FedCD: A Hybrid Federated Learning Framework for Efficient Training With IoT Devices

Jianchun Liu, *Member, IEEE*, Yujia Huo, Pengcheng Qu, Sun Xu, Zhi Liu, *Senior Member, IEEE*, Qianpiao Ma, and Jinyang Huang, *Member, IEEE*

***Abstract*—With billions of Internet of Things devices producing vast data globally, privacy and efficiency challenges arise in artificial intelligence applications. Federated learning (FL) has been widely adopted to train deep neural networks (DNNs) without privacy leakage. Existing centralized and decentralized FL (DFL) architectures have limitations, including memory burden, huge bandwidth pressure, and non independent and identically distributed (non-IID) data issues. This article introduces a novel hybrid FL framework, named FedCD, merging the benefits of both centralized and DFL architectures. FedCD strategically distributes the model based on layer sizes and consensus distances (i.e., the deviation between the local models and the global average models), effectively relieving network bandwidth pressures and accelerating training speed even under the non-IID setting. This method significantly mitigates resource constraints and improves model accuracy, offering a promising solution to the challenges in distributed machine learning. Extensive experimental results show the high effectiveness of FedCD. The total completion time of FedCD is reduced by 16.3%–53% and the average accuracy improvement is 1.85% compared to the baselines.**

***Index Terms*—Edge computing (EC), federated learning (FL), Internet of Things (IoT) devices, non independent and identically distributed (non-IID) data, resource constraints.**

Manuscript received 6 February 2024; accepted 18 February 2024. Date of publication 21 February 2024; date of current version 23 May 2024. This work was supported in part by the Jiangsu Province Science Foundation for Youths under Grant BK20230275; in part by the Xiaomi Young Talents Program; in part by the Fundamental Research Funds for the Central Universities; in part by the National Science Foundation of China (NSFC) under Grant 62302145; and in part by the Anhui Province Science Foundation for Youths under Grant 2308085QF230. This article was presented in part at the 29th IEEE International Conference on Parallel and Distributed Systems (ICPADS) 17–21 December, 2023, Ocean Flower Island, Hainan, China. *(Corresponding authors: Zhi Liu; Qianpiao Ma.)*

Jianchun Liu, Yujia Huo, Pengcheng Qu, and Sun Xu are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, Anhui, China, and also with the Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou 215123, Jiangsu, China (e-mail: jcliu17@ustc.edu.cn; yujia_huo@mail.ustc.edu.cn; qupengcheng@mail.ustc.edu.cn; xusun000@mail.ustc.edu.cn).

Zhi Liu is with the Department of Computer and Network Engineering, The University of Electro-Communications, Tokyo 1828585, Japan (e-mail: liu@ieee.org).

Qianpiao Ma is with the Future Network Research Center, Purple Mountain Laboratories, Nanjing 210000, China (e-mail: maqianpiao@pmlabs.com.cn).

Jinyang Huang is with the Anhui Province Key Laboratory of Affective Computing and Advanced Intelligence Machine and School of Computer and Information, Hefei University of Technology, Hefei 230601, China (e-mail: hjy@hfut.edu.cn).

Digital Object Identifier 10.1109/JIOT.2024.3368216

I. Introduction

BILLIONS of Internet of Things (IoT) devices globally generate substantial data, including photographs and voice samples, propelling the advancement of artificial intelligence (AI) [1]. Nonetheless, the process of cloud computing carries the inherent risk of privacy breaches since the data gathered by the cloud may contain sensitive and confidential user information. Also, transferring all data to a remote cloud server can increase latency and degrade user experience [2]. Therefore, edge computing (EC) [3], [4] has emerged as a solution to locally store data and shift high computing power applications from cloud servers to network edges [5]. Furthermore, to alleviate data privacy leakage concerns, federated learning (FL) [6], [7] is employed for distributed machine learning at the network edge across distributed data sets. Additionally, some techniques (e.g., differential privacy [8], [9]) also can be adopted to enhance individual privacy by adding noise to data in FL.

The most prevalent and widely used architecture in the existing FL mechanisms is the centralized FL (CFL) architecture [6], [10], which involves local training at the network edge and model aggregation at the parameter server (PS). Initially, each worker executes stochastic gradient descent (SGD) [6] on its local data set to minimize the loss function and then dispatches the updated model to the PS for global aggregation. Subsequently, the PS circulates the averaged model back to the workers for the following round of local training. However, the PS can become a bottleneck due to potential traffic congestion caused by numerous workers simultaneously communicating, leading to system breakdown if the PS is compromised.

Decentralized FL (DFL) [11], [12] serves as an alternative FL architecture. Here, each worker exchanges models with its neighbors and aggregates them for the subsequent local training. As there is no central PS, DFL eliminates the likelihood of traffic congestion and failure risks at the PS. Furthermore, communication between workers is faster than between workers and the PS, significantly reducing communication time. Despite these advantages, two challenges exist in DFL.

1) *Limited Memory Size:* Workers must receive and store neighbor models, which may strain memory resources because the memory size of a worker is always limited.
2) *Non Independent and Identically Distributed (Non-IID) Local Data:* The training data of a worker is always determined by the environment and users' preferences.

The data in each worker is non-IID in practice and cannot represent the overall data distribution. For example, in the garage, some cameras take more pictures of people, and some take more pictures of vehicles. This challenge also exists in CFL. But in DFL, each worker only exchanges models with a limited number of neighbors, and the training speed and the test accuracy are affected more by non-IID local data [13], [14].

To address the memory strain challenge raised by large model sizes, the model parallelism technique [15], [16] is suggested. This approach divides the model into submodels and distributes them across various devices, alleviating device resource consumption. For instance, RePurpose [15] adjusts neuron positions to decrease intermediate data transmission between workers. However, it still increases network bandwidth strain due to frequent data transmission. The pdADMM [16] divides the model by layers, allowing each layer to update the model independently without communicating with other layers. However, this method is only suitable for relatively small models. Current deep neural networks (DNNs) have large parameter sizes that continue to grow. Transmitting these large models to the PS or neighbors will inevitably consume substantial bandwidth resources, presenting a significant challenge for both centralized and decentralized architectures.

In this article, we propose a novel hybrid FL framework, named FedCD, which combines features of both centralized and decentralized architectures, to enhance model training with IoT devices. In FedCD, we leverage the model parallelism, i.e., some model layers (or submodel) adopt centralized aggregation, while the remaining model layers adopt decentralized aggregation. Specifically, we design a score for each layer according to the layer's location and size, and distribute the layers to the PS and the neighbors according to the scores at the beginning stage. Then, we use the intermediate data (i.e., consensus distances [17]) to form a new layer distribution method at the following stage. FedCD relieves network bandwidth pressure by transmitting submodels in both centralized and decentralized architectures. However, under the decentralized setup, each worker exchanges models with only a limited number of other workers, reducing model performance when local data is non-IID [14]. To address this, FedCD incorporates a centralized architecture to aggregate submodels into a global model at the PS, yielding good performance even with non-IID local data. Furthermore, only submodels are stored in the workers' memories which can relieve the burden of the memories. Consequently, our proposed FedCD can enhance model training even under non-IID settings and alleviate resource constraint pressures. The main contributions of our work can be summarized as follows.

1) We propose a novel hybrid FL framework, called FedCD, which facilitates the distribution of submodels to the PS and the workers' neighbors for efficient aggregation. We provide theoretical evidence affirming the convergence guarantee of model training with FedCD.
2) We design a novel algorithm that strategically determines the distribution of layers to the PS and the neighbors. This decision is initially based on the sizes and positions of the layers, and it is subsequently adjusted according to consensus distances, enabling accelerated convergence speed.
3) Experimental results on classical models and real-world data sets show the effectiveness of the proposed method. FedCD can accelerate the training speed by 16.3%–53% and reduce communication traffic compared to the existing FL systems.

The remainder of this article is organized as follows. Section II introduces some preliminaries and proposes the novel framework of FedCD. We also give the convergence analysis and formalize the problem in Section II. Besides, we provide the motivation for the algorithm design and propose the efficient algorithm for FedCD in Section III. In Section IV, the experiments are conducted and the corresponding results are presented. The related works are summarized in Section V. Finally, we conclude this article in Section VI.

II. Preliminaries and Problem Formulation

A. Federated Learning

Traditional CFL consists of N workers and a PS. Each worker i has a loss function based on the local data set D_i and the size of the data set is $|D_i|$. The loss function can be defined as

$$F_i\left(w_i^t\right) = \frac{1}{|D_i|} \sum_{\xi \in D_i} f_i\left(w_i^t, \xi\right) \tag{1}$$

where w_i^t is the model parameter of worker i at round t and ξ is a batch of the local data set D_i. $f_i(w_i^t, \xi)$ is the local loss function over ξ. To minimize the loss function $F_i(w_i^t)$, worker i uses SGD [6] to update the local model, which can be formulated as

$$w_i^{t+\frac{1}{2}} = w_i^t - \eta \nabla F_i\left(w_i^t\right) \tag{2}$$

where η is the learning rate, $w_i^{t+(1/2)}$ is the local model of worker i which finishes the local training after round t, and $\nabla F_i(w_i^t)$ is the gradient of the loss function.

After local training, each worker pushes the local model to the PS for global aggregation. This process can be formulated as

$$F\left(w^t\right) = \frac{1}{N} \sum_{i=1}^{N} F_i\left(w^t\right) \tag{3}$$

where $w^t = (1/N) \sum_{i=1}^{N} w_i^t$ and $F(w^t)$ is the global loss function. Then the PS sends the updated global model to the workers for the next training.

In DFL, the workers are connected in a network topology. This topology can be modeled as an undirected graph $G = (V, E)$, where $V = \{1, 2, \ldots, N\}$ and $E \in V \times V$. We use a matrix $A = \{A_{i,j} \in \{0, 1\}, 1 < i, j < N\}$ to represent the graph, where $A_{i,j} = 1$ represents there is a link between worker i and worker j. Otherwise, $A_{i,j} = 0$. Worker i first updates the local model using SGD, and then sends the local model to neighbors. Worker i receives the models from their neighbors and aggregates them as

$$w_i^{t+1} = w_i^t + \sum_{j \in N_i^t} u_{i,j}^t \left(w_j^{t+\frac{1}{2}} - w_i^{t+\frac{1}{2}} \right) \tag{4}$$

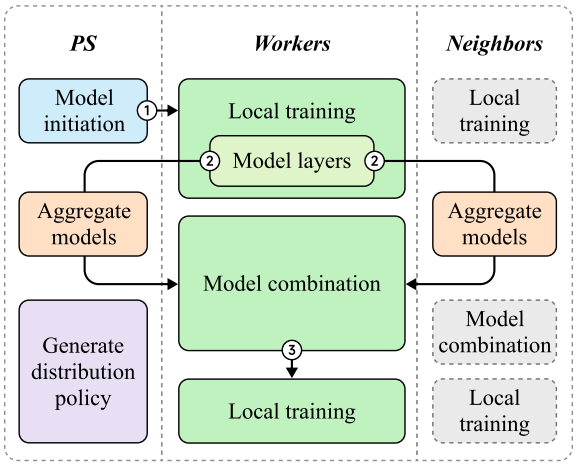


Fig. 1. Overview of FedCD.

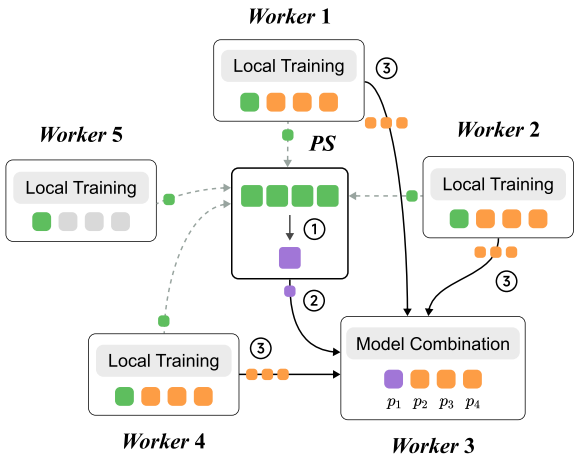


Fig. 2. Illustration example of FedCD.

where N_i^t is the neighbor set of worker i at round t. $u_{i,j}$ is the mixing weight for aggregating the model of worker j. After aggregating the received models, worker i adopts the aggregated model for the next local training.

B. Overview of FedCD

In this section, we will introduce our proposed framework FedCD, which includes an edge server (PS) and some workers. In FedCD, the PS periodically receives the status information (e.g., the consensus distance and the accuracy improvement). After that, the PS generates the layer distribution policy and sends it to the workers at regular intervals. Once the workers receive the policy, the workers will send some layers to the PS and send the rest layers to the neighbors according to the policy. This entire process operates over numerous communication rounds until the model achieves convergence.

We introduce more details about the training process of FedCD through the workflow in Fig. 1. First, in the initialization stage of the model, PS will initialize the entire global model and send it to all workers. Then, workers perform local model training with the fresh global model. Subsequently, PS generates the layer distribution policy, i.e., which layer sends to the PS and which layer sends to the neighbors, according to the status information and proposed algorithm in Section III. After that, the workers send some layers to the PS and send some layers to the neighbors. The PS aggregates the received models using $w^{g,t}(l) = (1/N)\sum_{i=1}^{N} w_i^t(l)$ at round t, where $w_i^t(l)$ is the layer l of trained model at worker i. Then, PS sends the updated model $w^g(l)$ to the workers. In addition, the workers will aggregate the models from the neighbors, and the aggregated model of worker i is denoted as $w_i^{n,t}(l)$. Finally, worker i performs model combination, which is defined as $w_i^{c,t} = \{w^{g,t}(\mathcal{L}_1), w_i^{n,t}(\mathcal{L}_2)\}$, wherein layers received from the PS are included in set $\mathcal{L}_1$, and those received from the neighbors are included in set $\mathcal{L}_2$. After combination, worker i uses the combined model $w_i^{c,t}$ for the next local training. The workers use SGD to update the models

$$w_i^{t+1} = w_i^{c,t} - \nabla f\left(w_i^{c,t}\right). \tag{5}$$

The goal of FedCD is to minimize the loss function

$$F(w) = \frac{1}{N}\sum_{i=1}^{N} f\left(w_i^{c,t}\right). \tag{6}$$

To better explain FedCD, we provide an example in Fig. 2. In this system, there are five workers and one PS. At the beginning, the PS initializes the model and distributes it to five clients. For worker 3, it has three neighbors (workers 1, 2, and 4) for model exchange. According to the strategy distributed by the server, these neighbors send layers 2–4 of the model to worker 3. Besides, layer 1 of all workers is transmitted to the PS for aggregation. Then, worker 3 combined layers 2–4 from neighbors and layer 1 from the PS into a fresh local model to continue training. The entire process will continue until global convergence.

C. Convergence Analysis

In this section, we propose the convergence analysis of FedCD and make four widely used assumptions as follows.

1) *Assumption 1 (Lipschitzian Gradient):* The loss function F_i is with L Lipschitzian gradients, i.e.,

$$\|\nabla F_i(w_1) - \nabla F_i(w_2)\|^2 \le L^2\|w_1 - w_2\|^2 \quad \forall w_1, w_2, i. \tag{7}$$

2) *Assumption 2 (Network Connectivity):* The network topology G is a connected topology.

3) *Assumption 3 (Bounded Gradient Variance):* The variance of stochastic gradients is bounded, i.e.,

$$E_{\xi\in D_i}\|\nabla F_i(w) - \nabla f_i(w;\xi)\|^2 \le \sigma^2 \quad \forall i, w \tag{8}$$

$$E_{i\in V}\|\nabla F_i(w) - \nabla F(w)\|^2 \le \varsigma^2 \quad \forall w. \tag{9}$$

4) *Assumption 4 (Bounded Model Variance):* The variance between the local model and global model is bounded by ϵ^2, i.e.,

$$E_{i\in V}\left\|w^t - w_i^t\right\|^2 \le \epsilon^2 \quad \forall t, i \tag{10}$$

where w^t is the average of the combined model of all the workers, and w_i^t is the combined model of worker i.

To express the relationship between the average of the combined models and the global model when one worker finishes local training, we adopt an upper bound α^2, i.e.,

$$\left\| w^{t+1} - w^{t+\frac{1}{2}} \right\|^2 \leq \alpha^2 \tag{11}$$

where $w^{t+(1/2)}$ represents the global model when one worker finishes the local training using SGD after round t, and w^{t+1} represents the average of the combined models in round $t+1$.

We conduct an experiment to test the change in the value of $||w^{t+1} - w^{t+(1/2)}||$. The results show that $||w^{t+1} - w^{t+(1/2)}||$ oscillates around 0.8 in independent and identically distributed (IID) settings and around 1 in non-IID settings with the increase of communication rounds. So we can use a small bound to limit $||w^{t+1} - w^{t+(1/2)}||^2$.

We also define the upper bound β^2 as

$$\left\| \nabla F\left(w^{t+\frac{1}{2}}\right) \right\|^2 - \left\| \nabla F\left(w^t\right) \right\|^2 \leq \beta^2. \tag{12}$$

We replace the parameter in [18] (*e.g.*, $\hat{M}_k$ by ϵ^2). The detailed proof is as follows: (w^0 is the initial model and w^* is the optimal model which minimizes F)

$$\begin{aligned} &E\left[F\left(w^{t+\frac{1}{2}}\right) - F\left(w^*\right)\right] \\ &\leq E\left[F\left(w^t\right) - F\left(w^*\right)\right] - \frac{\eta M}{2N} E\left\| \nabla F\left(w^t\right) \right\|^2 + \lambda \end{aligned} \tag{13}$$

where $\lambda \leq [(\eta M L^2 \epsilon^2)/(2N)] + [(\eta^2 L(\sigma^2 M + 6\varsigma^2 M^2))/(2N^2)] + [(6\eta^2 L^3 M^2 \epsilon^2)/N^2]$, and M is the size of the data used in each communication round by one worker. We further have

$$\begin{aligned} &E\left[F\left(w^{t+1}\right) - F\left(w^{t+\frac{1}{2}}\right)\right] \\ &\leq E < \nabla F\left(w^{t+\frac{1}{2}}\right), w^{t+1} - w^{t+\frac{1}{2}} > + \frac{L}{2}\left\| w^{t+1} - w^{t+\frac{1}{2}} \right\|^2 \\ &= \frac{1}{2}\left\| \nabla F\left(w^{t+\frac{1}{2}}\right) + w^{t+1} - w^{t+\frac{1}{2}} \right\|^2 - \frac{1}{2}\left\| \nabla F\left(w^{t+\frac{1}{2}}\right) \right\|^2 \\ &\quad - \frac{1}{2}\left\| w^{t+1} - w^{t+\frac{1}{2}} \right\|^2 + \frac{L}{2}\left\| w^{t+1} - w^{t+\frac{1}{2}} \right\|^2 \\ &\leq \frac{1}{2}\left\{ \left\| \nabla F\left(w^{t+\frac{1}{2}}\right) \right\|^2 + \left\| w^{t+1} - w^{t+\frac{1}{2}} \right\|^2 \right\} \\ &\quad + \frac{L}{2}\left\| w^{t+1} - w^{t+\frac{1}{2}} \right\|^2 \\ &\leq \frac{1}{2} E\left\| \nabla F(w^t) \right\|^2 + \frac{1}{2}\beta^2 + \frac{L+1}{2}\alpha^2. \end{aligned} \tag{14}$$

By adding (13) and (14), we obtain the convergence bound between two consecutive training rounds

$$\begin{aligned} &E\left[F\left(w^{t+1}\right) - F\left(w^*\right)\right] \\ &\leq E\left[F\left(w^t\right) - F\left(w^*\right)\right] - \frac{\eta M - N}{2N} E\left\| \nabla F\left(w^t\right) \right\|^2 \\ &+ \lambda + \frac{1}{2}\beta^2 + \frac{L+1}{2}\alpha^2 \\ &E\left[F\left(w^{t+1}\right) - F\left(w^t\right)\right] \\ &\leq -\frac{\eta M - N}{2N} E\left\| \nabla F\left(w^t\right) \right\|^2 + \lambda + \frac{1}{2}\beta^2 + \frac{L+1}{2}\alpha^2. \end{aligned} \tag{15}$$

We sum the results in (15) from $t = 0$ to $t = T-1$ and obtain:

$$\begin{aligned} &\sum_{t=0}^{T-1} E\left[F\left(w^{t+1}\right) - F\left(w^t\right)\right] = E\left[F\left(w^T\right) - F\left(w^0\right)\right] \\ &\leq -\frac{\eta M - N}{2N} \sum_{t=0}^{T-1} E\left\| \nabla F\left(w^t\right) \right\|^2 + T\left(\lambda + \frac{1}{2}\beta^2 + \frac{L+1}{2}\alpha^2\right) \\ &\frac{1}{T}\sum_{t=0}^{T-1} E\left\| \nabla F\left(w^t\right) \right\|^2 \\ &\leq \frac{2N\left(F\left(w^0\right) - F(w^*)\right)}{T(\eta M - N)} + \frac{2N}{(\eta M - N)}\left(\lambda + \frac{1}{2}\beta^2 + \frac{L+1}{2}\alpha^2\right). \end{aligned} \tag{16}$$

D. Problem Formulation

To train models among distributed workers by FL, it is inevitable to consume resources (e.g., CPU cycles and network traffic). Formally, we define the computing resource consumption of worker k in one round as c_k. Thus, the computing resource consumption of T rounds is Tc_k. We accumulate N workers' computing consumption and the total consumption should not exceed its budget B_c. Each worker's transmission workload of centralized architecture is defined as W_a, where $W_a = \sum_{i=1}^{L}(1 - x_i)M(i)$. We define the total size of the transmitted data of one peer in a decentralized architecture as W_b, where $W_b = \sum_{i=1}^{L} 2x_i M(i)$. $M(i)$ is the data volume of the ith layer model. x_i is a binary variable and it indicates whether layer i adopts centralized architecture or not (0 for adopting centralized architecture and 1 for adopting decentralized architecture). We define the total transmission budget of a round as B_b. Let B_1 and B_2 represent the inbound bandwidth between workers and the PS, and the outbound bandwidth between workers and the PS, respectively. We denote the bandwidth of worker j and worker m as B_{jm}. r_{jm} is a binary variable and it indicates whether there is a link between worker j and worker m. We define the capacity budget for PS node as C_a and the capacity budget for worker j as C_j. We formulate the problem as follows:

$$\begin{aligned} &\min\ \lambda H + (1-\lambda) f\left(w^t\right) \\ &\text{s.t.} \begin{cases} \sum_{k=1}^{N} Tc_k \leq B_c \\ \sum_{k=1}^{N} 2W_a + \sum_{j=1}^{N}\sum_{m=1}^{N} W_b r_{jm} \leq B_b \\ NW_a \leq C_a \\ W_b \sum_{m=1}^{N} r_{jm} \leq C_j \\ x_i = \{0, 1\} \\ r_{jm} = \{0, 1\}. \end{cases} \end{aligned} \tag{17}$$

Let $H = \max\left\{W_a/B_1 + W_a/B_2, \max\left\{W_b/B_{jm}\right\}\right\}$.

Our objective is to minimize the maximum communication time and maximize the training speed of FedCD. If λ is set as a large number, we pursue the minimum communication time of a round. If λ is set as a small number, we pursue the quick training speed of a round. The first inequality indicates that the computing workload of T rounds for N workers is less than a budget. The second inequality indicates the total transmission cost per round is less than a budget. The third inequality indicates the PS node's capacity is larger than the total transmission volume between the PS node and workers.

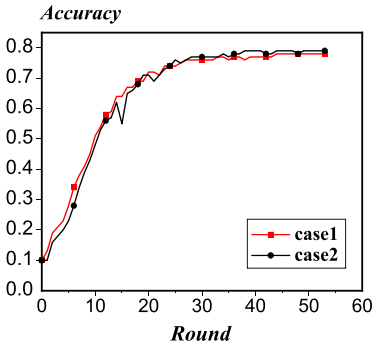


Fig. 3. Performance of AlexNet over CIFAR-10 under two different layer allocation strategies.

The fourth inequality indicates each worker's capacity is larger than the total sizes of the collected models of each worker.

III. Algorithm Design

A. Motivation for the Algorithm Design

This section provides a brief description of the inspiration sources in the early stage of the research, and designs LDLS algorithms based on some experimental phenomena in the early stage. Subsequent experiments have proven that the algorithm we designed is efficient. In early experiments, we evenly distributed the CIFAR10 [19] data set to ten clients with IID local data types using the AlexNet [20] model. We fixed the learning rate to 0.01 and conducted two experiments. The first experiment (case1) sent layers 0–8 to PS, layers 9–10 to neighbors, and the second experiment (case2) sent layers 0, 2, 4, 6, 8, and 10 to PS, layers 1, 3, 5, 7, and 9 were sent to neighbors to observe their aggregation and model accuracy in different rounds, and the results are shown in Fig. 3.

We observe that in the early stages of training (when the accuracy increases significantly), sending adjacent layers to the same target (case1), i.e., PS or neighbors, results in faster convergence speed compared to sending adjacent layers to different targets (case2). Therefore, we considered this discovery and designed the LDLS algorithm, which combines the size and position of each layer to determine whether to use CFL or DFL for each layer in the early training stage.The detailed algorithm design is shown in the following figure.

B. Layer Distribution Based on the Layer's Location and Size

We design two algorithms to decide which layer is sent to the PS and which layer is sent to the neighbors. We first introduce the layer distribution method which is based on the layers' locations and sizes in FedCD (LDLS in Algorithm 1). In LDLS, worker i first initializes two vectors μ_1 and μ_2, which represent the maximum time when each layer is sent to the PS and neighbors, respectively. FedCD uses sum_1 and sum_2 to express the total time when the layers are sent to the PS or the neighbors (line 1). FedCD also initializes the unassigned layers set Q_1 which contains all the layers (line 2). Since the network bandwidth always fluctuates, the architecture uses the average value to represent the network bandwidth. We assume that the inbound bandwidth between the workers and the PS is B_1, and the outbound bandwidth between the workers and the PS is B_2. Worker i computes the time when it transmits the layer l's parameter from the worker to the PS and downloads it from the PS to the worker, i.e., $\mu_1(l) = M(l)/B_1 + M(l)/B_2$. $M(l)$ is the size of layer l. We define the minimum bandwidth in the network as $B_{\min}$, the maximum time when each worker transmits the layer l's parameter using the decentralized method is $\mu_2(l) = M(l)/B_{\min}$. The layers sent to the PS are in the set L_1, and the layers sent to the neighbors are in the set L_2.

Algorithm 1 LDLS ($c_1 < c_2$)

1: Initialize $\mu_1, \mu_2, L_1, L_2, sum_1 = 0, sum_2 = 0$;
2: Initialize the set of unassigned layers $Q_1 = \{1, 2, ..., L\}$;
3: Sort all layers in non-decreasing order by $\mu_1(l)$;
4: Select the first element c_1;
5: Sort all layers in non-increasing order by $\mu_2(l)$;
6: Select the first element c_2;
7: $Q_1 \leftarrow Q_1 - \{c_1\}$; $Q_1 \leftarrow Q_1 - \{c_2\}$;
8: $L_1.insert(c_1)$; $L_2.insert(c_2)$;
9: $sum_1 += \mu_1(c_1)$; $sum_2 += \mu_2(c_2)$;
10: **while** $Q_1 \neq \phi$ **do**
11: **if** $sum_1 < sum_2$ **then**
12: search the element l_1 with minimum score in Q_1;
13: $L_1.insert(l_1)$;
14: $Q_1 \leftarrow Q_1 - \{l_1\}$;
15: $sum_1 += \mu_1(l_1)$;
16: **else**
17: search the element l_2 with maximum score in Q_1;
18: $L_2.insert(l_2)$;
19: $Q_1 \leftarrow Q_1 - \{l_2\}$;
20: $sum_2 += \mu_2(l_2)$;

We choose the layer with the minimum element in μ_1 and the layer is named c_1 (line 4), and choose the layer with the maximum element in μ_2 and the layer is named c_2 (line 6). Then, c_1 and c_2 are removed from Q_1 (line 7). We insert c_1 and c_2 to L_1 and L_2, respectively, (line 8). Then, we add $\mu_1(c_1)$ to sum_1 (line 9) and add $\mu_2(c_2)$ to sum_2 (line 9). d_1 is used to express the layer's distance from c_1. If the number of the layer is lower than c_1, d_1 will be lower than 0, and if the number of the layer is higher than c_1, d_1 will be higher than 0. d_2 is used to express the layer's distance from c_2. Then, we compute the score of each layer: we denote $m = (\sigma_1 d_1 + \sigma_2 d_2)M(l)$

$$\text{score}(l) = e^m. \tag{18}$$

Algorithm 1 introduces the LDLS method when $c_1 < c_2$. If $c_1 < c_2$, the centralized method will choose the layer with the minimum score, and the decentralized method will choose the layer with the maximum score. That is, because the scores of the layers next to c_1 are small and the scores of the layers next to c_2 are very large. If $sum_1 < sum_2$, we first choose the element in Q_1 with the minimum score and the layer will be named l_1. Then, we move the layer l_1 out of Q_1 and insert l_1 to L_1. And sum_1 will be added by $\mu_1(l_1)$ (lines 11–15). If $sum_1 > sum_2$, we will choose the element in Q_1 with the largest score and the layer will be named l_2. Then l_2 will be sent to the neighbors for aggregation. We move the layer l_2 out of Q_1

Algorithm 2 LDC

Input: the probability of the layers R_l; T_1, T_2; S_1, S_2
Output: L_1 and L_2.

1: **while** round t = nT **do**
2: **if** $\sum_{l\in L_1} R_l > T_1$ **then**
3: $S_1 = |L_1|$;
4: **else**
5: $S_1 = max\{S_1 - 1, 1\}$;
6: choose S_1 layers in L_1 with the largest R_l and these S_1 layers perform centralized FL (clear L_1 and insert these these S_1 layers to L_1).
7: **if** $\sum_{l\in L_2} R_{l(max)} > T_2$ **then**
8: $S_2 = max\{|L_2|/2, 1\}$;
9: **else**
10: $S_2 = 0$;
11: choose S_2 layers in L_2 and these S_2 layers perform centralized FL (insert these these S_2 layers to L_1);
12: clear L_2 and add the unselected layers to L_2;

and insert l_2 to L_2. And sum_2 will be added by $\mu_2(l_2)$ (lines 17–20). If $c_1 > c_2$, the centralized method will choose the layer with the maximum score, and the decentralized method will choose the layer with the minimum score. The algorithm will end when all the layers are distributed.

C. Layer Distribution Based on Consensus Distance

Note that LDLS may face performance degradation when the number of workers in the network is large and the number of epochs in a communication round is small. To this end, the layer distribution based on consensus distances [17] (LDC in Algorithm 2) is proposed. The consensus distance $D^t(l)$ represents the deviation between the local model and the average global model of layer l. If $D^t(l)$ is small, it represents that the local model is similar to the average global model and the workers do not need to send the layer l to the PS. On the contrary, if $D^t(l)$ is large, it means that the local model differs greatly from the average global model, so the workers need to send layer l to the PS. We define the consensus distance of worker i is $D^t(l) = (1/N)\sum_{i=1}^{N} D_i^t(l)$. If the layer uses centralized method, $D_i^{t+1}(l)$ is defined as $D_i^{t+1}(l) = ||\bar{w}^{t+1}(l) - w_i^{t+(1/2)}(l)||$, where $w_i^{t+(1/2)}$ is the model of worker i which finishes performing local training after round t and $\bar{w}^{t+1}(l) = (1/N)\sum_{i=1}^{N} w_i^{t+(1/2)}(l)$. If the layer uses decentralized method, $D_i^{t+1}(l)$ is defined as $D_i^{t+1}(l) = ||\bar{w}^{t+1}(l) - w_i^{t+1}(l)||$, where $w_i^{t+1}(l)$ is the model of worker i which finishes aggregation using the models received from neighbors after round t. However, in DFL, the average model $\bar{w}^{t+1}(l)$ is not available in practice. If the layer adopts a decentralized method to update the model, we will use the following method to calculate the consensus distance:

$$\begin{aligned} D_i^{t+1}(l) &= \left\| \bar{w}^{t+1}(l) - w_i^{t+1}(l) \right\| \\ &= \left\| \frac{1}{N}\sum_{j=1}^{N} w_j^{t+\frac{1}{2}}(l) - \left(w_i^{t+\frac{1}{2}}(l) + \sum_{j=1}^{N} u_{i,j}^t A_{i,j}\left(w_j^{t+\frac{1}{2}}(l) - w_i^{t+\frac{1}{2}}(l) \right) \right) \right\| \\ &= \left\| \sum_{j=1}^{N} \frac{w_j^{t+\frac{1}{2}}(l) - w_i^{t+\frac{1}{2}}(l)}{N} - u_{i,j}^t A_{i,j}\left(w_j^{t+\frac{1}{2}}(l) - w_i^{t+\frac{1}{2}}(l) \right) \right\|. \end{aligned}$$

We set $u_{i,j}^t = (1/N)$ for simplicity, and then $D_i^{t+1}(l)$ is the possible maximum value, thus it follows:

$$\begin{aligned} D_i^{t+1}(l) &= \left\| \sum_{j=1}^{N} \frac{(1 - A_{i,j})\left(w_j^{t+\frac{1}{2}}(l) - w_i^{t+\frac{1}{2}}(l) \right)}{N} \right\| \\ &\le \frac{1}{N}\sum_{j=1}^{N} (1 - A_{i,j}) D_{i,j}^{t+1} \end{aligned}$$

where $D_{i,j}^{t+1}(l) = ||w_i^{t+(1/2)}(l) - w_j^{t+(1/2)}(l)||$,

$$\begin{aligned} D_{i,j(\max)}^{t+1}(l) &= \left\| w_i^{t+\frac{1}{2}}(l) - w_k^{t+\frac{1}{2}}(l) + w_k^{t+\frac{1}{2}}(l) - w_j^{t+\frac{1}{2}}(l) \right\| \\ &\le \left\| w_i^{t+\frac{1}{2}}(l) - w_k^{t+\frac{1}{2}}(l) \right\| + \left\| w_k^{t+\frac{1}{2}}(l) - w_j^{t+\frac{1}{2}}(l) \right\| \\ &= D_{i,k}^{t+1}(l) + D_{k,j}^{t+1}(l) \\ D_{i,j(\min)}^{t+1}(l) &= \left\| w_i^{t+\frac{1}{2}}(l) - w_k^{t+\frac{1}{2}}(l) - (w_j^{t+\frac{1}{2}}(l) - w_k^{t+\frac{1}{2}}(l)) \right\| \\ &\ge \left| \left\| w_i^{t+\frac{1}{2}}(l) - w_k^{t+\frac{1}{2}}(l) \right\| - \left\| w_k^{t+\frac{1}{2}}(l) - w_j^{t+\frac{1}{2}}(l) \right\| \right| \\ &= \left| D_{i,k}^{t+1}(l) - D_{k,j}^{t+1}(l) \right|. \end{aligned}$$

Thus, we can estimate $D_{i,j(\max)}^t(l)$ as $\hat{D}_{i,j(\max)}^t(l)$

$$\hat{D}_{i,j(\max)}^t(l) = \min_{k\in[N]-\{i,j\}} \left(D_{i,k}^t(l) + D_{k,j}^t(l) \right) \tag{19}$$

we also can estimate $D_{i,j(\min)}^t(l)$ as $\hat{D}_{i,j(\min)}^t(l)$

$$\hat{D}_{i,j(\min)}^t(l) = \max_{k\in[N]-\{i,j\}} \left(\left\| D_{i,k}^t(l) - D_{k,j}^t(l) \right\| \right). \tag{20}$$

If $D_{i,j(\max)}^t(l)$ is used to calculate $D^t(l)$, $D^t(l)$ is recorded as $D_{\max}^t(l)$. $D_{\min}^t(l)$ is the same. t_{ps} and t_{nb} represent the total time when the layers are sent to the PS and the neighbors. If (t_{ps}/t_{nb}) is low, we need to arrange more layers to adopt centralized method. Thus, the probability of transmitting layer l to the PS p_l^t is inversely proportional to (t_{ps}/t_{nb}). We perform LDC when the round $t = nT$, where T is the interval of performing two adjacent LDC. Based on the above analysis, we determine the variable p_l^t by using the following equation:

$$p_l^t = \frac{\left[\lambda \sum_{h=(n-1)T}^{nT} D^{h+1}(l) + (1-\lambda)\left(e^{|d-d_m|} \right) \right] e^{\frac{1}{\triangle\xi}}}{e^{\sqrt{\frac{t_{ps}}{t_{nb}}}}}. \tag{21}$$

If the layer uses DFL, we will compute the $p_{l(\max)}^t$ and $p_{l(\min)}^t$ with $D_{\max}^h(l)$ and $D_{\min}^h(l)$ to decide whether the layer will use CFL. $\triangle\xi$ represents the average accuracy improvement of the models. The minimum $\triangle\xi$ is set as 20%. If the accuracy improvement is lower, we will increase the p_l^t to make more

layers use centralized mechanism. So we add $e^{[1/(\Delta\xi)]}$ to the equation. We compute the average number of the layers using the decentralized method and denote it as d_m, and use d to represent the number of layer l. We use p_l^t to determine the probability of using the centralized method for the layer l. The layer with higher p_l^t will be preferentially chosen to use the centralized method. We use the exponential moving average to smooth the probability

$$P_l = \varphi p_l^t + (1-\varphi)P_l \tag{22}$$

where φ $(0 \le \varphi \le 1)$ is a hyperparameter that reflects the weight of the previous probability and the newly computed probability. We use K to denote the number scale of P_l. For example, if $P_l = 5 \times 10^{-4}$, then $K = -4$. Let $V(l)$ represent the communication rounds when layer l uses CFL. We use $V_{\max}$ to represent the total communication rounds. We use $H(l)$ to denote the communication rounds when layer l uses the decentralized method. Since the layers that perform less CFL always have a high probability of performing CFL, we add a penalty item for the probability P_l. If the layer l is in L_1, we use (23) to calculate the decision variable R_l. If the layer l is in L_2, we use (24) to calculate the decision variable R_l

$$R_l = P_l + 10^K \sqrt{\ln(t+1)}\left(1 + \frac{V_{\max}}{V(l)+1}\right) \tag{23}$$

$$R_l = P_l + 10^K \frac{\sqrt{\ln(t+1)}}{1 + V_{\max} - H(l)}. \tag{24}$$

When performing LDC (Algorithm 2) for the first time, we first choose the minimum layer and choose another L-S_1-S_2-1 layers which are next to the minimum layer with the smallest sizes to use the decentralized method. The average number of these layers is denoted as d_m. The rest layers adopt a centralized method. When it is not the first time to perform LDC, we first sum up R_l of the layers in L_1. If the sum is larger than the threshold T_1, S_1 is set as $|L_1|$ (lines 2 and 3). S_1 is the number of the layers which continue to perform CFL. If the sum is lower than the threshold T_1. S_1 will minus 1 (line 5). Then, we choose S_1 layers in L_1 with the largest R_l and these S_1 layers perform CFL (line 7). Because the estimated values of $D^t(l)$ of the layers in L_2 differ significantly from the actual value, we will not compare them with the $D^t(l)$ of the layers in L_1 together. If we use $p^t_{l(\max)}$ to compute R_l, we denote R_l as $R_{l(\max)}$, and if we use $p^t_{l(\min)}$ to compute R_l, we denote R_l as $R_{l(\min)}$. We sum up $R_{l(\max)}$ of the layers in L_2 to compare it with the threshold T_2. If the sum is larger than T_2, S_2 is set to be $\max\{|L_2|/2, 1\}$; otherwise, $S_2 = 0$ (lines 8–11). And then we choose S_2 layers in L_2 to use the centralized method. If S_2 is not 0, we sort the layers by $R_{l(\max)}$ using descending order. If the difference between the value of the S_2th and the (S_2+1)th $R_{l(\max)}$ is less than a threshold, we choose $S_2 - 1$ layers with the highest $R_{l(\max)}$ and we choose one layer of the S_2th and the (S_2+1)th layers with higher $R_{l(\min)}$. Otherwise, we choose S_2 layers with higher $R_{l(\max)}$, and these layers perform the CFL (line 13). After that, we clear L_2 and add the unselected layers to L_2 (line 14). In the validation experiment, the LDC method can accelerate the training speed, but the final accuracy may be reduced. After performing LDC for some rounds, the workers will use the LDLS method to achieve higher accuracy.

TABLE I
LAYERS AND SIZE OF THE SELECTED MODELS

Model	# of Layers	Size (MB)
VGG-9	12	13.33
VGG-16	21	128.25
AlexNet	11	88.78

IV. EXPERIMENTS

This section introduces the experiment platform in Section IV-A. Then, we describe the data sets and models in Section IV-B. We introduce the baselines and metrics for performance comparison in Section IV-C. The experimental results are presented in Section IV-D.

A. Experiment Platform

We perform experiments on an AMAX deep learning workstation equipped with an Intel Core i9-10900X CPU, 4 NVIDIA GeForce RTX 2080Ti GPUs and 128-GB RAM. On the workstation, we implement 5–10 processes to simulate 5–10 workers and implement one process to simulate the PS. The execution of each worker's model training is based on the PyTorch framework [21]. The socket library of Python [21] is used to build up communication between workers and the PS.

B. Models and Data Sets

The experiments are conducted on three well-known DNNs: 1) VGG-9; 2) VGG-16 [22]; and 3) AlexNet [20], which represent the middle-size model (VGG-9) and the large-size models (VGG-16 and AlexNet). The sizes of DNNs are 13.33, 128.25, and 88.78 MB, respectively, as shown in Table I. AlexNet is trained over CIFAR-10, which includes 50 000 images for training and 10 000 for testing. The images in CIFAR-10 are $32 \times 32 \times 3$ dimensional and are labeled in ten classes. VGG9 and VGG16 are trained over the CIFAR-100 data set which is similar to CIFAR-10 but contains 100 classes. Then, we illustrate how to partition the training data under non-IID settings. For CIFAR-10, each class of the data set distributes χ data to one worker randomly, where χ represents a percentage. The rest of the $1-\chi$ data is equally distributed to the rest of the workers. For CIFAR-100, there are 100/N classes of the data set that distribute χ data to one worker, and the rest of the $1-\chi$ data is equally distributed to the rest of the workers. We employ the SGD [6] optimizer for AlexNet, VGG-16, and VGG-9 and the learning rates are initialized as 0.01. The models are trained with a batch size of 64.

C. Baselines and Metrics

Baselines: We choose three classical and efficient algorithms as baselines for performance comparison, which are summarized as follows.

1) *FedAvg* [6] is a famous algorithm in FL in which the workers send the entire model to the PS and download the models after aggregation at PS.

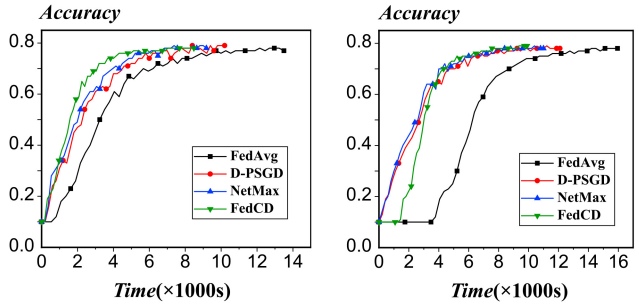


Fig. 4. AlexNet over CIFAR-10 under IID setting. (a) Node = 5. (b) Node = 10.

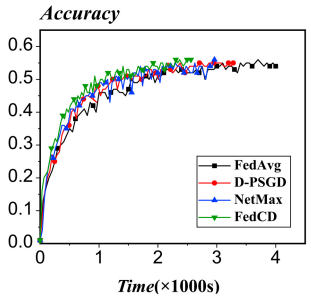


Fig. 5. VGG-9 over CIFAR-100.

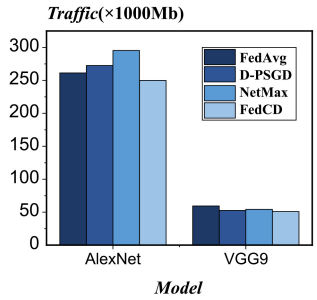


Fig. 6. Total traffic in the network.

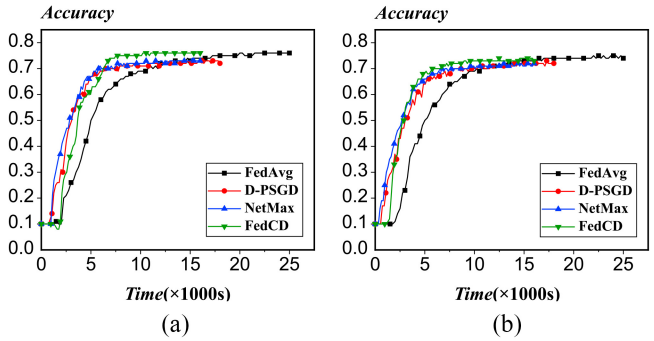


Fig. 7. AlexNet over CIFAR-10 under different non-IID levels. (a) Non-IID 50%. (b) Non-IID 70%.

2) *D-PSGD* [23] is a famous algorithm in DFL. Each worker sends the trained model to the neighbors and each worker aggregates the models locally.
3) *NetMax* [24] is a communication-aware DFL technique over the heterogeneous network. It enables each worker to asynchronously pull models from one peer for aggregation. The peers with higher bandwidth are selected with higher probabilities.

Metrics: We adopt the following metrics to evaluate the performance of our proposed FedCD and baselines.

1) *Test Accuracy* is the amount of the right data predicted by the model divided by the amount of all the data. It is used to test whether the method can converge or not.
2) *Completion Time* is the time when each worker finishes local training and model aggregation. It is used to evaluate the training speed.
3) *Network Traffic* is the total size of the models transmitted through the network, which is adopted to quantify the communication cost.

D. Overall Performance

Training Performance: We use LDLS to test the performance of FedCD under IID settings. Fig. 4 shows the training performance of AlexNet over CIFAR-10 with 5-10 workers under IID settings. Fig. 5 shows the performance of VGG-9 over CIFAR-100. As we can see in Figs. 4 and 5, our proposed FedCD framework converges faster than FedAvg, D-PSGD, and NetMax. This is because FedCD sends the model to the PS and the neighbors simultaneously which can save time. Fig. 6 shows the total traffic in the network when AlexNet reaches an accuracy of 70% and VGG9 reaches an accuracy of 50%. We can find out that FedCD consumes the least traffic. FedCD needs 22 rounds for AlexNet and 30 rounds for VGG9 to reach the target accuracy which uses the least rounds compared to the three baselines. In Fig. 4, we can find out that when the scale of workers becomes larger, our proposed FedCD still maintains its advantages over the baselines.

Performance on Non-IID Data: We use LDLS to test the performance of FedCD under non-IID settings. In our experiments, the number of workers and epochs in a communication round is set as 8 and 20, respectively. We distribute the data set into different Non-IID levels χ (e.g., 30%, 50%, and 70%) as suggested in [25]. As shown in Figs. 7 and 8, we train AlexNet over CIFAR-10 and VGG-16 over CIFAR-100 under non-IID settings. FedCD outperforms baselines at all non-IID levels. For example, FedCD achieves an accuracy of 76.67%, when training AlexNet over CIFAR-10 with $\chi = 50\%$, which is higher than that of FedAvg (76.42%), D-PSGD (73.37%) and NetMax (73.79%). This is because FedCD combines the advantage of centralized and decentralized methods which is more robust in non-IID settings. Second, as χ increases, the required completion time of each solution increases when achieving the same accuracy. However, Fig. 9 shows that FedCD is more robust without a significant increase in the completion time of AlexNet over CIFAR-10 when achieving the accuracy of 72%, compared with the baselines. For FedCD, the time consumption is 5860.8 s ($\chi = 30\%$), 6837.6 s ($\chi = 50\%$) and 7326 s ($\chi = 70\%$), while 7000.4, 9116.8, 13 349.6 s for NetMax. This is because the per-epoch completion time of FedCD is much shorter than other solutions, and FedCD iterates more epochs and achieves a better performance under a given time budget.

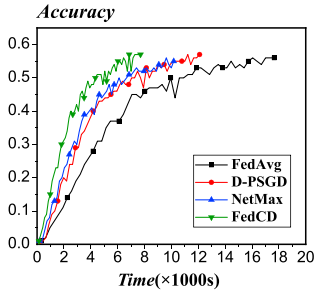


Fig. 8. VGG-16 non-IID 50%.

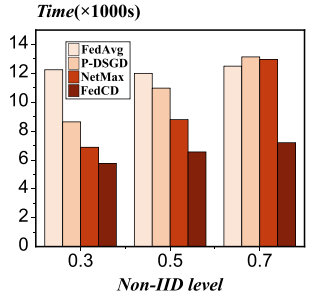


Fig. 9. Time cost under different non-IID levels.

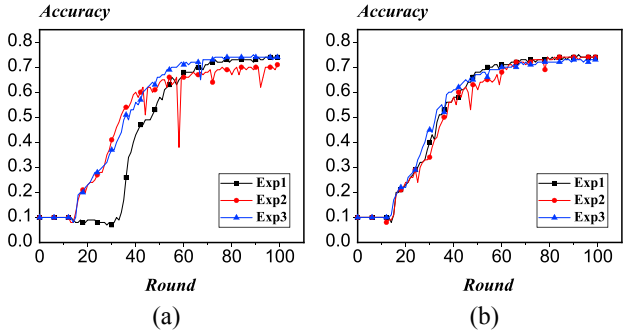


Fig. 10. Experiments using LDC. (a) Difference between using LDC and LDLS. (b) Performance on different section selection.

Performance of LDC: We conduct some experiments to test the effect of the LDC algorithm using AlexNet over CIFAR-10 with ten workers. In Fig. 10(a), the first experiment uses the LDLS algorithm to train the model, and the second experiment combines the LDC and LDLS algorithms. For the first 15 rounds, we use the LDLS algorithm and for the rest rounds, we use the LDC method and do not change the layer distribution after 15 rounds. The third experiment uses LDLS for the first 15 rounds. For the rounds 15–30, we use LDC, and for the rounds after 30, we use the LDLS method again. We can find out in Fig. 10(a) that in the first experiment, the improvement of accuracy is very slow at the beginning, experiments 2 and 3 converge faster than experiment 1. However, the final accuracy of experiment 2 is lower than that of experiments 1 and 3. Finally, experiment 3 performs well on both training speed and final test accuracy.

We continue to conduct three experiments to test the performance of LDC. In Fig. 10(b), all the three experiments use LDLS at the beginning. In the intermediate stage, all

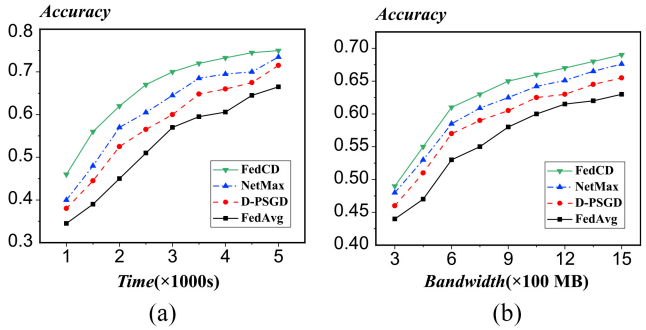


Fig. 11. Training performance under different resource budgets. (a) Test accuracy versus time. (b) Test accuracy versus bandwidth.

three experiments use LDC. Experiment 1 changes the layer distribution at rounds 15 and 30. Experiment 2 changes the layer distribution at rounds 15, 30, and 45. Experiment 3 changes the layer distribution at rounds 10, 20, and 30. We continue to use the LDLS method after round 45 in experiment 1, round 60 in experiment 2, and round 40 in experiment 3. We can observe in Fig. 10(b) that experiment 3 converges faster than its two counterparts at the beginning, however, its training speed falls behind after 50 rounds and the final test accuracy is the lowest in the three experiments. Experiment 1 converges faster than its two counterparts after round 50 and it reaches the highest test accuracy in the end. Thus, the number of training sections of LDC (when to change the layer distribution) is essential to achieve quick training speed and high test accuracy.

The Generalization of FedCD: We try to explore FedCD's performance in more applications, such as Human Activity Recognition, which is to identify a person's status (e.g., standing, sitting) based on the sensor data from IoT devices (e.g., smartphones or smartwatches). For this task, we adopt the HAR data set [26] collected from 30 individuals, including 7352 training samples and 2947 test samples across six categories. The model trained on HAR is a CNN model (denoted as CNN-H) with three 5×5 convolutional layers and two fully connected layers. We test the training performance of FedCD and baselines under different resource (e.g., completion time and network bandwidth) budgets. As shown in Fig. 11, FedCD always outperforms the baselines in terms of test accuracy. For example, given the bandwidth budget of 900 MB, the accuracy of FedCD is about 64.5%, while that of NetMax, D-PSGD, and FedAvg is about 62.2%, 60.7%, and 58.6%, respectively. In other words, FedCD can improve the accuracy by about 2.3%, 3.8%, and 5.9%, respectively. Thus, our proposed framework FedCD is also applicable to other applications.

V. Related Works

In the past few years, FL, which possesses unique advantages in terms of privacy protection, has gradually become a hot topic [6]. This has sparked profound research by numerous scholars in the field of FL. To address the FL challenges in various application scenarios, both CFL and DFL have been proposed, offering additional possibilities for advancements in the FL domain.

A. Centralized Federated Learning

The concept of CFL was initially introduced [27] by Google in 2016, and subsequently, McMahan and others developed the FedAvg algorithm [6]. CFL is currently the most extensively researched and widely used FL method. Specifically, in CFL, clients at the network edge locally train the models using their respective data sets. After a certain number of local training rounds, clients send the aggregated model over the network link to a central PS. The server collects parameters from different clients and completes the global model aggregation. In the subsequent training round, the updated global model is sent back to specific or all clients. CFL enables collaboration in training without revealing individual client data. However, challenges such as potential network link delays, data heterogeneity [14] among different clients, and device heterogeneity [28] still require further research in this technology. Yang et al. [29] proposed an iterative algorithm to address the problem of energy-efficient transmission and computation resource allocation in CFL. Luo et al. [30] developed an adaptive client sampling algorithm to mitigate the impact of data heterogeneity and structural heterogeneity on CFL. Li et al. [31] explored the impact of global aggregation rate and client weight on CFL.

B. Decentralized Federated Learning

The concept of DFL emerged after CFL. The primary goal of introducing this concept is to avoid potential network congestion and failure risks at the central server in CFL. By shifting communication from the client to the server to between clients, it further reduces communication costs and enhances communication efficiency. However, DFL has many problems, such as posing greater challenges to the memory of workers (in some cases edge devices). Roy et al. [32] designed BrainTorrent based on the idea of DFL, providing a highly dynamic peer-to-peer environment for clients to interact directly without relying on the central server. Hu et al. [33] proposed a segmented gossip approach, which fully utilizes the bandwidth between workers, avoiding dependence on highly concentrated topologies. Li et al. [34] proposed the BFLC framework, which combines blockchain technology with FL to implement a DFL method that can effectively reduce consensus computation and malicious attacks.

C. Hybrid Federated Learning

As we can see, both CFL and DFL have their own advantages and disadvantages, so it is extremely important to find a balance between the two. However, there is little research on the combination of CFL and DEL in existing work, and most of the work focuses on improving and analyzing these two types separately. Lalitha et al. [35] proposed the Fully DFL and analyzed the advantages of DFL compared to traditional FL algorithms. Chou et al. [36] designed a framework called fed P2P, which divides the network into multiple P2P networks and only allows some devices to communicate with the PS. Beltrán et al. [37] analyzed the existing framework of DFL, reviewed its application scenarios and studied the main aspects differentiating DFL and CFL. Our proposed FedCD framework, which combines the methods of CFL and DFL, adopts different strategies to determine the aggregation method of the current layer (sent to PS or neighbors) based on the position and consensus distance of the layers. This has achieved better training performance and aggregation speed during the experimental process, providing a new idea for the combination of CFL and DFL.

VI. Conclusion

In this article, we propose an FL framework named FedCD, which is a combination of centralized and decentralized architectures. The method addresses the challenges of the memory burden, huge bandwidth pressure, and non-IID local data. We have further proposed two algorithms to decide which layer is sent to the PS and which layer is sent to the neighbors. The experimental results show that FedCD significantly outperforms baselines.

References

[1] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Toward an intelligent edge: Wireless communication meets machine learning," *IEEE Commun. Mag.*, vol. 58, no. 1, pp. 19–25, Jan. 2020.

[2] A. Aral, M. Erol-Kantarci, and I. Brandić, "Staleness control for edge data analytics," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 2, pp. 1–24, 2020.

[3] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[5] X. Wei, Q. Li, Y. Liu, H. Yu, T. Chen, and Q. Yang, "Multi-agent visualization for explaining federated learning," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 6572–6574.

[6] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[7] J. Liu, J. Yan, H. Xu, Z. Wang, J. Huang, and Y. Xu, "Finch: Enhancing federated learning with hierarchical neural architecture search," *IEEE Trans. Mobile Comput.*, early access, Sep. 14, 2023, doi: 10.1109/TMC.2023.3315451.

[8] A. El Ouadrhiri and A. Abdelhadi, "Differential privacy for deep and federated learning: A survey," *IEEE access*, vol. 10, pp. 22359–22380, 2022.

[9] J. Huang et al., "PhyFinAtt: An undetectable attack framework against PHY layer fingerprint-based WiFi authentication," *IEEE Trans. Mobile Comput.*, early access, Dec. 4, 2023, doi: 10.1109/TMC.2023.3338954.

[10] M. Li et al., "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Symp. Oper. Syst. Design Implement.*, 2014, pp. 583–598.

[11] S. Savazzi, M. Nicoli, and V. Rampa, "Federated learning with cooperating devices: A consensus approach for massive IoT networks," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4641–4654, May 2020.

[12] J. Liu, J. Liu, H. Xu, Y. Liao, Z. Wang, and Q. Ma, "YOGA: Adaptive layer-wise model aggregation for decentralized federated learning," *IEEE/ACM Trans. Netw.*, early access, Nov. 6, 2023, doi: 10.1109/TNET.2023.3329005.

[13] M. Waqas, Y. Niu, M. Ahmed, Y. Li, D. Jin, and Z. Han, "Mobility-aware fog computing in dynamic environments: Understandings and implementation," *IEEE Access*, vol. 7, pp. 38867–38879, 2018.

[14] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-IID data," 2018, *arXiv:1806.00582*.

[15] A. Abdi, S. Rashidi, F. Fekri, and T. Krishna, "Restructuring, pruning, and adjustment of deep models for parallel distributed inference," 2020, *arXiv:2008.08289*.

[16] J. Wang, Z. Chai, Y. Cheng, and L. Zhao, "Toward model parallelism for deep neural network based on gradient-free ADMM framework," in *Proc. IEEE Int. Conf. Data Min. (ICDM)*, 2020, pp. 591–600.

[17] Y. Liao, Y. Xu, H. Xu, L. Wang, and C. Qian, "Adaptive configuration for heterogeneous participants in decentralized federated learning," in *Proc. IEEE Conf. Comput. Commun.*, 2023, pp. 1–10.
[18] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 3043–3052.
[19] A. Krizhevsky, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009.
[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1–9.
[21] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 8026–8037.
[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[23] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–33.
[24] P. Zhou, Q. Lin, D. Loghin, B. C. Ooi, Y. Wu, and H. Yu, "Communication-efficient decentralized machine learning over heterogeneous networks," in *Proc. IEEE 37th Int. Conf. Data Eng. (ICDE)*, 2021, pp. 384–395.
[25] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-IID data with reinforcement learning," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 1698–1707.
[26] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," in *Proc. Eur. Symp. Artif. Neural Netw.*, vol. 3, 2013, p. 3.
[27] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*.
[28] S. Wang, M. Lee, S. Hosseinalipour, R. Morabito, M. Chiang, and C. G. Brinton, "Device sampling for heterogeneous federated learning: Theory, algorithms, and implementation," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.
[29] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 1935–1949, Mar. 2021.
[30] B. Luo, W. Xiao, S. Wang, J. Huang, and L. Tassiulas, "Tackling system and statistical heterogeneity for federated learning with adaptive client sampling," in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 1739–1748.
[31] Z. Li, T. Lin, X. Shang, and C. Wu, "Revisiting weighted aggregation in federated learning with neural networks," 2023, *arXiv:2302.10911*.
[32] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "BrainTorrent: A peer-to-peer environment for decentralized federated learning," 2019, *arXiv:1905.06731*.
[33] C. Hu, J. Jiang, and Z. Wang, "Decentralized federated learning: A segmented gossip approach," 2019, *arXiv:1908.07782*.
[34] Y. Li, C. Chen, N. Liu, H. Huang, Z. Zheng, and Q. Yan, "A blockchain-based decentralized federated learning framework with committee consensus," *IEEE Netw.*, vol. 35, no. 1, pp. 234–241, Jan./Feb. 2020.
[35] A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, "Fully decentralized federated learning," in *Proc. 3rd Workshop Bayesian Deep Learn. (NeurIPS)*, vol. 2, 2018, pp. 1–9.
[36] L. Chou, Z. Liu, Z. Wang, and A. Shrivastava, "Efficient and less centralized federated learning," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, 2021, pp. 772–787.
[37] E. T. M. Beltrán et al., "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 4, pp. 2983–3013, 4th Quart., 2023.

Jianchun Liu (Member, IEEE) received the Ph.D. degree from the School of Data Science, University of Science and Technology of China, Hefei, China, in 2022.

He is currently an Associate Researcher with the School of Computer Science and Technology, University of Science and Technology of China. His main research interests are software-defined networks, network function virtualization, edge computing, and federated learning.

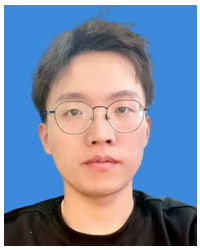
Dr. Liu is a member of ACM.

Yujia Huo received the B.S. degree from Nanjing Forest University, Nanjing, China, in 2023. He is currently pursuing the master's degree with the School of Computer Science, University of Science and Technology of China, Hefei, China.

His main research interests are edge computing and federated learning.

Pengcheng Qu received the B.S. degree from Shandong University, Jinan, China, in 2019. He is currently pursuing the master's degree with the School of Computer Science, University of Science and Technology of China, Hefei, China.

His main research interests are edge computing and federated learning.



Sun Xu received the B.S. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2022. He is currently pursuing the master's degree with the School of Computer Science, University of Science and Technology of China, Hefei, China.

His main research interests are quantum network and federated learning.

Zhi Liu (Senior Member, IEEE) received the Ph.D. degree in informatics from the National Institute of Informatics, Tokyo, Japan, in 2014.

He is currently an Associate Professor with The University of Electro-Communications, Tokyo. His research interests include video network transmission and mobile-edge computing.

Dr. Liu is currently an Editorial Board Member of *Wireless Networks* (Springer) and IEEE OPEN JOURNAL OF THE COMPUTER SOCIETY.

Qianpiao Ma received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China, Hefei, China, in 2014 and 2022, respectively.

He is currently a Postdoctoral Researcher with Purple Mountain Laboratories, Nanjing, China. His primary research interests include federated learning, mobile-edge computing, and distributed machine learning.

Jinyang Huang (Member, IEEE) received the Ph.D. degree from the School of Cyberspace Security, University of Science and Technology of China, Hefei, China, in 2022.

He is currently a Lecturer with the School of Computer and Information, Hefei University of Technology, Hefei. His research interests include wireless security and wireless sensing.

Dr. Huang is a TPC Member of IEEE ICME and Globecom. He is currently an Editorial Board Member of *Applied Sciences*.