Adaptive Asynchronous Federated Learning in Resource-Constrained Edge Computing

Jianchun Liu, *Student Member, IEEE*, Hongli Xu, *Member, IEEE*, Lun Wang, Yang Xu, Chen Qian, *Senior Member, IEEE*, Jinyang Huang, *Student Member, IEEE*, and He Huang, *Member, IEEE*

Abstract—Federated learning (FL) has been widely adopted to train machine learning models over massive data in edge computing. However, machine learning faces critical challenges, e.g., data imbalance, edge dynamics, and resource constraints, in edge computing. The existing FL solutions cannot well cope with data imbalance or edge dynamics, and may cause high resource cost. In this paper, we propose an adaptive asynchronous federated learning (AAFL) mechanism. To deal with edge dynamics, a certain fraction α of all local updates will be aggregated by their arrival order at the parameter server in each epoch. Moreover, the system can intelligently vary the number of local updated models for global model aggregation in different epochs with network situations. We then propose experience-driven algorithms based on deep reinforcement learning (DRL) to adaptively determine the optimal value of α in each epoch for two cases of AAFL, single learning task and multiple learning tasks, so as to achieve less completion time of training under resource constraints. Extensive experiments on the classical models and datasets show high effectiveness of the proposed algorithms. Specifically, AAFL can reduce the completion time by about 70 percent and improve the learning accuracy by about 28 percent under resource constraints, compared with the state-of-the-art solutions.

Index Terms—Edge computing, asynchronous federated learning, adaptive, resource constraint

1 Introduction

With the development of Internet of Things, many smart devices, e.g., mobile phones, and wearable devices, are generating a massive amount of data each day [1], [2], [3]. Due to the growing storage and computational power on these devices, it is increasingly attractive to store data locally and push more computation function to the network edge, which is called *edge computing* [4]. It motivates the application of federated learning (FL), which enables distributed machine learning at the network edge [5], [6], [7].

As shown in Fig. 1, a federated learning system is usually composed of one or more parameter servers and a large number of workers (e.g., edge nodes), following the typical parameter server architecture [8]. Each parameter server (PS) maintains a partition of the globally shared parameters. Each worker is responsible for computing local statistics such as gradients by training the local data, and communicates only with the parameter server. Specifically, the workers will send the local updated models to the parameter server, and receive the global updated model from the parameter server. Since the workers expose not their training data but the trained model to the parameter server, federated learning can efficiently protect users' privacy [9].

To implement highly efficient FL in edge computing, we should take into account the following constraints and factors from practical applications, e.g., mobile keyboard prediction [9], vehicle-to-vehicle (V2V) communication [10]. 1) *Data Imbalance*. The amount of data on the edge node(s) varies significantly with time and space. For example, due to device mobility, e.g., vehicles, each edge node will process data from varied numbers of devices at different times. The authors in [11] have shown that the amount of data on an edge node may vary from about 500MB to 50GB in a period of one hour. 2) *Edge Dynamics*. Since edge nodes are usually deployed outdoors, some nodes may fail to work occasionally because of system crash, dead battery, or network disconnection [12]. 3) *Resource Constraints*. Last but not least, edge nodes may frequently send and receive the updated models, which requires an enormous resource cost (e.g., network bandwidth) [13]. However, the bandwidth between edge nodes and remote parameter servers is constrained [14]. For example, the size of parameters in the AlexNet model is about 60MB [15]. Given the bandwidth constraint of 1GB, the network is easily

- *Jianchun Liu is with the School of Data Science, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with the Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu 215123, China. E-mail: jsen617@mail.ustc.edu.cn.*
- *Hongli Xu, Lun Wang, and Yang Xu are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with the Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu 215123, China. E-mail: {xuhongli, xuyangcs}@ustc.edu.cn, wanglun0@mail.ustc.edu.cn.*
- *Chen Qian is with the Department of Computer Science and Engineering, Jack Baskin School of Engineering, University of California Santa Cruz, Santa Cruz, CA 95064 USA. E-mail: cqian12@ucsc.edu.*
- *Jinyang Huang is with the CAS Key Laboratory of Electromagnetic Space Information, University of Science and Technology of China, Hefei 230026, China. E-mail: huangjy@mail.ustc.edu.cn.*
- *He Huang is with the School of Computer Science and Technology, Soochow University, Suzhou 215006, China. E-mail: huangh@suda.edu.cn.*

Manuscript received 5 Dec. 2020; revised 4 Mar. 2021; accepted 8 July 2021. Date of publication 14 July 2021; date of current version 6 Jan. 2023. (Corresponding author: Hongli Xu and Yang Xu.) Digital Object Identifier no. 10.1109/TMC.2021.3096846

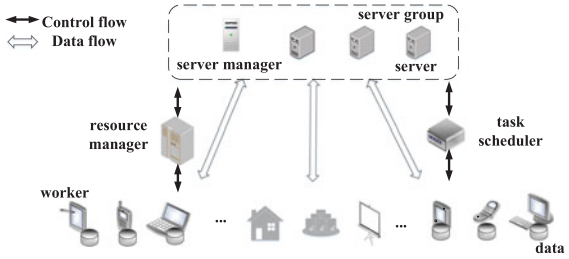


Fig. 1. Illustration of a typical parameter server architecture.

congested because of the frequent transmission of local and global models.

There are two main ways for federated learning in edge computing, including the synchronous scheme [16], [17], [18] and the asynchronous scheme [19], [20], [21], [22]. Under both schemes, on receiving the trained local models from a *fixed* number of edge nodes, the server will aggregate the models and send the updated global model to all the edge nodes. However, these solutions can not better conquer the aforementioned challenges.

- For the synchronous scheme, the parameter server will aggregate the local updated models from *all* or a specified set of edge nodes. Wang *et al*. [16] presented a control algorithm that aggregated the global model after receiving the trained local models from all the edge nodes in each epoch. Wang and Niu *et al*. [18] proposed a synchronous scheme, called FAVOR, aiming to improve the performance of training through intelligent edge nodes selection. However, due to data imbalance and edge dynamics caused by heterogeneous node capacities and network connections, the training time on different edge nodes (even on a set of selected edge nodes) may be varied significantly, e.g., from about 5 mins to 2 hours [23]. Due to *synchronization barrier* [24], the completion time of each epoch mainly depends on the maximum training time among these edge nodes, which will lead to long completion time. Moreover, it is difficult to determine the optimal number of local updated models for global aggregation in each epoch to improve the training speed, which increases the difficulty of the network management and system configuration.
- The asynchronous scheme is proposed for FL. In the previous solutions, a local updated model from an arbitrary edge node is gathered for global aggregation, which helps to conquer the edge dynamics [19], [20], [21]. The advantage of this solution is simple and with low management/configuration cost. However, there remain other challenges. There is only one local updated model involved in the global model aggregation in each epoch. Thus, more number of training epochs, as well as more training time are required to achieve similar training performance (e.g., loss and accuracy) of the synchronous scheme. Besides, the frequency of communication between the server and the workers is greatly increased, which will lead to massive bandwidth consumption. However, most of the existing solutions ignore the impact of limited network resources on training performance.

To accommodate data imbalance, edge dynamics and resource constraints, we propose an adaptive asynchronous federated learning (AAFL) mechanism for resource-constrained edge computing. Different from the existing schemes [16], [18] which assign the specified participating workers for model training, our proposed scheme adopts the model updates from one or several arbitrary workers. Specifically, the server will aggregate the local updated models only from a fraction of edge nodes by their arrival order in each epoch, and adaptively determine the optimal fraction values in different epochs according to real-time system situations (e.g., network resource and state). Thus, AAFL can effectively cope with the synchronization barrier problem, avoiding long waiting time for model aggregation. The main contributions of this paper are as follows:

- We design an adaptive asynchronous federated learning (AAFL) mechanism for edge computing, and formally prove the convergence of AAFL.
- We then propose experience-driven algorithms based on deep reinforcement learning (DRL) to adaptively determine the optimal fraction value α at each epoch for two cases of AAFL, single learning task and multiple learning tasks, so as to achieve less training time and bandwidth resource usage with low network management complexity.
- Extensive experiments on the typical models and datasets show the high effectiveness of the proposed algorithms. Specifically, AAFL can reduce the training time by about 70 percent and improve training accuracy by about 28 percent under resource constraints, compared with the state-of-the-art solutions.

2 Preliminaries and Problem Formulation

2.1 Federated Learning (FL)

2.1.1 The Goal of FL

Federated learning provides a decentralized computation strategy to train machine learning models [7]. Edge nodes, referred to as workers, generate large volumes of personal data for training. Instead of uploading data to the server for centralized training, workers process their local data and forward updated models to the parameter server [8], which maintains globally shared model parameters. On receiving local updated models from workers, the parameter server will perform the model aggregation using different algorithms, e.g., FEDAVG [17]. For ease of description, some key notations are listed in Table 1.

In edge computing, the sample data generated on the edge nodes may be inter-dependent and not obey the independent identical distribution (IID). For the classification problem, each worker has all labels (e.g., 10 classes) in IID setting, while there are only part of labels (e.g., 5 classes) on any worker in non-IID setting. Thus distributed machine learning (DML) cannot handle these data efficiently. On the contrary, FL can handle non-IID training data that are massively distributed on the workers. For convenience, given a training dataset Γ with $\mathcal{N}$ data points, $\Gamma = \{x_i, y_i\}_{i=1}^{\mathcal{N}}$, the parameters of the machine learning model $w \in \mathcal{R}^d$ are learned by minimizing a loss function $f(x_i, y_i; w)$ (or written as $f_i(w)$ in short) on Γ

TABLE 1
Key Notations

Symbol	Semantics
V	a set of edge nodes
Γ_i	the local dataset on the edge node v_i
D	the number of iterations in a local update
T	the total number of training epochs until the training terminates
$\mathbb{T}$	the vector's transposition
$\mathcal{K}$	the number of resource categories
g_k	the consumption of resource k for local updates on an edge node
b_k	the consumption of resource k for communication between a server and a worker
B_k	the total budget for each category of resource k
L	a set of learning tasks
α_j	a certain fraction of local updates that will be involved in the global aggregation of learning task j on the server
Φ	the set of α_j, with $j \in \{1, 2, \ldots, L\}$
$\widehat{w}^D$	the model parameter after D local updates
$F(w^T)$	the global loss function after T epochs
$F(w^*)$	the optimal value of loss function $F(w)$

$$\min_{w \in \mathcal{R}^d} \frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{N}} f_i(w), \tag{1}$$

where $\mathcal{R}^d$ denotes the d-dimension real-number space. In general, there is no closed-form solution for Eq. (1). To this end, learning starts from an initial model, and iteratively refines this model by processing the training data, to approach the solution. It terminates when a (near) optimal solution is found or the convergence is reached. For $\mathcal{N}$ input-output pairs $\{x_i, y_i\}_{i=1}^{\mathcal{N}}$, $x_i \in R^m$ is the input of the model (such as the pixels of an image) and $y_i \in R$ or $y_i \in \{-1, 1\}$ is the desired output of the model (such as the label of an image). Some typical examples of loss functions include

- Linear regression: $f_i(w) = \frac{1}{2}(x_i^{\mathbb{T}} w - y_i)^2, y_i \in R$
- Logistic regression: $f_i(w) = -\log\left(1 + \exp(-y_i x_i^{\mathbb{T}} w)\right)$, $y_i \in \{-1, 1\}$
- Support vector machines: $f_i(w) = \max\{0, 1 - y_i x_i^{\mathbb{T}} w\}$, $y_i \in \{-1, 1\}$

where $\mathbb{T}$ denotes the vector's transposition. More complicated non-convex problems arise in the context of neural networks, in which the network makes prediction through a non-convex function of the feature vector x_i, rather than via the linear-in-the-features mapping $x_i^{\mathbb{T}} w$.

2.2 Adaptive Asynchronous Federated Learning (AAFL)

Assume that there is a set of edge nodes $V = \{v_1, v_2, \ldots, v_n\}$, with $|V| = n > 2$ in edge computing. Each node trains the model over a local dataset $\Gamma_i, i \in \{1, 2, \ldots, n\}$, with its size $|\Gamma_i|$. For each node v_i, the loss function on the local dataset Γ_i is

$$F_i(w) = \frac{1}{|\Gamma_i|} \sum_{q_j \in \Gamma_i} f_j(w), \tag{2}$$

where q_j is a training sample in the local dataset Γ_i.

Algorithm 1. Adaptive Asynchronous Federated Learning

1: Initialize the model parameters w, fraction α
2: **for each** epoch $t \in \{1, 2, \ldots, T\}$ **do**
3: **Processing at the Parameter Server**
4: **if** the resource constraints are satisfied or the convergence threshold is not reached **then**
5: **for** each global update in the epoch t **do**
6: **while** No. of received local updates $< \alpha_t \cdot n$ **do**
7: Waiting for local updated models from workers
8: Perform the global update
9: Compute global loss function by Eq. (3)
10: Distribute the updated model to all the workers
11: Update the parameter α_{t+1}
12: Update the resource budgets
13: **Processing at the Edge Node** v_i
14: Receive the global model from the parameter server
15: **for** each local update $d \in \{1, 2, \ldots, D\}$ **do**
16: Update the local model w by stochastic gradient descent (SGD) [25]
17: Push local update to the server
18: Return the final model and loss function

In this section, we propose the adaptive asynchronous federated learning (AAFL) mechanism, described in Algorithm 1. Workers (and parameter servers) perform the local (and global) model updates. Let variable T denotes the total number of training epochs until the training terminates. Assume that there are $D(\geq 1)$ local updates (i.e., iterations) on each worker before one global update. Let $\alpha_t \in \{\frac{1}{n}, \frac{2}{n}, \ldots, 1\}$ be a certain fraction of local updated models from all edge nodes for global model aggregation on the parameter server in each epoch $t \in \{1, \ldots, T\}$. After the parameter server receiving $\alpha_t \cdot n$ local updated models from arbitrary workers in epoch t, it updates the global model through model aggregation (Line 4-9). For simplicity, we assume one single parameter server, and the solution can be easily extended to multiple parameter servers. Then, the server distributes the global updated model to all the workers (Line 10) and updates the parameter α_{t+1} for the next global update according to the control algorithm, which will be introduced in detail in Section 3 (Line 11). Besides, the resource budgets are also updated (Line 12) for the next training epoch at the edge nodes (Line 14-17). The global loss function $F(w^T)$ of the model after T epochs is

$$F(w^T) = \frac{\sum_{i=1}^{n} \sum_{q_j \in \Gamma_i} \beta_i^T f_j(w^T)}{\sum_{i=1}^{n} \beta_i^T |\Gamma_i|} = \frac{\sum_{i=1}^{n} \beta_i^T |\Gamma_i| F_i(w^T)}{\sum_{i=1}^{n} \beta_i^T |\Gamma_i|}, \tag{3}$$

where β_i^t is a binary variable to indicate whether the local update of worker v_i is involved in the global update or not in epoch t. Thus, it follows $\sum_{i=1}^{n} \beta_i^t \geq \alpha_t \cdot n, \forall t \in \{1, \ldots, T\}$. The whole training process will continue until the resource budgets are used up or the global convergence is achieved, i.e., $F(w^T) - F(w^*) < \varepsilon$, where ε is an arbitrarily small positive value, and w^* is the optimal solution for $F(w)$.

For a better explanation of AAFL, we illustrate the synchronous scheme (left plot) and AAFL (right plot) under the same time period in Fig. 2. Assume that there is one parameter server and four workers in the edge computing system. In the synchronous scheme, only when the server receives all local updated models from four workers, it will perform

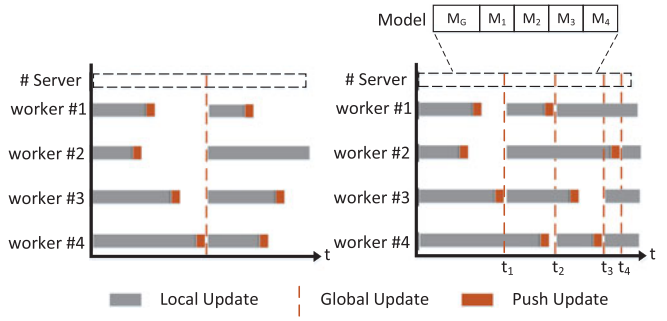


Fig. 2. Illustration of synchronous scheme and AAFL under the same time period. *Left plot: synchronous scheme; right plot: AAFL.* If we set $\alpha_1 = \frac{3}{4}$, on receiving three local updated models from workers #1, #2, #3 in turn, the PS performs the global update.

model aggregation to derive the updated global model. Thus, only one global update is performed during the training. In AAFL, the global update will be performed after the PS receives $4 \cdot \alpha$ local updated models from *an arbitrary combination* of edge nodes. Moreover, the value of α changes with training epochs according to the environment (e.g., loss value and resource usage). At the beginning of training, the global model needs more local updated models to achieve convergence quickly, with more resource consumption. The value of α_t will be determined in a real-time manner (e.g., $\alpha_1 = \frac{3}{4}, \alpha_2 = \alpha_3 = \frac{1}{2}$). As shown in the right plot of Fig. 2, as $\alpha_1 = \frac{3}{4}$, on receiving three local updated models from workers #1, #2, and #3 in turn, the PS performs the global update. As $\alpha_2 = \frac{1}{2}$, the PS aggregates two local models from workers #4 and #1. By this figure, AAFL will converge faster than the synchronous scheme with more global updates, making efficient resource usage.

Note that AAFL may suffer from another problem, *delayed update*. For example, when worker #4 sends its local updated model to the server at the first time, the server has aggregated the local updated models from workers #1, #2, and #3 at time point t_1. We adopt the *delay compensation* mechanism [26] to alleviate this problem. In Fig. 2, we use M_G to denote the current global model and $M_i, \forall i \in \{1, \ldots, 4\}$, to denote the latest local updated model from worker i. These models will be recorded on the server to perform delay compensation for outdated models. For example, considering a time point t between t_3 and t_4, worker #2 has sent the local updated model to the server only once, while the server has performed three global model aggregations. Then, the *staleness* of worker #2 is the gap between the number of global updates and the number of local model updates, e.g., here $3 - 1 = 2$. After the server receives the local model from worker #2 twice, the model M_2 will be updated with decay coefficient ς, with $0 < \varsigma < 1$, i.e., $M_2 = \varsigma^x \cdot M_2 + (1 - \varsigma^x) \cdot M_G$, where x denotes the staleness of worker #2. By this way, the impact of the outdated models can be alleviated.

Besides, we give an example to show how our proposed solution solves the problem of edge dynamics. As shown in Fig. 3, the parameter server cannot receive local updates from worker #4 because of system crash or network disconnection. Thus, there is no global model aggregation in the synchronous scheme (left plot). The parameter server perform global update after receiving local updates from worker #1, #2 and #3 ($\alpha = \frac{3}{4}$) in the first epoch, and worker #1, #3 ($\alpha = \frac{1}{2}$) in the second epoch. Even though we haven't received the local model update from the worker #4, AAFL (right plot) still has three global updates. Thus, our proposed solution can effectively solve the edge dynamics problem.

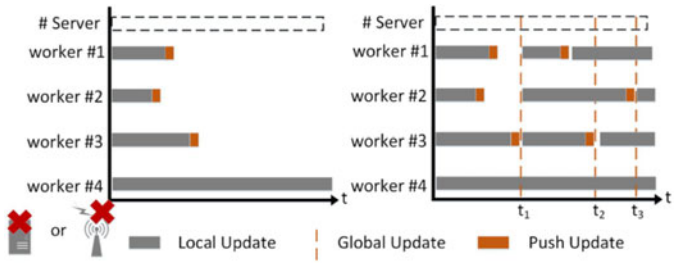


Fig. 3. Illustration of how AAFL handles edge dynamics. *Left plot: synchronous scheme; right plot: AAFL.*

2.3 Convergence Analysis

To analyze the feasibility of the proposed model training mechanism, we prove that AAFL can achieve a constant convergence bound. For convergence analysis, we make the following three assumptions according to [20].

Assumption 1 (Smoothness). *Let $L > 0$. Assume that the loss function f is L-smooth w.r.t. the model parameter if $\forall x, y$*

$$f(y) - f(x) \leq \langle \nabla f(x), y - x \rangle + \frac{L}{2} \|y - x\|^2. \tag{4}$$

Assumption 2 (Strong Convexity). *Let $\vartheta \geq 0$. Assume that the loss function f is ϑ-strongly convex if $\forall x, y$*

$$f(y) - f(x) \geq \langle \nabla f(x), y - x \rangle + \frac{\vartheta}{2} \|y - x\|^2. \tag{5}$$

This assumption can be satisfied for models with convex function, e.g., linear regression and SVM.

Assumption 3 (Existence of Global Optimization). *Assume that there exists at least one solution, denoted as w^*, to achieve the global minimum of the loss function $f(w)$.*

We analyze the convergence bound of AAFL based on the above assumptions. AAFL will perform T epochs until the convergence is achieved or the resource budget is used up. Moreover, D local updates will be performed on an edge node in each epoch. In the following, we first derive the convergence bound of each local update d (Theorem 1). Then, we prove the convergence of each epoch t and give the convergence bound after T epochs (Theorem 2). Due to the space limitations, we omit the detailed proof here.

Theorem 1. *Assume that the global loss function F is L-smooth and ϑ-strongly convex, and each worker executes D local updates before pushing the updated models to the parameter server. For $\forall w \in \mathcal{R}^m, q \in \Gamma_i, i \in \{1, \ldots, n\}$, it follows $\mathbb{E}\|\nabla f(w; q) - \nabla F(w)\|^2 \leq \mathcal{Z}$, where $\mathcal{Z}$ is a positive number. Let η denotes the learning rate which is used in SGD. When the following three conditions are satisfied:*

- $\eta < \frac{1}{L}$
- $\eta\vartheta > 1 - \sqrt[D]{\frac{1}{4p}}$

- $F(w^0) - F(w^*) > \frac{D\eta\mathcal{Z}}{4(1-\eta\vartheta)^D}$

we have $2\alpha_t n(1-\eta\vartheta)^D \in (0,1)$ *and*

$$\mathbb{E}[F(\widehat{w}^D) - F(w^*)] \leq (1-\eta\vartheta)^D[F(w^0) - F(w^*)] + \frac{D\eta\mathcal{Z}}{2}, \tag{6}$$

where $\widehat{w}^D$ *denotes the model parameter after* D *local updates and* w^0 *is the initial model parameter.*

Proof. Without loss of generality, we first consider the convergence analysis of the D local updates. For each loacl update $d \in \{1,\ldots,D\}$, using the assumptions of smoothness and strong convexity, we have

$$\begin{aligned}
&\mathbb{E}[F(\widehat{w}^d) - F(w^*)] \\
&\leq F(\widehat{w}^{d-1}) - F(w^*) - \eta\mathbb{E}[\langle \nabla F(\widehat{w}^{d-1}), \nabla f(\widehat{w}^{d-1}; q_d)\rangle] \\
&\quad + \frac{L\eta^2}{2}\mathbb{E}[\|\nabla f(\widehat{w}^{d-1}; q_d)\|^2] \\
&\leq F(\widehat{w}^{d-1}) - F(w^*) - \frac{\eta}{2}\|\nabla F(\widehat{w}^{d-1})\|^2 \\
&\quad + \frac{\eta}{2}\mathbb{E}[\|\nabla F(\widehat{w}^{d-1}) - \nabla f(\widehat{w}^{d-1}; q_d)\|^2] \\
&\leq F(\widehat{w}^{d-1}) - F(w^*) - \frac{\eta}{2}\|\nabla F(\widehat{w}^{d-1})\|^2 + \frac{\eta\mathcal{Z}}{2} \\
&\leq F(\widehat{w}^{d-1}) - F(w^*) - \eta\vartheta[F(\widehat{w}^{d-1}) - F(w^*)] + \frac{\eta\mathcal{Z}}{2} \\
&\leq (1-\eta\vartheta)[F(\widehat{w}^{d-1}) - F(w^*)] + \frac{\eta\mathcal{Z}}{2}.
\end{aligned} \tag{7}$$

We have derived the convergence result of each loacl update $d \in D$. By telescoping and taking total expectation, after D local updates, we have

$$\begin{aligned}
&\mathbb{E}[F(\widehat{w}^D) - F(w^*)] \\
&\leq (1-\eta\vartheta)[F(\widehat{w}^{D-1}) - F(w^*)] + \frac{\eta\mathcal{Z}}{2} \\
&\leq (1-\eta\vartheta)\left[(1-\eta\vartheta)[F(\widehat{w}^{D-2}) - F(w^*)] + \frac{\eta\mathcal{Z}}{2}\right] + \frac{\eta\mathcal{Z}}{2} \\
&\qquad\qquad \ldots\ldots \qquad \text{(Telescoping by Eq. (7))} \\
&\leq (1-\eta\vartheta)^D[F(w^0) - F(w^*)] + \frac{\eta\mathcal{Z}}{2}\sum_{d=1}^{D}(1-\eta\vartheta)^{d-1} \\
&\leq (1-\eta\vartheta)^D[F(w^0) - F(w^*)] + \frac{\eta\mathcal{Z}}{2}\frac{D\eta\vartheta}{1-(1-\eta\vartheta)} \\
&\leq (1-\eta\vartheta)^D[F(w^0) - F(w^*)] + \frac{D\eta\mathcal{Z}}{2}.
\end{aligned} \tag{8}$$

□

On the parameter server side, it will perform global model aggregation with $\alpha_t \cdot n$ updated models after D local updates at the epoch t, and we have

$$w^t = \frac{1}{\alpha_t n}\sum_{i=1}^{\alpha_t n}\widehat{w}_i^D, \tag{9}$$

where $\widehat{w}_i^D$ denotes the model in the worker node i after D local updates.

Theorem 2. *Let* $\alpha_m = \max_{t\in\{1,\ldots,T\}}\alpha_t$. *Based on Theorem 1, the convergence bound of AAFL after* T *epochs is*

$$\mathbb{E}[F(w^T) - F(w^*)] \leq \tau[F(w^0) - F(w^*)] + (1-\tau)\frac{D\eta\mathcal{Z}}{4\varphi}, \tag{10}$$

where $\varphi = (1-\eta\vartheta)^D$, *and* $\tau = (2\alpha_m n\varphi)^T$.

Proof. Combing with Eq. (9), for each global update $t \in T$, we have

$$\begin{aligned}
&\mathbb{E}[F(w^t) - F(w^*)] \\
&\leq \frac{1}{\alpha_t n}\sum_{i=1}^{\alpha_t n}[F(\widehat{w}_i^D) - F(w^*)] \\
&\leq \frac{1}{\alpha_t n}\sum_{i=1}^{\alpha_t n}\left[(1-\eta\vartheta)^D(F(w_i^0) - F(w^*)) + \frac{D\eta\mathcal{Z}}{2}\right] \\
&\leq \alpha_t n\left[(1-\eta\vartheta)^D(F(w^0) - F(w^*)) + \frac{D\eta\mathcal{Z}}{2}\right] \\
&\leq \alpha_t n\Bigg[(1-\eta\vartheta)^D(F(w^0) - F(w^{t-1}) + F(w^{t-1}) + F(w^*)) \\
&\quad + \frac{D\eta\mathcal{Z}}{2}\Bigg] \\
&\leq \alpha_t n(1-\eta\vartheta)^D(F(w^0) - F(w^{t-1})) \\
&\quad + \alpha_t n(1-\eta\vartheta)^D(F(w^{t-1}) - F(w^*)) + \frac{\alpha_t n D\eta\mathcal{Z}}{2} \\
&\leq \alpha_t n(1-\eta\vartheta)^D(F(w^{t-1}) - F(w^*)) \\
&\quad + \alpha_t n(1-\eta\vartheta)^D(F(w^{t-1}) - F(w^*)) + \alpha_t\frac{nD\eta\mathcal{Z}}{2} \\
&\leq (2\alpha_t n(1-\eta\vartheta)^D)[F(w^{t-1}) - F(w^*)] + \alpha_t\frac{nD\eta\mathcal{Z}}{2}.
\end{aligned} \tag{11}$$

Then, the convergence bound after T training epochs can be derived as follows:

$$\begin{aligned}
&\mathbb{E}[F(w^T) - F(w^*)] \\
&\leq (2\alpha_T n(1-\eta\vartheta)^D)[F(w^{T-1}) - F(w^*)] + \alpha_T\frac{nD\eta\mathcal{Z}}{2} \\
&\qquad\qquad \ldots\ldots \qquad \text{(Telescoping by Eq. (11))} \\
&\leq (2\alpha_m n(1-\eta\vartheta)^D)^T[F(w^0) - F(w^*)] \\
&\quad + \frac{(1-(2\alpha_m n(1-\eta\vartheta)^D)^T)\alpha_m nD\eta\mathcal{Z}}{2(1-(2\alpha_m n(1-\eta\vartheta)^D))} \\
&(\triangleright\ \eta\vartheta > 1 - \sqrt[D]{1/4n}, \\
&\quad 1-(2\alpha_m n(1-\eta\vartheta)^D) > 2\alpha_m n(1-\eta\vartheta)^D) \\
&\leq (2\alpha_m n(1-\eta\vartheta)^D)^T[F(w^0) - F(w^*)] \\
&\quad + \frac{(1-(2\alpha_m n(1-\eta\vartheta)^D)^T)\alpha_m nD\eta\mathcal{Z}}{4\alpha_m n(1-\eta\vartheta)^D} \\
&\leq (2\alpha_m n(1-\eta\vartheta)^D)^T[F(w^0) - F(w^*)] \\
&\quad + \frac{D\eta\mathcal{Z}}{4(1-\eta\vartheta)^D}(1-(2\alpha_m n(1-\eta\vartheta)^D)^T),
\end{aligned} \tag{12}$$

where $\alpha_m = \max_{t\in\{1,\ldots,T\}}\alpha_t$. To simplify expression, let $\varphi = (1-\eta\vartheta)^D$, we write it as follows:

$$\mathbb{E}[F(w^T) - F(w^*)] \leq \tau[F(w^0) - F(w^*)] + (1-\tau)\frac{D\eta\mathcal{Z}}{4\varphi}, \tag{13}$$

where $\tau = (2\alpha_m n\varphi)^T$. □

We note that the convergence bound (i.e., optimality gap), $F(w^T) - F(w^*)$, is related to the value of $\alpha_t, \forall t \in \{1, \ldots, T\}$ and the number of epochs T by Eq. (13). Furthermore, the optimality gap becomes smaller when both α_t and T become larger, which may violate the resource constraints. Thus, it is a challenge to determine the optimal values of α_t and T under resource constraints.

2.4 Problem Formulation

We define the adaptive asynchronous federated learning with resource constraints (AAFL-RC) problem as below. Given a federated learning task, we will determine the values of α_t, where $t \in \{1, 2, \ldots, T\}$, and the number of epochs T so as to minimize its training time with resource constraints and convergence guarantee (i.e., accuracy constraint). The local updates on the workers and global updates on the parameter server need to consume several categories of resources (e.g., network bandwidth, CPU, *etc.*). Assume that there are totally $\mathcal{K}$ different categories of resources. Let g_k denote the consumption of resource $k \in \{1, 2, \ldots, \mathcal{K}\}$ for local updates on an edge node. Meanwhile, b_k denotes the consumption of resource k for model exchanging once between an edge node and a parameter server. Since the computing power on the server is sufficient compared with workers, the consumption of computing resource for global model aggregation can be ignored [27]. Thus, for each category of resource k, the total resource consumption of local updates and global updates at all nodes after T epochs is $T \cdot n \cdot g_k$ and $\sum_{t=1}^{T}(\alpha_t + 1) \cdot n \cdot b_k$, respectively. Some entities (e.g., resource manager and task scheduler) in the proposed architecture are deployed on the edge nodes or on the cloud servers (i.e., parameter server) [7]. If these two modules are deployed on an edge node, it may bring significant processing overhead on this edge node with limited computing capacity between the edge node and the parameter server. Thus, we deploy these two modules on the cloud server by default in this paper. Let B_k denotes the total budget for each category of resource k. Accordingly, we formulate the AAFL-RC problem as follows:

$$\min \sum_{t=1}^{T} H_t$$
$$s.t. \begin{cases} F(w^T) \leq \mathcal{F} & \\ \sum_{t=1}^{T} n \cdot [g_k + (\alpha_t + 1) \cdot b_k] \leq B_k, & \forall k \\ \sum_{i=1}^{n} \beta_i^t \geq \alpha_t \cdot n, & \forall t \\ \beta_i^t \in \{0, 1\}, & \forall i, \forall t \\ \alpha_t \in \{\frac{1}{n}, \frac{2}{n}, \ldots, 1\}, & \forall t \end{cases} \quad (14)$$

where H_t denotes the completion time of epoch t, i.e., the time that $\alpha_t \cdot n$ workers complete their local training after the last epoch. The first inequality expresses the convergence requirement, where $\mathcal{F}$ is the convergence threshold of the loss value of the learning task after T training epochs. The second set of inequalities expresses the constraints of resource $\forall k \in \{1, \ldots, K\}$ during totally T training epochs. The third set of inequalities tells that the parameter server will receive at least $\alpha_t \cdot n$ local updated models from all edge nodes for model aggregation in each epoch, where $t \in \{1, \ldots, T\}$. The objective of the AAFL-RC problem is to minimize the completion time.

In order to solve the problem, we discuss the following two issues. On one hand, AAFL-RC is a sequential decision problem, which can hardly be solved by the existing dynamic programming (DP) algorithms, due to the absence of a deterministic mapping from the current status and operations in DP. Thus, we propose an experience-driven algorithm based on deep reinforcement learning (DRL) to solve Eq. (14). Note that the resource budget may have been used up, while the loss value has not reached the convergence threshold. In order to achieve model convergence in the shortest time under resource constraints, we adopt the reward with penalty mechanism, which will be introduced in details in Section 3.

On the other hand, by the second set of constraints in Eq. (14), we find that two parameters α_t and T are dependent. For simplicity, we consider how to determine the value of α_t at each epoch t for two cases, single learning task and multiple learning tasks, respectively. Then, the value of parameter T can be accordingly determined. To avoid the complex management and configuration, we adopt DRL based control algorithm to determine the value of α_t at each epoch t.

3 Algorithm Design for AAFL-RC

In this section, we first briefly introduce the agent deep reinforcement learning (DRL) technique based algorithm (Section 3.1). Then, we describe the training methodology of the algorithm for single learning task in detail (Section 3.2). Finally, we extend the algorithm for the general case with multiple learning tasks (Section 3.3).

3.1 The DRL Agent

We consider a simple case with only one single learning task in edge computing. To achieve training convergence with less completion time under resource constraints, we need to adaptively determine the value of $\alpha_t, t \in \{1, \ldots, T\}$. For ease of description, we just consider the bandwidth resource, which is the widely considered communication bottleneck in edge computing [14]. Our solution can also be applied to other resource constraints. We present the DRL technique based asynchronous federated learning (DAFL) algorithm.

We illustrate the architecture of the DRL system in Fig. 4. The standard reinforcement learning has an *agent* which interacts with an *environment* over a number of discrete training epochs. The agent mainly consists of three components: state, policy network, and action probability. In each training epoch t, the policy network in the agent receives a *state* s_{t-1} (e.g., completion time, loss function and resource usage) and outputs the probability 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 to maximize the expected return from each state s_t. We describe the state, action and reward of our DRL system in detail as follows.

State. We use a vector $s_t = (t, H_t, w^t, F_t, \triangle F_t, \mathcal{F}, b_t, \mathcal{G}_t)$ to denote the state of epoch t. Here t is the training epoch index. H_t is the completion time of the learning task at epoch t and reflects the training speed under the previous action a_{t-1}. w^t and F_t denote the model parameter and loss

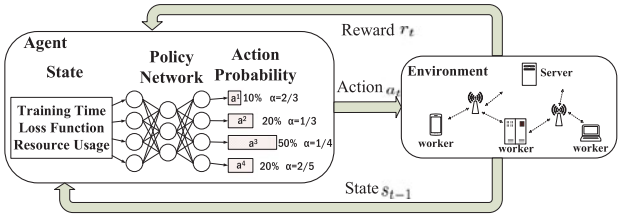


Fig. 4. The architecture of the DRL system. It mainly consists of two parts: the agent and environment. The agent includes the state, policy network and action probability. It receives the state and reward from the environment and returns the action for the next training.

function after epoch t, respectively. $\triangle F_t$ is the difference between current loss value and the target loss value $\mathcal{F}$, i.e., $\triangle F_t = \mathcal{F} - F_t$, where threshold $\mathcal{F}$ reflects the convergence degree of training. Besides, bandwidth consumption at each epoch t is represented as b_t. We use $\mathcal{G}_t$ to denote the remaining resource budget at the end of epoch t. With the progress of the learning task, more and more resources are consumed, and the remaining resource budget decreases.

Action. At each epoch t, a fraction, i.e., $\alpha_t \in \{\frac{1}{n}, \ldots, 1\}$, of local updated models, which are involved in the global model update, will be determined by the agent, called an action a_t. Given the current state, the DRL agent chooses an action based on a policy, expressed by a probability distribution $\pi(a|s)$ over the whole action space. We use neural network [28] to represent the policy π, where the adjustable parameters of the neural network are referred to as the policy parameter θ. Our policy can be represented as $\pi(a_t|s_t;\theta) \rightarrow [0,1]$, which is the probability of taking the action a_t at the state s_t. For example, as shown in Fig. 4, the agent derives the probability of different actions (e.g., $\pi(a_t = a^1|s_t;\theta) = 0.1$ and $\pi(a_t = a^3|s_t;\theta) = 0.5$) according to the current state.

Algorithm 2. DRL-Based Asynchronous FL (DAFL)

1: Initialize the set of agents ϕ, the global parameters of actor network θ and critic network ψ
2: **for each** epoch $t \in \{1, 2, \ldots, T\}$ **do**
3: **for each** agent in ϕ **do**
4: $d\theta \leftarrow 0, d\psi \leftarrow 0$
5: Pull the global parameters of actor θ' and critic ψ'
6: The agent interacts with the environment
7: Actor is updated towards the target
8: $\frac{\partial \log \pi(a_t|s_t;\theta')}{\partial \theta} A(s_t, a_t; \theta, \psi)$
9: Critic is updated to minimize mean square error (MSE) w.r.t target
10: $L_{\psi'} = (A(s_t, a_t; \theta, \psi))^2$
11: Compute gradients w.r.t. θ' and ψ'
12: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_t|s_t;\theta') A(s_t, a_t; \theta, \psi)$
13: $d\psi \leftarrow d\psi + \nabla L_{\psi'}$
14: Push the update to the global parameter θ and ψ
15: Select the optimal action, i.e., the optimal value of α'
16: Update $\alpha_{t+1} \leftarrow \alpha'$ in line 11 of AAFL

Reward. When an action a_t is applied at epoch t, the agent will receive a reward r_t from the environment. We define the reward r_t as the combination of the completion time, the difference of loss value and the resource usage at epoch t, i.e.,

$$r_t = -\Upsilon^{\frac{H_t - \overline{H}_{t-1}}{\overline{H}_{t-1}}} + \frac{\triangle F_t}{\mathcal{F}} - \frac{b_t}{\mathcal{G}_t}, \quad t < T, \tag{15}$$

where Υ is a positive constant so as to enusre that r_t decreases exponentially with the completion time H_t. $\overline{H}_t$ is the moving average time which can alleviate the impact of data jitter at epoch t and follows $\overline{H}_t = \varpi H_t + (1-\varpi)\overline{H}_{t-1}$, with $\varpi \in (0,1)$. The longer training time the epoch t takes or more resource the task consumes, the less reward the agent will receive. In contrast, the closer to the degree of convergence, i.e., the smaller value of $|\triangle F_t|$, the more reward the agent will obtain. At epoch T, the learning task will stop because we have $F(w^T) \leq \mathcal{F}$ or the resource budget is used up (i.e., $\mathcal{G}_T \leq 0$). Then, the reward in the final epoch T is defined as

$$r_T = \begin{cases} \mathcal{L}_T + \mathcal{C} & \text{if } F(w^T) \leq \mathcal{F} \text{ and } \mathcal{G}_T \geq 0, \\ \mathcal{L}_T - \mathcal{C} & \text{otherwise.} \end{cases} \tag{16}$$

where $\mathcal{L}_T = -\Upsilon^{\frac{H_T - \overline{H}_{T-1}}{\overline{H}_{T-1}}} + \frac{\triangle F_T}{\mathcal{F}} - \frac{b_T}{\mathcal{G}_T}$ and $\mathcal{C}$ is a positive real number. If the learning task stops with success, i.e., the convergence threshold $\mathcal{F}$ is met without overrunning the resource budget, the reward will be added by $\mathcal{C}$. Otherwise, if the learning task fails, i.e., it dose not achieve convergence under the resource budget, a negative value $\mathcal{C}$ will be added to the reward. The reward will be sent to the agent for the next decision. If the reward of the current action is larger compared with that of the other actions, the probability of this action being selected in the next epoch will increase. Otherwise, it will decrease.

3.2 Training Methodology of DAFL

In this section, we describe the techniques to train the DRL agent. The goal of DRL is to maximize the expected value of cumulative discounted reward. We train the agent using the cost-effective and time-efficient asynchronous advantage actor-critic (A3C) algorithm [29] in DAFL. A3C creates a master agent and multiple subagents, and performs in parallel and asynchronously. It can be run on a single machine with a standard multi-core CPU, rather than those with GPUs or a cluster [29].

In our proposed DAFL algorithm, A3C maintains a policy which has a softmax output $\pi(a_t|s_t;\theta)$ (the actor network) and an estimate of the value function $V(s_t;\psi)$ that will output the linear value (the critic network). Here θ is the policy parameter and ψ is the value function parameter. The value function is represented with a function approximator (e.g., neural network) and estimated as

$$V(s_t;\psi) \approx \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-t} r_T | s_t]. \tag{17}$$

The policy and the value function perform updates after every t_{max} actions or when a terminal state (e.g., the convergence threshold is achieved or the resource budget is used up) is reached, where t_{max} is the given number of maximum global iterations. The actor network and the critic network share the previous part of network parameters except for the last output layer. The update can be seen as $\nabla_\theta \log \pi(a_t|s_t;\theta) A(s_t, a_t; \theta, \psi)$, where $A(s_t, a_t; \theta, \psi)$ is an estimate of the advantage function given by $\sum_{i=0}^{q-1} \gamma^i r_{t+i} + \gamma^q V(s_{t+q};\psi) - V(s_t;\psi)$, and q varies from state to state and is upper-bounded by t_{max}. The model updates both the policy and the value function based on the returns of every t_{max} actions or until reaching the terminal state.

The DAFL algorithm is described in Algorithm 2. At the beginning, the DAFL algorithm initializes some variables, e.g., the set of subagents ϕ, the global parameters of actor network θ and critic network ψ (Line 1). Each subagent initializes the gradients of two networks and pulls the global parameters from the master agent (Line 4-5). The subagent will interact with the environment and independently update two networks with different targets (Line 6-10). Then the gradients will be computed and updated through gradient ascent optimization process (Line 11-13). After that the subagents push their updates to the master agent as an input (Line 14). At the end of each epoch, the optimal value of α will be selected for the next epoch of traning by the output of actor network in the master agent (Line 15-16).

3.3 Extending to the General Case

In many practical scenarios, there are usually multiple simultaneous learning tasks in edge computing [30], [31], [32]. Thus, we consider the more general version, denoted as AAFL-RCG, for multiple learning tasks. In this paper, we focus our attention on independent learning tasks, while the case of multiple dependent learning tasks will be regarded as our future work.

3.3.1 Problem Formulation

There are multiple learning tasks $L = \{l_1, \ldots, l_m\}$ in edge computing. To minimize the maximum completion time of all learning tasks with resource constraints, we will determine the proper values of two parameters T_j and α_j^t for each task l_j, where T_j denotes the total number of epochs and α_j^t is the fraction of local updated models for global aggregation in epoch t. Accordingly, the AAFL-RCG problem can be described as follows:

$$\min \max_{j\in\{1,\ldots,m\}} \sum_{t=1}^{T_j} H_j^t$$
$$s.t. \begin{cases} \max_{j\in\{1,\ldots,m\}} F_j(w^{T_j}) \le \mathcal{F} & \\ \sum_{j\in\{1,\ldots,m\}} \sum_{t=1}^{T_j} (1+\alpha_j^t)\cdot n \cdot b^j \le B & \\ \sum_{i=1}^{n} \beta_{i,j}^t \ge \alpha_j^t \cdot n & \forall j,t \quad , \\ \beta_{i,j}^t \in \{0,1\} & \forall i,j,t \\ \alpha_j^t \in \{\frac{1}{n},\ldots,1\} & \forall j,t \end{cases} \tag{18}$$

where b^j denotes the bandwidth cost for forwarding a global update to an edge node of task l_j, and B is the total bandwidth budget. $\beta_{i,j}^t$ is a binary variable to indicate whether the local updated model of task l_j at edge node v_i is involved in the global update or not at epoch t. The first set of inequalities denotes that the maximum loss value of these tasks should be less than the convergence threshold $\mathcal{F}$. The second set of inequalities means that the bandwidth consumption by all learning tasks should not exceed the budget B. The third set of equations tells that the parameter server will receive at least $\alpha_j^t \cdot n$ local updated models from all workers of each task l_j for model aggregation in each epoch t.

3.3.2 Algorithm for AAFL-RCG

In this section, we extend the DAFL algorithm for multiple learning tasks, called DAFL-G. Each task l_j maintains a variable q_j, which denotes the current epoch index. When the global model aggregation of a task l_j is finished, we update the variable $q_j = q_j + 1$. If we find that q_j is the smallest epoch index among all learning tasks, DAFL-G will be triggered. Besides, we use $\mathcal{X}_k$ and t'_k to denote the smallest epoch index and the corresponding time point, respectively, when DAFL-G is triggered at the kth time. The training speed and convergence degree of all tasks may be different, which determines the number of local updated models involved in the global update. Therefore, let Φ_k be the set of $\xi_j^{\mathcal{X}_k}$, where $\xi_j^{\mathcal{X}_k} \in \{\frac{1}{n},\ldots,1\}$ denotes the fraction of local updated models involved in the global update of task l_j at the time point $\mathcal{X}_k$.

Similar to DAFL, we also adopt A3C, which can train multiple learning tasks without changes in the network architecture [33], to determine the optimal value of Φ_k at the kth time point. DAFL-G trains the global actor network and critic network with several subagents, deriving the optimal action for each learning task at each epoch. Different from DAFL, the operations of multiple learning tasks increase the difficulty of management. Thus, we need to keep the agent and environment stable as soon as possible, instead of changing frequently.

To this end, we need to modify some agent settings, including state, reward and action, for multiple learning tasks. Then, the value of $\alpha_j^{q_j+1}$ for task l_j at epoch q_j will be updated, i.e., $\alpha_j^{q_j+1} = \xi_j^{\mathcal{X}_k}$. During the training, the agent interacts with the environment and observes the state. Under this setting, we redefine the state $s_{t'_k} = (t'_k, \mathbb{H}_{t'_k}, \mathbb{B}_{t'_k}, \mathbb{F}_{t'_k}, \Omega_{t'_k}, \mathcal{F}, \mathbb{R}_{t'_k})$. We use $\mathbb{H}_{t'_k}$ and $\mathbb{B}_{t'_k}$ to record the completion time and bandwidth consumption of each task during epochs $\mathcal{X}_{k-1}$ and $\mathcal{X}_k$, respectively. $\mathbb{F}_{t'_k}$ is the set of loss function of all learning tasks at time point t'_k and $\Omega_{t'_k} = \{\triangle F_{t'_k}^1, \ldots, \triangle F_{t'_k}^m\}$ denotes the set of differences between the current loss value and the target loss value $\mathcal{F}$ for all m learning tasks. Each task keeps training until the maximum loss value of these tasks reaches the threshold $\mathcal{F}$. Finally, the remaining bandwidth budget after the time point t'_k is denoted as $\mathbb{R}_{t'_k}$.

Due to the different training speed of these tasks, some of them may have reached the convergence, while others may still need to train several epochs. Thus, we redefine the reward for task l_j at time point t'_k (or epoch $\mathcal{X}_k$) as

$$r_{t'_k}^j = \begin{cases} -\Upsilon^{\Xi} + \frac{\triangle F_{t'_k}^j}{\mathcal{F}} - \frac{b_{t'_k}^j}{\mathbb{R}_{t'_k}} + \mathcal{M} & \text{if } l_j \le \mathcal{F}, \\ -\Upsilon^{\Xi} + \frac{\triangle F_{t'_k}^j}{\mathcal{F}} - \frac{b_{t'_k}^j}{\mathbb{R}_{t'_k}} - \mathcal{M} & \text{otherwise.} \end{cases} \tag{19}$$

where $\Xi = \frac{H_{\mathcal{X}_k}^j - \overline{H}_{\mathcal{X}_{k-1}}^j}{\overline{H}_{\mathcal{X}_{k-1}}^j}$ and $\mathcal{M}$ is a positive number. When the loss value of task l_j has reached the convergence threshold $\mathcal{F}$ under the resource constraint, the reward will be added by $\mathcal{M}$ so that the probability of the current action will increase. Thus, the actions will not change frequently and the agent tends to be stable gradually.

Note that our action space is a discrete space including as many as n^m possible choices, where n is the number of workers and m is the number of learning tasks. For example, given 3 tasks and 100 edge nodes, there are 1,000,000 possible actions in each epoch. Training a DRL agent with such a large action space would be costly. Thus, we redefine

the policy π as a Gaussian probability density over a real-valued space [34] as follows:

$$\pi(a|s,\theta) = \frac{1}{\sigma(s,\theta)\sqrt{2\pi}} \exp\left[-\frac{\left(a-\mu(s,\theta)\right)^2}{2\sigma(s,\theta)^2}\right], \tag{20}$$

where μ denotes the expectation and σ is the standard deviation. The agent can choose an action a_{χ_k} based on the conditional probability $\pi(a_{\chi_k}|s_{\chi_{k-1}},\theta)$. To make DAFL-G more efficient, we just need to find the parameters $(\mu(s,\theta),\sigma(s,\theta))$ in a 2-D continuous space, instead of learning the probability mass function over a large discrete action space.

3.4 Discussion

In this section, we make some discussions about the proposed mechanism and algorithms. In addition to bandwidth resource, our proposed experience-driven solution also can be applied to other resource budget constraints. For example, there are two categories of resources (e.g., network bandwidth and energy). Let ϱ_t denote the energy consumption by the workers at each epoch t. We use $\mathcal{E}_t$ to denote the remaining energy budget at the end of epoch t. We need to collect more network information from the workers for DRL training. To this end, we just need to modify some agent settings, including state and reward. We add the energy consumption ϱ_t and remaining energy budget $\mathcal{E}_t$ to the state vector. Besides, the reward is redefined as

$$r_t = -\Upsilon^{\frac{H_t-\overline{H}_{t-1}}{\overline{H}_{t-1}}} + \frac{\triangle F_t}{\mathcal{F}} - \frac{b_t}{\mathcal{G}_t} - \frac{\varrho_t}{\mathcal{E}_t}, \quad t < T. \tag{21}$$

In other words, more resources the task consumes, the less reward the agent will receive. Thus, state and reward in DRL agent can be easily modified, even if more network resources budgets are considered.

4 Performance Evaluation

This section first introduces the metrics and benchmarks for performance comparison (Section 4.1). Then, we describe the datasets and models for simulations (Section 4.2). Besides, simulation settings and results are also given (Section 4.3). Finally, we evaluate our proposed algorithm through a real small-scale test-bed and give the results (Section 4.4).

4.1 Performance Metrics and Benchmarks

In this paper, we design the DRL-based asynchronous federated learning algorithm for efficient model learning in edge computing. We adopt the following metrics to evaluate the efficiency of our proposed algorithm. (1) *Training loss* is the quantification difference of probability distributions between model output and observation results. The loss value reflects the quality of model learning and whether convergence has been achieved or not as described in Section 2.2. (2) *Reward* of DRL is the return of the reward function in DRL training. (3) As one of the most common performance metrics for classification, *accuracy* is measured by the proportion between the amount of the right data by the model and that of all data. (4) We adopt the *completion time* to estimate the training speed of a learning task. (5) When there are multiple learning tasks in the network, we measure the *maximum loss* and *minimum accuracy* of all tasks to evaluate the training performance.

We choose two recent algorithms as benchmarks for performance comparison. The first one, called ADP-FL [16], belongs to the synchronous FL scheme. In an epoch, the number of local updates before one global update on each edge node are adaptively determined by the coordinator and can be found via linear search to minimize the loss function under given resource constraints. The second one, called AFO [20], is an asynchronous federated optimization algorithm with provable convergence, in which the parameter server will perform the global update on receiving only one local updated model from an arbitrary worker. Thus, we choose these two algorithms, the synchronous scheme ($\alpha = 1$) and the asynchronous scheme ($\alpha = \frac{1}{n}$), as benchmarks in our work.

4.2 Datasets and Models

We evaluate the training process with two open-source datasets, i.e., Fashion-MNIST [35], and CIFAR-10 [36], constructed for image classification tasks. The Fashion-MNIST dataset (referred to as FMNIST) contains 70,000 images of fashion handwritten digits (60,000 for training and 10,000 for testing), each of which is a 28×28 grayscale image associated with a label from 10 classes. CIFAR-10 includes 60,000 32×32 color images (50,000 for training and 10,000 for testing) of 10 different classes, with 6,000 images per class.

We choose two classical models with different structures and parameters. One is the logistic regression (referred to as LR in short) for the FMNIST dataset. LR is constructed of a fully connected network with two hidden layers, each of which has 512 units. The other is the convolutional neural networks (CNN) for both FMNIST and CIFAR-10. CNN has two 5×5 convolution layers, a fully connected layer with 512 units, and a softmax output layer with 10 units.

4.3 Simulation Evaluation

4.3.1 Evaluation Settings

The simulations are conducted on an AMAX deep learning workstation[1] (CPU: Intel(R) E5-2620v4, GPU:NVIDIA GeForce RTX 2080Ti), where we build an FL simulation environment and implement all models with PySyft [37], a Python library for privacy-preserving deep learning including FL, under the PyTorch framework.

Network Resources. In the simulations, we mainly focus on the bandwidth and time resource cost in edge computing. Specifically, the bandwidth consumption can be measured by the size of the model parameters transmitted. We train some models under a fixed amount of resource budget (e.g., network bandwidth and completion time). In order to implement the resource efficient asynchronous federated learning, we set the value of parameters $\eta = 0.01, \varsigma = 0.9, \Upsilon = 2, \varpi = 0.3$ and estimate the values of parameters L, ϑ in real time according to [38].

As suggested in [18], in order to efficiently simulate the training processing in FL of our proposed solution and benchmarks, a total of 100 edge nodes are generated in the

1. https://www.amax.com/products/gpu-platforms/

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. We partition the training datasets for the workers with a non-IID setting. At the beginning, each worker node is allocated with the same amount of images which are grouped by their labels as its local data. Besides, the test datasets are allocated to the server for evaluating and testing the global models.

Data Distribution on Workers. To simulate the data imbalance, we assign different amounts of data into the workers based on the random distribution with parameter $\mathbb{A}$ for the maximum value and $\mathbb{B}$ for the minimum value. For example, given 60,000 data samples and 10 workers, let $\mathbb{A}$=1,000 and $\mathbb{B}$=9,000. The amount of data in the worker may be 1,000 $\sim$ 9,000, i.e., the maximum amount of data in the worker is 9,000 and the minimum amount of data is 1,000. We mainly consider four different ways of data distribution (Case 1-4).[2]

Model Training. Two models are separately trained using ADP-FL, AFO and AAFL. For ease of interpretation of results, AAFL performs only one local update between two global updates, i.e., $D = 1$. Mini-batch SGD with batch size of 50 is applied to optimize the local models. We repeat each simulation 10 times and compute the average results.

4.3.2 Simulation Results

1) *Training the DRL Agent.* The experience-driven method can be divided into two phases: (1) an offline training phase first simulates the environment and generates a DRL policy network based on rewards; and (2) an online running phase, which deploys the policy network at the server to online determine the value of α_t for FL system. Both two phases are performed on the parameter server (e.g., cloud [39], [40]), which usually has sufficient computation resource compared with workers (or edge devices). Thus, the main workload of training a DRL policy network can be luckily completed offline on the server. We first test the performance of DRL training, including the training loss and reward. The DRL training is conducted in a simulation system with 5 edge nodes for 100 episodes. We first observe the change of training loss with the increasing number of episodes in DRL. The left plot of Fig. 5 shows that the training loss decreases quickly in the earlier stages of DRL training process. That is because the DRL agent has no information of the FL environment that causes a large loss value. Thanks to the efficient explorations of agent, the loss can be rapidly minimized with the model training procession. After less than 20 episodes, the training loss becomes stabilized, which means that the DRL agent learns to adapt to the FL environment. By the right plot of Fig. 5, we observe the change of reward with the increasing of epochs in one DRL training episode. Specifically, the reward value increases with the epochs and gradually tends to be stable. That is because the DRL model is enabled to learn a better policy to achieve a better reward. When reaching 200 epochs, the reward starts to saturate with slight fluctuations.

2) *Single Learning Task.* The first set of simulations evaluates the performance of the classification models (e.g., CNN) without resource constraints. We train each set of model and dataset, including CNN over FMNIST and CNN over CIFAR-10, for all three solutions (i.e., AFO, ADP-FL and AAFL) with 600 epochs. As shown in Figs. 6 and 7, AAFL can achieve similar training performance with ADP-FL, which adopts the synchronous scheme. Besides, AAFL can achieve better training performance than AFO with an increasing number of epochs. For example, when CNN training runs 600 epochs and over the FMNIST dataset, the accuracy achieved by AAFL is about 75 percent, while that by ADP-FL and AFO is about 78 and 65 percent, respectively. We can conclude that our proposed AAFL framework can improve the accuracy of the classification model by about 10 percent compared with AFO.

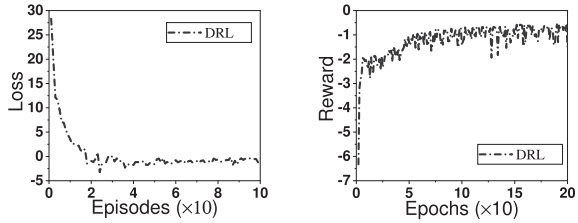


Fig. 5. Training convergence of DRL agent. *Left plot*: Loss versus No. of episodes; *Right plot*: Reward versus No. of epochs.

The second set of simulations observes the performance of the classification models (e.g., CNN) with resource constraints. In practice, some training tasks often need to be completed within a specified time. We train CNN over FMNIST with a limited completion time constraint (e.g., 1,800s). Due to synchronization barrier, the completion time of each epoch mainly depends on the maximum training time among these workers, which will lead to long completion time. Thus, ADP-FL does not achieve convergence within given completion time constraint. AFO will run averagely 4 times as many training epochs as both AAFL and ADP-FL within the same time constraint by Fig. 8. However, the training performance (e.g., loss or accuracy) of AFO is worse than that of AAFL, which achieves better performance than ADP-FL. For example, given the fixed time constraint, the loss of AAFL is 0.74, while the minimum loss of AFO and ADP-FL is about 1.03 and 0.79, respectively. Accordingly, the accuracy of AAFL, AFO and ADP-FL is about 74, 65 and 71 percent, respectively. Thus, AAFL can improve the accuracy by 9 and 3 percent compared with AFO and ADP-FL, respectively.

The communication between the parameter server and the workers will cause huge bandwidth consumption. We measure the training performance of CNN over FMNIST with a limited bandwidth constraint (e.g., 10Gb). As shown in Fig. 9, AAFL can achieve higher accuracy than both AFO and ADP-FL. For example, the accuracy of CNN over FMNIST using AAFL is about 74 percent, while that by AFO and ADP-FL is about 66 and 72 percent, respectively. Thus, the proposed AAFL framework can improve the accuracy by about 8 and 2 percent compared with AFO and ADP-FL, respectively.

The third set of simulations observes the change of parameter of α with the increasing number of epochs in AAFL. The value of α is fixed in AFO ($\alpha = 0.1$) and ADP-FL ($\alpha = 1$). Our proposed algorithm can adaptively adjust the value of α according to the environment. At the beginning of training, the parameter server aggregates more local updates so as to accelerate the convergence of model

2. Case 1: 5,500 $\sim$ 6,500; Case 2: 5,000 $\sim$ 7,000; Case 3: 3,000 $\sim$ 9,000; Case 4: 1,000 $\sim$ 11,000;

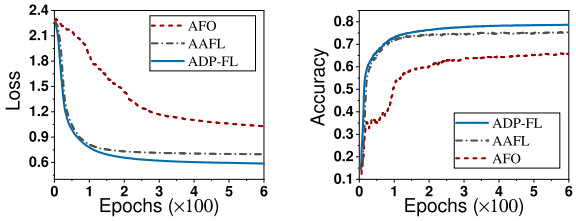


Fig. 6. Training performance of CNN over FMNIST without resource constraints. *Left plot*: Loss; *Right plot*: Accuracy.

training. When the model tends to converge, less local updates are required for global model aggregation, which saves a lot of network resources (e.g., network bandwidth). In Fig. 10, the value of α in AAFL is decreasing with the training epochs and tends to be stable.

The fourth set of simulations tests the completion time of three solutions with different data distributions at the workers. The amount of data on the workers is always changing dynamically. Fig. 11 shows that the completion time increases for all solutions under four cases. However, the increasing ratio of AAFL is much slower than that of the other two benchmarks. In comparison, AAFL requires less completion time than AFO and ADP-FL. For example, by the right plot of Fig. 11, the completion time of AAFL is about 340s, while ADP-FL and AFO need about 690s and 750s, respectively. In other words, AAFL can reduce the completion time by about 51 and 55 percent compared with ADP-FL and AFO, respectively.

In the fifth set of simulations, we observe the impact of different number of workers on training performance (CNN over FMNIST) within given bandwidth budget (e.g., 5Gb). We adopt five different number of workers (e.g., 10, 20, 30, 50 and 100) to test the loss value and completion time. The testing results in Fig. 12 show that the training performance of two schemes is rarely improved, or even worse, when there are more than 20 workers. For example, in the left plot of Fig. 12, we test the loss value of two schemes within given bandwidth budget. In AAFL, the parameter server will aggregate more (not all) local updates in each epoch with the increasing number of workers in edge computing. Thus, the global training model can well achieve convergence, and the loss value gradually decreases. However, more workers will also bring more resource consumption and waiting time. The total number of training epochs will be reduced in ADP-FL, leading to poor training performance. Besides, the testing results, in the right plot of Fig. 12, show that the completion time is gradually increasing in AAFL and ADP-FL, when the number of edge nodes is more than 20. Accordingly, we choose 20 as the limitation of workers to participate in the model training in the simulations.

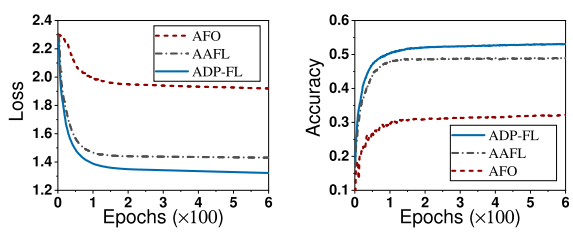


Fig. 7. Training performance of CNN over CIFAR-10 without resource constraints. *Left plot*: Loss; *Right plot*: Accuracy.

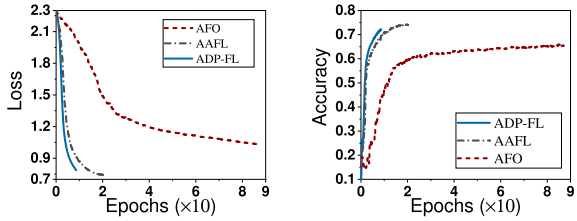


Fig. 8. Training performance of CNN over FMNIST with completion time constraint. *Left plot*: Loss; *Right plot*: Accuracy.

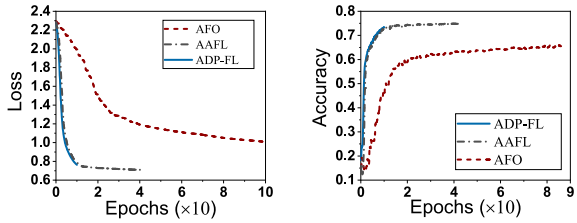


Fig. 9. Training performance of CNN over FMNIST with bandwidth constraint. *Left plot*: Loss; *Right plot*: Accuracy.

3) *Multiple Learning Tasks.* The sixth set of simulations observes the performance of multiple learning tasks without resource constraints. We run two models over three different datasets, including LR over FMNIST, CNN over FMNIST and CIFAR-10, respectively. Each learning task performs 300 epochs. Fig. 13 shows that the maximum loss and minimum accuracy of the three learning tasks. By this figure, AAFL achieves a little worse training performance (e.g., loss or accuracy) than ADP-FL, but better than AFO. For example, given 300 training epochs, the accuracy of AAFL is about 75 percent, while that of AFO and ADP-FL is about 61 and 77 percent, respectively.

In the seventh set of simulations, we test the performance of multiple learning tasks with limited bandwidth constraints. By Fig. 14, the maximum loss becomes smaller and the minimum accuracy becomes higher by changing the bandwidth constraint from 300 Mbps to 1,500 Mbps for all three solutions.

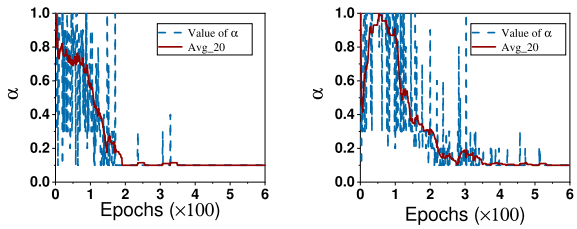


Fig. 10. Value of α versus No. of Epochs. *Left plot*: LR-FMNIST; *right plot*: CNN-FMNIST.

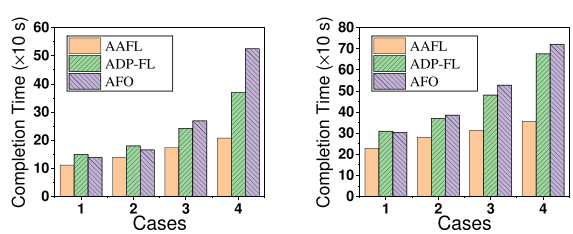


Fig. 11. Completion time versus different cases. *Left plot*: LR-FMNIST; *right plot*: CNN-FMNIST.

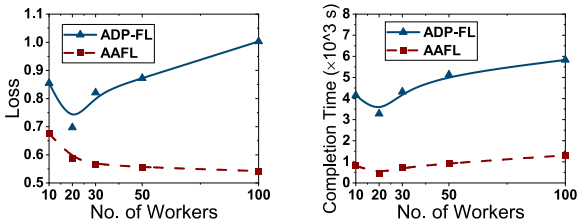


Fig. 12. Loss and completion time versus different number of workers within given constraint. *Left plot*: Bandwidth; *right plot*: Accuracy.

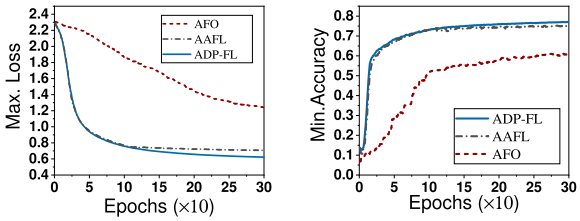


Fig. 13. Max. loss and Min. accuracy for multiple learning tasks without resource constraints.

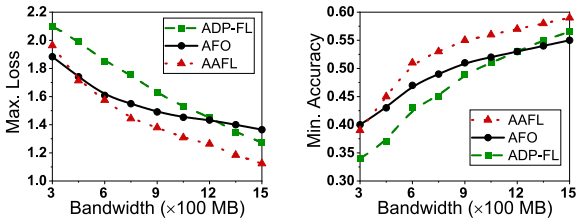


Fig. 14. Max. loss and Min. accuracy with bandwidth constraint for multiple learning tasks.

Our proposed AAFL framework can achieve less loss and higher accuracy compared with the other two benchmarks. For example, when the bandwidth constraint is 900 Mbps, the minimum accuracy of AAFL is about 55 percent, while that of ADP-FL and AFO is only about 49 and 51 percent, respectively. In other words, AAFL can improve the minimum accuracy by about 6 and 4 percent compared with ADP-FL and AFO, respectively.

The last set of simulations observes the performance of multiple learning tasks with a limited completion time constraint. We test three learning tasks by changing the completion time constraint from 600s to 3,000s. Fig. 15 shows that AAFL can achieve significantly higher minimum accuracy than both ADP-FL and AFO. For example, when the completion time constraint is 1,800s, the minimum accuracy of three learning tasks by AAFL is 69 percent, while that of AFO and ADP-FL is about 51 and 59 percent. Thus, our

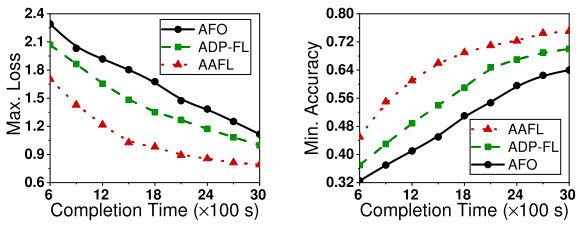


Fig. 15. Max. loss and Min. accuracy with completion time constraint for multiple learning tasks.

Fig. 16. The test-bed platform.

AAFL framework can improve the minimum accuracy by about 18 and 10 percent compared with AFO and ADP-FL, respectively. These results show that AAFL can significantly improve the classification accuracy compared with two benchmarks under resource constraints.

In conclusion, our proposed AAFL mechanism can reduce the completion time by about 55 percent compared with the existing schemes even under data imbalance and edge dynamics in the network. Moreover, AAFL can improve the maximum loss value and minimum classification accuracy compared with AFO and ADP-FL under the resource constraints.

4.4 Test-Bed Evaluation

4.4.1 *Implementation on the Platform*

We implement AFO, ADP-FL and AAFL on a real small-scale test-bed in Fig. 16, which is composed of two main parts: a deep learning workstation with four NVIDIA GeForce RTX Titan GPUs and 10 Jetson TX2 developers[3] (CPU: ARMv8 Cortex-A57, RAM: 8GB). 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 Jetson TX2 developer as a worker to locally train the model and send the updates to the parameter server for aggregation. We develop a distributed model training framework with pytorch[41]. The workers and the parameter server are physically connected through wireless network under the same router. Besides, they are logically connected through *torch.distributed* package (*gloo* back-end). 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. All our code is publicly available at github.[4]

Two typical CNN models with different types and structures are implemented for CIFAR10 and FMNIST,

3. https://developer.nvidia.com/embedded/jetson-tx2-developer-kit

4. https://github.com/lyl617/AAFL

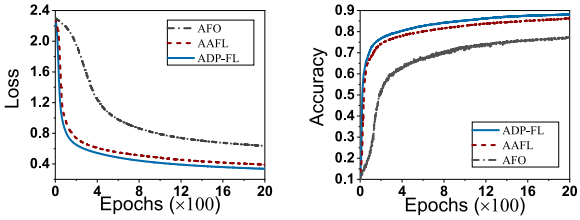


Fig. 17. Loss and accuracy with CNN over FMNIST in test-bed.

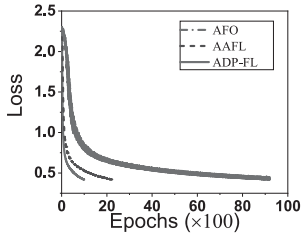


Fig. 19. Loss versus Epochs with balanced data in test-bed.

respectively, on the test-bed. The first CNN model is used for CIFAR10 dataset. It has two 5×5 convolution layers (64, 64 channels, each followed by 3×3 max pooling), two dense layers with 384 and 192 units and a softmax output layer with 10 units. The second CNN model which has two 5×5 convolution layers (32, 64 channels, each followed by 2×2 max pooling), a dense layer with 1024 units and a softmax output layer with 10 units (related to the 10 classes in FMNIST), is used for the FMNIST dataset.

The different data distributions (e.g., quantity and category) among the workers have great impact on the performance of model training. In the test-bed, we mainly consider the following two data distributions. First, the distribution of data amount is often imbalanced, and significantly varies with time and space on the workers. We adopt three different cases of data amount distributions to simulate the data imbalance. (1) Case 1 (balance): We allocate the same amount (e.g., 6,000) of training data among the ten workers; (2) Case 2 (weak imbalance): There is little difference in the amount of data among workers (e.g., 4,000-8,000); (3) Case 3 (strong imbalance): The amount of data on these workers varies greatly (e.g., 1,000-11,000). Second, different categories of data distributions, i.e., IID and non-IID, among the workers also emerge different effects on the performance of model training. For example, in the case of IID, each worker has all categories of data samples (e.g., 10 classes), but in the case of non-IID, each worker may have only part of categories (e.g., 5 classes). We adopt four different cases to verify the effect of data distributions on model training, including IID data and the three different levels of non-IID data. I): Each data sample is randomly assigned to a worker, thus each worker has uniform (but not full) information, i.e., IID data; II): Each worker has 5 categories of data samples; III): Each worker has 2 categories of data samples; IV): Each worker only has 1 category of data samples.

4.4.2 Testing Results

In the first set of experiments, we test the balanced and uniform data with CNN training over FMNIST and CIFAR10, respectively. We run two groups of experiments with 2,000 training epochs. In Figs. 17 and 18, the training performance (i.e., loss and accuracy) of AAFL is very close to that of ADP-FL and much better than that of AFO. For example, given 2,000 epochs in CNN training over the FMNIST dataset, the loss value of AAFL is 0.3919, while that of ADP-FL and AFO is 0.3376 and 0.6375, respectively. Accordingly, the training accuracy of AAFL is about 86.4 percent, and that of ADP-FL and AFO is about 87.9 and 75.2 percent, respectively. Thus, our proposed mechanism can improve the training accuracy by about 11 percent compared with AFO. Besides, we also observe the performance of DRL training on the test-bed, including training loss and reward. After less than about 30 episodes, the training loss becomes stabilized, which is shown in the left plot of Fig. 5. It means that the DRL agent learns to adapt to the FL environment. By the right plot of Fig. 5, we observe the change of reward with the increasing of epochs in one DRL training episode. Specifically, the reward value also increases with the epochs and gradually tends to be stable. The two figures show that a better policy can be learned to achieve a better reward by DRL training in the test-bed.

In the second set of experiments, we observe the performance of model training (CNN over FMNIST) under three different distributions of data amount (cases 1-3). In each case, we run the ADP-FL algorithm with 1,000 training epochs as baseline. Fig. 19 shows that more training epochs (about 2,195) are needed by AAFL to reach the loss value of the baseline under case 3. Moreover, AFO needs 9,199 training epochs to reach the same performance of training loss. That's because the server only aggregates the updated local model from the arbitrary one worker at a time in AFO. In other words, AFO needs 9× training epochs compared with ADP-FL, while AAFL only needs 2× epochs compare with the baseline. The training loss of AAFL under the three different cases is shown in Fig. 20. Because AAFL can efficiently alleviate the problem of synchrnoization barrier caused by imbalanced data. The results show that the performance of training loss under cases 2 and 3 (imbalanced data) is very close to that of case 1 (balanced data).

We also observe the other performance metrics of training, such as accuracy, time and bandwidth, under the cases 1-3.

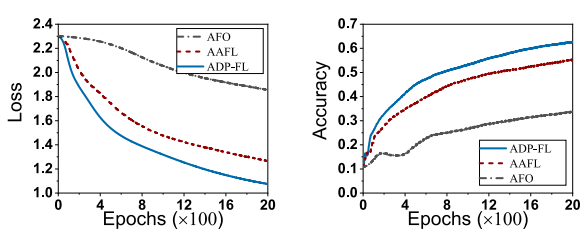


Fig. 18. Loss and accuracy with CNN over CIFAR10 in test-bed.

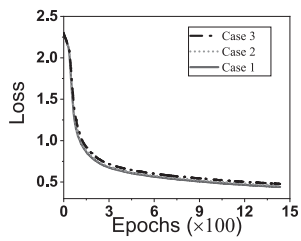


Fig. 20. Loss versus Epochs under cases 1-3 with IID data.

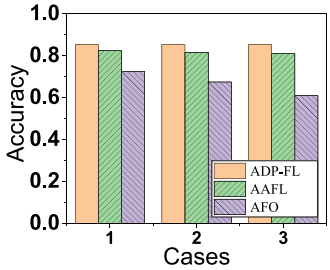


Fig. 21. Training accuracy under cases 1-3 in test-bed.

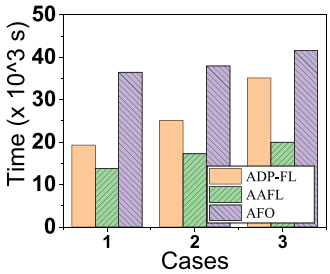


Fig. 22. Training time under cases 1-3 in test-bed.

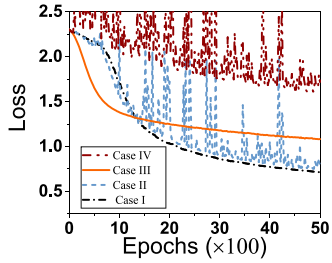


Fig. 24. Loss versus Epochs under case I-IV with balanced data.

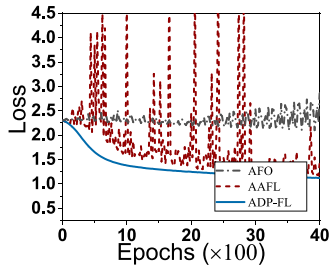


Fig. 25. Loss versus Epochs of three solutions under case II.

Fig. 21 shows the training accuracy after 1,000 training epochs under different cases. In AAFL, more updated models ($\alpha \geq 0.1$) from workers are involved in the model aggregation than that of AFO ($\alpha = 0.1$) in each epoch. In each case, AAFL always achieves better accuracy compared with AFO, and similar performance of ADP-FL. As shown in Fig. 22, our proposed mechanism achieves the minimum training time while reaching the same training performance (loss and accuracy) of the baseline among the three solutions. For example, for case 1, the training time of AAFL is about 11,244s, while that of ADP-FL and AFO is about 22,324s and 36,483s, respectively. In other words, AAFL can reduce the training time about 49.6 and 69.2 percent compared with ADP-FL and AFO, respectively. We test the network bandwidth consumption of three solutions. The server only aggregates one arbitrary local updated model from the workers. Fig. 23 shows that the bandwidth consumption of AFO is the least among the three schemes. However, the bandwidth consumption of AAFL is very close to that of AFO. For example, in case 2, the bandwidth consumption of AAFL is about 1.73Gb, while that of AFO and ADP-FL is about 2.51Gb and 3.79Gb, respectively. In other words, AAFL can improve the performance of bandwidth consumption by about 54.3 percent compared with ADP-FL.

The last set of experiments tests the performance of model training (CNN over FMNIST) under four different categories of data distributions (cases I-IV). We first test the training performance of AAFL under four different cases of data categories distribution (cases I-IV). Fig. 24 shows that the distribution of data categories will emerge the effects on the speed of model training. For example, the training loss of the experiment under case IV by running 5,000 epochs is about 0.4138, while that of case I is about 0.8272. In other words, the training performance with non-IID data is worse than that of IID data. Then, we test the training performance under case II. As shown in Fig. 25, the training loss of AAFL is very close to that of ADP-FL. For example, given 4,000 epochs, the loss value of AAFL and ADP-FL is about 0.6372 and 0.6268, respectively. However, the loss value of AFO has no obvious downward trend and tends to be stable. Thus, the solution AFO cannot well handle non-IID training data, but our proposed AAFL can well handle it.

Besides, the training performance (accuracy, time and bandwidth) of three solutions under cases I-IV is shown in Figs. 26, 27, and 28. Given 1,000 training epochs, we test the training accuracy under cases I-IV. As shown in Fig. 27, the training accuracy achieved by AAFL is always better than AFO in various cases. For example, the accuracy of AAFL is about 68.9 percent under case III, while the accuracy of ADP-FL and AFO is about 72.6 and 23.5 percent, respectively. In other words, our proposed mechanism can improve the training accuracy by about 45 percent compared with AFO. We

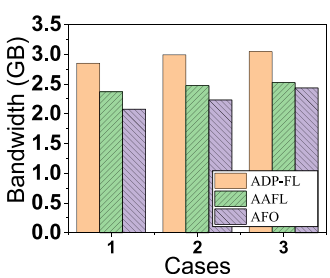


Fig. 23. Network bandwidth under cases 1-3 in test-bed.

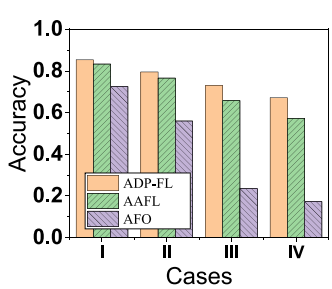


Fig. 26. Training accuracy under cases I-IV in test-bed.

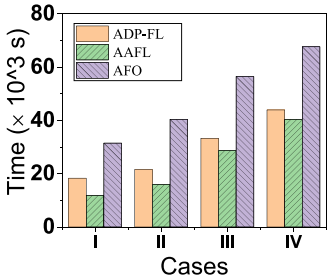


Fig. 27. Training time under cases I-IV in test-bed.

then evaluate the training time of ADP-FL, AAFL and AFO under cases I-IV. Our proposed solution can efficiently avoid the straggler problem caused by ADP-FL. Fig. 27 shows that the training time required by AAFL is always less than that of ADP-FL in each case. For example, the required training time by AAFL is about 14,913s under case II, while that of ADP-FL and AFO is 29,684s and 40,394s, respectively. Thus, RE-AFL can reduce the training time by about 49.8 and 63.1 percent compared with ADP-FL under Case II. Finally, we test the bandwidth consumption of the three solutions under cases I-IV. As shown in Fig. 28, the bandwidth consumption of AAFL is much less than that of ADP-FL, although it is slightly larger than that of AFO. For example, the bandwidth consumption of AAFL is about 2.07Gb under case I, while that of ADP-FL and AFO is about 2.85Gb and 1.78Gb, respectively. That means AAFL can improve the network bandwidth by about 27.3 percent compared with ADP-FL. In conclusion, our proposed mechanism achieves better performance (e.g., loss, time and bandwidth) of model training compared with the benchmarks under different cases of data distribution.

5 Related Works

Recently, federated learning (FL) [42] has been widely mentioned and studied in both academia and industry fields.

One research area related to FL is distributed machine learning (DML) through worker machines and parameter servers [8]. Bao *et al.* [43] propose an online algorithm for scheduling the arriving jobs and deciding the numbers of concurrent workers and parameter servers for each job over its course, so as to maximize the overall utility of all jobs. Ho *et al.* [44] design a parameter server system which maximizes the time computational workers spend doing useful work on algorithms for DML, and the system followed a Stale Synchronous Parallel (SSP) model of computation. The authors [45] 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, improve runtime performance and achieve good model accuracy.

The above works mainly study efficient solutions of DML in datacenters. Under this scenario, shared storage is usually adopted. But in edge computing, no storage will be shared among edge nodes. 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 [46]. Smith *et al.* [12] 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. Our method and theory consider issues of high communication cost, stragglers, and fault tolerance for distributed multi-task learning. The authors [34] propose an asynchronous and distributed machine learning framework based on the emerging serverless architecture, with which stateless functions can be executed in the cloud without the complexity of building and maintaining virtual machine infrastructures. Shi *et al.* [47] merge some short communication tasks into a single one to reduce the overall communication time and formulate an optimization problem to minimize the training iteration time. The authors [48] introduce a new and increasingly relevant setting for distributed optimization in machine learning, where the data for the optimization of training are distributed over an extremely large number of nodes. However, most of these solutions ignore the impact of limited resource constraints on training performance, leading to massive resource consumption on edge computing systems. Konevcny *et al.* [27] proposed two ways to reduce the uplink communication costs. The first one is structured updates, where they directly learn an update from a restricted space parametrized using a smaller number of variables, e.g., either low-rank or a random mask. The second is sketched updates, where they learn a full model update and then compress it using a combination of quantization, random rotations, and subsampling before sending it to the server. Xie [20] proposes a new asynchronous federated optimization algorithm. We prove that the proposed approach has near-linear convergence to a global optimum, for both strongly and non-strongly convex problems, as well as a restricted family of non-convex problems.

Last but not least, some works [7], [18] similar to our research will be introduced. The authors [7] perform FL efficiently while actively managing workers based on their resource conditions. Specifically, the proposed solution solves a worker selection problem with resource constraints, which allows the server to aggregate as many local updates as possible and to accelerate performance improvement in ML models. Wang [18] propose an experience-driven control framework that intelligently chooses the workers to participate in each round of federated learning to counterbalance the bias introduced by non-IID data and to speed up convergence of model training. However, after selecting the subset of workers to participate in the model training, the parameter server only

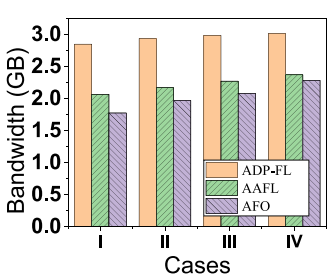


Fig. 28. Network bandwidth under cases I-IV in test-bed.

perform model aggregation while receiving all local updates from these workers. In other words, the *synchronous scheme* is adopted by these works for global updating on the server. Compared with our proposed asynchronous scheme, these researches cannot solve synchronization barrier problem which will lead to longer training time and worse training performance under given resource budget.

To our best knowledge, we are the first to address the problem of determining the number of received local updates from the workers to optimize the training performance of learning tasks, with a given resource budget for federated learning in edge computing systems.

6 Conclusion

In this paper, we present the adaptive asynchronous federated learning (AAFL) mechanism for edge computing, and analyze the convergence bound. The adaptive asynchronous federated learning with resource constraints (AAFL-RC) problem is formulated for minimizing the completion time of model training. We further design experience-driven algorithms based on deep reinforcement learning (DRL) to adaptively determine the optimal values of parameters in AAFL for single learning task and multiple learning tasks, respectively. The simulation and experimental results show that AAFL can achieve significantly higher accuracy and less completion time of model training under resource constraints, compared with the existing solutions. In the future work, we will study the multiple dependent learning tasks in the edge computing system.

Acknowledgments

This work was supported in part by the National Science Foundation of China (NSFC) under Grants 61822210, 61936015, and U1709217 in part by Anhui Initiative in Quantum Information Technologies under Grant AHY150300.

References

[1] A. Pantelopoulos and N. G. Bourbakis, "A survey on wearable sensor-based systems for health monitoring and prognosis," *IEEE Trans. Syst., Man, Cybern., Part C Appl. Rev.*, vol. 40, no. 1, pp. 1–12, Jan. 2010.

[2] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," *Esann*, vol. 3, p. 3, 2013.

[3] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Towards an intelligent edge: Wireless communication meets machine learning," 2018, *arXiv:1809.00343*.

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

[5] H. B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," 2017. [Online]. Available: http://www.googblogs.com/federated-learning-collaborative-machine-learning -without-centralized-training-data/

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

[7] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–7.

[8] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *Proc. 11th {USENIX} Symp. Operating Syst. Des. Implementation*, 2014, pp. 583–598.

[9] A. Hard *et al.*, "Federated learning for mobile keyboard prediction," 2018, *arXiv:1811.03604*.

[10] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Federated learning for ultra-reliable low-latency v2v communications," in *Proc. IEEE Glob. Commun. Conf.* 2018, pp. 1–7.

[11] W. Xu, H. Zhou, N. Cheng, F. Lyu, W. Shi, J. Chen, and X. Shen, "Internet of vehicles in big data era," *IEEE/CAA J. Automatica Sinica*, vol. 5, no. 1, pp. 19–35, Jan. 2018.

[12] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4424–4434.

[13] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, "ENORM: A framework for edge node resource management," *IEEE Trans. Services Comput.*, vol. 13, no. 6, pp. 1086–1099, Nov./Dec. 2020.

[14] 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.*, 2014, pp. 19–27.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[16] 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.

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

[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.*, 2020, pp. 1698–1707.

[19] Y. Chen, Y. Ning, and H. Rangwala, "Asynchronous online federated learning for edge devices," 2019, *arXiv:1911.02134*.

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

[21] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, "Differentially private asynchronous federated learning for mobile edge computing in urban informatics," *IEEE Trans. Ind. Informat.*, vol. 16, no. 3, pp. 2134–2143, Mar. 2020.

[22] W. Wu, L. He, W. Lin, R. Mao, C. Maple, and S. A. Jarvis, "SAFA: a semi-asynchronous protocol for fast federated learning with low overhead," *IEEE Trans. Comput.*, vol. 70, no. 5, pp. 655–668, May 2021.

[23] H. Lim, D. G. Andersen, and M. Kaminsky, "3LC: Lightweight and effective traffic compression for distributed machine learning," 2018, *arXiv:1802.07389*.

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

[25] L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer, 2012, pp. 421–436.

[26] S. Zheng *et al.*, "Asynchronous stochastic gradient descent with delay compensation," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 4120–4129.

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

[28] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural Networks for Perception*. Boston, MA, USA: Acad. Press, 1992, pp. 65–93.

[29] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[30] L. Jacob, J.-P. Vert, and F. R. Bach, "Clustered multi-task learning: A convex formulation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 745–752.

[31] Y. Zhang and D.-Y. Yeung, "A convex formulation for learning task relationships in multi-task learning," 2012, *arXiv:1203.3536*.

[32] A. R. Gonçalves, F. J. Von Zuben, and A. Banerjee, "Multi-task sparse structure learning with gaussian copula models," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1205–1234, 2016.

[33] M. A. Birck, U. B. Correa, P. Ballester, V. O. Andersson, and R. M. Araujo, "Multi-task reinforcement learning: An hybrid A3C domain approach," *Eniac*, 2017.

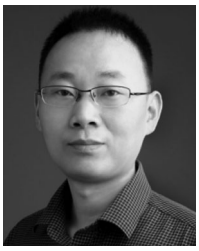
[34] H. Wang, D. Niu, and B. Li, "Distributed machine learning with a serverless architecture," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1288–1296.

[35] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.

[36] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009. [Online]. Available: https://www.cs.toronto.edu/kriz/learning-features-2009-TR.pdf

[37] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," 2018, *arXiv:1811.04017*.
[38] S. Wang *et al.*, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 63–71.
[39] R. Zhang, F. R. Yu, J. Liu, T. Huang, and Y. Liu, "Deep reinforcement learning (DRL)-based device-to-device (D2D) caching with blockchain and mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 10, pp. 6469–6485, Oct. 2020.
[40] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 883–893, Jun. 2020.
[41] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.
[42] H. B. McMahan *et al.*, "Communication-efficient learning of deep networks from decentralized data," 2016, *arXiv:1602.05629*.
[43] Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 495–503.
[44] Q. Ho, "More effective distributed ML via a stale synchronous parallel parameter server," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 1223–1231.
[45] 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*, 2016, pp. 171–180.
[46] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," 2018, *arXiv:1806.00582*.
[47] S. Shi, X. Chu, and B. Li, "MG-WFBP: Efficient data communication for distributed synchronous SGD algorithms," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 172–180.
[48] J. Konečnỳ, B. McMahan, and D. Ramage, "Federated optimization: Distributed optimization beyond the datacenter," 2015, *arXiv:1511.03575*.

Jianchun Liu (Student Member, IEEE) received the BS degree from the North China Electric Power University in 2017. He is currently working toward the PhD degree with the School of Data Science, University of Science and Technology of China (USTC). His main research interests include software-defined networks, network function virtualization, edge computing, and federated learning.

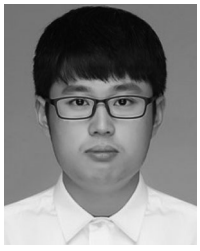


Hongli Xu (Member, IEEE) received the BS and PhD degrees in computer science from the University of Science and Technology of China (USTC), China, in 2002 and 2007, respectively. He is currently a professor with the School of Computer Science and Technology, USTC. He has authored or coauthored more than 100 papers in famous journals and conferences, including the *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Parallel and Distributed Systems*, Infocom, and ICNP, and also has more than 30 patents. His main research interest is software-defined networks, edge computing, and Internet of Thing. He was the recipient of the Outstanding Youth Science Foundation of NSFC in 2018 and the Best Paper Award or the Best Paper Candidate in several famous conferences.

Lun Wang received the BS degree from the University of Electronic Science and Technology of China in 2019. He is currently working toward the MS degree with the School of Computer Science and Technology, University of Science and Technology of China. His research interests include mobile edge computing and federated learning.

Yang Xu received the BS degree from the Wuhan University of Technology in 2014 and the PhD 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.

Chen Qian (Senior Member, IEEE) received the BS degree in computer science from Nanjing University in 2006, the MPhil degree in computer science from The Hong Kong University of Science and Technology in 2008, and the PhD degree in computer science from The University of Texas at Austin in 2013. He is currently an assistant professor with the Department of Computer Engineering, University of California at Santa Cruz. He has authored more than 60 research papers in highly competitive conferences and journals. His research interests include computer networking, network security, and Internet of Things. He is a member of the ACM.



Jinyang Huang (Student Member, IEEE) received the BEng degree from Anhui University, China, in 2017. He is currently working toward the PhD degree with the School of Cyberspace Security, University of Science and Technology of China. His research interests lie human–computer interaction, wireless sensing, wireless communication, and machine learning.

He Huang (Member, IEEE) received the PhD degree from the School of Computer Science and Technology, University of Science and Technology of China, 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.