Enhancing Federated Learning with Intelligent Model Migration in Heterogeneous Edge Computing

Jianchun Liu[1,3] Yang Xu[2,3] Hongli Xu[2,3] Yunming Liao[2,3] Zhiyuan Wang[2,3] He Huang[4]
[1]School of Data Science, University of Science and Technology of China, China
[2]School of Computer Science and Technology, University of Science and Technology of China, China
[3]Suzhou Institute for Advanced Research, University of Science and Technology of China, China
[4] School of Computer Science and Technology, Soochow University, China

***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 constraints, our *FedMigr* will intelligently guide one client to forward its local model to another client after local updating, rather than 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 it over more data from different clients, contributing to alleviating the influence of non-IID issue. We prove that *FedMigr* can help to reduce the parameter divergences between different local models and the global model from a theoretical perspective, even 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*—Non-IID data, 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) [1], 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) [2].

Recently, mobile Internet has been boosting the growth explosion of user data. More and more data-driven machine learning applications (*e.g.*, machine translation [3] or sentiment analysis [4]) are becoming appealing for consumers and researchers. It is predictable that machine learning tasks will be a dominant workload in MEC systems [5]. 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 [6]. 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) [6], which enables the so-called on-device intelligence [7]. Specif-

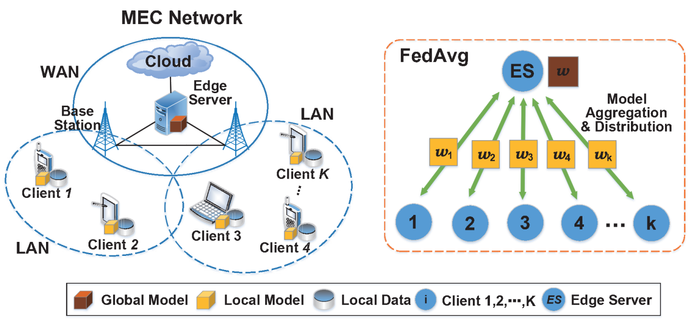

Fig. 1: Illustrative depictions of MEC network and *FedAvg*.

coordinator (*e.g.*, an edge server) maintains a globally shared model. As shown in Fig. 1, the traditional *FederatedAveraging* (or *FedAvg*) [6] algorithm lets the server iteratively aggregate all the local models to update the global model, and distributes the fresh global model back to the clients for further training. 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 typical distributed machine learning in high-performance datacenters, FL will face two major challenges in MEC [6]. 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, different clients probably have very different data distributions from others. Data samples of different clients are usually not independent and identically distributed (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 [8]. 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) [9]. Generally, the communication bandwidth between clients and the edge server is relatively more scarce than that within a datacenter or among the clients over LANs [10]. Thus, communication is likely to be the bottleneck of the MEC network and the distributed model training, since the client-to-server (C2S) communication is probably more time-consuming than a single training iteration on each client [5].

'o address the non-IID challenge, Zhao *et al.* [8] and Huang *et al.* [11] propose to improve the performance of model training by distributing a globally shared proxy data to all clients, which requires extra efforts to maintain such auxiliary data for dynamic scenarios carefully [12]. Li *et al.* [13] propose a framework called *FedProx* to tackle systems heterogeneity and statistical heterogeneity, *i.e.*, non-IID data, in federated networks. *FedProx* can be viewed as a generalization and re-parametrization of *FedAvg*, which makes only minor modifications to it. The vanilla data augmentation technique is adopted by [14] to increase the diversity of training data by random transformation or knowledge transfer, which can also be used to mitigate non-IID issues in FL. 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 a enormous amout of traffic at parameter server (PS) [15]. With the advanced reinforcement learning (RL) techniques [16], [17], Wang *et al.* [18] develope an experience-driven control framework to counterbalance the bias induced by non-IID data. But the proposed framework cannot reduce the communication overhead for model delivery.

As for the communication concern, especially in MEC, some researchers study the communication-efficient FL schemes. For example, based on the pioneer *FedAvg* algorithm [6], Wang *et al.* [5], [19] design a control algorithm to dynamically adjust the frequency of global aggregation 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 under the non-IID setting. Xie *et al.* [20] design an asynchronous federated optimization algorithm with low bandwidth consumption, in which the parameter server will perform the global update with only one local updated model from an arbitrary client. But the asynchronous method, which does not aggregate local models from all clients with different data distributions, cannot well deal with the non-IID issue [8]. 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* for performance improvement. 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, which has been validated through extensive simulations in Section IV.

Motivated by this, given clients with non-IID local datasets and limited communication budget, we propose the *FedMigr* framework, which integrates the model migration strategy into the *FedAvg* algorithm 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 client, which is termed as model migration and helps to alleviate the C2S communication, since the client-to-client (C2C) communication can occur over LANs. Besides, migrating a local model from one client to another is equivalent to training it over more data from different clients, which can alleviate the influence of non-IID issue. Powered by the advanced deep reinforcement learning (DRL), *FedMigr* will intelligently and dynamically determine the migration policy which indicates the migration destinations for the local models in terms of the state information (*e.g.*, difference of data distribution and resource usage) from the real-time FL environment. Thus, our *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. The main contributions of this paper are summarized below.

- To approach the challenges of non-IID data and limited communication resource raised by FL in MEC, we propose an efficient FL framework, termed *FedMigr*, in which model migration is integrated in *FedAvg*.
- We analyze the effectiveness of *FedMigr* from a theoretical perspective, and explain that the model migration can help to reduce the parameter divergences between different local models and the global model, enhancing the federated training given non-IID local data.
- We propose experience-driven algorithms based on DRL to adaptively determine the optimal migration policy in *FedMigr*, so as to achieve less training time and bandwidth resource 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 communication resource consumption for global communication by 42% on average, compared with baselines.

The rest of this paper is organized as follows. Section II first introduces some preliminaries, including distributed machine learning and federated learning. We then propose the *FedMigr* framework, give the convergence analysis, and formalize the problem in Section II. The efficient algorithms for the problem are proposed in Section III. We report our simulation and experimental results in Section IV. We conclude the paper in Section V.

II. Preliminaries and Problem Formulation

A. Distributed Machine Learning vs. Federated Learning

Distributed Machine 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 $f_i(w) = \ell(w, x_i, y_i)$ is defined to measure the error of the predicted output made with model parameter w. The ning 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 get 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 [24], [25]. Considering the distributed architecture based on parameter server (PS) [15], given K clients (or edge nodes), each client is allocated with a dataset $\mathcal{D}_k$ ($k \in \{1, 2, ..., K\}$). According to [5], we mainly consider the synchronous implementation due to its popularity and satisfied performance in practice [6], [15], leaving the asynchronous setting as our future direction. For ease of description, we assume the intersection between any two allocated datasets is empty, which means $N = \sum_{k=1}^{K} n_k$, and n_k is size of $\mathcal{D}_k$ [26]. 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. Let w_k and w_g separately denote the parameters of the local model and the global model. All local models are initialized with the same parameter of the global model when $t = 0$, *i.e.*, $w_k(0) = w_g(0)$, $\forall k \in \{1, ..., K\}$. Subsequently, there are three main processes in each training epoch $t \in \{1, ..., T\}$, where T is the total number of training epochs. (1) *Data & Model Distribution*: The server first randomly distributes n_k IID data samples along with 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

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

Federated Learning: With the development of MEC and AI chipsets, model training can be implemented from clouds to edges. McMahan *et al.* [6] proposed an emerging alternative of DML, termed federated learning (FL). In MEC, all the clie

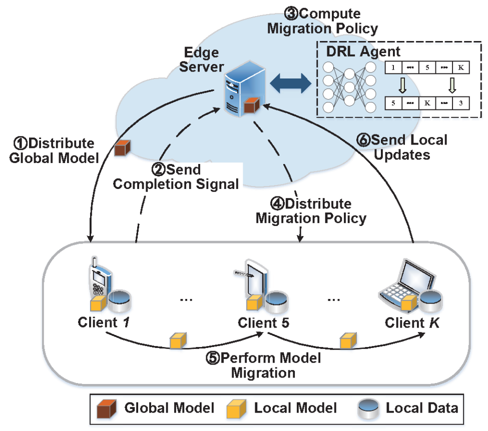

Fig. 2: The workflow of *FedMigr*.

are interconnected with the edge server via the mobile networks (*e.g.*, cell networks), and each client has the ability to store its local data and performs ML tasks. 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.

Typically, *FedAvg* consists of three core processes, *i.e.*, *Model Distribution*, *Local Updating* and *Global Aggregation*. Two extra parameters are introduced to reduce computation and communication costs, namely α, the fraction of clients that perform *Local Updating* in each global iteration; and τ, the number of local training epochs each client runs over its local dataset. Therefore, in each global iteration, for a selected client k (αK out of K clients are selected) 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. The theoretical convergence bound of *FedAvg* can be found in [5].

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 $g \in \{1, ..., G\}$, where G is the total number of global iterations. Fig. 2 shows the general 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 of size n_k (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), it computes the *migration policy* $\mathbf{P} = [p_{i,j}^t], \forall i,j \in \{1,...,K\}$. 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 algorithm based on DRL technique, which will be introduced in 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, which is regarded as *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 [6].
- After that, client j again performs local updating on the basis of the model of client i after epoch t 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, *Global Aggregation* is performed by the server to aggregate the local updates from the clients and derive 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 consumption 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 mainly analyze how *FedMigr* reduces the distance between the distribution of local data and the population distribution, and helps to improve model accuracy. Considering a classification problem defined over the dataset $\mathcal{D}$ with n^l ($l \in \{1,...,L\}$) samples for type 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 parameter of the centralized model. While 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 [8], 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 all the clients have the same data distribution as 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. From this point of view, we regard *Local Updating* and *Model Migration* as an integrated process of local model updating. Therefore, the convergence can be guaranteed in terms of the demonstrations in [5], [19]. 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 of model migration, 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 can observe that

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

$\|q_k'(y=l) - q(y=l)\| < \|q_k(y=l) - q(y=l)\|$. In terms of Eqs. (14) and (15), we find that *FedMigr* can help 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*.

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. For the other three processes, the communication cost is dominated. 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 $b_{0,k} = b_{k,0} > 0$ (for ease description, written as b_k). To be noted that, considering the sufficient computing power on the server, the computation cost for aggregating local updates and computing the latest global model is also neglected [1]. Besides, the communication cost for delivering the model from client i to client j in the m-th ($\forall m \in \{1, 2, ..., M\}$) 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:

$$\min_{T\in\{1,2,3,...\}} F(w^T)$$

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

where $p_{i,j}^m$ denotes whether the model on client i will be migrated to client j 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 of the FLMM problem is to minimize the loss function of federated learning.

In fact, it is difficult to directly solve the FLMM problem in Eq. (16). Since the model migration decision variable is a bool one, this is a typical integer programming problem. In general, finding the optimal solution for an integer programming problem is NP-hard [27], 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 reinforcem

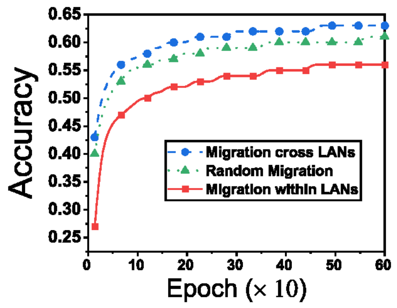

Fig. 3: Test accuracy of model training with *FedMigr* given different model migration strategies.

learning based method to learn an effective solution for the FLMM problem.

III. Algorithm Design

In this section, we first give an example to motivate our algorithm design (Section III-A). We briefly introduce the basic mechanism of deep reinforcement learning (DRL) in Section III-B. Then, we describe the model design of the DRL agent in detail (Section III-C). The training methodology of the experience-driven algorithm for migration policy generation (EMPG) is presented in Section III-D. Finally, we discuss some practical issues to enhance the proposed solution (Section III-E).

A. Motivation for Algorithm Design

FL always relies on SGD which has been widely used in training deep networks with good empirical performance [28]. 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 local 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 [10], [29]. 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 [8]. To this end, we perform two groups of tests to illustrate the motivation for algorithm design.

We first observe the performance of model training with *FedMigr* under three different model migration strategies, *i.e.*, migration cross LANs, random migration and migration within LANs. 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. 3, the model accuracy of migration cross LANs is better than that of random migration and migration within LANs. For instance, given 500 epochs, the accuracy of model migration cross LANs is about 63.6%, while that of model migration within LANs and random migration 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.

ıttp://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. Our *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 [30]), 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 can not perceive the detailed and real-time state information (*e.g.*, the resource consumption and process of training) of network environment [17], [31]. 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 [17]. 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 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 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 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. $D_t = [d^t_{i,j}], \forall i, j \in \{1, ..., K\}$ 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 $[p^t_{i,j}], \forall i, j \in \{1, ..., K\}$, 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 $\{1, 2, ..., K\}$ 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 [32] 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 $\pi(s_t|\theta) \to [0, 1]$, 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. (16), 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 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 [33]. 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) [34], where ψ is the weight vector of the DQN. The critic function returns Q for a given state-action pair at epoch t as follow:

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

The actor function $\pi(s_t|\theta)$ can be implemented using a Deep Neural Networks (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:

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

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

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 ϱ-greedy method [35] to meet our demands, where $\varrho \in [0, 1]$ denotes an adjustable parameter. It means that with ϱ probability, the agent derives actions by solving the FLMM problem in Eq. (16); and with $(1 - \varrho)$ 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. (16), which is NP-Hard due to the integer variables [27]. To solve the FLMM problem, we first deal with the integer variable by relaxing it to be any fractional value in $[0, 1]$ [36]. 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 [37]). . By adjusting parameter ϱ, 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 [38], which specifies samples with careful considerations 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 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:

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

The priority of the transition data z is given by:

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

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:

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

where the parameter ξ controls to what extent the prioritization is used. If $\xi = 0$, 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 eriences since even those with low TD errors can still have

Algorithm 1 Experience-Driven Migration Policy Generation

```
1:  Initialization
2:  Initialize the migration policy matrix $\mathbf{P} = [p^0_{i,j}], \forall i, j$
3:  Initialize critic networks $Q(\cdot)$ and actor networks $\pi(\cdot)$ with
    weights $\psi$ and $\theta$, 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 $\mathbf{P}$
```

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 accumulate weight-change for critic and actor networks are:

$$\Xi_\psi := \Xi_\psi + \mu_z \cdot \phi_z \cdot \nabla_\psi Q\left(s^z, a^z\right) \tag{27}$$

$$\Xi_\theta := \Xi_\theta + \mu_z \cdot \nabla_a Q(s^z, a^z) \tag{28}$$

where μ_z is the important-sample weight:

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

The accumulated weight-change will be used for updating the critic network and actor network. The EMPG algorithm is described in Alg. 1.

The EMPG algorithm includes three stpng: 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 upd 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 (or migration policy) will be selected for the next training epoch by the output of actor network (Lines 21-22).

E. Discussion

In this section, we discuss some practical issues to enhance the proposed method.

1) Note that some C2C communications (*e.g.*, C2C communication across LANs) may be slower than C2S communications. How to determine the model migration strategy depends on some important factors (*e.g.*, communication budgets and link speed). For making decisions of model migration, the DRL agent has the ability to analyze the impact of link speed of both C2C and C2S communication on the completion time of federated training. Based on the analysis results, the algorithm takes the speed of C2C/C2S communication links and communication budgets as the input, and selects efficient links for model migration. In the extreme case of very low C2C communication links, there is no model migration between clients. As a result, the worst-case cost of FedMigr will not exceed that of FedAvg, where clients only forward the local models to parameter servers. The experimental results in Section IV also show the effectiveness of C2C communications in our proposed method.

2) FL enables local training on workers without exchanging personal data between the server and clients, thereby protecting clients' data from being eavesdropped by hidden adversaries. Our proposed method migrates not data, but models between clients. Nevertheless, private information may still be divulged to some extent from adversaries' analyzing on the differences of related model parameters, *e.g.*, parameters trained in neural networks. It naturally can prevent information leakage by adding artificial noises, known as differential privacy (DP) techniques, including local DP (LDP) [39] and centralized DP [40]. The previous works also give the theoretical analyses for federated data privacy [39]. In fact, these privacy preserving techniques (*e.g.*, LDP) can be introduced to enhance the privacy preserving of transmitted models or gradients in our proposed method.

Here, we define a (ϵ, δ)-LDP requirement for *FedMigr*, where ϵ is the privacy budget. $\delta \in (0, 1)$ accounts for the probability that plain ϵ-DP is broken. In *FedMigr*, the local model w_i of client i will be clipped after model updating on the local dataset at epoch t:

$$w_i^t = w_i^t / \max(1, \frac{\|w_i^t\|}{C}) \tag{30}$$

where C is a clipping threshold for bounding w_i. Then, the noise ζ_i^t will be added to the parameters according to the Gaussian Machanism (GM) [41] before model migration between clients or model aggregation at the server:

$$\widetilde{w}_i^t = w_i^t + \zeta_i^t \tag{31}$$

where $\zeta_i^t \sim \mathcal{N}(0, \chi^2)$ and χ is a preset parameter of gaussian mechanism. Generally, the parameter χ increases with the decerasing privacy budget, and the performance of privacy preserving can be significantly improved [39].

In order to evaluate the privacy implications of our proposed hod, we perform CNN over CIFAR10 dataset with different

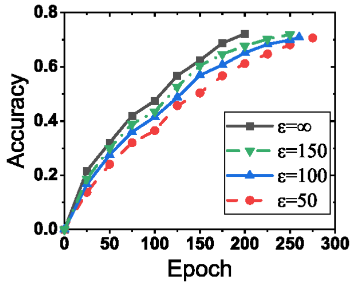

Fig. 4: The training performance of our proposed method with different privacy budgets.

privacy budgets ϵ. The model training will be performed without privacy budget if $\epsilon = \infty$. Fig.4 shows that the training accuracy is slightly degraded with the decreasing privacy budgets. For example, given 200 training epochs, the accuracy of *FedMigr*($\epsilon = \infty$) is about 72.4% while those of *FedMigr*($\epsilon = 150$) and *FedMigr*($\epsilon = 100$) are about 69.2% and 67.6%, respectively. However, the protection capability for model migration in *FedMigr* can be significantly improved. Thus, the (ϵ, δ)-LDP techniques can be well combined to enhance privacy preserving of transmitted models or gradients in our proposed method.

IV. 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* [13], as baselines for performance comparison. 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 picture on the entire dataset, so as to reduce weight divergence. However, model swapping at the server may still bring an enormous amount of traffic workload to the PS. 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 classificat 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 colour 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 224x224 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, for some experiments a subset of this dataset, the ImageNet-100 dataset is adopted. It features only 100 randomly drawn classes from the complete ImageNet dataset.

Models: Three deep learning models with different structures and parameters are implemented based on the datasets: 1) A Convolutional Neural Network (CNN) has the same structure as that in [6], *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 by 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].

Clients and Servers: As suggested in [18], in order to efficiently simulate the training processing in FL 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 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 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)

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

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-uniform data on clients [5]. 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 fraction α of the selected clients is set as 1, *i.e.*, all clients are selected for model training in a global iteration. Besides, local iteration τ is also set as 1, which indicates that each local model is trained once over the local data in a single *Local Updating* process. 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.

2) Simulation Results: We perform five groups of simulations to verify the efficiency of our proposed framework, and the simulation results are as follows:

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

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-art performances 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 the 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 intelligent model migration strategy, *FedMigr* outperforms the other four baselines on all the 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. Therefore, *FedMigr* improves the performance (test accuracy) over the other four baselines by as much as 16.4%, 9.8%, 3.2% and 13% on these baselines. The testing results demonstrate the significant effectiveness of our *FedMigr* framework.

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

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 global communications and bandwidth consumption for federated training, leading to a slower convergence rate. In 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 contribute to further shortening the distribution divergence

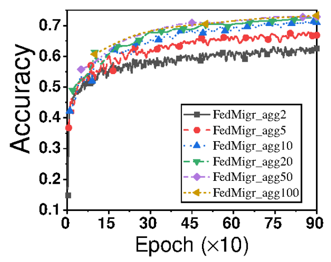

Fig. 5: Ratios of local migration vary with training epochs for different models.

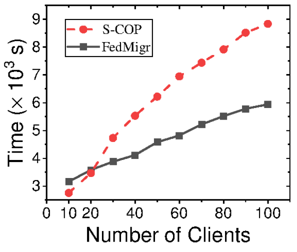

Fig. 6: Completion time vs. Number of clients under two different schemes.

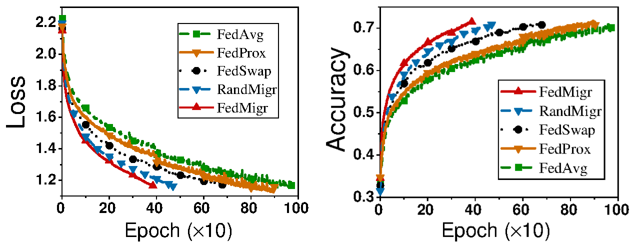

Fig. 7: Training performance of loss and accuracy with CNN trained over CIFAR10 in test-bed.

as well as the parameter divergence, 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.

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 be 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 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.*, 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.

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 and 15 Xavier NX[5]. 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. Each NX 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 PS which is responsible for the model aggregation and verifying the training performance of global model. We adopt a TX2 or NX as a

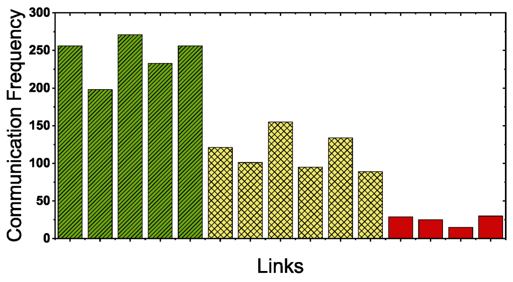

Fig. 8: The communication frequencies for 15 sampled C2C links. All sampled links are divided into three parts according to their link speeds: fast(green), moderate(yellow), slow(red).

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 PS 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.

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 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 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, well reduce the influence of non-IID issue through model

[5]https://www.nvidia.com/en-us/autonomous-machines/embedded-systems

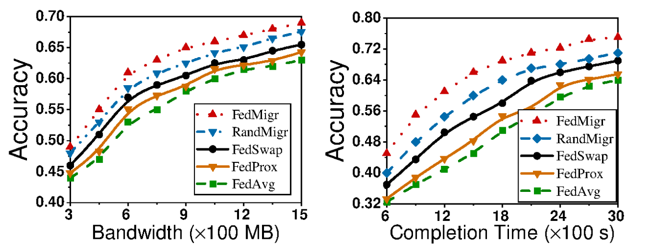

Fig. 9: Test accuracy with bandwidth and completion time constraint (CNN over CIFAR10) in test-bed.

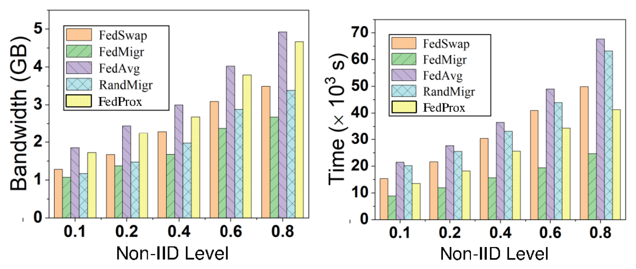

Fig. 11: Bandwidth consumption and completion time of C10-CNN with different Non-IID levels in test-bed.

migration, accelerating the process of model training. Fig. 7 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.

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. 9 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. 9, 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. Thus, *FedMigr* can improve the accuracy by about 5.1%, 7.5%, 10% and 12.3% compared with the four baselines.

Different Non-IID Levels: The last set of experiments t

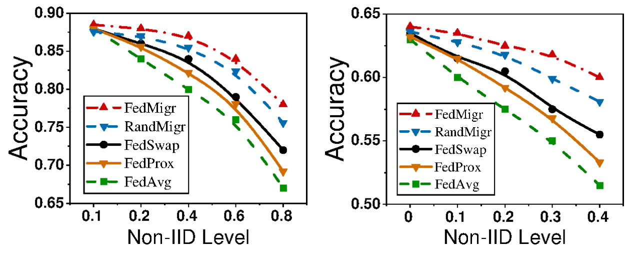

Fig. 10: Test accuracy of C10-CNN and C100-CNN with different Non-IID Levels in test-bed.

the training performance (CNN trained over CIFAR10 and CIFAR100) of five schemes under different non-IID levels. As shown in Fig. 10, 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 the four baselines, respectively.

Besides, we observe the bandwidth consumption and completion time of federated training (CNN trained over CIFAR10) with different non-IID levels. Fig. 11 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.

V. 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.

Acknowledgement

The corresponding authors of this paper are Yang Xu and Hongli Xu. This research is supported in part by the National Key R&D Program of China (No. 2021YFB3301501), and the National Science Foundation of China (NSFC) under Grants 02391, 62132019 and 61936015.

REFERENCES

[1] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

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

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

[4] R. K. Bakshi, N. Kaur, R. Kaur, and G. Kaur, "Opinion mining and sentiment analysis," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2016, pp. 452–455.

[5] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[6] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[7] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

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

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

[10] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 629–647.

[11] L. Huang, Y. Yin, Z. Fu, S. Zhang, H. Deng, and D. Liu, "Loadaboost: Loss-based adaboost federated machine learning on medical data," *arXiv preprint arXiv:1811.12629*, 2018.

[12] 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 Communications Surveys Tutorials*, vol. 22, no. 1, pp. 38–67, 2020.

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

[14] M. Duan, D. Liu, X. Chen, Y. Tan, J. Ren, L. Qiao, and L. Liang, "Astraea: Self-balancing federated learning for improving classification accuracy of mobile deep learning applications," in *2019 IEEE 37th international conference on computer design (ICCD)*. IEEE, 2019, pp. 246–254.

[15] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, 2014, pp. 583–598.

[16] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations ICLR*, 2016.

[18] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1698–1707.

[19] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 63–71.

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

[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 of Things Journal*, vol. 7, no. 10, pp. 9266–9277, 2020.

[22] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[23] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on Machine Learning*. Omnipress, 2011, pp. 265–272.

[24] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[25] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 37–48.

[26] I. Sabek and M. F. Mokbel, "Machine learning meets big spatial data," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1782–1785.

[27] J. Liu, H. Xu, G. Zhao, C. Qian, X. Fan, and L. Huang, "Incremental server deployment for scalable nfv-enabled networks," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2361–2370.

[28] A. Rakhlin, O. Shamir, and K. Sridharan, "Making gradient descent optimal for strongly convex stochastic optimization," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, 2012, pp. 1571–1578.

[29] Y. Wang, Y. Tong, D. Shi, and K. Xu, "An efficient approach for cross-silo federated learning to rank," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 1128–1139.

[30] 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 *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1468–1476.

[31] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE Transactions on Computers*, vol. 69, no. 6, pp. 883–893, 2020.

[32] 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, 2017.

[33] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

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

[35] M. Tokic and G. Palm, "Value-difference based exploration: adaptive control between epsilon-greedy and softmax," in *Annual conference on artificial intelligence*. Springer, 2011, pp. 335–346.

[36] J. Devriendt, A. Gleixner, and J. Nordström, "Learn to relax: Integrating 0-1 integer linear programming with pseudo-boolean conflict-driven search," *Constraints*, pp. 1–30, 2021.

[37] M. Grant and S. Boyd, "Cvx: Matlab software for disciplined convex programming, version 2.1," 2014.

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

[39] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 1054–1067.

[40] S. Paul and S. Mishra, "Ara: Aggregated rappor and analysis for centralized differential privacy," *SN Computer Science*, vol. 1, no. 1, pp. 1–10, 2020.

[41] F. Liu, "Generalized gaussian mechanism for differential privacy," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 4, pp. 747–756, 2018.

[42] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[43] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[44] X. Zhang, Z. Li, C. Change Loy, and D. Lin, "Polynet: A pursuit of structural diversity in very deep networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 718–726.

[45] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," *arXiv preprint arXiv:1811.04017*, 2018.

[46] J. Wang, A. K. Sahu, Z. Yang, G. Joshi, and S. Kar, "Matcha: Speeding up decentralized sgd via matching decomposition sampling," in *2019 Sixth Indian Control Conference (ICC)*. IEEE, 2019, pp. 299–300.