

SLIC is an adaptation of k-means, specifically developed for superpixel generation, and it limits its search space to a region proportional to the superpixel size [43]. SLIC involves five main steps: (1) Uniformly initializing K_{SLIC} centroids in the image; (2) Calculating pixel-to-centroid distances in a five-dimensional space, which includes color space and spatial dimensions; (3) Assigning pixels to the nearest centroids; (4) Updating centroid locations; (5) Iterating steps two to four for a set number of rounds.

SLIC, unlike traditional k-means, calculates distances to only up to 8 neighboring centroids [43]. To prevent underestimation in the FLOP analysis, we equate SLIC's computational complexity with k-means. This assumption involves calculating distances between each pixel and all centroids, representing a worst-case scenario for SLIC. It is crucial to recognize that the actual FLOP count in practical applications of SLIC is significantly lower than our estimate [51], [52]. This indicates that the computational efficiency observed in our method is indeed a realistic assessment.

In the CIFAR-10 experiment, after completing pretraining, each client collects a maximum of 20 images per category, in $\mathbb{R}^{\gamma \times w \times h}$, where $\gamma = 3$ corresponds to the number of color space, $w = 32$, and $h = 32$. $D_{\text{pos}} = 2$ represents the number of spatial dimensions. These images are segmented into 10, 15, or 20 parts using the SLIC.

The computational approach adopted in SLIC is fundamentally based on the k-means methodology. A detailed explanation of the k-means computation is provided in Sect. A.3, as detailed in equation (A· 2). During the SLIC segmentation process, the computational cost for K_{SLIC} segmentations, expressed in FLOPs, is calculated as follows:

$$\begin{aligned}\text{FLOP}_{\text{SLIC}} = T_{\text{SLIC}} \Bigg\{ & \overbrace{(w \times h) K_{\text{SLIC}} \left[3\left(\gamma + D_{\text{pos}}\right) + 1\right]}^{\text{Distance Calculation}} \\ & \underbrace{+ K_{\text{SLIC}} \left(\gamma + D_{\text{pos}}\right) \left[\frac{w \times h}{K_{\text{SLIC}}} + 1\right]}_{\text{Centroid Updating}} \Bigg\} \end{aligned} \tag{A· 1}$$

where T_{SLIC} is the cluster iterations and is set to a default value of 10.

In this formula, the term before the addition accounts for the FLOPs required for distance calculation, while the term following represents updating centroids. We omit the FLOPs for centroid initialization and pixel-to-centroid assignment, as these processes typically require minimal computational resources. Thus, we could calculate that for each client, the computational cost during the SLIC stage amounts to 0.15 gigaFLOPs (GFLOPs).

A.2 Activations and Gradients in the Euclidean Space

We adopt the same VGG-11 structure as [13], featuring three fully connected layers at the end as a classifier. We designate the layer preceding the classifier as the Euclidean space for feature mapping. According to [50], [53], [54], for VGG-11, processing an image of size $(3, 32, 32)$ requires 2.76549632×10^8 FLOPs. In this phase, 900 segmented image patches are processed through VGG-11. Additionally, 200 random images are also propagated through the neural network to classify identical semantic features within the Euclidean space. Our calculations also include the backpropagation of gradients for activations in the bottleneck layer. Therefore, the computational cost for this step is approximately 0.305 teraFLOPs (TFLOPs).

A.3 Clustering Activation with k-means

One of the most popular unsupervised clustering methods, k-means, relies on a predefined number of clusters K_{kmeans} to execute its algorithm [55]. The sequential implementation of the k-means algorithm is divided into the following five steps: (1) Initialization, where K_{kmeans} initial centroids are selected; (2) Calculation of the distance between N_{feature} data points and each centroid, with each data point having a dimension of D_{feature}; (3) Assignment of each data point to the nearest centroid, forming K_{kmeans} clusters; (4) Calculation of new centroids for each cluster; (5) Iteration of steps two to four until the centroids no longer change significantly or a preset number of iterations T_{kmeans} is reached. If the computational cost of each iteration is constant, then the main computational cost primarily depends on the distance calculation and centroid updating.

In our analysis, we calculate the total computational complexity of sequentially executing the k-means algorithm following [33]. This methodology is also widely referenced and applied in other works, such as [56]–[59].

For a data point in a D_{feature} space, calculating its Euclidean distance to a centroid typically involves the following number of operations: D_{feature} subtractions, D_{feature} multiplications, $D_{\text{feature}} - 1$ additions, and 1 square root operation. Additionally, identifying the nearest centroid to a data point would require K_{kmeans} comparison operations. Therefore, the computational cost for the distance calculation step should be $N_{\text{feature}} K_{\text{kmeans}} (3D_{\text{feature}} + 1)$ FLOPs. In the centroid updating step, the calculation of a new centroid involves computing the mean of all dimensions for all points in a cluster. Assuming there are N_{feature} points in a cluster, the computation of the average value for a single dimension would include $N_{\text{feature}} - 1$ additions, 1 division, and 1 reassignment operation, totaling $K_{\text{kmeans}} D_{\text{feature}} (N_{\text{feature}} / K_{\text{kmeans}} + 1)$ FLOPs.

By summing all the calculated FLOPs, the total computational cost required to sequentially execute the k-means algorithm is

$$\mathrm{FLOP}_{\mathrm{kmeans}} = T_{\mathrm{kmeans}} \Bigg[\overbrace{N_{\mathrm{feature}} K_{\mathrm{kmeans}} (3D_{\mathrm{feature}} + 1)}^{\text{Distance Calculation}} + \underbrace{K_{\mathrm{kmeans}} D_{\mathrm{feature}} (N_{\mathrm{feature}} / K_{\mathrm{kmeans}} + 1)}_{\text{Centroid Updating}} \Bigg]. \tag{A· 2}$$

More details can be confirmed in [33]. Some relevant studies optimize distance calculations [60], however, as a worst-case analysis, we use the more complex computation formula.

In our implements, the vector dimension of the activations within the Euclidean space is $D_{\mathrm{feature}} = \Gamma = 512$. The number of activations N_{feature} is 900. The number of cluster centroids for k-means, K_{kmeans}, is set to a default of 25. The clustering iterations T_{kmeans}, is 300. Hence, during the k-means clustering step, our algorithm incurs a total of 10.52 GFLOPs.

A.4 Computing Concept Activation Vectors (CAVs)

We employ a linear classifier within the Euclidean space, separating activations from the semantically similar cluster and random samples. For each classification, the total number of activations from both types of samples is $N_{\mathrm{linear}} = 40$. Consequently, the FLOPs consumed in calculating the Concept Activation Vectors (CAVs) for each iteration can be expressed using the formula [13],

$$\mathrm{FLOP}_{\mathrm{CAV}} = n \times T_{\mathrm{linear}} \times 2N_{\mathrm{linear}} \underbrace{\times D_{\mathrm{feature}}}_{\text{in}} \underbrace{\times 2}_{\text{out}} \underbrace{\times 3}_{\text{forward, backward, update}}, \tag{A· 3}$$

where times 3 implies the process of forward pass, backward pass, and parameter updating. Assuming we have $K_{\mathrm{kmeans}} = 25$ clusters of semantically similar activations, each cluster will be classified with random samples $n = 10$ times. The number of iterations for the linear classification, T_{linear}, is default set to 1000. Thus, during the computing CAVs step, our algorithm incurs 61.44 GFLOPs.

Appendix B: Longer Training

Our extended experiments show that networks pruned via DDPruneFL require a longer period to achieve their best performance. Figure A· 1 demonstrates that extending the training duration to 20,000 rounds leads to continued improvement in both loss and test accuracy, with the latter surpassing its initial peak at around 16,000 rounds. This finding highlights the potential of the DDPruneFL method when allowed additional training rounds.

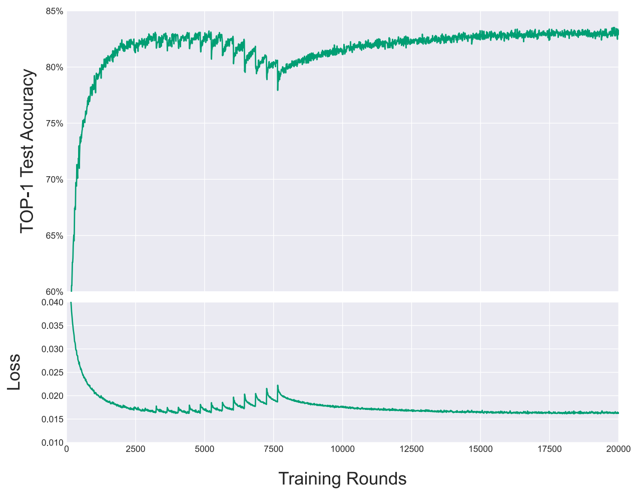

Fig. A· 1 Test Top-1 ACC and Loss of longer training duration to 20,000 rounds on FEMNIST

Yinan Yang received the MSc degree in Computer Science, from the University of Bristol in 2020. He is currently a Ph.D. Candidate with the Graduate School of Information Science and Engineering, Ritsumeikan University.