YOGA: Adaptive Layer-Wise Model Aggregation for Decentralized Federated Learning

Jun Liu, Jianchun Liu, *Member, IEEE, ACM*, Hongli Xu, *Member, IEEE*, Yunming Liao, Zhiyuan Wang, and Qianpiao Ma

***Abstract*— Traditional Federated Learning (FL) is a promising paradigm that enables massive edge clients to collaboratively train deep neural network (DNN) models without exposing raw data to the parameter server (PS). To avoid the bottleneck on the PS, Decentralized Federated Learning (DFL), which utilizes peer-to-peer (P2P) communication without maintaining a global model, has been proposed. Nevertheless, DFL still faces two critical challenges, *i.e.*, limited communication bandwidth and not independent and identically distributed (non-IID) local data, thus hindering efficient model training. Existing works commonly assume *full* model aggregation at periodic intervals, *i.e.*, clients periodically collect models from peers. To reduce the communication cost, these methods allow clients to collect model(s) from selected peers, but often result in a significant degradation of model accuracy when dealing with non-IID data. Alternatively, the layer-wise aggregation mechanism has been proposed to alleviate communication overhead under the PS architecture, but its potential in DFL remains rarely explored yet. To this end, we propose an efficient DFL framework YOGA that adaptively performs layer-wise model aggregation and training. Specifically, YOGA first generates the ranking of layers in the model according to the learning speed and layer-wise divergence. Combining with the layer ranking and peers' status information (*i.e.*, data distribution and communication capability), we propose the max-match (MM) algorithm to generate the proper layer-wise model aggregation policy for the clients. Extensive experiments on DNN models and datasets show that YOGA saves communication cost by about 45% without sacrificing the model performance compared with the baselines, and provides 1.53-3.5× speedup on the physical platform.**

***Index Terms*— Decentralized federated learning, edge computing, heterogeneity, layer-wise aggregation.**

Manuscript received 24 June 2023; revised 7 October 2023; accepted 27 October 2023; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor G. Fanti. Date of publication 6 November 2023; date of current version 18 April 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB3301501; in part by the National Science Foundation of China (NSFC) under Grant 61936015, Grant 62132019, and Grant 62102391; in part by the Jiangsu Province Science Foundation for Youths under Grant BK20210122 and Grant BK20230275; and in part by Xiaomi Young Talents Program. *(Corresponding authors: Jianchun Liu; Hongli Xu.)*

Jun Liu, Jianchun Liu, Hongli Xu, Yunming Liao, and Zhiyuan Wang are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with the Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou, Jiangsu 215123, China (e-mail: lj18090759057@mail.ustc.edu.cn; jcliu17@ustc.edu.cn; xuhongli@ustc.edu.cn; liaoyun@mail.ustc.edu.cn; cswangzy@mail.ustc.edu.cn).

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

Digital Object Identifier 10.1109/TNET.2023.3329005

I. Introduction

THE emergence of the Internet of Things (IoTs) has led to a surge in generating of large amounts of data from ubiquitous devices, promoting the application of artificial intelligence (AI) [1], [2]. Traditional Federated Learning (FL) [3] enables a mass of edge clients to collaboratively train a model without sharing their data with the central server. Since only the models are required to be transmitted between the server and edge clients, FL can effectively safeguard user privacy.

The traditional FL relies on a parameter server (PS) to periodically receive/send the updated models from/to the local clients. However, the PS may become a bottleneck for training due to network congestion and pose security risks if compromised [4]. To overcome the problem of PS-based architecture, Decentralized Federated Learning (DFL) [5], which enables the clients to collaborate and learn from each other by exchanging model parameters directly, has been proposed. Since DFL does not require forwarding local models from clients to a central server, it eliminates the potential for a single point of failure or network congestion on the PS, thereby accelerating the training process.

Although DFL has demonstrated its advantages, it still faces the following two challenges for efficient training: (1) *Limited communication capability*. In contrast to data centers with sufficient communication resources, the available bandwidth between clients in edge networks is often limited [3], [6]. For example, the network bandwidth within WANs averagely ranges from 5 to 25 Mb/s [3], which is significantly lower than that within data centers (*e.g.*, over 10 Gb/s [7]). (2) *Non-independent and distributed (non-IID) data*. As the training data on individual clients always depends on their local environments and user preferences, the local data distributions will vary heavily among different clients [3]. For example, in the context of patient monitoring, the data from different patients demonstrates a notable level of heterogeneity stemming from a range of factors such as diverse human biological features, varying physical environments, and even sensor biases [8]. The non-IID data will significantly degenerate the convergence rate and even degrade the accuracy of trained models [2], [9] [10].

In the traditional DFL scheme, *e.g.*, D-PSGD [11], each client collects *full* models from all connected peers for model aggregation. However, this natural solution may result in excessive traffic among clients, which will cause significant communication delays. To address the aforementioned challenges, existing works [12], [13], [14], [15] carefully select peers, also called peer selection, based on data distribution or link speed to facilitate model convergence. In [12], Kong et al. identify that a large consensus distance (correlated to data distribution) is beneficial for decentralized training. In [13], Wang et al. propose CoCo to preferentially select peers with large divergence in data distribution to overcome the challenge of non-IID data. However, the communication delay in each round always depends on the slowest link, resulting in a long communication time and slow convergence speed. As a special case, some works [14], [15] allow each client to select only one peer for model aggregation to mitigate the impact of low-speed links and accelerate the training process. For example, NetMax [15] selects a peer with the highest-speed link to perform model aggregation. However, if a client has extremely high bandwidth and its peers choose to aggregate the model through the highest-speed link, it will lead to deviation from the global optimization direction. Thus, these methods suffer from poor model convergence and performance degradation when dealing with non-IID data [16]. In summary, the existing methods commonly assume that each client periodically collects *full* model(s) from the selected peer(s), and do not provide comprehensive solutions to address the aforementioned challenges.

To conquer the aforementioned challenges, we propose an efficient DFL framework, called YOGA, to adaptively perform la**y**er-wise m**o**del a**g**greg**a**tion, *i.e.*, each client collects various *layers* from all connected peers for model aggregation. At each round, YOGA first generates the ranking of layers of each client according to the learning speed and layer-wise divergence. Combined with the layer ranking and peers' status information (*i.e.*, data distribution and communication capability), YOGA then adaptively generates the proper layer-wise model aggregation policy for clients. Then, a client will collect layers from its peers and rebuild a combined model for model aggregation.

YOGA offers many advantages compared with the existing schemes. On the one hand, the bandwidth resource of the entire system can be effectively utilized, and the bandwidth pressure on each client is significantly alleviated. On the other hand, YOGA can accelerate model convergence and effectively cope with non-IID data. However, there remains one key challenge in building an effective layer-wise model aggregation framework: *how to adaptively generate a layer-wise model aggregation policy, i.e., pull the proper subset of layers from the connected peers, for each client?* The main contributions of this paper are summarized as follows:

- We propose an efficient DFL framework, called YOGA, which adaptively performs layer-wise aggregation and training to better overcome the challenges of limited communication capability and non-IID issues. Besides, we theoretically prove the convergence of YOGA.
- To tackle the challenges of system design, we present an efficient algorithm, termed Max-Match, which customizes the proper layer-wise model aggregation policy for the clients based on data distribution and communication capacity.
- We evaluate our proposed framework and algorithm through extensive test-bed experiments. The results show that YOGA provides 1.53-2.47$\times$ speedup on the physical platform and saves communication cost by about 45% without sacrificing the model performance compared with the baselines.

The paper is organized as follows. Section II provides the background and motivation for our research. Section III presents the design of YOGA, including the framework workflow and the proposed algorithm. Section IV provides a theoretical analysis and demonstrates the convergence of YOGA. Section V presents the experimental results and performance analysis. Section VI reviews related work in the field. Finally, Section VII concludes the paper.

II. Background and Motivation

A. Decentralized Federated Learning

DNN model. Modern artificial intelligence (AI) applications typically adopt deep neural network (DNN) models, which can achieve satisfactory model performance. Concretely, a specific DNN model is composed of a sequence of multiple layers with different types (*e.g.*, convolutional layers and fully-connected layers [17]). Let L be the number of layers of the DNN model $\theta = \{\theta(1), \theta(2), \ldots, \theta(L)\}$ and $|\theta(l)|$ be the size of $\theta(l), \forall l \in \{1, 2, \ldots, L\}$. Thus, the size of the whole model is denoted as $|\theta| = \sum_{l=1}^{L} |\theta(l)|$.

Network model. For a conventional DFL system, there is a set of edge clients $\mathcal{V} = \{1, 2, \ldots, N\}$, where $|\mathcal{V}| = N > 1$ is the number of clients. At the training stage, the clients collaboratively train the DNN model with its local private data by peer-to-peer (P2P) communications, *i.e.*, each client exchanges the model parameters with its connected peers in a network topology. We represent the decentralized communication topology with an undirected graph using the adjacency matrix $A \in \mathbb{R}^{N \times N}$. A is a symmetric doubly matrix: (i) $A_{ij} \in \{0, 1\}, \forall i, j$, (ii) $A_{ij} = A_{ji}, \forall i \neq j$ and $A_{ii} = 0, \forall i$, (iii) $\sum_j A_{ij} = |\sigma_i|, \forall i$, while σ_i is the set of connected peers of client i and $|\sigma_i|$ is the cardinality of the set. $A_{ij} = 1$ indicates the presence of a physical link between client i and client j, and vice versa.

Training procedure. Assume that model training will continue T rounds until the model convergence. At each round, the clients perform two main steps: local training and model aggregation. During the local model training, each client is associated with a local loss function based on the local dataset D_i, *i.e.*, $f_i(\theta_i) = \frac{1}{|D_i|} \sum_{\xi \in D_i} F_i(\theta_i; \xi)$, where θ_i is the model of the client i, ξ is a batch of data samples in D_i, and $F_i(\theta_i; \xi)$ is the loss over ξ. In general, client i updates the local model through stochastic gradient descent (SGD) [18] to minimize its local objective function $f_i(\theta_i)$. Specifically, the client updates its local model by $\hat{\theta}_i = \theta_i - \eta \nabla f_i(\theta_i)$, where $\hat{\theta}_i$ is the updated local model of the client i, η is the learning rate, and $\nabla f_i(\theta_i)$ is the gradient of the loss for the current model θ_i. After local training, each client collects model(s) from the connected (or selected) peers and then aggregates the received model(s) with the local model. When all clients have completed the aggregation, the client will then perform the next round of local training with the fresh aggregated model.

B. Layer-Wise Aggregation

The growing parameter size of DNN models poses a communication challenge in edge networks. Deep learning has evolved rapidly during the past few years. With advancements in DNN model performance to accommodate more complex tasks, the model size increases dramatically. For example, AlexNet [19] contains 61M parameters, and VGG [20] contains 138M parameters for image classification. Recently, the parameter size of the DNN model has grown exponentially (*e.g.*, The number of parameters in a DNN surpasses hundreds of billions [21].) In contrast, the communication capability of clients in the edge network is strictly limited (typically bandwidth is only about 5∼25Mb/s [3]). What's worse, training a model over federated learning requires frequent cross-client communications, which further exacerbates the dilemma of clients' limited communication capability. Thus, training large DNN models efficiently over FL requires minimizing clients' communication overhead, particularly for DFL [11], which involves more complex communication than centralized federated learning.

Layer-wise aggregation is a better alternative. In order to reduce the communication cost, Lee et al. [22] propose a layer-wise model aggregation scheme, called FedLAMA, which finely controls the aggregation interval of layers in PS-based FL. In [23], Ma et al. propose pFedLA, that selectively aggregates the layers contributing more to the model convergence while skipping the less important ones under PS architecture. For DFL, Tang et al. have proposed GossipFL [24], which allows clients to accelerate DFL model training by communicating with a highly sparse model from a single peer. Besides, Hu et al. have presented Combo [25], a segmented gossip method designed to leverage node-to-node bandwidth efficiently while maintaining strong training convergence. In addition, Barbieri et al. have introduced CFL-LS [26], a method for client-driven DNN fragment selection. It uses a layer optimizer to rank layers based on their impact on model quality (measured by a normalized squared gradient of local loss) and incorporates a fairness scheme for balanced parameter selection and enhanced performance. However, all these aforementioned works have neither considered the convergence patterns within the model (*e.g.*, different layers of a DNN model have distinct learning speeds and divergences) nor addressed the impact of non-IID issue. These three observations motivate us to adopt layer-wise model aggregation in DFL:

- *Observation 1: The parameter size of layers in a model varies dramatically [20], [27].* The parameter size of different layers in a model is not the same, and the difference may be orders of magnitude [20], [27].
- *Observation 2: The functionality of different layers in a DNN model is not the same [20], [23], [27].* In a conventional DNN model, each layer serves a unique purpose. Shallow layers are primarily responsible for local feature extraction, while deeper layers are designed to extract more global features [20], [23], [27].
- *Observation 3: Different layers of a DNN model have distinct learning speeds and divergences [22], [28].* All layers in the model have different learning speeds to converge to the final representation and various degrees of model discrepancy [28].

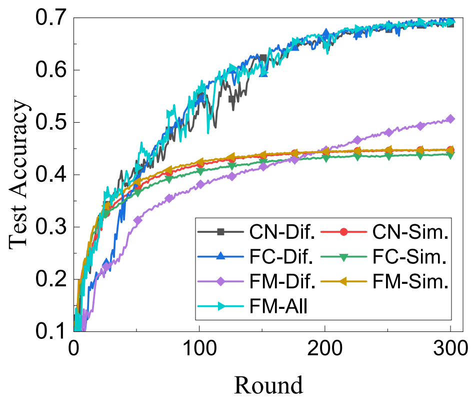


Fig. 1. The test accuracy for AlexNet on CIFAR-10 with three model aggregation strategies.

C. Impact of Non-IID Data on Layer-Wise Aggregation

A natural approach to perform layer-wise aggregation is randomly pulling the specified layers from different peers to form a complete model and aggregating it with the local model. However, this naive approach may result in accuracy degradation under non-IID setting. Our observation indicates that the data distributions will significantly impact the convergence progress and the final performance of the model trained by layer-wise aggregation. Intuitively, by aggregating the layers from the peers with different data distributions, the local model on the client will learn more information about the global data distribution, which can well deal with the non-IID issue [29].

We further conduct a set of experiments with three model aggregation methods (*i.e.*, *CN*, *FC*, and *FM*) with *similar* (*Sim*) and *different* (*Dif*) data distributions, corresponding to the IID and non-IID data, to demonstrate the impact of non-IID data. For example, *CN-Sim* denotes that the client only aggregates the convolutional layers of all peers with similar data distribution at each round. In addition, we conduct *FM-All* allowing each client to receive models from all peers (both similar and different data distributions) for comparison. The empirical results in Fig. 1 demonstrate that:

- Aggregating layers (models) among the clients with heterogeneous data distributions yields a discernible improvement in model convergence speed and final accuracy performance. For instance, training AlexNet with *CN-Dif*, *FC-Dif*, and *FM-Dif* exhibits better model convergence than *CN-Sim*, *FC-Sim*, and *FM-Sim*, respectively. Besides, *CN-Dif* and *FC-Dif* have a similar convergence process as *FM-All*, while significantly reducing the communication cost. These results indicate that layer-wise aggregation is correlated with data distribution.

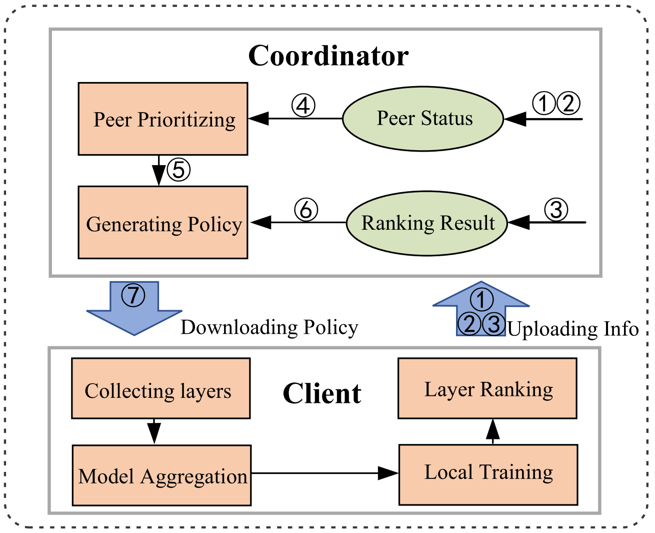


Fig. 2. System overview of YOGA.

Aggregating the layers from the peers with different data distributions facilitates model convergence.

- Training with layer-wise aggregation strategy shows better performance than full model aggregation strategy when peers have heterogeneous data distributions, bringing a significant reduction in communication cost. For example, *CN-Dif* and *FC-Dif* show better final accuracy performance than *FM-Dif* after the same number of training rounds. *CN-Dif* and *FC-Dif* reduce the communication cost by 41% and 59% compared with *FM-Dif*, respectively.

These results suggest that the layer-wise aggregation can handle non-IID without sacrificing the model performance. Therefore, the key challenge of layer-wise model aggregation and training is *how can we generate a proper layer-wise aggregation policy for each client to address the non-IID issue and fully utilize the limited communication capability?*

III. Design of YOGA

A. System Overview

In this section, we give a brief introduction to our proposed system YOGA, which mainly involves two key roles, *i.e.*, the coordinator and a set of clients. In YOGA, the coordinator will periodically collect the network status information (*e.g.*, available bandwidth and data distribution) on the clients and generate a layer-wise aggregation policy. On receiving the policy from the coordinator, each client pulls the specified layers from the connected peers for model aggregation and training. The system overview is presented in Fig. 2. The whole process involves several training rounds until model convergences. In each round, YOGA mainly consists of the following steps:

Local Training. At each round, the clients perform local training by aggregating the local model and the received layers from the selected peers. Once the local training has finished after a given number (*e.g.*, $\mathcal{T}$) of rounds, the clients upload the network bandwidth (①), current data distribution information (②) and the layer ranking (③) to the coordinator for a further operation (*e.g.*, layer-wise aggregation policy generation).

Layer Ranking. According to Section II-B, it is evident that the proficiency of different layers in a model varies significantly. The impact of each layer on the training process varies significantly. Therefore, YOGA ranks layers of a DNN model based on their learning speed and discrepancy, including the parameter size, to generate the aggregation policy.

Peer Prioritizing. Based on ① and ②, YOGA prioritizes the client's connected peers (④). The peer with more available bandwidth and larger data distribution differences will be assigned a higher priority. To fully utilize the limited communication capability and handle the non-IID issue, the higher ranking layer (*i.e.*, layer with a faster learning speed or more parameters) should be preferentially pulled from the peer with a higher priority.

Generating Aggregation policy. Combined with the peers' priorities (⑤) and layer ranking (⑥), YOGA generates layer-wise model aggregation policy for each client utilizing the max-match algorithm, which will be introduced in Section III-D. Then YOGA sends the policy back to the client (⑦) to indicate layer-wise aggregation.

Performing Model Aggregation. Based on the layer-wise aggregation policy from the coordinator, each client adaptively collects different layers from various peers to perform layer-wise model aggregation. The client rebuilds a combined model after collecting layers from peers, and performs model aggregation. This aggregated model is used for further training in the next round.

B. Layer Ranking

Different layers of a DNN model converge at inconsistent speeds [30], [31]. For example, Brock et al. [32] have shown that the shallow layers converge faster than the deep layers. Previous studies mostly adopt gradient magnitude as an indicator of model convergence, but it may not always be a reliable measure [33]. On the one hand, small gradients do not necessarily mean that the parameters are close to the optimum, *e.g.*, they could indicate the presence of saddle points. On the other hand, large gradients do not necessarily indicate fast convergence. Parameters may oscillate around the optimum due to the variance of stochastic optimization [34], *e.g.*, SGD. Since gradients within the same layer have similar magnitudes [33], and the inconsistencies of gradients are mainly between layers [31], we adopt the layer-wise learning speed [30] $P_i^t = \{P_{i,1}^t, P_{i,2}^t, \ldots, P_{i,L}^t\}$ to indicate the convergence status of layers on client i, where

$$P_{i,l}^t = \frac{\left\|\sum_{\Delta=0}^{r-1} \Lambda_l^{t-\Delta}\right\|}{\epsilon + \sum_{\Delta=0}^{r-1} \left\|\Lambda_l^{t-\Delta}\right\|}, \quad \forall l. \tag{1}$$

We use t to denote the training round, and $\Lambda_l^t = \theta_i^t(l) - \theta_i^{t-1}(l)$ is the update of layer l at round t and r is the observation window. A large observation window r indicates that the learning speed takes into account more updates from previous rounds. The symbol ϵ represents a small positive value that is commonly used to avoid division-by-zero errors. The range of $P_{i,l}^t$ is between 0 and 1, reflecting the convergence status of the layer at round t on client i. Generally, a small learning speed of a layer suggests that the layer is close to its optimum, and vice versa [30]. For example, if the gradients are consistently pointing in the same direction within the observation window $[t-r+1, t]$, then $P^t_{i,l} = 1$, indicating that the learned layer is far from its optimum. Conversely, if the updates in the observation window completely cancel each other out (*i.e.*, the parameters oscillate around a point), then $P^t_{i,l} = 0$, indicating that the parameters of layer l are close to the optimum.

The parameter size of layers in a model varies dramatically. To complement the layer-wise learning speed, we calculate the layer-wise discrepancy of the layers as an essential indicator for layer ranking, *i.e.*,

$$\Psi^t_{i,l} = \left\|\theta^t_i(l) - \theta^{t-1}_i(l)\right\|, \quad \forall l \tag{2}$$

where $\Psi^t_{i,l}$ denotes the discrepancy of layer l at training round t. The layer-wise discrepancy $\Psi^t_{i,l}$ captures both the divergence degree between layers and the parameter size of the layers, providing a comprehensive measure. For example, when two layers show similar convergence degrees, the layer with more parameters exhibits a greater layer-wise discrepancy compared to the other layer. Besides, a higher layer-wise discrepancy also indicates that the layer has not yet converged and requires further optimization.

Combined with layer-wise learning speed $P^t_{i,l}$ and discrepancy $\Psi^t_{i,l}$, YOGA computes the layer-wise priority $\mathcal{P}^t_i = \{\mathcal{P}^t_{i,1}, \mathcal{P}^t_{i,2}, \ldots, \mathcal{P}^t_{i,L}\}$, where

$$\mathcal{P}^t_{i,l} = \frac{P^t_{i,l} + \Psi^t_{i,l}}{2}, \quad \forall l. \tag{3}$$

Further, we rank the layers of the model according to the layer-wise priorities, and obtain the layer ranking $\hat{\mathcal{P}}^t_i = \{r_1, r_2, \ldots, r_L\}$, where $r_j \in \{1, \cdots, L\}$ is the index of layer $\theta^t_i(r_j)$ ranks j^{th} among all layers in the model.

C. Peer Prioritizing

Peer Status. The coordinator periodically receives and maintains the status information of clients uploaded by clients. Formally, we represent the local data distribution with a vector $d_i = \{d^1_i, d^2_i, \ldots, d^C_i\}, \forall i \in N$, where $C = |d_i|$ is the total number of classes in the learning problem and $d^c_i, \forall c \in \{1, 2, \ldots, C\}$ is the proportion of class c in the local data samples in D_i. Thus, the distribution divergence over classes between the local datasets of the client i and its peer j can be represented as [35]:

$$\mathbb{D}_{i,j} = \sum_{c=1}^{C} \|d^c_i - d^c_j\|. \tag{4}$$

Intuitively, by periodically aggregating the local model from the peer j with higher distribution divergence $\mathbb{D}_{i,j}$, the local model of client i can learn more knowledge from its peers so that the learned local model has a good representative of the global model. Moreover, we normalize the distribution divergence $\hat{\mathbb{D}}_i = \{\hat{\mathbb{D}}_{i,1}, \hat{\mathbb{D}}_{i,2}, \ldots, \hat{\mathbb{D}}_{i,|\sigma_i|}\}$ of connected peers for peer prioritizing, where

$$\hat{\mathbb{D}}_{i,j} = \frac{\mathbb{D}_{i,j}}{\sum_{k\in\sigma_i} \mathbb{D}_{i,k}}, \quad j \in \{1, 2, \ldots, |\sigma_i|\}. \tag{5}$$

To efficiently collect layers from peers with limited communication capability, it is necessary to prioritize peers based on their current bandwidth. This allows for the most efficient use of available network resources for layer collection. We use $\mathbb{B}_i$ to denote current bandwidth of client i and use $p^t_i = \{p^t_{i,1}, p^t_{i,2}, \ldots, p^t_{i,|\sigma_i|}\}$ to indicate the bandwidth of peers of client i at round t, where

$$p^t_{i,j} = \frac{\mathbb{B}^t_j}{\sum_{k\in\sigma_i} \mathbb{B}^t_k}, \quad j \in \{1, 2, \ldots, |\sigma_i|\} \tag{6}$$

Prioritizing the connected peers. Based on data distribution divergence $\hat{\mathbb{D}}_{i,j}$ and available bandwidth p^t_i at round t, YOGA calculates the connected peers' priorities $\mathbb{P}^t_i = \{\mathbb{P}^t_{i,1}, \mathbb{P}^t_{i,2}, \ldots, \mathbb{P}^t_{i,|\sigma_i|}\}$ of client i, where

$$\mathbb{P}^t_{i,j} = \hat{\mathbb{D}}_{i,j} \cdot w_p + p^t_{i,j} \cdot (1 - w_p), \quad j \in \{1, 2, \ldots, |\sigma_i|\} \tag{7}$$

and w_p is the weight parameter to balance the impact of data distribution and bandwidth, with the default value of $w_p = 0.5$. $\mathbb{P}^t_i$ represents the sequence of priorities assigned to all connected peers of client i at round t. To generate the layer-wise aggregation policy, YOGA ranks and normalizes the peers' priorities, *i.e.*, $\hat{\mathbb{P}}^t_i = \{\hat{\mathbb{P}}^t_{i,1}, \hat{\mathbb{P}}^t_{i,2}, \ldots, \hat{\mathbb{P}}^t_{i,|\sigma_i|}\}$, where $\hat{\mathbb{P}}^t_{i,j} > \hat{\mathbb{P}}^t_{i,j+1}$ and

$$\hat{\mathbb{P}}^t_{i,j} = \frac{\mathbb{P}^t_{i,j}}{\sum_{k\in\sigma_i} \mathbb{P}^t_{i,k}}. \tag{8}$$

D. Generating Aggregation Policy

Based on the priorities of layers $\hat{\mathcal{P}}^t_i = \{\hat{\mathcal{P}}^t_{i,1}, \hat{\mathcal{P}}^t_{i,2}, \ldots, \hat{\mathcal{P}}^t_{i,L}\}$ and the connected peers $\hat{\mathbb{P}}^t_i = \{\hat{\mathbb{P}}^t_{i,1}, \hat{\mathbb{P}}^t_{i,2}, \ldots, \hat{\mathbb{P}}^t_{i,|\sigma_i|}\}$, YOGA generates the layer-wise aggregation policy using a greedy-based algorithm, termed max-match (MM), as described in Algorithm1. Specifically, according to the index γ of layers in $\hat{\mathcal{P}}^t_i$ and the priority $\hat{\mathbb{P}}^t_{i,j}$ of peer j, MM firstly generates the range of the layers that should be collected from peer j (Line 2). Then, based on $\hat{\mathcal{P}}^t_i$, MM gets the corresponding layers and generates the layers that client i will pull from peer j at round t (Line 3). We use the upper bound of $\hat{\mathbb{P}}^t_{i,j} \times L$ (*i.e.*, $\lceil\hat{\mathbb{P}}^t_{i,j} \times L\rceil$ (where $\lceil\cdot\rceil$ is the round-up operator) to generate the number of layers should be pulled from peer j (Line 4). YOGA allows overlap layers between two connected peers when $\lceil\hat{\mathbb{P}}^t_{i,j} \times L\rceil$ is not an integer, and overlap only happens between two peers with consecutive priority, YOGA allows overlapping layers between two connected peers when the result of $\lceil\hat{\mathbb{P}}^t_{i,j} \times L\rceil$ is not an integer. This overlap only occurs between two peers with consecutive priorities, *e.g.*, $\Gamma^t_1 \bigcup \Gamma^t_2 \neq \phi$, where ϕ denotes the empty set. Peer with higher priority gets more layers, which ranks higher among layers in the local model. We then update the index γ for the following generation. The layer-wise aggregation policy $\Gamma^t(i)$ specifies the peers j and corresponding layers $\Gamma^t_j(i)$ that client i should adopt for layer-wise model aggregation, *i.e.*,

$$\Gamma^t(i) = \{\Gamma^t_1(i), \Gamma^t_2(i), \ldots, \Gamma^t_{|\sigma_i|}(i)\}. \tag{9}$$

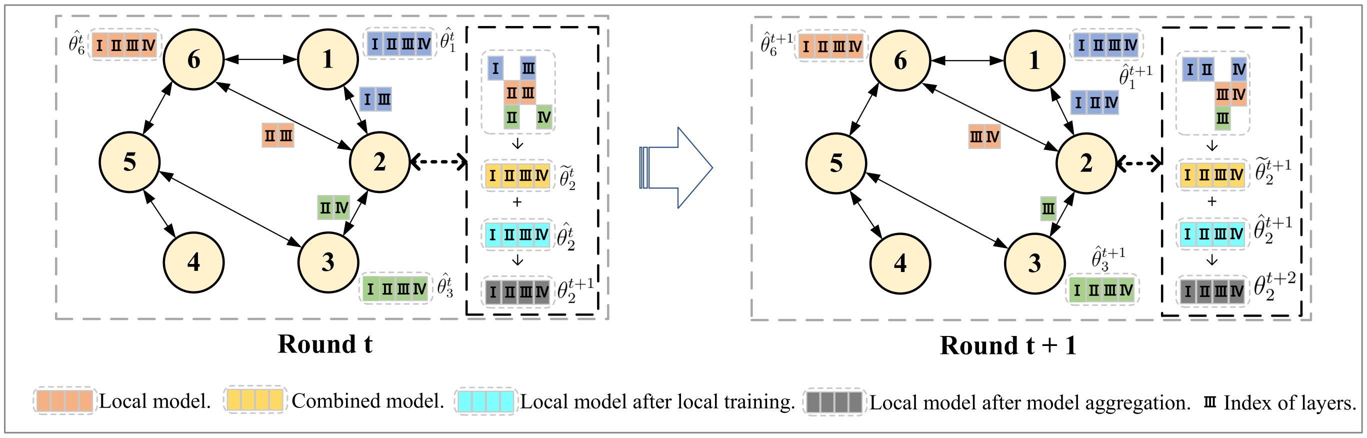

Round t
Round t + 1
Local model.
Combined model.
Local model after local training.
Local model after model aggregation.
Index of layers.

Fig. 3. Illustration of layer-wise model aggregation in YOGA.

Algorithm 1 Greedy-Based Max-Maxth

```
Require: The ranking of layers $\hat{\mathcal{P}}_i^t$; the priority of connected peers $\hat{\mathbb{P}}_i^t$; the set of connected peers $\sigma_i$; the number of the layers in current model $L$.
Ensure: The layer-wise model aggregation policy $\Gamma^t(i)$.
    Initialize index $\gamma$ of layers in $\hat{\mathcal{P}}_i^t$: $\gamma = 1$
1: for each peer $j \in \sigma_i$ do
2:     Calculate the range of layers from peer $j$: $s_j = \gamma$, $e_j = min(s_j + \lceil \hat{\mathbb{P}}_{i,j}^t \times L \rceil, L)$.
3:     Get corresponding indices of layers according to the ranking of layers $\hat{\mathcal{P}}_i^t$: $\{r_{s_j}, \cdots, r_{e_j}\} = \{\hat{\mathcal{P}}_i^t[s_j], \cdots, \hat{\mathcal{P}}_i^t[e_j]\}$.
4:     Generate layers pulling from peer $j$: $\Gamma_j^t(i) = \{\theta_j^t(r_{s_j}), \cdots, \theta_j^t(r_{e_j})\}$
5:     Update index: $\gamma = \gamma + \lfloor \hat{\mathbb{P}}_{i,j}^t \times L \rfloor$
6: end for
```

$\Gamma_j^t(i), \forall j \in \sigma_i$ is a bunch of layers that client i pulls from connected peer j, *i.e.*,

$$\Gamma_j^t(i) = \{\theta_j^t(s), \ldots, \theta_j^t(e)\}, \quad 0 < s,\ e \le L \tag{10}$$

where the subscript s and e is determined by both layer ranking on client i and priority of peer j.

E. Performing Model Aggregation

Based on the layer-wise aggregation policy, the client collects layers from peers and incorporates them into a new combined model $\widetilde{\theta}_i^t$, which is a full DNN model. For some layers received from more than one peer, the layers in the combined model $\widetilde{\theta}_i^t$ are the aggregation of the same layers by layer-wise averaging [22]. Then, the client aggregates its local model $\hat{\theta}_i^t$ after local training with the combined model $\widetilde{\theta}_i^t$. We use an averaging approach to aggregate models, *i.e.*,

$$\theta_i^{t+1} = \hat{\theta}_i^t \cdot w_\theta + \widetilde{\theta}_i^t \cdot (1 - w_\theta) \tag{11}$$

where w_θ is used for aggregating the combined model and the local model, with the default value of $w_\theta = 0.5$. After aggregation, client i will perform the next round of local training. We provide an example below to better illustrate the training process of YOGA. As shown in Fig. 3, we take client 2 as an example. At the round t, client 2 utilizes its local dataset to update the local model θ_2^t and gets updated model $\hat{\theta}_2^t$. After the update is completed, it will upload some key information to the coordinator to generate an aggregation policy. Once the client receives the aggregation policy from the coordinator, it actively pulls the layer parameters of the model from the corresponding peer. For example, as shown in the left subgraph of Fig. 3, at round t, client 2 will pull layers 1 and 3 from client 1, layers 2 and 4 from client 3, and layers 2 and 3 from client 6. After all the layer parameters are received, these parameters are combined to create a hybrid model $\widetilde{\theta}_2^t$, which will be aggregated with the local model $\hat{\theta}_2^t$. The client will continue training with the aggregated model until convergence.

IV. Convergence Analysis

This section provides an analysis of the convergence of the YOGA algorithm. Throughout this paper, we make the following commonly used assumptions:

Assumption 1: **Lipschitzian gradient.** The loss function f_i is with $\mathcal{L}$-Lispschitzian gradients, $\forall \theta_1, \theta_2$, *i.e.*,

$$\|\nabla f_i(\theta_1) - \nabla f_i(\theta_2)\|_2 \le \mathcal{L} \|\theta_1 - \theta_2\|_2, \quad \forall i \tag{12}$$

where $\|\cdot\|_2$ is the L_2 norm of vector.

Assumption 2: **Unbiased estimation.** For each client i, we assume that the expectation of the stochastic gradient of F on each data sample is equal to $f_i(\theta)$, *i.e.*,

$$\mathbb{E}_{\xi \sim D_i} \nabla F_i(\theta; \xi) = \nabla f_i(\theta). \tag{13}$$

And further, for all clients, we assume that,

$$\mathbb{E}_{i \in \mathcal{V}} \nabla f_i(\theta) = \nabla f(\theta). \tag{14}$$

Assumption 3: **Bounded variance.** We assume the variance of stochastic gradients $\mathbb{E}_{i \in \mathcal{V}} \mathbb{E}_{\xi \in D_i} \|\nabla F_i(\theta; \xi) - \nabla f(\theta)\|^2$ is bounded for any θ with i uniformly sampled from $\mathcal{V}$ and ξ from the distribution D_i, *i.e.*

$$\mathbb{E}_{\xi \sim D_i} \|\nabla F_i(\theta; \xi) - \nabla f_i(\theta)\|^2 \le \varrho^2, \quad \forall i, \theta \tag{15}$$

$$\mathbb{E}_{i \in V} \|\nabla f_i(\theta) - \nabla f(\theta)\| \le \varsigma^2, \quad \forall \theta. \tag{16}$$

Note that if the local dataset on each client is homogeneously distributed, then $\varsigma = 0$.

Assumption 4: **Connected topology.** We assume that the global topology G in YOGA is a connected topology.

We prove the convergence of YOGA in two steps focusing on an arbitrary client i. Firstly, we study the model divergence between the model before model aggregation and the model after model aggregation. Then, followed by the previous work [11], we obtain the expectation of gradient after T training rounds are bounded.

Theorem 1: After T training round, the average gradient $\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f(\theta^t)\right\|^2$ is bounded, *i.e.*,

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f(\theta^t)\right\|^2 \le \frac{4N(f(\theta^0)-f(\theta^*)}{4N\eta(1-\eta)-1} + \frac{4N}{4N\eta(1-\eta)-1}(\frac{\mathcal{L}+2N}{8N^2}\alpha^2 + \frac{\beta^2}{4N} + \frac{\eta^2\mathcal{L}\varrho^2}{2N}), \tag{17}$$

where $\eta = (\frac{1}{2\mathcal{L}+\varrho\sqrt{T/N}})^3$, and T is sufficiently large.

Proof: We use the α^2 to denote the upper bound of model divergence of the combined model collected from peers $\widetilde{\theta}_i^t$ and local model after local training $\hat{\theta}_i^t$, *i.e.*, $\|\widetilde{\theta}_i^t - \hat{\theta}_i^t\| \le \alpha^2, \forall t, i$. According to Assumption 3, we use β^2 to represent the upper bound of model divergence between local model before and after local training, *i.e.*, $\|\nabla f(\theta^t)\|^2 - \|\nabla f(\hat{\theta}^t)\|^2 \le \beta^2, \forall t$.

Based on the model aggregation in (11), we study the divergence of the local model before aggregation and the model after aggregation is bounded, *i.e.*,

$$\begin{aligned}\mathbb{E}[f(\theta^{t+1} - f(\hat{\theta}^t))] &= \mathbb{E}[f(\hat{\theta}^t - \frac{1}{2N}(\widetilde{\theta}_t^i - \hat{\theta}_i^t) - f(\hat{\theta}^t))] \\ &\le -\frac{1}{2N}\mathbb{E} < \nabla f(\hat{\theta}^t), \widetilde{\theta}_t^i - \hat{\theta}_i^t > \\ &\quad + \frac{\mathcal{L}}{8N^2}\|\widetilde{\theta}_t^i - \hat{\theta}_i^t\|^2 \\ &= -\frac{1}{4N}(\|\nabla f(\hat{\theta}^t) + \widetilde{\theta}_t^i - \hat{\theta}_t^i\|^2 - \|\nabla f(\hat{\theta}^t)\|^2 \\ &\quad - \|\widetilde{\theta}_t^i - \hat{\theta}_t^i\|^2) + \frac{\mathcal{L}}{8N^2}\|\widetilde{\theta}_t^i - \hat{\theta}_i^t\|^2 \\ &\le \frac{1}{4N}\mathbb{E}\|\nabla f(\theta^t)\|^2 + \frac{\mathcal{L}+2N}{8N^2}\alpha^2 + \frac{\beta^2}{4N}\end{aligned} \tag{18}$$

Followed by previous work in [11], we derive the bound of $\mathbb{E}[f(\hat{\theta}^t) - f(\theta^*)]$, *i.e.*,

$$\mathbb{E}[f(\hat{\theta}^t) - f(\theta^*)] \le \mathbb{E}[f(\theta^t - \theta^*)] - (\eta + \frac{\eta^2}{2})\mathbb{E}\|\nabla f(\theta^t)\|^2 + \frac{\eta^2\mathcal{L}\varrho^2}{2N} \tag{19}$$

By adding Eq. (18) and Eq. (19) and rearranging, we obtain the convergence bound between two consecutive training rounds, *i.e.*,

$$\mathbb{E}[f(\theta^{t+1}) - f(\theta^t)] \le (\frac{1}{4N} - \eta + \eta^2)\mathbb{E}\|\nabla f(\theta^t)\|^2 + \frac{\mathcal{L}+2N}{8N^2}\alpha^2 + \frac{\beta^2}{4N} + \frac{\eta^2\mathcal{L}\varrho^2}{2N}. \tag{20}$$

Finally, by summing the results in (20) during the whole training process (from $t = 0$ to $T - 1$), we have

$$\begin{aligned}\sum_{t=0}^{T-1}\mathbb{E}[f(\theta^{t+1}) - f(\theta^t)] &= \mathbb{E}[f(\theta^*) - f(\theta^0)] \\ &\le T((\frac{1}{4N} - \eta + \eta^2)\mathbb{E}\|\nabla f(\theta^t)\|^2 \\ &\quad + \frac{\mathcal{L}+2N}{8N^2}\alpha^2 + \frac{\beta^2}{4N} + \frac{\eta^2\mathcal{L}\varrho^2}{2N}).\end{aligned} \tag{21}$$

Dividing both sides of (21) by T and rearranging the inequality above, we obtain:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\|\nabla f(\theta^t)\right\|^2 \le \frac{4N(f(\theta^0)-f(\theta^*)}{4N\eta(1-\eta)-1} + \frac{4N}{4N\eta(1-\eta)-1}(\frac{\mathcal{L}+2N}{8N^2}\alpha^2 + \frac{\beta^2}{4N} + \frac{\eta^2\mathcal{L}\varrho^2}{2N}), \tag{22}$$

which completes the proof of Theorem 1.

The results of layer ranking and peer prioritizing are inputs of the proposed MM algorithm, ensuring that the local model closely approximates the global model, leading to smaller values of α^2 and β^2, thereby guaranteeing faster model convergence. The convergence bound is associated with α^2 and β^2, which are determined by both the local model and the combined model. The combined model is composed of different layers pulled by the client from various peers, reflecting the optimization direction of the global model, whereas the local model reflects the optimization direction of the local model. YOGA utilizes the MM algorithm, derived from layer ranking and peer priority, ensuring that the local model closely approximates the global model. Consequently, α^2 and β^2, computed from the local model and the mixed model, remain within a relatively narrow range, guaranteeing the convergence upper bound and thus accelerating the model convergence process.

V. Performance Evaluation

A. Datasets and Models

Datasets: We adopt three real-world classical datasets to conduct extensive experiments:

- **EMNIST [36]** comprises a collection of handwritten characters, including 731,668 training samples and 82,587 test samples. It encompasses 62 categories, which include 10 digits and 52 characters encompassing lowercase and uppercase letters.
- **CIFAR-10 [37]** is composed of 60,000 color images with a resolution of 32 × 32 pixels. It is divided into 10 classes, each containing 6,000 images. The dataset is further split into 50,000 training images and 10,000 test images.
- **ImageNet [38]** is a visual recognition dataset consisting of 1,281,167 training images, 50,000 validation images, and 100,000 test images, spread across 1,000 categories. To accommodate the limited resources of edge clients,

we have created a subset called ImageNet-100. This subset focuses on 100 categories selected from the original 1,000 categories. Furthermore, each sample in ImageNet-100 is resized to a shape of $64 \times 64 \times 3$, optimizing it for efficient processing on edge devices.

To simulate the non-IID setting, we generate synthetic non-IID datasets with varying levels of class distribution skews, as described in [2] and [35], *e.g.*, a single client may have a larger proportion of data for one or a few specific classes compared to others. Specifically, we divide a unique class into equal portions among five clients, with a proportion p (e.g., 0.1, 0.2, 0.4, 0.6, and 0.8). The remaining samples of each class are then evenly distributed among the other clients. Consequently, the non-IID levels of a dataset are denoted as $p = 0.1, 0.2, 0.4, 0.6$ and 0.8, respectively. When $p = 0.1$, samples are distributed uniformly and the distribution of the training dataset is IID for 50 clients. For fair comparisons, the full test datasets are used across all clients.

Models: To evaluate the performance, three different DNN models with distinct structures are implemented on the aforementioned datasets:

- **CNN on EMNIST.** The plain CNN model [3], tailored for the EMNIST dataset, comprises two convolutional layers of size 5×5, a fully connected layer with 512 units, and a softmax output layer with 62 units.
- **AlexNet on CIFAR-10.** The AlexNet model [19] is an 8-layer deep neural network (DNN) model adopted for the CIFAR-10 dataset. It consists of three 3×3 convolutional layers, one 7×7 convolutional layer, one 11×11 convolutional layer, two fully-connected hidden layers, and a fully-connected output layer.
- **VGG-16 on ImageNet-100.** The famous VGG-16 model [20], which consists of 13 convolutional layers with a kernel of 3×3, followed by two dense layers and a softmax output layer, is adopted for the ImageNet-100 dataset.

B. Baselines and Metrics

Baselines: We adopt three classical algorithms as baselines:

- **GossipFL** [24] enables each client to communicate with just one peer during each communication round, exchanging a highly compressed model.
- **CFL-LS** [15] independently selects fragments of the DNN to be shared with peers at each round. The selection process relies on a local optimizer that balances model quality with sideline communication resources.
- **Rand-YOGA** pulls each layer of a DNN model from a randomly selected peer and performs the same layer-wise model aggregation process as YOGA. This baseline is used to demonstrate the effectiveness of the MM algorithm.

Metrics: We utilize the following three metrics to assess the performance of YOGA and the baseline algorithms.

- **Test accuracy.** At each training round, we evaluate the local model on the test dataset by measuring the proportion of correctly predicted data. To be specific, we calculate the mean test accuracy of all clients' models on the test datasets.
- **Completion time.** The completion time, which refers to the total training duration required to attain the desired test accuracy, is recorded and utilized to assess the training speed.
- **Communication traffic.** The total communication cost for transmitting the model (or layers) is recorded upon reaching the target test accuracy. This metric reflects the amount of data exchanged during the training process.

TABLE I
TECHNICAL SPECIFICATIONS OF JETSON DEVELOPER KITS

	AI Performance	GPU Type
Jetson TX2	1.33 TFLOPS	256-core Pascal
Jetson NX	21 TOPS	384-core Volta
AGX Xavier	22 TOPS	512-core Volta
	CPU Type	**ROM**
Jetson TX2	Denver 2 and ARM 4	8 GB LPDDR4
Jetson NX	6-core Carmel ARM 8	8 GB LPDDR4x
Jetson AGX	8-core Carmel ARM 8	32 GB LPDDR4x
	CPU Frequency	**GPU Frequency**
Jetson TX2	2.0GHz	1.12GHz
Jetson NX	1.9GHz	1.1GHz
Jetson AGX	2.2GHz	1.37GHz

TABLE II
TRAINING DETAILS OF DIFFERENT MODELS/DATASETS

Model/Dataset	η	η_{min}	decay rate
CNN/EMNIST	0.1	0.001	0.993
AlexNet/CIFAR-10	0.1	0.001	0.98
VGG-16/ImageNet-100	0.1	0.001	0.993
Model/Dataset	**local iteration**	**batch size**	**epoch**
CNN/EMNIST	30	32	200
AlexNet/CIFAR-10	50	32	300
VGG-16/ImageNet-100	50	64	300

C. Experimental Settings

Test-bed Setup To better resemble the DFL setting, we evaluate the performance of YOGA through physical experiments. The test-bed experiments are performed on 20 NVIDIA Jetson TX2, 20 NVIDIA Jetson Xavier NX, and 10 NVIDIA Jetson AGX Xavier Developer Kits, where we randomly select a client as the coordinator. More details of the Jetson devices can be found in TABLE III. We configure Jetson devices to effectively operate with varying models and datasets. The experimental network is set up using a router, allowing clients to connect through wireless links. Our software implementation utilizes Docker Swarm for distributed software development [39], [40], and PyTorch deep learning framework [41]. Docker Swarm-based distributed software development makes it easier to build a distributed scheme and detect the working status of each client. The PyTorch deep learning framework enables clients to train the models efficiently. Besides, the communication among clients is facilitated by the MPI (Message Passing Interface) [42], which

TABLE III

The Communication Cost of Four Schemes Under IID Setting When Achieving Target Accuracies, *e.g.*, 85% on EMNIST, 65% on CIFAR-10 (YOGA Provides the Baseline Communication cost)

Dataset	EMNIST	CIFAR-10
YOGA	**100%**	**100%**
Rand-YOGA	112%	111%
GossipFL	206%	127%
CFL-LS	110%	106%

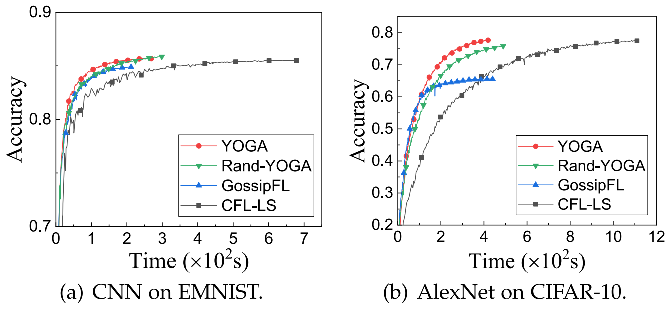


Fig. 4. Test Accuracy of four schemes on the IID datasets.

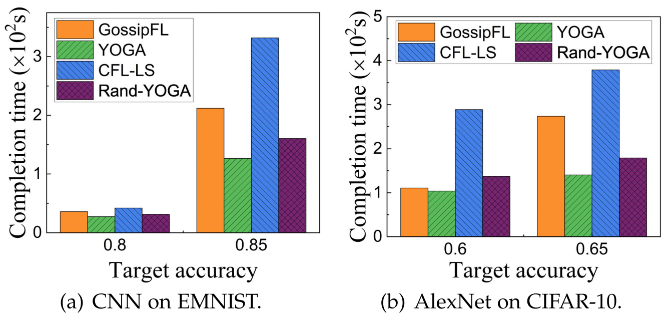


Fig. 5. Completion time of four schemes when achieving different target accuracy.

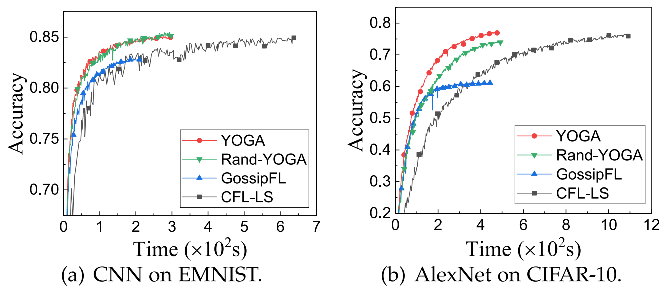


Fig. 6. Test accuracy of four schemes with non-IID level p = 0.4.

optimizes parallel communication through a set of functions such as sending and receiving.

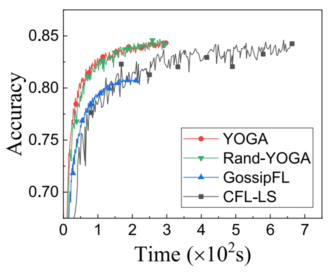
Training Details: We adopt the decaying learning rate technique [43] for training and use the SGD optimizer to update the parameters of the DNN model. If not specified, all experiments are conducted with the default settings as shown in Table II (η stands for learning rate and η_{min} is the minimum of the learning rate). For example, we train AlexNet on CIFAR-10 with a batch size of 32 and a local iteration of 50. We use a learning rate of 0.1 and a decaying learning rate of 0.98. The learning rate decays over the course of training and the minimum learning rate is set to 0.001. The total training round is 300 rounds. Each experiment is conducted with 50 clients.

D. Overall Performance

Convergence Performance. Firstly, we implement a set of experiments on the IID datasets with four schemes. The training processes of YOGA and the baselines are presented in Fig. 4. The results show that YOGA can speed up the training process under the IID setting. For example, by Figs. 4(a) and 4(b), on EMNIST and CIFAR-10, the model convergence rate of is the fastest compared to baselines. Additionally, we show the completion time of different schemes upon achieving various target accuracies in Fig. 5. YOGA demonstrates the fastest convergence rate and significantly outperforms other schemes in terms of completion time. For example, by Fig. 5(a), YOGA takes only 126s to achieve 85% accuracy for CNN on EMNIST, while Rand-YOGA, GossipFL and CFL-LS take 160s, 211s, and 331s, respectively. Compared with Rand-YOGA, GossipFL, and CFL-LS, YOGA achieves 1.3×, 1.7× and 2.6× speedup, respectively. By Fig. 5(b), YOGA speeds up training by about 1.3×, 1.9×, and 2.7× when achieving 65% on CIFAR-10, compared with Rand-YOGA, GossipFL, and CFL-LS, respectively. These results demonstrate the advantage of YOGA in accelerating model training by collecting layers from peers to perform model aggregation.

Communication Cost. In TABLE III, we compare the communication cost of four schemes when achieving different target accuracies under the IID setting. YOGA outperforms the other schemes in terms of communication cost when achieving the target accuracies, *e.g.*, 85% for EMNIST, 65% for CIFAR-10. For example, by TABLE III, YOGA requires 6.25GB to achieve 85% accuracy on the EMNIST dataset, while Rand-YOGA, GossipFL, and CFL-LS requires 7.03GB, 12.9GB, and 6.89GB, respectively. Compared with Rand-YOGA, GossipFL, and CFL-LS, YOGA reduces the communication cost on EMNIST by about 12.3%, 106%, and 10.1%, separately. YOGA reduces communication cost on CIFAR-10 when achieving 65% by about 10.7%, 27.1%, and 6% compared with Rand-YOGA, GossipFL, and CFL-LS, respectively. These results demonstrate the advantage of YOGA in saving communication cost and efficiency of YOGA dealing with the IID dataset.

Secondly, we implement a set of experiments for four schemes on non-IID datasets. Figs. 6 and 7 show the results of model training on two different datasets (EMNIST and CIFAR-10) under two non-IID settings, *i.e.*, $p = 0.4$ and $p = 0.6$, respectively. YOGA accelerates the training process without sacrificing accuracy under the non-IID setting. For example, by Fig. 6(a), when achieving 80% accuracy on EMNIST, YOGA speeds up the training process by 1.2×, 1.7× and 2.4×, compared with Rand-YOGA, GossipFL and CFL-LS, individually. By Fig. 6(b), when achieving 55% accuracy on CIFAR-10, YOGA takes 97s, while Rand-YOGA, GossipFL and CFL-LS take 131s, 124s and 250s, respectively.


(a) CNN on EMNIST.

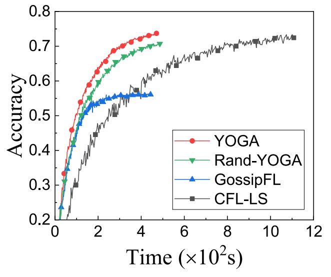
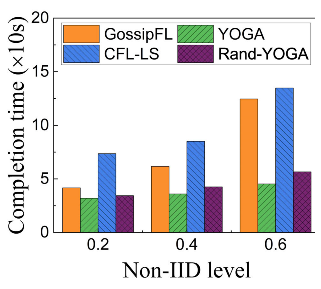

(b) AlexNet on CIFAR-10.

Fig. 7. Test accuracy of four schemes on the three datasets with non-IID level p = 0.6.


(a) CNN on EMNIST.

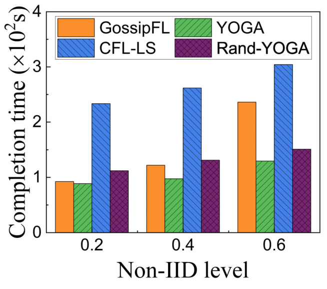
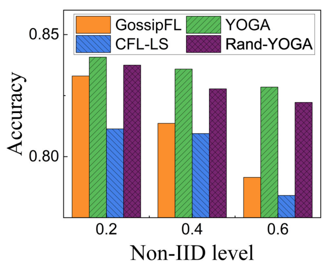

(b) AlexNet on CIFAR-10.

Fig. 8. Completion time of four schemes when achieving 80% accuracy on EMNIST and 55% accuracy on CIFAR-10 under different non-IID levels.


(a) EMNIST (within 100s)

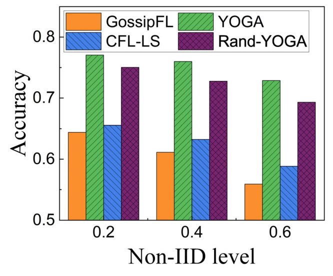
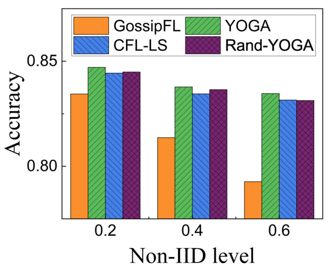

(b) CIFAR-10 (within 400s).

Fig. 9. Accuracy within the same time of four schemes under different non-IID levels.


(a) EMNIST (within 6GB).

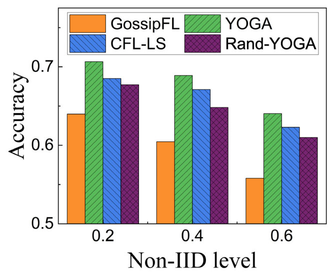
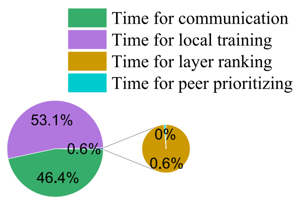

(b) CIFAR-10 (within 20GB).

Fig. 10. Accuracy within the same cost of four schemes under different non-IID levels.


(a) CNN on EMNIST.

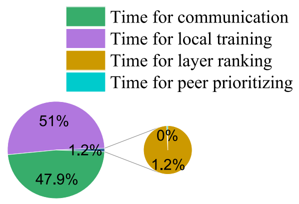

(b) AlexNet on CIFAR-10.

Fig. 11. Time slots of the main procedure required for YOGA to converge.

By Fig. 7(a), YOGA achieves 1.2×, 2.7× and 2.7× speedup on EMNIST, compared with Rand-YOGA, GossipFL, CFL-LS, separately. By Fig. 7(b), YOGA achieves 1.2×, 1.7× and 2.3× speedup on CIFAR-10, compared with Rand-YOGA, GossipFL and CFL-LS, respectively. These results significantly demonstrate the advantage of YOGA in dealing with non-IID data.

E. The Effect of Non-IID Data

To demonstrate the effectiveness of YOGA on the non-IID data, we compare the completion time required to achieve the target accuracy with baselines at different non-IID levels ($p =$ $0.2, 0.4,$ and 0.6) in Fig. 8. First, as the non-IID level increases, all schemes require more time to reach the target accuracy. For example, by Fig. 8(a), the time to reach 80% accuracy on EMNIST with $p = 0.6$ is separately 1.26×, 1.42× more than that with $p = 0.4$ and $p = 0.2$ for YOGA. Second, YOGA outperforms other schemes in terms of completion time when achieving the same target accuracy. For example, as shown in Fig. 8(a), YOGA provides a speedup of 1.87-2.9× under all non-IID settings compared with baselines. By Fig. 8(b), YOGA consistently outperforms the baseline schemes under different non-IID levels, and provides a speedup of 2.68-3.5×. In addition, we show the test accuracy within the same time of YOGA and the baselines in Fig. 9. The results reveal that YOGA consistently outperforms the baselines in test accuracy. For example, by Fig. 9(a), YOGA achieves 3.7%∼4.4% higher accuracy than the baselines on EMNIST within 100s under the non-IID level of $p = 0.6$. By 9(b), YOGA achieves 3.6%∼17% higher accuracy than other schemes on CIFAR-10 within 400s under the non-IID level of $p = 0.6$. These results demonstrate the effectiveness and efficiency of YOGA dealing with non-IID data.

Additionally, we show the performance comparison with different non-IID levels under the same communication budget. As shown in Fig. 10, we record the test accuracy with the communication budget of 6GB and 20GB on EMNIST and CIFAR-10, respectively. In various non-IID settings, YOGA consistently achieves superior performance compared to baselines with the same communication cost. For example, given the communication budget of 6GB with $p = 0.6$ on EMNIST, YOGA achieves an accuracy improvement of 1.27%, 0.27%, and 0.22% compared to GossipFL, CFL-LS, and Rand-YOGA, respectively. Besides, given the communication budget of 20GB for CIFAR-10, YOGA exhibits notable accuracy improvements of 6.7%, 2.1%, and 2.9% in comparison to GossipFL, CFL-LS, and Rand-YOGA with $p = 0.6$, respectively. These results demonstrate that YOGA handles non-IID issue effectively and save communication cost without sacrificing model performance in DFL.

Besides, to demonstrate that layer ranking and peer prioritizing do not introduce extra computational complexity, we recorded the time required for the key steps during the convergence process of YOGA. In Fig. 11, we present the time slots for the main processes of YOGA, *i.e.*, local training, communication, layer ranking, peer prioritizing, training CNN on EMNIST and AlexNet on CIFAR-10. Apparently, as our analysis suggests, YOGA does not introduce significant additional computation complexity. For example, when training AlexNet on CIFAR-10, the time spent on layer ranking and peer prioritizing accounts for only 1.2% of the total training time, which is significantly lower than the time spent on local training and communication.

VI. Related Works

The introduction of FL by McMahan et al. in [3] has opened up a new paradigm for collaborative training with isolated datasets, drawing considerable attention from the research community. While FL offers several advantages, such as the preservation of data privacy and reduction of communication cost, it also faces challenges such as system and data heterogeneity and limited communication bandwidth [44]. Numerous techniques have been proposed to address the challenge of communication constraints in FL, including model compression [45], [46], [47] and pruning [48], [49]. However, most of these previous works commonly assume that periodically full model aggregation.

Earlier works [22] focus on reducing the communication cost by layer-wise model aggregation under PS architecture. Lee et al. propose FedLAMA [22], a technique that dynamically fine-tunes the aggregation period of each layer, accounting for the layer-wise unit model discrepancy to curtail communication expenses. In [23], Ma et al. propose pFedLA, which selectively aggregates the layers contributing more to the model convergence while skipping the less important ones under PS architecture. However, these studies predominantly revolve around PS-based FL, exposing them to the inherent problem of a single point of failure. For DFL, Tang et al. propose GossipFL [24], which allows clients to accelerate DFL model training by communicating with a highly sparse model from a single peer. Besides, Hu et al. present Combo [25], a segmented gossip method designed to leverage node-to-node bandwidth efficiently while maintaining strong training convergence. Combo allows the client to pull different parts of the model parameters from various workers and rebuild a mixed model for aggregation. In addition, Barbieri et al. introduce CFL-LS [26], a method for client-driven DNN fragment selection. It uses a layer optimizer to rank layers based on their impact on model quality (measured by normalized squared gradient of local loss) and introduces a fairness scheme for balanced parameter selection and enhanced performance. However, existing DFL methods mainly focus on optimizing network bandwidth utilization and reducing communication overhead, often overlooking the impact of non-IID issue. This oversight may result in significant performance degradation and poor convergence rates.

Regarding non-IID issue in DFL, numerous studies have been conducted [12], [13] aiming to improve model performance and convergence speed. In [12], Kong et al. conclude that a large consensus distance (correlated to heterogeneous data distribution) can be beneficial for the performance of decentralized training. Wang et al. propose CoCo [13] to preferentially select peers with large divergence in data distribution to overcome the challenge of non-IID data. However, all of these existing works commonly assume the full model as the basic transmission unit, which may not be flexible enough for bandwidth-constrained edge devices, and how layer-wise model aggregation addresses non-IID data remains unexplored.

We summarize the primary differences between the existing works and our work as follows. On one hand, existing research [12], [13], [24] that employs the *full model* as the basic transmission unit typically utilizes compression or quantization techniques to reduce the size of the DNN model. However, this inevitably leads to a decrease in model accuracy, requiring more training rounds to converge. Besides, such approaches overlook the learning dynamics of a DNN throughout training [28] (*e.g.*, different layers of a DNN model have distinct learning speeds and divergences), and using the full model as the transmission unit may not be flexible enough for bandwidth-constrained edge devices. On the other hand, existing research [25], [26] that uses *layers* as the basic transmission unit primarily focuses on bandwidth utilization but often neglects the impact of non-IID data. These approaches may exhibit slow convergence rates and degradation of model accuracy when dealing with non-IID data. In contrast, YOGA investigates the relationship between layer-wise model aggregation and data distribution. Through layer-wise model aggregation, YOGA not only makes full utilization of node-to-node bandwidth, but also addresses the non-IID issue.

VII. Discussion

Since layer-wise model aggregation is not the only way to reduce communication, in this section, we discuss the differences and connections between the layer-wise method and traditional communication reduction methods. In addition to layer-wise model aggregation, model compression or quantization techniques are widely adopted to reduce the communication cost of FL [12], [13], [24], [50], [51]. These existing works usually propose to periodically exchange the full model and reduce communication cost by representing the model with fewer parameters. For example, RandomK and TopK [50] select a portion of the model's parameters for transmission. However, this inevitably leads to a decrease in model accuracy and more training rounds to converge, which increases the communication cost during the training process. Besides, such approaches always overlook the learning dynamics of a DNN throughout training [28] (*e.g.*, different layers of a DNN model have distinct learning speeds and divergences), and using the full model as the transmission unit may not be flexible enough for bandwidth-constrained edge devices. Through layer-wise model aggregation, YOGA offers greater flexibility in utilizing network bandwidth and substantially reduces bandwidth overhead on each link, thus mitigating communication cost for edge devices. Importantly, our method is also orthogonal to these traditional techniques (*e.g.*, model compression or quantization). For example, combined with the TopK technique, YOGA can dynamically set thresholds to control the compression rate. For layers with a larger number of parameters, higher compression rates may be applied, while layers with fewer parameters may adopt lower compression rates. This area holds promise for future research.

VIII. Conclusion

In this work, we have proposed the YOGA framework, which integrates the idea of layer-wise model aggregation into DFL, to fully utilize the limited communication capability of edge clients while dealing with the non-IID data challenge. We have designed a heuristic algorithm (MM) to adaptively determines the layer-wise model aggregation policy for each client. We have built a test-bed environment and evaluated the performance of YOGA. The results demonstrate the effectiveness of YOGA in accelerating the model convergence and saving communication cost without sacrificing model performance.

References

[1] W. Y. B. Lim et al., "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2031–2063, 3rd Quart., 2020.

[2] 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.*, Jul. 2020, pp. 1698–1707.

[3] 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*. PMLR, 2017, pp. 1273–1282.

[4] H. Zhang, J. Bosch, and H. H. Olsson, "Federated learning systems: Architecture alternatives," in *Proc. 27th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2020, pp. 385–394.

[5] E. T. M. Beltrán et al., "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges," 2022, *arXiv:2211.08413*.

[6] J. Liu et al., "Adaptive asynchronous federated learning in resource-constrained edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 2, pp. 674–690, Feb. 2023.

[7] P. Goyal et al., "Accurate, large minibatch SGD: Training ImageNet in 1 hour," 2017, *arXiv:1706.02677*.

[8] S. Rajendran, Z. Xu, W. Pan, A. Ghosh, and F. Wang, "Data heterogeneity in federated learning with electronic health records: Case studies of risk prediction for acute kidney injury and sepsis diseases in critical care," *PLOS Digital Health*, vol. 2, no. 3, 2023, Art. no. e0000117.

[9] Q. Ma, Y. Xu, H. Xu, Z. Jiang, L. Huang, and H. Huang, "FedSA: A semi-asynchronous federated learning mechanism in heterogeneous edge computing," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3654–3672, Dec. 2021.

[10] 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.*, May 2023, pp. 1–10.

[11] 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 *Advances in Neural Information Processing Systems*, vol. 30. 2017.

[12] L. Kong, T. Lin, A. Koloskova, M. Jaggi, and S. Stich, "Consensus control for decentralized deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 5686–5696.

[13] L. Wang, Y. Xu, H. Xu, M. Chen, and L. Huang, "Accelerating decentralized federated learning in heterogeneous edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 9, pp. 5001–5016, Sep. 2022.

[14] 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.

[15] 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)*, Apr. 2021, pp. 384–395.

[16] Q. Luo, J. He, Y. Zhuo, and X. Qian, "Prague: High-performance heterogeneity-aware asynchronous decentralized training," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 401–416.

[17] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," 2021, *arXiv:2106.11342*.

[18] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. 19th Int. Conf. Comput. Statist.* Cham, Switzerland: Springer, 2010, pp. 177–186.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 2, pp. 84–90, Jun. 2012.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[21] OpenAI, "GPT-4 technical report," 2023, *arXiv:2303.08774*.

[22] S. Lee, T. Zhang, C. He, and S. Avestimehr, "Layer-wise adaptive model aggregation for scalable federated learning," 2021, *arXiv:2110.10302*.

[23] X. Ma, J. Zhang, S. Guo, and W. Xu, "Layer-wised model aggregation for personalized federated learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 10082–10091.

[24] Z. Tang, S. Shi, B. Li, and X. Chu, "GossipFL: A decentralized federated learning framework with sparsified and adaptive communication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 3, pp. 909–922, Mar. 2023.

[25] C. Hu, J. Jiang, and Z. Wang, "Decentralized federated learning: A segmented gossip approach," 2019, *arXiv:1908.07782*.

[26] L. Barbieri, S. Savazzi, and M. Nicoli, "A layer selection optimizer for communication-efficient decentralized federated deep learning," *IEEE Access*, vol. 11, pp. 22155–22173, 2023.

[27] 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.

[28] M. Raghu, J. Gilmer, J. Yosinski, and J. Sohl-Dickstein, "SVCCA: Singular vector canonical correlation analysis for deep learning dynamics and interpretability," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[29] K. Hsieh, A. Phanishayee, O. Mutlu, and P. Gibbons, "The non-IID data quagmire of decentralized machine learning," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 4387–4398.

[30] X. Xiao, T. B. Mudiyanselage, C. Ji, J. Hu, and Y. Pan, "Fast deep learning training through intelligently freezing layers," in *Proc. Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData)*, Jul. 2019, pp. 1225–1232.

[31] C. Ye, Y. Yang, C. Fermuller, and Y. Aloimonos, "On the importance of consistency in training deep neural networks," 2017, *arXiv:1708.00631*.

[32] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "FreezeOut: Accelerate training by progressively freezing layers," 2017, *arXiv:1706.04983*.

[33] B. Singh, S. De, Y. Zhang, T. Goldstein, and G. Taylor, "Layer-specific adaptive learning rates for deep networks," in *Proc. IEEE 14th Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2015, pp. 364–368.

[34] X. Qian and D. Klabjan, "The impact of the mini-batch size on the variance of gradients in stochastic gradient descent," 2020, *arXiv:2004.13146*.

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

[36] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: Extending MNIST to handwritten letters," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 2921–2926.

[37] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," Toronto, ON, Canada, 2009.

[38] O. Russakovsky et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[39] D. Merkel et al., "Docker: Lightweight Linux containers for consistent development and deployment," *Linux J.*, vol. 239, no. 2, p. 2, 2014.

[40] N. Naik, "Building a virtual system of systems using Docker swarm in multiple clouds," in *Proc. IEEE Int. Symp. Syst. Eng. (ISSE)*, Oct. 2016, pp. 1–3.

[41] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.

[42] E. Gabriel et al., "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proc. Eur. Parallel Virtual Mach./Message Passing Interface Users' Group Meeting*. Cham, Switzerland: Springer, 2004, pp. 97–104.

[43] S. L. Smith and Q. V. Le, "A Bayesian perspective on generalization and stochastic gradient descent," 2017, *arXiv:1710.06451*.

[44] Y. Liao, Y. Xu, H. Xu, Z. Yao, L. Wang, and C. Qiao, "Accelerating federated learning with data and model parallelism in edge computing," *IEEE/ACM Trans. Netw.*, early access, Aug. 24, 2023, doi: 10.1109/TNET.2023.3299851.

[45] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, “Deep gradient compression: Reducing the communication bandwidth for distributed training,” 2017, *arXiv:1712.01887*.

[46] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, “Robust and communication-efficient federated learning from non-IID data,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3400–3413, Sep. 2020.

[47] Y. Xu, Y. Liao, H. Xu, Z. Ma, L. Wang, and J. Liu, “Adaptive control of local updating and model compression for efficient federated learning,” *IEEE Trans. Mobile Comput.*, vol. 22, no. 10, pp. 5675–5689, Oct. 2023.

[48] Y. Jiang et al., “Model pruning enables efficient federated learning on edge devices,” *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Apr. 25, 2022, doi: 10.1109/TNNLS.2022.3166101.

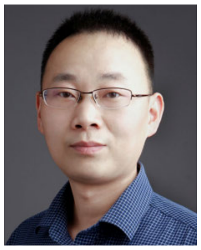
[49] S. Vahidian, M. Morafah, and B. Lin, “Personalized federated learning by structured and unstructured pruning under data heterogeneity,” in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst. Workshops (ICDCSW)*, Jul. 2021, pp. 27–34.

[50] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, “Sparsified SGD with memory,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018.

[51] 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.

Jun Liu received the B.S. degree from Central South University in 2019. He is currently pursuing the master’s degree with the School of Computer Science and Technology, University of Science and Technology of China (USTC). His current research interests include edge computing and federated learning.

Jianchun Liu (Member, IEEE) received the Ph.D. degree from the School of Data Science, University of Science and Technology of 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 current research interests include software-defined networks, network function virtualization, edge computing, and federated learning. He is member of ACM.



Hongli Xu (Member, IEEE) 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 (USTC), China, in 2002 and 2007, respectively. He is currently a Professor with the School of Computer Science and Technology, USTC. He has published more than 100 papers in famous journals and conferences, including IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, Infocom, and ICNP. He has also held more than 30 patents. His current research interests include software-defined networks, edge computing, and the Internet of Things. He was awarded the Outstanding Youth Science Foundation of NSFC in 2018. He has won the best paper award or the best paper candidate at several famous conferences.

Yunming Liao received the B.S. degree from the University of Science and Technology of China in 2020, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Technology. His current research interests include mobile edge computing and federated learning.

Zhiyuan Wang received the B.S. degree from Jilin University in 2019. He is currently pursuing the master’s degree with the School of Computer Science, University of Science and Technology of China (USTC). His current research interests include edge computing, deep learning, and federated learning.

Qianpiao Ma received the B.S. degree in computer science from the University of Science and Technology of China in 2014 and the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2022. He is currently a Post-Doctoral Researcher with Purple Mountain Laboratories. His current research interests include federated learning, mobile edge computing, and distributed machine learning.