Federated Learning With Experience-Driven Model Migration in Heterogeneous Edge Networks

Jianchun Liu, *Member, IEEE, ACM*, Shilong Wang, Hongli Xu, *Member, IEEE*, Yang Xu, *Member, IEEE, ACM*, Yunming Liao, Jinyang Huang, *Member, IEEE*, and He Huang, *Senior Member, IEEE, Member, ACM*

***Abstract*— To approach the challenges of non-IID data and limited communication resource raised by the emerging federated learning (FL) in mobile edge computing (MEC), we propose an efficient framework, called *FedMigr*, which integrates a deep reinforcement learning (DRL) based model migration strategy into the pioneer FL algorithm *FedAvg*. According to the data distribution and resource budgets, our *FedMigr* will intelligently guide one client to forward its local model to another client after local updating, before directly sending the local models to the server for global aggregation as in *FedAvg*. Intuitively, migrating a local model from one client to another is equivalent to training the model over more data from different clients, alleviating the influence of non-IID issue. To this end, we propose an experience-driven method to make proper decisions for model migrations while satisfying the resource constraints. We also prove that *FedMigr* can help to reduce the parameter divergences between different local models and the global model from a theoretical perspective under the non-IID setting. Extensive experiments on three popular benchmark datasets demonstrate that *FedMigr* can achieve an average accuracy improvement of around 13%, and reduce bandwidth consumption for global communication by 42% on average, compared with the baselines.**

***Index Terms*— Edge computing, federated learning, model migration, deep reinforcement learning.**

I. Introduction

DRIVEN by the development of mobile cloud computing (MCC) and Internet of Things (IoT), a new computing paradigm, termed mobile edge computing (MEC) [2], has emerged. In the canonical MCC, end users need to deliver their requests and data to the remote clouds through several networks like radio core network and Internet, leading to high delivery latency and worse user experience. On the contrary, MEC is proposed to push the computation, storage and network functions from clouds to the network edges, which range from specialized base stations, home gateways to ubiquitous mobile devices (such as mobile phones, laptops and wearables) [3].

Recently, mobile Internet has been boosting the growth explosion of user data. More and more data-driven machine learning applications (*e.g.*, machine translation [4] or sentiment analysis [5]) are becoming appealing to consumers and researchers. It is predictable that machine learning tasks will become a dominant workload in MEC systems [6]. However, it usually consumes a large amount of bandwidth when directly sending the total user data to MEC servers for model training, which also raises the risk of exposing users' privacy [7]. As computing resources on clients are becoming increasingly powerful with the emergence of AI chipsets, distributed model training can be implemented on clients by the technique termed *federated learning* (FL) [7], [8], which enables the so-called on-device intelligence [9]. Specifically, each client in MEC holds its local data and a centralized coordinator (*e.g.*, an edge server) maintains a globally shared model. During the runtime, only the global model and the local models are exchanged while the local data residing on each client are never uploaded to the server, which significantly relieves the risk of privacy leakage.

Compared with distributed machine learning in high-performance datacenters, FL will face two major challenges in MEC. 1) **Non-IID Local Data**. The local data are usually collected based on the usage and/or locations of clients. For example, two surveillance cameras, separately deployed in a station hall and on a street-side, may capture quite different views. Therefore, the data distributions of different clients are significantly varied from others. The data sampling of different clients does not follow independent identically distribution (*i.e.*, Non-IID), and any of them cannot be representative of the population distribution (*i.e.*, the distribution of the total data from all the clients), which hurts the accuracy or the convergence rate of the global model training [10]. 2) **Limited Communication Resource**. In MEC, the clients are located in different local area networks (LANs), and communicate with the edge server over the wide area networks (WANs) [11]. Generally, the communication bandwidth between clients and the edge server is relatively more scarce than that within

Manuscript received 18 September 2023; revised 6 March 2024; accepted 11 April 2024; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor J. P. Jue. This work was supported in part by the National Science Foundation of China (NSFC) under Grant 61936015 and Grant 62132019, in part by Jiangsu Province Science Foundation for Youths under Grant BK20230275, in part by the Fundamental Research Funds for the Central Universities, and in part by the University of Science and Technology of China (USTC) Research Funds of the Double First-Class Initiative under Grant WK2150110030. Some earlier results of this paper were published in the Proceedings of IEEE ICDE 2022 [DOI: 10.1109/ICDE53745.2022.00164]. *(Corresponding author: Hongli Xu.)*

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

Jinyang Huang is with the S^2AC Laboratory, the Key Laboratory of Knowledge Engineering with Big Data, and the School of Computer and Information, Hefei University of Technology, Hefei, Anhui 230002, China (e-mail: hjy@hfut.edu.cn).

He Huang is with the School of Computer Science and Technology, Soochow University, Suzhou 215006, China (e-mail: huangh@suda.edu.cn).

Digital Object Identifier 10.1109/TNET.2024.3390416

a datacenter or among the clients over LANs [12]. Thus, communication is likely to become the bottleneck of the distributed model training, since the client-to-server (C2S) communication is probably more time-consuming than a single training iteration on each client [6].

To address the non-IID challenge, Zhao et al. [13] propose to distribute a globally shared proxy dataset to all clients, which requires extra efforts to maintain such auxiliary data for dynamic scenarios carefully [14]. Li et al. [15] propose a framework called *FedProx* to tackle system heterogeneity and statistical heterogeneity, *i.e.*, non-IID data, in federated learning. *FedProx* can be viewed as a generalization and re-parametrization of *FedAvg* with some minor modifications. Besides, the vanilla data augmentation technique [16] is adopted to increase the diversity of training data by random transformation or knowledge transfer, which also helps to mitigate non-IID issues. However, each client needs to send its local model and label distribution information like the number of data samples for each class to the server, leading to an enormous amount of traffic at the parameter server (PS). With the advanced reinforcement learning (RL) techniques [17], Wang et al. [18] propose an experience-driven control framework to counterbalance the bias induced by non-IID data, but will bring enormous bandwidth consumption for model delivery.

As for the communication concern, especially in MEC, some researchers study the communication-efficient FL schemes. For example, Wang et al. [6], [19] design a control algorithm to dynamically adjust the frequency of global aggregation in the pioneer *FedAvg* algorithm [7] to reduce the bandwidth consumption during model training. However, the control algorithm is specialized for better utilizing the bandwidth resource rather than handling non-IID issue, leading to worse training performance. Xie et al. [20] design an asynchronous federated optimization algorithm with low bandwidth consumption, in which the parameter server performs the global update with only one local updated model from an arbitrary client. But the asynchronous method does not aggregate local models from all clients with different data distributions, and cannot well deal with the non-IID issue [13]. In a nutshell, none of the aforementioned works can fully address the two critical challenges for FL. Thus, it is of significant importance to design a communication-efficient FL scheme, especially for non-IID data.

The most related work with our paper is *FedSwap* [21]. In addition to the traditional operation of *FedAvg*, model swapping between any two of all clients at the PS is also adopted by *FedSwap*. However, model swapping at the server still brings an enormous amount of C2S communication traffic to the PS as in *FedAvg*, which will become the bottleneck of FL. More importantly, without considering the data distributions on the clients, random model swapping between two clients cannot well deal with the non-IID issue, which has been validated through simulations in Section V.

Motivated by this [21], given clients with non-IID local datasets and limited communication budget, we propose an efficient FL framework, called *FedMigr*, which integrates the model migration strategy into *FedAvg* to simultaneously cope with the two challenges. Rather than directly sending the local models to the server for global aggregation, *FedMigr* guides one client to forward its local model to another, termed model migration, which helps to alleviate the C2S communication overhead, since the client-to-client (C2C) communication can occur over LANs. Besides, migrating a local model from one client to another is equivalent to model training over more data from different clients, which can alleviate the influence of non-IID issue on the training performance. Powered by the advanced deep reinforcement learning (DRL), *FedMigr* will intelligently and dynamically determine the migration policy in terms of state information (*e.g.*, difference of data distribution and resource usage) from the real-time FL environment. Thus, *FedMigr* will significantly speed up the process of federated training with less resource cost, and enhance the trained model even under the non-IID setting. To accommodate network dynamics, we also present a general version of *FedMigr*, called *FedMigr-G*, in which there are fewer clients participating in the model migration, improving the training performance and resource utilization. The main contributions of this paper are summarized below.

- To approach the challenges of non-IID data and limited communication resource in MEC, we propose an efficient FL framework, termed *FedMigr*, in which model migration is integrated into *FedAvg*.
- We analyze the effectiveness of *FedMigr* from a theoretical perspective, and explain that model migration can help to reduce the parameter divergences between different local models and the global model, enhancing the federated training even under non-IID local data.
- We propose experience-driven algorithms based on DRL to adaptively determine the optimal migration policy, so as to achieve less training time and bandwidth usage.
- We build simulated and real FL environments to evaluate the performance of the proposed algorithm via extensive experiments with three popular benchmark datasets. The results demonstrate that *FedMigr* can achieve an accuracy improvement of around 13%, and reduce bandwidth consumption for global communication by 42% on average, compared with baselines.

II. Preliminaries and Problem Formulation

A. Federated Learning

Traditional machine learning (ML) aims to learn the probability distribution of the training data located on centralized machine(s). Given a training dataset $\mathcal{D} = \{(x_i, y_i)|i \in [1, N]\}$, the item $x_i \in \mathbb{R}^d$ is an input vector, and the corresponding scalar $y_i \in \mathbb{R}$ is the expected output. N denotes the size of the dataset $\mathcal{D}$, and d indicates the number of dimensions. For each data pair (x_i, y_i), the loss function $f_i(w) = \ell(w, x_i, y_i)$ measures the error of the predicted output made with model parameter w. The learning objective is formulated as finding the optimal parameter w^* so that the empirical loss $F(w)$ *w.r.t.* the total training data is minimized.

$$w^* = \arg\min_w F(w) \equiv \frac{1}{N}\sum_{i=1}^{N} f_i(w), \tag{1}$$

Generally, Eq. (1) can be solved by gradient descent (GD) or stochastic gradient descent (SGD) algorithms [22], which iteratively compute the first-order gradient of $F(w)$ *w.r.t.* total (in GD) or partial (in SGD or mini-batch SGD) training data. The parameter w is updated as follows

$$w^t = w^{t-1} - \eta \nabla F(w^{t-1}), \tag{2}$$

where t indicates the iteration index, $\eta > 0$ is the update step size (or learning rate), and $\nabla F(w)$ denotes gradient of $F(w)$.

It has been shown that the performances of machine learning models can be well improved by increasing the scale of training data and model parameters [23], [24]. In order to efficiently cover much more data and speed up model training, distributed machine learning (DML) has been proposed [23], [25]. Considering the distributed architecture based on parameter server (PS) [26], given K clients (or edge nodes), each client is allocated with a dataset $\mathcal{D}_k$ ($k \in \{1, 2, \ldots, K\}$). According to [6], we mainly consider the synchronous implementation due to its popularity and satisfied performance in practice [7], [26], leaving the asynchronous setting as our future direction. For ease of description, we assume the intersection between any two allocated datasets is empty [27]. That is $N = \sum_{k=1}^{K} n_k$, where n_k is size of $\mathcal{D}_k$. Therefore, Eq. (1) can be solved in a distributed manner, *i.e.*,

$$\min_{w} F(w) \equiv \sum_{k=1}^{K} \frac{n_k}{N} F_k(w), \tag{3}$$

where $F_k(w) = \frac{1}{n_k} \sum_{i=1}^{n_k} f_i(w)$ represents the empirical loss *w.r.t.* the data of client k.

With the development of MEC and AI chipsets, model training can be implemented from clouds to edges. McMahan et al. [7] propose an emerging alternative of DML, termed federated learning (FL). In MEC, all the clients are interconnected with the edge server via the mobile networks, and each client has the ability to store its local data. FL can learn a globally shared model in a collaborative fashion, where distributed clients compute local statistic updates based on their own local datasets and communicate with a central coordinator to derive a high-quality global model with the *FedAvg* algorithm. During the runtime, the local data of each client is always kept locally and never sent outwards. The data residing on different clients are usually non-IID, and the communication bandwidth between clients and the server is much more scarce in FL than that in DML.

Let w_k and w_g separately denote the parameters of the local model in client k and the global model. The parameters of all local models are initialized with the same parameter of the global model when $t = 0$, *i.e.*, $w_k(0) = w_g(0)$, $\forall k \in \{1, \ldots, K\}$. Typically, *FedAvg* consists of three core processes, *i.e.*, *Model Distribution*, *Local Updating* and *Global Aggregation*, in each training epoch $t \in \{1, \ldots, T\}$, where T is the total number of training epochs. (1) *Model Distribution*. The server first randomly distributes the latest global model parameter w_g^{t-1} to the k^{th} client. (2) *Local Updating*. Each client computes the gradient of the local empirical loss based on its dataset $\mathcal{D}_k$ and updates the local model parameter w_k^t, *i.e.*,

$$w_k^{t-1} = w_g^{t-1}, \tag{4}$$

$$\nabla F_k(w_k^{t-1}) = \frac{1}{n_k} \sum_{i=1}^{n_k} \nabla f_i(w_k^{t-1}), \tag{5}$$

$$w_k^t = w_k^{t-1} - \eta \nabla F_k(w_k^{t-1}). \tag{6}$$

(3) *Global Aggregation*. At the end of epoch t, the server aggregates the local models from all clients and computes the global model by weighted averaging. Thus, the latest parameter of the global model can be expressed as

$$w_g^t = \sum_{k=1}^{K} \frac{n_k}{N} w_k^t, \tag{7}$$

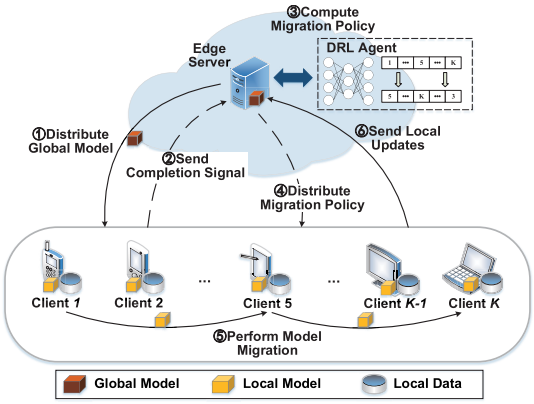


Fig. 1. The workflow of *FedMigr*.

In each global iteration, for a selected client k with n_k samples, given the local mini-batch size b, the local parameter w_k will be updated over $\tau \frac{n_k}{b}$ mini-batches before global aggregation, where τ is the number of local training epochs each client runs over its local dataset during the two consecutive global iterations.

B. FL With Intelligent Model Migration

In this section, we propose the *FedMigr* framework, which integrates intelligent model migration into the traditional *FedAvg* algorithm. Different from *FedAvg*, our *FedMigr* mainly consists of four processes, *i.e.*, *Model Distribution*, *Local Updating*, *Model Migration* and *Global Aggregation*, in a single global iteration. Fig. 1 shows the workflow of *FedMigr*. Similar to *FedAvg*, in the process of *Model Distribution*, the server distributes the latest global model to all K clients. Then, each client k performs the single-machine model training and updates its model parameter for τ local iterations over a local dataset (note that we regard one local iteration as one training epoch).

The major difference between *FedAvg* and our proposed framework comes at the end of *Local Updating*. Concretely,

- Each client sends a completion signal rather than its local update to the server at an interval period T_s (one or several epochs), which can be adjusted according to the network conditions. For example, T_s can be a small value (*e.g.*, one epoch) to react to the network dynamics if the number of clients or data distributions change quickly. When the server receives the completion signals of all clients (herein, assume all clients participate in model training and the case study with fewer participating clients will be discussed in Section IV), it computes the *migration policy* $\mathbf{P} = [p_{i,j}^t], \forall i, j \in \{1, \ldots, K\}$. Specifically, $p_{i,j}^t = 1$ if the local model on client i will be migrated to client j at epoch t. The migration policy is computed on the server by an efficient DRL-based algorithm (Section III).
- In terms of migration policy, we perform the *Model Migration* process. Specifically, the server first sends the migration policy to all clients, and then each client i delivers its local model parameter to client j if $p_{i,j}^t = 1$. If client i and client j are within a LAN, the model migration is termed as *local migration*. Otherwise, the model migration needs to be performed by gateways or the edge server, called *global migration*. Since the completion signals and migration policy can

be represented by some bool values or/and IP addresses, their communication cost can be ignored in comparison with that for model delivery [28].

- After that, client j again performs local updating on the basis of the model of client i if $p_{i,j}^t = 1$. We use M to denote the total number of model migrations in a global iteration. Finally, when $\tau(M+1)$ times of local updating and M times of model migration are finished, the server aggregates the local updates from the clients and derives an up-to-date global model. Thus, the total number of local iterations is $T = G(M+1)\tau$.

In this work, we mainly focus on saving the resource cost and improving the test accuracy of federated training through model migration, even under Non-IID settings. According to the analysis, the proper migration policy will be determined by our proposed method. Meanwhile, the number of model migrations for each client also will be counted by the client or server. Then, the optimal or suboptimal number of model migrations can be achieved.

C. Convergence Analysis

In this section, we analyze how *FedMigr* reduces the distance between the local data distribution and the population distribution, and helps to improve model accuracy. Considering a classification problem over the dataset $\mathcal{D}$ with n^l samples for type $l \in \{1, \ldots, L\}$), where L is the number of label types, and $\sum_{l=1}^{L} n^l = N$. The learning objective in Eq. (1) and Eq. (2) can be rewritten as

$$\min_{w_c} F(w_c) \equiv \sum_{l=1}^{L} q(y=l) \frac{1}{n^l} \sum_{i=1}^{n^l} f_i(w_c), \tag{8}$$

$$w_c^t = w_c^{t-1} - \eta \sum_{l=1}^{L} q(y=l) \frac{1}{n^l} \sum_{i=1}^{n^l} \nabla f_i(w_c^{t-1}), \tag{9}$$

where $q(y=l) = \frac{n^l}{N}$ indicates the distribution probability of samples labeled with l in the dataset, and w_c denotes the parameter of the centralized model. In *FedAvg*, the local updating of client k at iteration t can be rewritten as

$$w_k^t = w_k^{t-1} - \sum_{l=1}^{L} q_k(y=l) \frac{1}{n_k^l} \sum_{i=1}^{n_k^l} \nabla f_i(w_k^{t-1}), \tag{10}$$

where $q_k(y=l) = \frac{n_k^l}{n_k}$, and n_k^l is the number of samples labeled with l in the local dataset $\mathcal{D}_k$. In [13], given the same model initialization for all clients, it demonstrates that the parameter divergence $\|w_g^t - w_c^t\|$ between the global model w_g^t and the centralized model w_c^t is dominated by the earth mover's distance (EMD) between the label distributions on each client and the population distribution, *i.e.*, $\sum_{l=1}^{L} \|q_k(y=l) - q(y=l)\|$, and

$$\|q_k(y=l) - q(y=l)\| = \|\frac{n_k^l}{n_k} - \frac{n^l}{N}\| = \|\frac{Nn_k^l - n_k n^l}{Nn_k}\|, \tag{11}$$

If the data distributions of all the clients follow the population distribution, then $\|q_k(y=l) - q(y=l)\| = 0$, $\|w_g^t - w_c^t\|$ becomes small or negligible, and the global model has the similar performance as the centralized model. However, under the non-IID data, the distribution distance $\|q_k(y=l) - q(y=l)\|$ gets larger, which results in large divergence between w_g^t and w_c^t as well as accuracy degradation of the global model.

As for our learning strategy, the process of model migration among the clients is equivalent to training a model of one client with more local data [10]. From this point of view, we regard *Local Updating* and *Model Migration* as an integrated process of local model updating. We first demonstrate that a smaller distribution distance can achieve a better convergence guarantee (*e.g.*, bound). We make the following assumptions on the loss functions $f_i(w), i \in \{1, 2, \ldots, K\}$, which are widely used in the existing literature [29].

Assumption 1: (Smoothness) $f_i(w)$ is ϱ-smooth with $\varrho > 0$, i.e., $\forall w_1, w_2, \|\nabla f(w_1) - \nabla f(w_2)\| \leq \varrho \|w_1 - w_2\|$. Then, by the property of ϱ-smooth function, we have, $\forall w_1, w_2$, $f_i(w_2) - f_i(w_1) \leq \langle \nabla f_i(w_1), w_2 - w_1 \rangle + \frac{\varrho}{2} \|w_2 - w_1\|^2$.

Assumption 2: (Strong convexity) $f_i(w)$ is μ-strongly convex with $\mu > 0$, i.e., $\forall w_1, w_2$, $f_i(w_2) - f_i(w_1) \geq \langle \nabla f_i(w_1), w_2 - w_1 \rangle + \frac{\mu}{2} \|w_2 - w_1\|^2$.

Let $\vartheta \triangleq \sum_{l=1}^{L} \|q_k(y=l) - q(y=l)\|$ and δ denote the residual error, which represents the loss function can converge to a δ-neighborhood of the optimal value. Combining the Assumptions 1-2, we can derive the following Corollary:

Corollary 1: The greater the degree of data Non-IID among clients, the larger the value of ϑ, and the higher residual error δ. Given IID data among clients, then $q_k(y=l) = q(y=l)$ and $\vartheta = 0$, the residual error δ can be reduced.

Due to space limitations, we omit the detailed proof, which can be referred to in Section IV of our work [30]. Herein, we have proven that a smaller distribution distance is more conducive to the convergence of the model training. In the rest of this section, we mainly demonstrate that model migration can help to decrease the distribution distance $\|q_k(y=l) - q(y=l)\|$. For the sake of analysis, we consider the random model migration strategy (we will illustrate that experience-driven model migration is much more effective than the random strategy in Section III). At the beginning, the model of each client will be sent to another client or keep intact with the uniform probability of $\frac{1}{K}$. In other words, each model has the probability of $\frac{1}{K}$ to be trained on the dataset of another client after model migration. Thus, the final local model on client k after $\tau(M+1)$ times of local updating and M times of model migration is equivalent to being trained on a larger but virtual dataset, whose data distribution can be expressed as

$$q_k'(y=l) = \frac{n_k^l + M \sum_{k'=1}^{K} \frac{1}{K} n_{k'}^l}{n_k + M \sum_{k'=1}^{K} \frac{1}{K} n_{k'}}. \tag{12}$$

Since M and $\frac{1}{K}$ are constant, $\sum_{k'=1}^{K} n_{k'}^l = n^l$, and $\sum_{k'=1}^{K} n_{k'} = N$, Eq. (12) can be rewritten as

$$q_k'(y=l) = \frac{n_k^l + \frac{M}{K} n^l}{n_k + \frac{M}{K} N} = \frac{Kn_k^l + Mn^l}{Kn_k + MN}. \tag{13}$$

Then, we have

$$\begin{aligned} \|q_k'(y=l) - q(y=l)\| &= \left\| \frac{Kn_k^l + Mn^l}{Kn_k + MN} - \frac{n^l}{N} \right\| \\ &= \left\| \frac{K(Nn_k^l - n_k n^l)}{N(Kn_k + MN)} \right\| \\ &= \left\| \frac{Nn_k^l - n_k n^l}{Nn_k + \frac{MN^2}{K}} \right\|. \end{aligned} \tag{14}$$

Usually $N > K > 0$. We have $\frac{MN^2}{K} > MN > 0$ if $M \geq 1$. Then, we observe that

$$\left\| \frac{Nn_k^l - n_k n^l}{Nn_k + \frac{MN^2}{K}} \right\| < \left\| \frac{Nn_k^l - n_k n^l}{Nn_k} \right\|, \tag{15}$$

i.e., $\|q_k'(y = l) - q(y = l)\| < \|q_k(y = l) - q(y = l)\|$. In terms of Eqs. (14) and (15), we find that *FedMigr* helps to shorten the probability distance between the data distribution on each client and the population distribution. In other words, *FedMigr* has the potential to reduce the parameter divergence of the derived global model and the centralized model, and obtain better performance in comparison with *FedAvg*. If the local data on all clients follow the IID assumption, then $Nn_k^l - n_k n^l = 0$, and the property of *massive clients* mentioned above has little influence on the accuracy of the global model. When increasing the numbers of *model migration*, it is equivalent to increasing the probability of training each local model with much more data of many different clients, which can contribute to further reducing the distribution divergence as well as the parameter divergence.

D. Problem Formulation

In this section, we give the definition of federated learning with model migration (FLMM) problem. Without loss of generality, two major kinds of resources, computation and communication, are taken into considerations in this work. Specifically, for *Local Updating*, the communication cost is neglected while the computation cost on client k at each epoch, denoted as c_k, is proportional to the volume of local training data. In *Model Distribution* and *Global Aggregation*, the costs for client k receiving the latest model and sending local update are denoted as $b_{0,k}$ and $b_{k,0}$, and usually $b_{0,k} = b_{k,0} > 0$ (written as b_k for simplicity). Considering the sufficient computing power on the server, the computation cost for aggregating local updates and computing the latest global model is also neglected [2]. Besides, the communication cost for delivering the model from client i to client j in the m-th ($\forall m \in \{1, 2, \ldots, M\}$) round of *Model Migration* is denoted as $b_{i,j}^m$. If $i = j$, $b_{i,j}^m = 0$. Assume that the total budgets of computation and communication resources are denoted as B_c and B_b in the network, respectively. Accordingly, the FLMM problem can be formulated as follows:

$$\min_{T \in \{1,2,3,\ldots\}} F(w^T)$$
$$s.t. \begin{cases} \sum_{k=1}^{K} T \cdot c_k \leq B_c, \\ G\left(2\sum_{k=1}^{K} b_k + \sum_{m=0}^{M-1}\sum_{i=1}^{K}\sum_{j=1}^{K} p_{i,j}^m b_{i,j}^m\right) \leq B_b, \\ T = G \cdot (M+1) \cdot \tau, \\ p_{i,j}^m \in \{0, 1\}, \qquad \forall i, j, \forall m \end{cases} \tag{16}$$

where $p_{i,j}^m$ denotes whether the model on client i will be migrated to client j or not at epoch m. The first set of inequalities expresses the computation resource constraints during totally T training epochs. The second set of inequalities ensures the bandwidth constraints in the network. The objective is to minimize the loss function of federated learning.

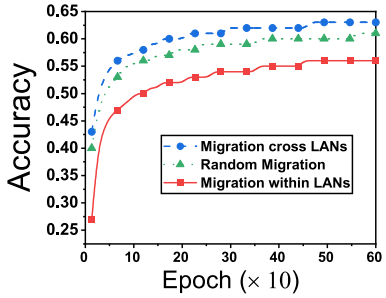


Fig. 2. Test accuracy of model training with *FedMigr* given different model migration strategies.

Since the model migration decision is a bool variable, this is a typical integer programming problem. In general, finding the optimal solution for an integer programming problem is NP-hard [31], thus solving the integer programming problem at every epoch will be time-consuming. Meanwhile, the time-varying network conditions also aggravate the difficulty for this problem. On the contrary, with the abundant network and model training information, we believe that an experience-driven method will be helpful to make an efficient decision for model migration while satisfying the resource constraints at each epoch. Hence, we propose to design a deep reinforcement learning based method to learn an effective solution for the FLMM problem.

III. Algorithm Design

A. Motivation for Algorithm Design

FL always relies on SGD which has been widely used in training deep networks with good empirical performance [32]. The IID sampling of the training data is essential to ensure that the stochastic gradient is an unbiased estimate of the full gradient. However, it is unrealistic to assume that the data on each client is always IID in practice. Generally, the data collected by the clients within a LAN often have similar features and labels, while the data collected by the clients in different LANs greatly vary [12]. The algorithm in our proposed framework may prefer to perform migration among the clients with different data distributions (*e.g.*, cross LANs) to accelerate the convergence of model training, improving the performance (*e.g.*, test accuracy) of FL in heterogeneous edge computing [13]. To this end, we perform two groups of tests to illustrate the motivation for algorithm design.

We first observe the training performance with *FedMigr* under three different model migration strategies, *i.e.*, migration cross LANs (denoted as migration-C), random migration (migration-R) and migration within LANs (migration-W). Let AlexNet[1] train on CIFAR10[2] with 600 epochs. The data distributions of the clients within a LAN are the same. As shown in Fig. 2, the model accuracy of migration-C is better than that of migration-R and migration-W. For instance, given 500 epochs, the accuracy of migration-C is about 63.6%, while those of migration-W and migration-R are about 56.2% and 60.7%, respectively. Thus, performing model migration between the clients with different data distributions will significantly improve the training performance.

[1]AlexNet consists of 8 weight layers including 5 convolutional layers and 3 fully-connected layers, and three max-pooling layers are used following the first, second and fifth convolutional layers.

[2]http://www.cs.toronto.edu/ kriz/cifar.html

TABLE I
COMPLETION TIME AND TRAFFIC CONSUMPTION UNDER DIFFERENT SCHEMES GIVEN A TARGET ACCURACY

Schemes	Target Accuracy=80%	
	Completion Time (s)	Traffic Consumption (MB)
FedAvg	13,927	328
FedMigr	6,584	175

In order to better illustrate the effectiveness of *FedMigr*, we then test the training performance of *FedAvg* and *FedMigr* given a target accuracy requirement (*e.g.*, 80%). Table I shows the completion time and traffic consumption of model training under two schemes. *FedMigr* will significantly reduce the resource cost compared with *FedAvg*. For example, the completion time and traffic consumption of *FedMigr* are about 6,584s and 175MB, while those of *FedAvg* are about 13,927s and 328MB, respectively. In other words, *FedMigr* can reduce the time and bandwidth cost by about 53% and 47% compared with *FedAvg*, respectively. In a nutshell, migrating models among the clients with different data distributions will efficiently speed up the process of federated training with less resource cost.

B. Deep Reinforcement Learning (DRL)

Our *FedMigr* adopts a deep reinforcement learning (DRL) based algorithm to generate migration policy, according to the data distributions and network resource. We adopt not the other RL algorithms (*e.g.*, MAB [33]), but the DRL approach mainly due to system dynamics. Specifically, edge nodes (*e.g.*, mobile devices), located at diverse geographical positions, will dynamically join/leave the system, and the wireless connections between these devices may be time-varying due to the background noise in edge computing scenarios. Hence, the network conditions may frequently change during training. In fact, MAB usually determines the proper action over the relatively steady system states, but cannot perceive the detailed and real-time state information (*e.g.*, the resource consumption and process of training) of network environment [34]. Thus, it is difficult for MAB to provide fine-grained control of model training or migration in dynamic edge computing. On the contrary, the experience-driven models are very effective for discovering the complicated underlying relation [34]. The deep neural network in DRL can better perceive the underlying relation among the model training, network states and migration policy, and is more robust to the environment dynamics. Besides, the training of DRL agent can be performed offline in the simulation environment which has sufficient resources before being deployed in practice.

DRL is the learning process of an *agent* that acts in corresponds to the *environment* to maximize its *rewards*. The agent mainly involves three components: state, policy network, and action probability. At each epoch t, the agent observes a state s_{t-1}, takes an action a_t and receives a reward r_t by executing the action. The workflow of the DRL method is illustrated in Fig. 3. At each training epoch t, the policy network in the agent receives a *state* s_{t-1} (*e.g.*, data distribution, loss function and resource usage) and outputs the probabilities of some *actions*, called *policy* π, which is a mapping from state s_{t-1} to actions $\mathcal{A}$. Then, an action a_t will be picked from $\mathcal{A}$ according to the policy π. In return, the agent receives the next state s_t and a scalar *reward* r_t. The

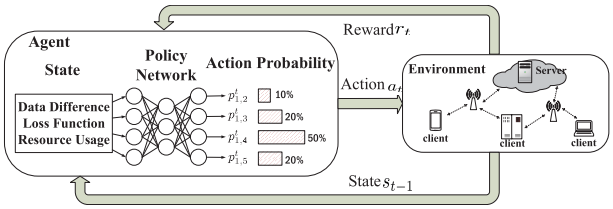


Fig. 3. The architecture of the DRL system. It mainly consists of two parts: the agent and environment. The agent includes the state, policy network and action probability.

return reward $R_t = \sum_{d=0}^{T-t} \gamma^d r_{t+d}$ is the total accumulated return from epoch t with a discount factor $\gamma \in (0, 1]$. The goal of the agent is to maximize the expected return from each state s_t. The detailed description about the state, action and reward is given in the next section.

C. Model Design for DRL

To set up the DRL system, we elaborate the state space, action space, and reward function as follows.

State Space: We use a vector $s_t = (t, w^t, F_t, D_t, \mathcal{R}_t, \mathcal{G}_t)$ to denote the state at epoch t. Here t is the training epoch index. w^t and F_t denote the model parameter and loss function after epoch t, respectively. $D_t = [d^t_{i,j}], \forall i, j \in \{1, \ldots, K\}$ is a $K \times K$ symmetric matrix which reflects the differences of data distributions among the clients after t epochs. Besides, computation and communication cost at each epoch t is represented as $\mathcal{R}_t = \{c^t, b^t\}$. We use $\mathcal{G}_t = \{\mathcal{B}_c, \mathcal{B}_b\}$ to denote the remaining resource budgets at the end of epoch t. With the progress of the federated training, more and more resources are consumed, and the remaining resource budgets decrease.

Action Space: At epoch t, a migration policy $\mathbf{P}$ will be determined by the system agent, called an action a_t. Note that the model migration decision at epoch t is denoted as $[p^t_{i,j}], \forall i, j \in \{1, \ldots, K\}$, resulting in a large action space of $K \times K$, which complicates DRL training. Similar to [18], we try to reduce the action space size while still leveraging the intelligent control provided by the DRL agent. Specifically, at epoch t, the agent will find the client j for model migration with the maximum expected return reward for only one client i per round, *i.e.*, $p^t_{i,j} = 1$. Thus, the action space is reduced to $\{1, 2, \ldots, K\}$ for client i, where action $a = j$ means that the model will be migrated from client i to client j. Given the current state, the DRL agent chooses an action based on a policy network, expressed by a probability distribution $\pi(s_t|\theta)$ over the whole action space. We use deep neural network [35] to represent the policy π in the DRL agent, where the adjustable parameters of the neural network are referred to as the policy parameter θ. The policy can be represented as $\pi(s_t|\theta) \rightarrow [0, 1]$, which is the probability of taking the action a_t at the state s_t.

Reward Function: At training epoch t, the agent will get a reward $r(s_t; a_t)$ under a certain state s_t after taking action a_t. In practice, the reward function should be positively correlated with the system objective. As in Eq. (30), the objective function is to minimize the loss function under resource constraints. The better training performance (*e.g.*, loss value) may be achieved when the model migration occurs between the two clients with larger difference of data distributions. We define the reward r_t as the combination of the difference

of loss value and resource usage at epoch t, *i.e.*,

$$r_t = -\Upsilon^{\frac{\triangle F_t}{F_{t-1}}} - \frac{c^t}{\mathcal{B}_c} - \frac{b^t}{\mathcal{B}_b}, \quad t < T \tag{17}$$

where Υ is a positive constant so as to ensure that r_t decreases exponentially with the loss value F_t. Here $\triangle F_t = F_t - F_{t-1}$ denotes the difference between the current and the previous loss values, which reflects the performance of federated training at epoch t. The better training performance it achieves, *i.e.*, the smaller value of $\frac{\triangle F_t}{F_{t-1}}$, the more reward the agent will obtain. In contrast, the more resources (*e.g.*, computation and communication) the task consumes, the less reward the agent will obtain. The weights in the function are determined by the specific requirements to balance objectives. For example, when communication cost dominates the resource consumption of model training, we will set a larger weight for traffic consumption in the function. At epoch T, the learning task will stop as either the model converges or the resource budget is used up (*i.e.*, $\min \mathcal{G}_T \leq 0$). Then, the reward in the final epoch T is defined as

$$r_T = \begin{cases} \mathcal{L}_T + \mathcal{C} & \text{if } \min \mathcal{G}_T \geq 0, \\ \mathcal{L}_T - \mathcal{C} & \text{otherwise.} \end{cases} \tag{18}$$

where $\mathcal{L}_T = -\Upsilon^{\frac{\triangle F_T}{F_{T-1}}} - \frac{c^T}{\mathcal{B}_c} - \frac{b^T}{\mathcal{B}_b}$ and $\mathcal{C}$ is a positive real number. If the learning task stops with success, *i.e.*, the convergence of model training is reached without overrunning the resource budget, the reward will be added by $\mathcal{C}$. Otherwise, if the learning task fails, *i.e.*, no convergence guarantee under the resource budget, a negative value $\mathcal{C}$ (or reward penalty) will be added to the reward function. The reward will be sent to the agent for the following decision. If the reward of the current action is larger than that of other actions, the probability of this action being selected in the next epoch will be increased. Otherwise, it will be decreased.

Note that the edge computing system (*e.g.*, the number of clients or the amount of local data on the clients) varies significantly with time and space. In fact, we consider the search space as large as possible (*i.e.*, all clients in the network, not just those activated clients participating in model training) when the pre-training of the DRL agent is performed. Thus, the agent can well adapt to the changes of clients. The effectiveness of model migration will be well perceived by the DRL agent through status and reward from the network environment. Moreover, the DRL agent has the ability to adjust the decision of model migration according to the reward, depending on the training data, even if the local data of a client is updated. Thus, there is no need to retrain the DRL system when the edge computing system varies with time and space.

D. Algorithm Description

In this section, we describe the experience-driven migration policy generation (EMPG) algorithm in detail. We train the DRL agent using the cost-effective and time-efficient Deep Deterministic Policy Gradient (DDPG) method [36]. The basic idea inside the EMPG algorithm is to maintain a parameterized critic function and a parameterized actor function, respectively. The critic function $Q(s_t, a_t|\psi)$ can be implemented using Deep Q-Network (DQN) [37], where ψ is the weight vector of the DQN. The critic function returns Q for a given state-action pair at epoch t as follows:

$$Q(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t], \tag{19}$$

The actor function $\pi(s_t|\theta)$ can be implemented using a Deep Neural Network (DNN), and be updated by applying the chain rule to the expected cumulative reward $\mathcal{R}$:

$$\mathcal{R} = \mathbb{E}[\nabla_\theta Q(s, a|\psi)|_{s=s_t, a=\pi(s_t)} \cdot \nabla_\theta \pi(s|\theta)|_{s=s_t}], \tag{20}$$

where θ is the weight vector of the DNN and $\pi(s_t) = \arg\max Q(s_t, a_t)$. Let h_t be the target value as:

$$h_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \pi(s_{t+1}|\theta)|\psi), \tag{21}$$

where γ is the discount factor for the future reward. The DQN of the critic function can be learned by minimizing the loss function:

$$L(\psi) = \mathbb{E}[h_t - Q(s_t, a_t|\psi)], \tag{22}$$

To effectively train the DRL agent for migration policy generation, two critical problems should be taken into consideration:

1) Action Exploration: To obtain a suitable policy through DRL, we need to ensure that the action space is adequately explored. To this end, the actions with high reward will be sufficiently produced. For effective exploration, we apply the modified ϵ-greedy method [38] to meet our demands, where $\epsilon \in [0, 1]$ denotes an adjustable parameter. It means that with ϵ probability, the agent derives actions by solving the FLMM problem in Eq. (30); and with $(1 - \epsilon)$ probability, the agent directly derives actions from the policy network $Q(s_t, a_t)$. However, the exploration is based on the solution of Eq. (30), which is NP-Hard due to the integer variables [31]. To solve the FLMM problem, we first deal with the integer variable by relaxing it to be any fractional value in $[0, 1]$. Then, the original problem is transformed into a quadratic programming (QP) problem with linear constraints, which can be easily solved by the convex problem solver (*e.g.*, CVX [39]).. By adjusting parameter ϵ, we can achieve a tradeoff between exploration and exploitation.

2) Experience Replay: In order to replay the experience, we adopt the prioritized experience replay strategy [40], which specifies samples with careful consideration for both the actor and critic networks. Specifically, a priority will be assigned for each transition sample. Based on this priority, the replay buffer will be sampled at each epoch. For each transition sample $z = (s_t, a_t, r_t, s_{t+1})$, we define a function called Temporal-Difference (TD) error, which corresponds to the training of the critic deep network:

$$\phi_z = h^z - Q(s^z, a^z), \tag{23}$$

where h^z is the target value for training the critic network in Eq. (21). The value of TD error acts as the correction for the estimation and may implicitly reflect to what extent an agent can learn from the experience. The bigger the magnitude of absolute TD error is, the more aggressive the correction for the expected action-value is. In this condition, experiences with high TD-errors are more likely to be of high value and associated with very successful attempts. Besides, more frequently replaying these experiences will help the agent gradually realize the true consequence of the wrong behavior under the corresponding states, as well as avoid making the wrong behavior in these conditions again, which can improve

the overall performance. Then, we put the TD error with the Q gradient:

$$\nabla_a Q(s^z, a^z) = \nabla_\theta Q(s, a|\psi)|_{s=s^z}^{a=\pi(s^z)} \cdot \nabla_\theta \pi(s|\theta)_{s=s^z}, \quad (24)$$

The priority of the transition data z is given by:

$$\rho_z = \varepsilon \cdot (|\phi_z|) + (1-\varepsilon) \cdot |\nabla_a Q(s^z, a^z)|, \quad (25)$$

where ε is the control parameter between TD error and gradient. $|\nabla_a Q(s^z, a^z)|$ is the absolute value of the gradient. At last, we define the probability $\mathbb{P}(z)$ of sampling transition z as follows:

$$\mathbb{P}(z) = \frac{\rho_z^\delta}{\sum_{j=0}^{|\mathbb{B}|} \rho_j^\delta}, \quad (26)$$

where the parameter δ controls to what extent the prioritization is used. If $\delta = 0$, then it becomes uniform sampling. $\mathbb{B}$ is the buffer of samples with size of $|\mathbb{B}|$ and ρ_z can be computed by Eq. (25). The definition of the sampling probability can be seen as a method of adding stochastic factors in selecting experiences since even those with low TD errors can still have a probability to be replayed, which guarantees the diversity of sampled experiences. Such diversity can help prevent the neural network from being over-fitting. Then the accumulated weight-change for critic and actor networks are:

$$\Delta_\psi := \Delta_\psi + \mu_z \cdot \phi_z \cdot \nabla_\psi Q\left(s^z, a^z\right) \quad (27)$$

$$\Delta_\theta := \Delta_\theta + \mu_z \cdot \nabla_a Q(s^z, a^z) \quad (28)$$

where μ_z is the important-sample weight:

$$\mu_z = \frac{(|\mathbb{B}| \cdot \mathbb{P}(z))^{-\delta}}{\max_{j \in \mathbb{B}} w_j}, \quad (29)$$

The accumulated weight-change will be used for updating the critic network and actor network.

The EMPG algorithm in Alg. 1 includes three steps: initialization, Actor-Critic network learning and network updating. First, the algorithm initializes the policy matrix for model migration (Line 2). Besides, all the weights θ of the actor network, and ψ of the critic network are also initialized (Line 3). To apply an off-policy training method, we employ target network $Q'(\cdot)$ and $\pi'(\cdot)$ to improve the training speed. The target network is a clone of the origin network (Line 4) and will be slowly following updated. In each training epoch of FL, the agent will interact with the environment to receive a state and push it into the buffer (Line 8). Then, we sample a transition data from the replay buffer to train the actor-critic network (Lines 11-17). For each transition data z in the sample buffer $\mathbb{B}$, we first compute its important-sample weight μ_z by Eq. (29), which is used to correct the bias introduced by prioritized replay (Line 12). Then, the TD error is computed by Eq. (23) based on the target value y_z by Eq. (21). Moreover, the policy gradient should be updated by the chain rule (Line 13) and the transition priority is updated by Eq. (25) (Line 16). The weight-changes are accumulated for updating the actor-critic network (Line 17). Based on the weight-changes, the critic network, actor network, and target network are updated (Lines 19-20). Finally, the optimal action will be selected for the next training epoch by the output of actor network (Lines 21-22).

Algorithm 1 Experience-Driven Migration Policy Generation

1: **Initialization**
2: Initialize the migration policy matrix $\mathbf{P} = [p_{i,j}^0], \forall i, j$
3: Initialize critic networks $Q(\cdot)$ and actor networks $\pi(\cdot)$ with weights ψ and θ, respectively;
4: Initialize target networks $Q'(\cdot)$ and actor networks $\pi'(\cdot)$ with weights $\psi' = \psi$ and $\theta' = \theta$, respectively;
5: Initialize replay buffer $\mathbb{B}$ and $\rho_1 = 1$;
6: **Migration policy generation for federated training**
7: **for** $t = 1$ to T **do**
8: The agent interacts with the environment
9: **Actor-Critic network learning**
10: **for** $k = 1$ to $|\mathbb{B}|$ **do**
11: Sample a transition $z = (s_t, a_t, r_t, s_{t+1})$ from $\mathbb{B}$;
12: Update important-sample weight by Eq. (29);
13: Compute TD error of sample z by Eq. (23);
14: Compute target value for critic network by Eq. (21)
15: Compute Q gradient by Eq. (24);
16: Update the transition priority by Eq. (25);
17: Accumulate weight-change for critic network by Eq. (27) and actor network by Eq. (28);
18: **Actor-Critic network update**
19: Update the actor network and critic network;
20: Update the target network;
21: Select the optimal action, *i.e.*, the optimal policy
22: Update the migration matrix $\mathbf{P}$

IV. A General Version of *FedMigr*

In this section, we consider a general version of *FedMigr*, denoted as *FedMigr-G*, in which a smaller number of model migrations will be performed among the clients. The clients are usually deployed outdoors in edge computing, some clients may fail to work occasionally because of a system crash, edge dynamics, or network disconnection. Some unnecessary model migration among these clients may not bring the improvement of training performance, but more resource consumption. As a result, we just select partial (not all) clients participating in the model migration. To this end, we define the client's activity, ranging from 0 to 1, as the combination of two parts: (1) the frequency of participating in global update; and (2) the frequency of model migration. Herein we assume that both have the same impact on the client's activity, and set their weights to 0.5 respectively. Besides, we set an activity threshold (*e.g.*, 0.5) for each client. Specifically, a client is positive if the activity is greater than the threshold, *i.e.*, $\varphi \geq 0.5$. Otherwise, the client is negative. The DRL agent will compute the migration policy for these positive clients instead of these negative clients. Thus, the decision-making efficiency of DRL will be significantly improved even in the dynamic and heterogeneous edge networks. We will observe the impact of the activity threshold on the training performance through evaluation in Section V.

In order to solve the problem of long training time caused by the synchronization barrier [15] or node failure, we set a hard time threshold $\mathbb{T}$. The global aggregation will be performed within the time threshold, even though the parameter server has not received the local updates from all clients. To determine the set of clients for model migration, we count the consecutive number of local updates (*e.g.*, ϱ) from these clients not participating in global aggregation. Specifically, if a client has not sent the local updates to the

Algorithm 2 *FedMigr-G*: A General Version of *FedMigr*

1: Initialize the model parameters w, the activity φ of each client, the number of participating in global updates per client $\mathbb{G}$, the number of model migrations per client $\mathbb{M}$
2: **Processing at the Parameter Server**
3: **if** the resource constraints are satisfied **then**
4: **for** each global update $g \in G$ **do**
5: **while** waiting time $< \mathbb{T}$ and No. of received local updates $< n$ **do**
6: Waiting for local updated models from workers
7: Perform the global updating
8: Compute global loss function
9: **if** Received the completion signal **then**
10: Call the EMPG-G algorithm to compute the migration policy with the activity of each client
11: Update the sets $\mathbb{G}$ and $\mathbb{M}$
12: Update the activity φ of each client
13: Update the resource budgets
14: Distribute the updated global model and migration policy to the clients
15: **Processing at the Clients**
16: **if** the resource constraints are satisfied **then**
17: Receive the updated global model and migration policy from the server
18: **for** each local update $d \in \{1, 2, \ldots, \tau\}$ **do**
19: Update the local model w by stochastic gradient descent (SGD) [41]
20: Perform the model migration according the policy
21: **if** No. of training epochs % T_s == 0 **then**
22: Send the completion signal to the server
23: Push local update to the server
24: Return the final model and loss function

parameter server for model aggregation within 10 consecutive epochs, *i.e.*, $\varrho \geq 10$, the activity of this client will be set to 0. Therefore, it is no need to compute the migration policy for these clients.

We give the definition of the FLMM-G problem. We denote $\mathbb{K}_t \in K$ as the set of clients participating in the model migration at epoch t. Let c_k and b_k denote the computation cost and communication cost of client k, respectively. During the model training, the total consumption of computation and communication cannot exceed the given resource budgets, denoted as B_c and B_b respectively. Accordingly, the FLMM-G problem can be formulated as follows:

$$\min_{T \in \{1,2,3,\ldots\}} F(w^T)$$
$$s.t. \begin{cases} \sum_{k=1}^{K} T \cdot c_k \leq B_c, \\ 2 \cdot G \sum_{k=1}^{K} b_k + \sum_{t=1}^{T} \sum_{i=1}^{\mathbb{K}_t} \sum_{j=1}^{\mathbb{K}_t} p_{i,j}^t b_{i,j}^t \leq B_b, \\ T = G \cdot (M+1) \cdot \tau, \\ p_{i,j}^t \in \{0,1\} \qquad \forall i, j, \forall t \end{cases} \tag{30}$$

The first set of inequalities expresses the computation resource constraints during total T training epochs. The second set of inequalities ensures the bandwidth constraints in the network. The objective of the FLMM problem is to minimize the loss function of federated learning.

In order to solve the FLMM-G problem, we make some small changes to EMPG, denoted as EMPG-G. The main difference between EMPG and EMPG-G is the size of state and action spaces. Let $s_t' = (t', w'^t, F_t', D_t', \mathcal{R}_t', \mathcal{G}_t')$ denote the state in EMPG-G at epoch t, where $D_t' = [d_{i,j}^t], \forall i, j \in \{1, \ldots, \mathbb{K}_t\}$ is a $\mathbb{K}_t \times \mathbb{K}_t$ symmetric matrix. For the actions in the DRL system, we use $[p_{i,j}^t], \forall i, j \in \{1, \ldots, \mathbb{K}_t\}$ to denote the model migration policy. Since the state and action space of EMPG-G is much smaller than that of EMPG, the complexity of action exploration and experience replay becomes lower, respectively. The detailed *FedMigr-G* solution with EMPG-G is described in Alg. 2.

At the beginning of *FedMigr-G*, we first initialize the model parameter and the counter for the client's activity φ, *i.e.*, the number of participating in global updates and model migrations (Line 1). The PS first aggregates these local models received from the clients (Line 4-8). Then, the EMPG-G algorithm will be called to compute the migration policy if the PS has received the completion signals from the clients (Line 10). Subsequently, the activity of each client and resource budget are updated by the server (Line 11-13). Finally, the PS will distribute the updated global model and migration policy to the clients (Line 14). After receiving the updated global model, each client first performs model training on the local dataset (Line 17-19). Then, the local models will be migrated among the clients according to the policy and sent to the server (Line 20-23). The algorithm will terminate when the model achieves convergence or resources are exhausted. Note that clients always dynamically join or exit model training, resulting in model migration failure in practice. Therefore, the server needs to frequently update the migration policy against the dynamic network, which will be our future work.

V. Performance Evaluation

A. Performance Metrics and Baselines

In this paper, we adopt the following metrics to evaluate the efficiency of our proposed framework. (1) *Training loss* is the quantification difference of probability distributions between model output and observation results. The loss value reflects the quality and convergence of model learning. (2) As one of the most common performance metrics for classification, *accuracy* is measured by the proportion between the amount of the correct data through model inference and that of all data. (3) The total amount of traffic, *i.e.*, *bandwidth consumption*, is measured between the server and clients for model training. (4) We adopt the *completion time*, which consists of computation time and communication time, to estimate the training speed of an FL task.

We adopt three typical FL schemes, *i.e.*, *FedAvg*, *FedSwap* [21] and *FedProx* [15], as baselines for performance comparison. The key idea of *FedSwap* is to perform model swapping between any two of all clients through the PS, instead of running *FedAvg* every iteration. This operation of swapping the models between the clients gives each model a bigger picture on the entire dataset, so as to reduce weight divergence. In *FedProx*, a proximal term is added to the

objective that helps to improve the stability of federated training. This term provides a principled way for the server to account for heterogeneity associated with partial information. Besides, we perform the random model migrations among the clients (termed *RandMigr*), instead of experience-driven model migrations, to verify the efficiency of our proposed model migration algorithm.

B. Datasets and Models

Datasets: In the experiments, three popular benchmark datasets, *i.e.*, CIFAR10 and CIFAR100 [42], ImageNet ILSVRC-2012 dataset [43] constructed for image classification tasks, are employed to test the performances of *FedMigr*, *FedSwap*, *RandMigr* and *FedAvg*. Concretely, 1) CIFAR10 (referred to as C10) consists of 60,000 32×32 color images (50,000 training images and 10,000 test images) in 10 classes, with 6,000 images per class. 2) CIFAR100 (referred as C100) is similar to C10, but has 100 classes, each of which has 600 images. 3) ImageNet ILSVRC-2012 dataset features RGB-images of 224×224 pixels belonging to 1,000 different classes. In all, there are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images. Usually, edge nodes may be resource-limited devices, *e.g.*, mobile phones, gateways or servers. Based on this, a subset of this dataset, *i.e.*, the ImageNet-100 dataset is adopted for some experiments. It features only 100 randomly drawn classes from the complete ImageNet dataset.

Models: Three deep learning models with different structures and parameters are implemented: 1) A Convolutional Neural Network (CNN) has the same structure as that in [7], *i.e.*, two 5×5 convolution layers (32, 64 channels, each followed with 2×2 max pooling), a full connected layer with 512 units, and a softmax output layer with 10 units. This model, which is specialized for the C10 dataset, is called C10-CNN. 2) The second model is a CNN for the C100 dataset (C100-CNN). Different from C10-CNN, it involves two fully connected layers (with 512 units each) following the convolution layers, and an output layer with 100 units. 3) The third model is ResNet-152 [44] which will be adopted to perform the image classification tasks based on ImageNet-100 dataset (denoted as Res-ImageNet).

C. Simulation Evaluation

1) Evaluation Settings: All the experiments are conducted on an AMAX deep learning workstation[3] (CPU: Intel(R) E5-2620v4, GPU: NVIDIA GeForce RTX 2080Ti), where we build an FL simulation environment and implement all models with PySyft [45], a Python library for privacy-preserving deep learning including FL, under the PyTorch framework.[4] 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.

Clients and Servers: As suggested in [18], in order to efficiently simulate the training processing of our proposed solution and baselines, totally 100 clients are generated in the simulation, and 10 (or 20) of them are randomly activated to participate in the model training. The solution can be easily extended to the case of more edge nodes. For training

[3]https://www.amax.com/products/gpu-platforms/

[4]https://pytorch.org/

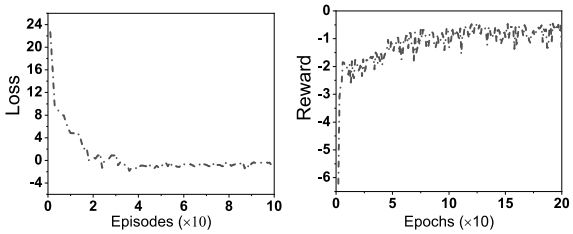


Fig. 4. Training convergence of DRL agent. *Left plot*: loss vs. No. of episodes; *Right plot*: reward vs. No. of epochs.

C10-CNN, we adopt one edge server and 10 clients with induces $[1, \cdots, 10]$, which are split into 3 different LANs with the groups ($[1, 2, 3, 4]$, $[5, 6, 7]$, $[8, 9, 10]$) in the simulation environment. While for C100-CNN, one edge server as well as 20 clients are adopted, and these clients are evenly arranged in 5 LANs with 4 clients per LAN. The communication cost within a LAN is supposed to be cheaper than the communication cost across the LANs or with the server.

Data Partition: In order to study the impact of non-IID data on the performance of model training, for C10-CNN, we partition the training datasets on the clients in two ways. 1) IID: each client is evenly and randomly allocated with the same amount of images (5,000 images per client for C10-CNN); 2) non-IID: the images are grouped by their labels, and the images of one class are only distributed to a certain client as its local data. For each client, it only holds the images of one class, which represents non-IID data on clients [6]. For C100-CNN, each client is allocated with 1) (IID) 2,500 randomly sampled training images, or 2) (non-IID) the images labeled with 5 distinct classes. Similar to the above non-IID setting, the images with 100 classes (*e.g.*, Res-ImageNet) are partitioned into 20 shards in terms of their labels, each of which contains the images labeled with 5 classes and is distributed to a specific client. In addition, the test datasets are allocated to the server for evaluating and testing the global models.

Model Training: The aforementioned three models are separately trained using *FedAvg*, *FedSwap*, *RandMigr*, *FedProx* and *FedMigr*. To better analyze the effectiveness of the stochastic model migration strategy, we unify some of the hyperparameters in the common three processes of these schemes, *i.e.*, *Model Distribution*, *Global Aggregation* and *Local Updating*. Concretely, the local iteration τ is also set as 1, which indicates that each local model is trained once between the two consecutive global updating. The frequency of model migration within a global iteration is empirically set as 49, thus the local models are aggregated every 50 epochs (as mentioned in Section II, herein, one local updating iteration is regarded as one epoch). For the sake of comparison, we are going to evaluate the performances (*e.g.*, test accuracy) of *FedAvg*, *FedSwap*, *RandMigr*, *FedProx* and *FedMigr* within the same number of training epochs. The three models and datasets (C10-CNN, C100-CNN and Res-ImageNet) are trained for 2000 epochs. Mini-batch SGD with a batch size of 64 is applied to optimize the local models.

Training the DRL Agent: The experience-driven method can be divided into the offline and online training phases. The offline training phase first simulates the environment and generates a DRL policy network based on rewards. Then, the online running phase deploys the policy network at the server and determines the migration policy for FL systems. Both two phases are performed on the parameter server (*e.g.*, cloud),

TABLE II
TEST ACCURACY (%) OF DIFFERENT MODELS TRAINED WITH FIVE SCHEMES UNDER DIFFERENT DATA SETTINGS

	C10-CNN		C100-CNN		Res-ImageNet	
	IID	non-IID	IID	non-IID	IID	non-IID
FedAvg	62.5	28.3	42.2	36.8	55.3	45.4
FedSwap	62.8	34.9	42.6	37.5	56.2	48.6
RandMigr	63.2	41.5	42.8	38.9	57.4	49.2
FedProx	62.8	31.7	42.5	37.7	55.8	47.8
FedMigr	63.7	44.7	43.2	40.7	57.9	52.4

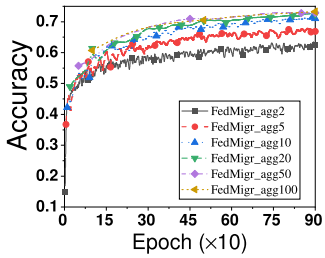


Fig. 5. Ratios of local migration vary with training epochs for different models.

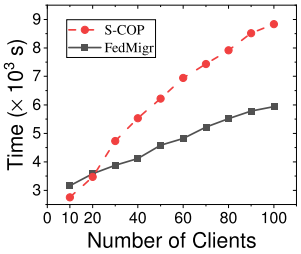


Fig. 6. Completion time vs. Number of clients under two different schemes.

which usually has sufficient computation resource compared with clients (or edge devices). Thus, the main workload of training a DRL policy network can be completed offline on the server. We first test the performance of DRL training, including the training loss and reward. In the DRL training, we use a 2-layer fully-connected feedforward neural network to serve as the actor network, which includes 128 neurons in both the first and second layers respectively. In the final output layer, we employ Relu and tanh as activation functions to ensure the output values equal to one. For the critic network, we also adopt a 2-layer fully-connected feedforward neural network, with 128 neurons in both the first and second layers respectively. The DRL training is conducted in a simulation system with 5 clients for 100 episodes. We first observe the change of training loss with the increasing number of episodes in DRL. The left plot of Fig. 4 shows that the training loss decreases quickly in the earlier stages of the DRL training process. That is because the DRL agent has no information of the FL environment which causes a large loss value at the beginning of federated training. Thanks to the efficient explorations of agents, the loss can be rapidly minimized with the model training process. After less than 30 episodes, the training loss becomes stabilized, which means the DRL agent learns to adapt to the FL environment. By the right plot of Fig. 4, we observe that the reward value increases with the epochs and gradually tends to be stable. That is because the DRL model enables one to learn a better policy for a better reward. After training 200 epochs, the reward starts to saturate with slight fluctuations.

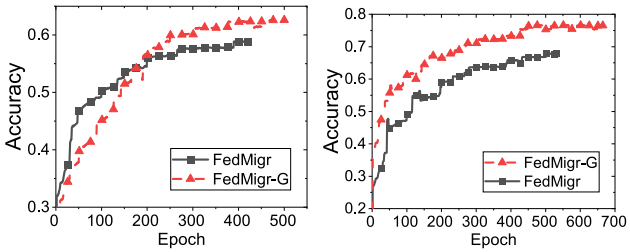


Fig. 7. Training performance of *FedMigr* and *FedMigr-G* within fixed resource budege. Left: bandwidth; right: completion time.

TABLE III
RESOURCE CONSUMPTION, *i.e.*, TRAFFIC (GB) AND TIME (H), OF FIVE DIFFERENT SOLUTIONS UNDER NON-IID SETTING

	C10-CNN		C100-CNN		Res-ImageNet	
	Traffic	Time	Traffic	Time	Traffic	Time
FedAvg	2.87	9.38	2.91	9.72	4.62	17.34
FedSwap	2.44	7.88	2.64	8.19	3.84	16.85
RandMigr	1.57	5.64	1.71	5.93	3.37	15.23
FedProx	2.65	8.62	2.77	9.15	4.35	17.16
FedMigr	1.73	4.59	1.89	4.87	2.44	13.42

2) Simulation Results: **IID vs. Non-IID:** Given a fixed number (*e.g.*, 1,000) of training epochs, the test accuracy of the three models trained with five schemes on both IID and non-IID data are shown in Table II. Since we mainly concentrate on comparing five different schemes, training models with state-of-the-art performance are beyond the scope of this work. In terms of the results, we can find that: 1) When the local data of clients follow the IID setting, the models trained with five schemes may have similar training performances, which is in accordance with the theoretical analysis in Section II-C. 2) Compared with the IID setting, the test accuracies of all the models trained on non-IID data show diverse degrees of decline. However, with the intelligent model migration strategy, *FedMigr* outperforms the other four baselines on all three models. Concretely, *FedAvg*, *FedSwap*, *RandMigr*, *FedProx* and *FedMigr* separately achieve the accuracy of 28.3%, 34.9%, 41.5%, 31.7% and 44.7% on C10-CNN. In other words, *FedMigr* improves the performance (test accuracy) over the other four baselines by about 16.4%, 9.8%, 3.2% and 13% on these baselines. The testing results demonstrate the significant effectiveness of *FedMigr*.

Resource Consumption: We test the resource consumption (*e.g.*, bandwidth and time) of five different schemes with a fixed accuracy requirement (*e.g.*, 80%). Given the simulated network topologies specified above, during the period of model training, some model migrations happen within a LAN with *RandMigr* and *FedMigr* frameworks under non-IID settings. However, for *FedSwap* and *FedAvg*, local updates always need to be transmitted to the server every epoch, which brings much more bandwidth consumption for federated training, leading to a slower convergence rate. Under our experimental setting, *FedMigr* can help to save C2S bandwidth resource for model delivery and take full advantage of the local communication in comparison with *FedSwap* and *FedAvg*. Table III shows that *FedMigr* can significantly reduce the resource consumption, including network bandwidth and completion time, of federated training compared with the other four baselines. For CIFAR100 trained over CNN, the bandwidth consumption of *FedMigr* is 1.89GB, while that of

RandMigr, *FedSwap*, *FedProx* and *FedAvg* is 1.71GB, 2.64GB, 2.77GB and 2.91GB, respectively. Therefore, *FedMigr* can reduce the bandwidth consumption by about 39.6%, 46.6% and 53.9% compared with *FedSwap*, *FedProx* and *FedAvg*, respectively. Moreover, our proposed framework can also reduce the completion time by about 21.8%, 40.5%, 46.8% and 49.9% compared with the four baselines, respectively.

Effect of Model Migration: To further evaluate the effect of the *Model Migration* process, we train multiple experiments with different configurations. We modify the frequency of model migration with diverse values when training different models, and perform *Global Aggregation* every 2 times ('agg2'), 5 times ('agg5'), 10 times ('agg10'), 20 times ('agg20'), 50 times ('agg50') and 100 times ('agg100'). The simulation results in Fig. 5 indicates that the model accuracy can get well improved with more rounds of *Model Migration* in a global iteration (from 'agg2' to 'agg100', the accuracy increases from 63% to 73%). With the increasing frequency of *Model Migration*, it is equivalent to increasing the probability of training each local model with much more data on many different clients, which can contribute to further shortening the distribution divergence as well as the parameter divergence, and improving the test accuracy.

Scalability Simulation: In order to evaluate the scalability of our proposed algorithm, we test the time required for decision making with different simulation scales, *i.e.*, different number of clients in the network. In the previous works, there is a core step in the design of approximate algorithms, that is, solving a convex optimization problem (S-COP) through programming [46], which is the most time-consuming part of an approximate algorithm. We have conducted a set of experiments to compare the consumed time for making decisio of different algorithms. With the increasing number of clients from 10 to 100, we compare the time costs of S-COP and the model inference. The experimental results in Fig. 6, indicate that the inference time of the deep model increases much more slowly with the increasing scale of clients, compared with that of S-COP.

Improvement of DRL Strategy: In order to prove that the DRL strategy does help in our proposed framework *FedMigr*, we perform the comparison between *FedMigr* and random migration schemes with different seeds (*e.g.*, 10, 20, 50), termed as *RandMigr*-10, *RandMigr*-20 and *RandMigr*-50. We separately tested the training performance of all schemes under the given bandwidth budget (500MB) and completion time (3000s). As shown in Fig. 7, the performance of our proposed schemes which adopt a DRL-based strategy always outperforms that of random model migration schemes. For example, given a fixed bandwidth budget, the accuracy of FedMigr is about 58.9%, while that of RandMigr-10, RandMigr-20 and RandMigr-50 is about 56.3%, 55.6% and 54.8%, respectively. In other words, our proposed framework which adopts the DRL strategy can averagely improve the accuracy by about 3.3% compared to the random model migration schemes.

We also test the training performance of *FedMigr-G*. Let the ResNet model be trained over the ImageNet-100 dataset with fixed bandwidth and completion time budgets. In the left plot of Fig. 7, given the bandwidth budget of 500MB, the training performance (*i.e.*, accuracy) of *FedMigr* is better than that of *FedMigr-G*. However, the increasing ratio of *FedMigr* becomes slower because it consumes more network bandwidth than *FedMigr-G*. For example, when the bandwidth budget is exhausted, the accuracy of *FedMigr* and *FedMigr-G* is about 58.9% and 62.5%, respectively. In other words, the proposed *FedMigr-G* scheme will improve the test accuracy by about 3.6% with the fixed bandwidth budget. The right plot of Fig. 7 reveals the training performance of these two schemes with a fixed completion time. *FedMigr* requires more clients than *FedMigr-G* to participate in the model training, leading to a longer completion time of each training epoch. Thus, the test accuracy of *FedMigr* is worse than that of *FedMigr-G*. For example, given 500 training epochs, the accuracy of *FedMigr* is about 68.2%, while that of *FedMigr-G* is about 75.3%. In conclusion, the performance of *FedMigr-G* is a little better than that of *FedMigr* with the constrained resource.

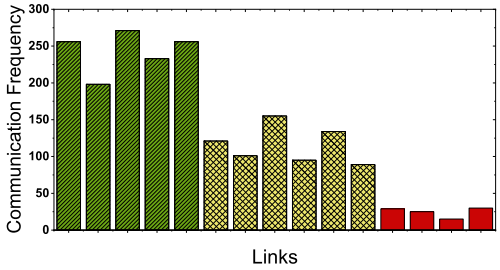


Fig. 8. The communication frequencies for 15 sampled C2C links. All sampled links are divided into three parts according to their link speeds: fast(green), moderate(yellow), slow(red).

Impact of Link Speed: Generally, the link speed of C2C communication within LANs is faster than that of C2S communication across LANs in edge computing. However, there may exist some slow C2C communication links (*e.g.*, across LANs) in practice. In order to evaluate the performance of *FedMigr* in presence of slow C2C communication, we perform a classification task with CNN over CIFAR10 dataset. Given 500 training epochs, we record the communication frequency of each C2C link. As shown in Fig. 8, the faster links are often selected for model migration with higher probabilities, *i.e.*, with the larger communication frequency. That is because the DRL agent in *FedMigr* has the ability to analyze the impact of C2C link speed on the completion time of federated training while making decisions for model migration. Thus, *FedMigr* can still achieve great performance of federated training even if there are some slow C2C communication links in the network.

Impact of activity φ: In order to evaluate the impact of φ on the performance of *FedMigr-G*, we perform the model training with CNN over the CIFAR10 dataset. We distribute the Non-IID data to the clients. In *FedMigr-G* and *RandMigr*, a larger value of φ means that fewer clients will participate in the model migration and training. In other schemes, the parameter server will aggregate more local updates from the clients when the value of φ becomes smaller. As shown in Fig. 9, the test accuracy of all schemes will decrease by changing the value of φ from 0.1 to 0.9 with the fixed bandwidth (500MB) and completion time (1,000s) budgets. However, the decreasing ratio of *FedMigr-G* is slower than that of other schemes because of the intelligent model migration among the clients. For example, given φ of 0.5, the accuracy of *FedMigr-G* is about 72.8%, while for the other four schemes, the accuracies are about 68.4%, 64.2%, 63.5% and 58.8%, respectively. In other words, our proposed scheme can improve the test accuracy by about 9% on average compared with these benchmarks with the constrained bandwidth budget.

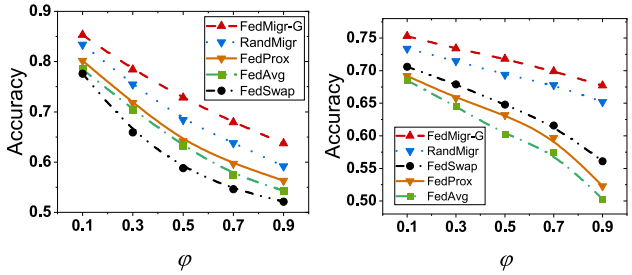


Fig. 9. Training performance by changing the value of φ with fixed resource budget. Left: bandwidth;right: completion time.

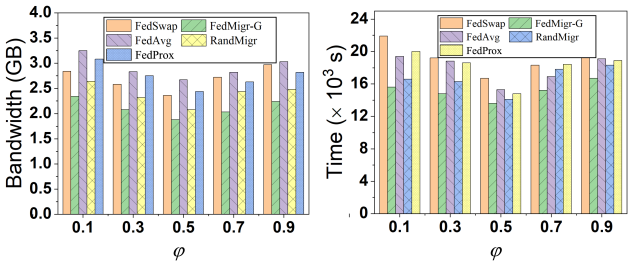
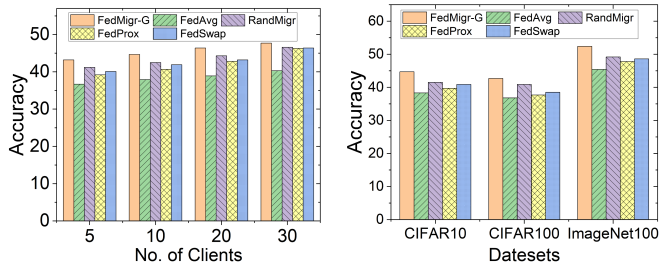


Fig. 10. Bandwidth consumption and completion time by changing the value of φ with fixed target accuracy.



(a) No. of clients vs. Accuracy (b) Datasets vs. Accuracy

Fig. 11. Scalability performance of our proposed framework and baselines.

Besides, we observe the resource consumption (network bandwidth and completion time) of model training when the target accuracy requirement is 90%. The bandwidth consumption for model training is shown in the left plog of Fig. 10. Since our proposed *FedMigr-G* scheme requires fewer training epochs than other benchmarks through intelligent model migration, especially under Non-IID settings, it requires less network bandwidth. For example, given φ of 0.5, the bandwidth consumption of *FedMigr-G* is about 1.87GB, while those of *RandMigr*, *FedSwap*, *FedProx* and *FedAvg* are about 2.03GB, 2.36GB, 2.42GB and 2.67GB, respectively. Thus, *FedMigr-G* reduces the required network bandwidth by about 7.9%, 20.8%, 22.7% and 30% compared with these schemes, respectively. Moreover, *FedMigr-G* can also reduce the completion time of model training, which is shown in the right plot of Fig.9. For example, the completion time of *FedMigr-G* is 13,624s, while that of *RandMigr*, *FedSwap*, *FedProx* and *FedAvg* is about 14,135s, 16,757s, 14,824s and 15,393s, respectively. In conclusion, our proposed scheme can significantly reduce resource consumption, including network bandwidth and completion time of model training, under different values of φ.

Scalability evaluation: In order to evaluate the scalability of our proposed algorithm, we perform two groups of experiments (CNN trained over CIFAR10) in terms of the number of clients and the size of the dataset. First, we observe the test results with a varied number of clients (*e.g.*, 5, 10, 20 and 30). As shown in Fig. 11(a), with the increase in the number of clients, training performance significantly improves, however, resulting in more resource consumption. Our proposed framework FedMigr-G can achieve superior training performance with less resource consumption compared to the baselines under non-IID settings. Then, we also test the performance of our proposed framework and baselines with different sizes of the dataset (*i.e.*, CIFAR10, CIFAR100 and ImageNet100). Fig. 11(b) shows that the training performance of our proposed framework FedMigr-G outperforms the baseline in datasets of varying sizes. In summary, our proposed framework demonstrates excellent scalability, both in terms of varying numbers of clients and datasets of different sizes.

D. Test-Bed Evaluation

1) Implementation on the Platform: We implement five different schemes on a real test-bed environment, which is composed of two main parts: a deep learning workstation with four NVIDIA GeForce RTX Titan GPUs and 30 devices, including 15 NVIDIA Jetson TX2[5] and 15 NVIDIA Jetson Xavier NX.[6] Each TX2 has one GPU and one CPU cluster, which consists of a 2-core Denver2 and a 4-core ARM CortexA57 with 8GB RAM, and its OS is Ubuntu 18.04.4 LTS. Each NX with Ubuntu 18.04.5 LTS is equipped with a 6-core NVIDIA Carmel ARMv8.2 CPU and a 384-core NVIDIA Volta GPU with 8GB RAM. Specifically, the workstation acts as the parameter server which is responsible for the model aggregation and verifying the training performance of global model. We adopt a TX2 or NX as a worker to locally train the model and send the updates to the parameter server for aggregation. To represent the real-world edge computing environment where the parameter server is always located at remote cloud and communicates with the edge servers (devices) at network edge via WAN, we place them at different locations at least 2,000 meters apart and let them communicate through the link with a bandwidth of about 50Mbps, which is measured by iperf3.[7] Meanwhile, since the transmission delay between devices and edge servers is much shorter in practical scenario, we arrange those equipment at different locations in a laboratory of about 400m^2 and let them communicate through the highspeed 5GHz WiFi.

We develop a distributed model training framework with PyTorch.[8] The clients and the parameter server are connected via a router over the same wireless network. Besides, they are logically connected for communication through *torch.distributed* package. Specifically, the IP address of the server and a designated port are combined to establish a connection between the server and the worker through TCP protocol. After the connection is established, the server segments the training and testing datasets, and sends the segmentation results to each worker. After receiving the results, the worker generates the local dataset for training.

Data Distribution on Clients: The different categories of data distributions, *i.e.*, IID and non-IID, among the clients have a great impact on the performance of model training. In the experiments, we mainly consider the following five different cases to verify the effect of data distributions on model training, including IID data and the four different

[5]https://developer.nvidia.com/embedded/jetson-tx2-developer-kit
[6]https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/
[7]http://software.es.net/iperf/index.html
[8]https://pytorch.org/

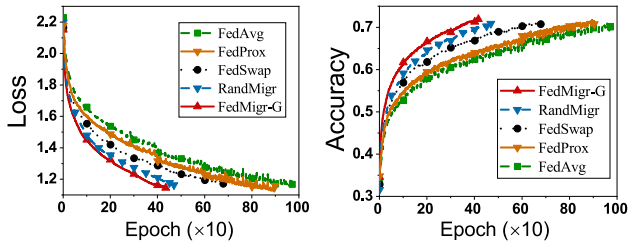


Fig. 12. Training performance of loss and accuracy with CNN trained over CIFAR10 in test-bed.

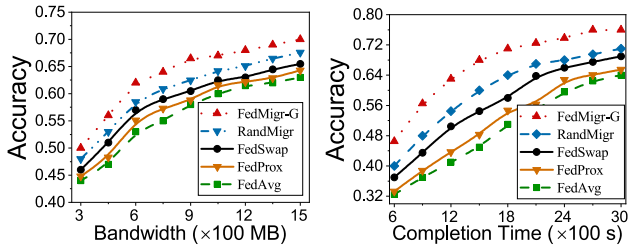


Fig. 13. Test accuracy with bandwidth and completion time constraint (CNN over CIFAR10) in test-bed.

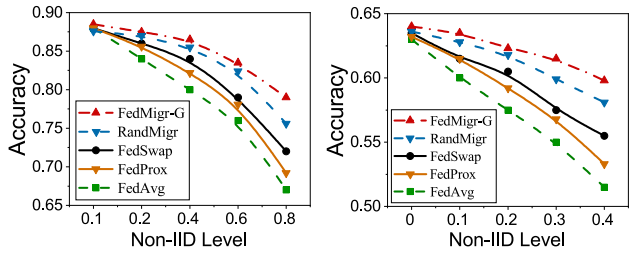


Fig. 14. Test accuracy of C10-CNN and C100-CNN with different Non-IID Levels in test-bed.

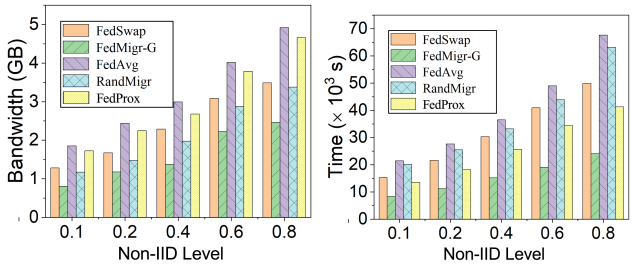


Fig. 15. Bandwidth consumption and completion time of C10-CNN with different Non-IID levels in test-bed.

levels of non-IID data. For CIFAR10, each worker has $p\%$ ($p = 10, 20, 40, 60$ and 80) of a unique class in 10 classes and the remaining samples of each class are partitioned to other clients uniformly. Note that $p = 10$ is a special case, where the distribution of the training dataset is IID. We denote the five different cases of data distributions as 0.1, 0.2, 0.4, 0.6 and 0.8 over CIFAR10. For CIFAR100, each worker lacks p ($p = 0, 10, 20, 30$ and 40) classes of data samples, and the samples of one class are distributed on only $(10-p/10)$ clients uniformly. Particularly, $p = 0$ also represents uniform data distribution. We denote the cases of data distributions as 0, 0.1, 0.2, 0.3 and 0.4 over CIFAR100.

2) Testing Results: We perform three groups of experiments to evaluate the efficiency of our proposed framework.

Convergence Performance: In the first set of experiments, we observe the training performance of five different schemes with a fixed accuracy requirement (*e.g.*, 80%). The model migration in *FedMigr* and *RandMigr*, which is equivalent to training the local model over more data on different clients, can well reduce the influence of non-IID issue through model migration, accelerating the process of training. Fig. 12 shows that the required number of training epochs of *FedMigr* is less than the other four baselines. For *FedMigr*, given the accuracy requirement of 80%, the required number of epochs is about 385, while that of *RandMigr*, *FedSwap*, *FedProx* and *FedAvg* is about 468, 679, 884 and 972, respectively. In other words, *FedMigr* can reduce the required number of epochs by about 17.7%, 43.3%, 56.4% and 60.4% compared with *RandMigr*, *FedSwap*, *FedProx* and *FedAvg*, respectively. When a given accuracy is achieved, our proposed framework can complete model convergence at a faster rate.

Effect of Resource Constraints: The second set of experiments observes the performance (CNN trained over CIFAR10) of federated training with resource constraints (*e.g.*, network bandwidth and completion time). Local model migration between the clients in *FedMigr* and *RandMigr* will accelerate the training process with less number of epochs compared with *FedAvg*, *FedProx* and *FedSwap*, which can reduce the bandwidth consumption and completion time of federated training. The left plot of Fig. 13 shows that the test accuracy increases for all solutions with the increasing bandwidth budget. However, the training performance of *FedMigr* is better than the other four schemes. For instance, given the bandwidth budget of 1GB, the test accuracy of *FedMigr* is about 65.7%, while that of *RandMigr*, *FedSwap*, *FedProx* and *FedAvg* is about 63.3%, 60.5%, 58.8% and 57.4%, respectively. Therefore, *FedMigr* can improve the test accuracy by about 2.4%, 5.2%, 6.9% and 8.3% compared with *RandMigr*, *FedSwap*, *FedProx* and *FedAvg*.

As shown in the right plot of Fig. 13, more time budgets will significantly improve the test accuracy of all solutions. However, our *FedMigr* will achieve higher test accuracy than the four baselines with the same completion time (*e.g.*, 3,000s). For example, the accuracy of *FedMigr* is about 75.8%, while that of *RandMigr*, *FedSwap*, *FedProx* and *FedAvg* is about 70.7%, 68.3%, 65.8% and 63.5%, respectively. That is, *FedMigr* can improve the accuracy by about 5.1%, 7.5%, 10% and 12.3% compared with the baselines. Therefore, regardless of whether resources are abundant or scarce, our proposed framework always achieves better performance than the baselines.

Different Non-IID Levels: The last set of experiments tests the training performance (CNN trained over CIFAR10 and CIFAR100) of five schemes under different non-IID levels. As shown in Fig. 13, the training performance (*e.g.*, test accuracy) will be degraded with the increasing of non-IID levels. The local model migration will significantly improve the weight divergence caused by non-IID data. Thus, *FedMigr* and *RandMigr* can achieve better performance of federated training than the other three baselines, especially under a large non-IID level. For example, given C100-CNN with 1,000 epochs and non-IID level of 0.4, the test accuracy of *FedMigr* is about 61.7%, while that of *RandMigr*, *FedSwap*, *FedProx* and *FedAvg* is about 58.2%, 55.3%, 53.5% and 51.6%. In other words, *FedMigr* will improve the test accuracy by about 3.5%, 6.4%, 8.2% and 10.1% compared with *RandMigr*, *FedSwap*, *FedProx* and *FedAvg*, respectively.

Besides, we observe the bandwidth consumption and completion time of federated training (CNN trained over CIFAR10) with different non-IID levels. Fig. 15 shows that

both bandwidth consumption and completion time of model training increase with the level of non-IID. However, the increasing ratio of *FedMigr* is much slower than the other four baselines. In comparison, *FedMigr* requires less bandwidth consumption and completion time than *RandMigr*, *FedSwap*, *FedProx* and *FedAvg*. For instance, given 2000 epochs, the completion time of *FedMigr* is about 19,372s if the level of non-IID is 0.6, while that of *RandMigr*, *FedSwap* and *FedAvg* is about 34,384s, 40,929s, 43,914s and 48,942s, respectively. Therefore, *FedMigr* can reduce the completion time of model training by about 43.7%, 52.7%, 55.8% and 60.4%, respectively. In summary, the experimental results show that, under various non-IID settings, our proposed framework significantly outperforms the baselines in both convergence performance and resource consumption. This also demonstrates that the model migration in our proposed framework can indeed reduce the parameter divergence caused by non-IID data, which is consistent with the theoretical analysis in Section II-C.

VI. Related Works

Recently, federated learning (FL) [47] has been widely studied in both academia and industry fields. One research area related to FL is distributed machine learning (DML) on worker machines and parameter servers [26]. Bao et al. [48] propose an online algorithm for scheduling the arriving jobs and deciding the number of concurrent workers and parameter servers for each job over its course, so as to maximize the overall utility of all jobs. The authors in [49] propose a parameter server based distributed computing framework for training large-scale deep neural networks. Besides, the authors introduce a new learning rate modulation strategy to counter the effect of stale gradients, and propose a new synchronization protocol that effectively bound the staleness in gradients, improving runtime performance and testing accuracy. Li et al. [50] describe a third-generation parameter server framework that introduces two relaxations to balance system performance and algorithm efficiency for distributed machine learning. The authors also propose a new algorithm that takes advantage of this framework to solve non-convex and non-smooth problems with a convergence guarantee.

The above works mainly study efficient solutions of DML in datacenters where the shared storage is usually adopted. The worker machines will not keep persistent data storage, but fetch the data from the shared storage at the beginning of the learning process. As a result, the data samples on different workers are usually IID in datacenters.

In federated learning, the data are collected at the edge directly and stored persistently at edge nodes, thus the data distribution at different edge nodes is usually non-IID and imbalanced, which is different from DML in datacenters [13]. Regarding the performance degradation incurred by non-IID data, several researches have been made in the literature. For example, Li et al. [51] analyze the convergence of FedAvg on non-IID data and establish a convergence rate for convex and smooth problems. Zhao et al. [13] show that the non-IID CIFAR-10 dataset at clients degrades the accuracy of FedAvg. To address this issue, Zhao et al. [13] allow the public available data to be distributed to clients such that the influence of Non-IID is alleviated. However, clients are in general reluctant to receive unknown data on their devices for security reasons. FedMA [52] proposes an aggregation strategy for non-iid data partition that shares the global model in a layer-wise manner. Li et al. [15] propose to add a proximal term to the local optimization sub-problems in order to limit the impact of variable local updates. This problem is phrased as client drift in [53] and a set of control variates are delivered between the clients and server to correct for this drift. The work in [54] proposes a Hybrid-FL, which allows a few number of clients to upload their data to the server, so as to form an approximately IID dataset. Smith et al. [55] show that multi-task learning is naturally suited to handle the statistical challenges of this setting, and propose a novel systems-aware optimization method that is robust to practical systems issues. Sattler et al. [56] construct a sparse ternary compression (STC) for FL, which requires both fewer gradient evaluations and communication resource to converge to a target accuracy.

Finally, we introduce some works (*e.g.*, FedSwap [21]) similar to our research. The key idea of FedSwap is to perform model swapping between any two of all clients at the PS, instead of running FedAvg every iteration. This operation of swapping the models between the clients gives each model a bigger view on the entire dataset, so as to reduce weight divergence. However, model swapping at the server still brings an enormous amount of C2S communication traffic to the PS as in FedAvg, which will become the bottleneck of FL. Besides, without considering the data distributions on the clients, random model swapping between two clients cannot well deal with the non-IID issue.

Compared with previous works, our proposed method *FedMigr* integrates the model migration strategy into the *FedAvg* algorithm to simultaneously cope with the challenges of non-IID data and limited communication resource. *FedMigr* will significantly speed up the process of federated training with less resource cost, and enhance the trained model even under the non-IID setting.

VII. Conclusion

In this work, we propose the *FedMigr* framework, which integrates an intelligent model migration strategy into the *FedAvg* algorithm to address the challenges of non-IID data and limited bandwidth resource raised by emerging FL in heterogeneous edge computing. We have built a simulated and a real test-bed FL environment to evaluate the performance of *FedMigr* via extensive experiments with three popular benchmark datasets. The results demonstrate the effectiveness of *FedMigr* for improving the model accuracy and reducing resource consumption (*e.g.*, bandwidth and time) in heterogeneous edge computing.

References

[1] J. Liu, Y. Xu, H. Xu, Y. Liao, Z. Wang, and H. Huang, "Enhancing federated learning with intelligent model migration in heterogeneous edge computing," in *Proc. IEEE 38th Int. Conf. Data Eng. (ICDE)*, May 2022, pp. 1586–1597.

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[3] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[4] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*.

[5] R. K. Bakshi, N. Kaur, R. Kaur, and G. Kaur, "Opinion mining and sentiment analysis," in *Proc. 3rd Int. Conf. Comput. Sustain. Global Develop. (INDIACom)*, Mar. 2016, pp. 452–455.

[6] S. Wang et al., "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.

[7] 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.*, vol. 54, 2017, pp. 1273–1282.

[8] 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.*, vol. 23, no. 5, pp. 6012–6026, May 2024.

[9] J. Konečný, 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*.

[10] Z. Yao, J. Liu, H. Xu, L. Wang, C. Qian, and Y. Liao, "Ferrari: A personalized federated learning framework for heterogeneous edge clients," *IEEE Trans. Mobile Comput.*, 2024.

[11] K. Dolui and S. K. Datta, "Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing," in *Proc. Global Internet Things Summit (GIoTS)*, Jun. 2017, pp. 1–6.

[12] K. Hsieh et al., "Gaia: Geo-distributed machine learning approaching LAN speeds," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 629–647.

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

[14] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, "Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 38–67, 1st Quart., 2020.

[15] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," 2018, *arXiv:1812.06127*.

[16] M. Duan et al., "Astraea: Self-balancing federated learning for improving classification accuracy of mobile deep learning applications," in *Proc. IEEE 37th Int. Conf. Comput. Design (ICCD)*, Nov. 2019, pp. 246–254.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[18] 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.

[19] S. Wang et al., "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 63–71.

[20] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," 2019, *arXiv:1903.03934*.

[21] T.-C. Chiu, Y.-Y. Shih, A.-C. Pang, C.-S. Wang, W. Weng, and C.-T. Chou, "Semisupervised distributed learning with non-IID data for AIoT service platform," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9266–9277, Oct. 2020.

[22] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. 19th Int. Conf. Comput. Statist.*, Paris, France, 2010, pp. 177–186.

[23] J. Dean et al., "Large scale distributed deep networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, vol. 1, 2012, pp. 1223–1231.

[24] J. Yan, J. Liu, H. Xu, Z. Wang, and C. Qiao, "Peaches: Personalized federated learning with neural architecture search in edge computing," *IEEE Trans. Mobile Comput.*, 2024.

[25] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proc. 22nd Int. Conf. World Wide Web*. New York, NY, USA: ACM, May 2013, pp. 37–48.

[26] 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.

[27] I. Sabek and M. F. Mokbel, "Machine learning meets big spatial data," in *Proc. IEEE 36th Int. Conf. Data Eng. (ICDE)*, Apr. 2020, pp. 1782–1785.

[28] J. Huang et al., "PhyFinAtt: An undetectable attack framework against PHY layer fingerprint-based WiFi authentication," *IEEE Trans. Mobile Comput.*, 2023.

[29] 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.

[30] Q. Ma, Y. Xu, H. Xu, J. Liu, and L. Huang, "FedUC: A unified clustering approach for hierarchical federated learning," *IEEE Trans. Mobile Comput.*, 2024.

[31] J. Liu, H. Xu, G. Zhao, C. Qian, X. Fan, and L. Huang, "Incremental server deployment for scalable NFV-enabled networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Jul. 2020, pp. 2361–2370.

[32] A. Rakhlin, O. Shamir, and K. Sridharan, "Making gradient descent optimal for strongly convex stochastic optimization," in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 1571–1578.

[33] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jun. 2019, pp. 1468–1476.

[34] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, 2016.

[35] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, Apr. 2017.

[36] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[37] M. Roderick, J. MacGlashan, and S. Tellex, "Implementing the deep Q-network," 2017, *arXiv:1711.07478*.

[38] M. Tokic and G. Palm, "Value-difference based exploration: Adaptive control between epsilon-greedy and softmax," in *Proc. Annu. Conf. Artif. Intell.* Springer, 2011, pp. 335–346.

[39] M. Grant and S. Boyd. (2014). *Cvx: MATLAB Software for Disciplined Convex Programming, Version 2.1*. [Online]. Available: https://cvxr.com/cvx/

[40] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*.

[41] L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks Trade*, 2nd ed. Cham, Switzerland: Springer, 2012, pp. 421–436.

[42] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Toronto, ON, Canada, 2009.

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

[44] X. Zhang, Z. Li, C. C. Loy, and D. Lin, "PolyNet: A pursuit of structural diversity in very deep networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 3900–3908.

[45] T. Ryffel et al., "A generic framework for privacy preserving deep learning," 2018, *arXiv:1811.04017*.

[46] J. Wang, A. K. Sahu, Z. Yang, G. Joshi, and S. Kar, "MATCHA: Speeding up decentralized SGD via matching decomposition sampling," in *Proc. 6th Indian Control Conf. (ICC)*, Dec. 2019, pp. 299–300.

[47] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016, *arXiv:1602.05629*.

[48] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 495–503.

[49] S. Gupta, W. Zhang, and F. Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in *Proc. IEEE 16th Int. Conf. Data Mining (ICDM)*, Dec. 2016, pp. 171–180.

[50] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 27, 2014.

[51] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of FedAvg on non-IID data," 2019, *arXiv:1907.02189*.

[52] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," 2020, *arXiv:2002.06440*.

[53] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning," in *Proc. 37th Int. Conf. Mach. Learn.*, vol. 119, Jul. 2020, pp. 5132–5143.

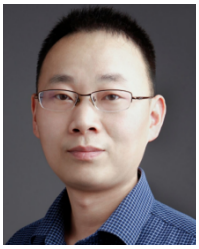
[54] N. Yoshida, T. Nishio, M. Morikura, K. Yamamoto, and R. Yonetani, "Hybrid-FL for wireless networks: Cooperative learning mechanism using non-IID data," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–7.

[55] V. Smith, C. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," in *Proc. 30th Annu. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4424–4434.

[56] 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. 2019.

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 main research interests are software defined networks, network function virtualization, edge computing, and federated learning. He is a member of ACM.

Shilong Wang received the B.S. degree from Sichuan University in 2022. He is currently pursuing the master's degree with the School of Computer Science, University of Science and Technology of China (USTC). His main research interests are edge computing, deep learning, and federated learning.



Hongli Xu (Member, IEEE) received the B.S. degree in computer science from the University of Science and Technology of China, China, in 2002, and the Ph.D. degree in computer software and theory from the University of Science and Technology of China (USTC), China, in 2007. 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 main 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 in several famous conferences.

Yang Xu (Member, IEEE) received the B.S. degree from Wuhan University of Technology in 2014 and the Ph.D. degree in computer science and technology from the University of Science and Technology of China in 2019. He is currently an Associate Researcher with the School of Computer Science and Technology, University of Science and Technology of China. His research interests include ubiquitous computing, deep learning, and mobile edge computing. He is a member of ACM.

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

Jinyang Huang (Member, IEEE) received the Ph.D. degree from the School of Cyberspace Security, University of Science and Technology of China, in 2022. He is currently a Lecturer with the School of Computer and Information, Hefei University of Technology. His research interests include wireless security and wireless sensing. He is a TPC Member of ACM MM, IEEE ICME, and Globecom. He is also an Editorial Board Member of *Applied Sciences*.

He Huang (Senior Member, IEEE) received the Ph.D. degree from the School of Computer Science and Technology, University of Science and Technology of China (USTC), in 2011. He is currently a Professor with the School of Computer Science and Technology, Soochow University, China. His current research interests include traffic measurement, computer networks, and algorithmic game theory. He is a member of ACM.