# Energy-Efficient Dynamic Asynchronous Federated Learning in Mobile Edge Computing Networks

Guozeng Xu[1], Xiuhua Li[1*], Hui Li[1], Qilin Fan[1], Xiaofei Wang[2], and Victor C. M. Leung[3, 4]
[1]School of Big Data & Software Engineering, Chongqing University, Chongqing, China
[2]TKLAN, College of Intelligence & Computing, Tianjin University, Tianjin, China
[3]College of Computer Science & Software Engineering, Shenzhen University, Shenzhen, China
[4]Department of Electrical & Computer Engineering, The University of British Columbia, Vancouver, Canada
[*]Corresponding author: Xiuhua Li
Email:x.guozeng@stu.cqu.edu.cn, {lixiuhua, h.li, fanqilin}@cqu.edu.cn, xiaofeiwang@tju.edu.cn, vleung@ieee.org

ICC 2023 - IEEE International Conference on Communications | 978-1-5386-7462-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICC45041.2023.10278887

***Abstract*—To break data silos and address the challenge of green communication, federated learning (FL) is widely used at network edges to train deep learning models in mobile edge computing (MEC) networks. However, many existing FL algorithms do not fully consider the dynamic environment, resulting in slower convergence of the model and larger training energy consumption. In this paper, we design a dynamic asynchronous federated learning (DAFL) model to improve the efficiency of FL in MEC networks. Specifically, we dynamically choose a certain number of mobile devices (MDs) by their arrival order to participate in the global aggregation at each epoch. Meanwhile, we analyze the energy consumption model of local update and upload update, and formulate the problem as a dynamic sequential decision problem to minimize the energy consumption, which is NP-hard. To address it, we propose an energy-efficient algorithm based on deep reinforcement learning named DDAFL, to intelligently determine the number of MDs participating in global aggregation according to the state of MEC networks at each epoch. Compared with baseline schemes, the proposed algorithm can significantly reduce energy consumption and accelerate model convergence.**

## I. Introduction

Recently, with many novel technologies (e.g., computer vision, natural language processing, and recommended system) emerging, artificial intelligence (AI) has entered the vigorous development period[1]. However, due to the challenges of data silos and green communication, it is difficult to deal with data distributed across all mobile devices (MDs) by traditionally training AI models in a centralized manner or on MDs individually[2]. Mobile edge computing (MEC) is an emerging technology with potential, which can process data locally or offloads computing task to network edges (e.g., base stations (BSs))[3]. By deploying the FL framework in MEC networks, it is efficient to obtain a converged model based on data distributed across all MDs in a decentralized manner[4].

FL was proposed to build a distributed machine learning (ML) model based on multi-party data[5], [6]. Typically, the FL system contains at least one parameter server (PS), and many workers (i.e., MDs). Each worker and PS are responsible for updating the model locally and aggregating the model, respectively. Specifically, each worker trains the model locally (i.e., local update), and then uploads the model to PS, which aggregates the received models weighted based on some strategy (i.e., global update) and then sends aggregated model to each worker. The content transmitted between each worker and PS only consists of model parameters and no specific data, so that the model can be trained in a decentralized manner, which greatly improves communication efficiency and protects the privacy of MDs[7]. FL was adopted by many industrial practitioners, including Google's Gboard mobile keyboard, Apple's QuickType keyboard, and so on. However, MDs with different computing resources in the MEC networks generally have great uncertainty, such as offline, power outage, etc[8], [9]. At the same time, the amounts of data in MDs are unevenly distributed and will change with time. To improve the efficiency and stability of FL in MEC networks, it is necessary to consider resource constraints and the dynamic network environment[10], [11]. Asynchronous FL is suitable for dynamic network environments and can significantly improve the efficiency of federated aggregation in MEC networks.

Many researchers have studied different asynchronous aggregation methods in FL. McMahan *et al.*[12] presented a federated averaging (FedAvg) algorithm, in which PS would wait for all MDs to upload their models before a global update at each epoch. It is inefficient to use FedAvg in MEC networks due to the time required for different MDs to perform a round of local updates. To obtain a global model with a smaller loss under the premise of certain resource constraints, Wang *et al.*[13] proposed an algorithm to adaptively control the frequency of global updates according to the current network environment. For improving flexibility and scalability, Xie *et al.* [14] proposed a new asynchronous federated optimization algorithm (FedAsync), where PS would perform a global update once receiving one model from MD and not wait for other MDs. However, the convergence of the model will be unstable if there is only one MD involved in the global update at one epoch, which means that more training is required for the model to converge and more energy consumption. Ma *et al.*[15] presented a semi-asynchronous federated learning mechanism to shorten overall training time, which has good performance in multi-task dynamic application scenarios. These works mainly optimize the training effect and training time of federated learning and have relatively little research on energy consumption in the training process[16].

978-1-5386-7462-8/23/$31.00 ©2023 IEEE

In this paper, we are motivated to design a DAFL model based on FedAsync to adapt to the dynamic environment of MEC networks. Different from FedAsync, DAFL can efficiently solve the difficulty in model convergence. In addition, we proposed an energy-efficient DRL-based algorithm DDAFL to determine the number of MDs participating in global aggregation to minimize energy consumption. Particularly, the main contributions of this paper are as follows:

- We design a DAFL model in MEC networks and analyze the energy consumption of each stage of local update and upload update, respectively, for effectively improving the efficiency of FL, toward future green MEC networks.
- We propose an energy-efficient DRL-based algorithm DDAFL to intelligently determine the number of MDs for global update according to the state of MEC networks at each epoch to significantly reduce energy consumption.
- We evaluate the proposed algorithm through extensive experiments on a typical model and dataset. Simulation results demonstrate that the proposed algorithm can significantly reduce energy consumption and accelerate model convergence for FL in MEC networks.

The remainder of this paper is organized as follows. Section II introduces the system model and formulates the optimization problem. We present our proposed algorithm in Section III and evaluate the performance of our proposed algorithm in Section IV. Finally, Section V concludes this paper.

## II. System Model and Problem Formulation

### A. Federated Learning in MEC Networks

As shown in the Fig. 1, we consider a MEC network that consists of $N$ MDs (denoted by $\mathcal{N} = \{1, 2, \ldots, N\}$) and a BS. Each BS is equipped with an edge server (ES) that has certain computing capacities to perform the global model aggregation and broadcast for each MD. Each MD has limited computing/communication resources in different areas and can generate a large amount of raw data. However, due to the requirement of green communication and privacy issues, these data cannot be uploaded to the ES for model training. The ES and MDs are connected through cellular links and the MDs can upload local model parameters to the ES for aggregation or get the latest model parameters.

FL provides a decentralized training strategy to reasonably process multi-party data distributed and make the model have better generalization. Specifically, each MD trains local data and uploads the model to the ES, then the ES will perform the global model aggregation and broadcast the updated model to each MD. ML updates model parameters $w$ by training on sample data[13]. In order to make the model have higher accuracy, a loss function $f(w, x_j, y_j)$, abbreviated as $f_j(w)$, is defined for each sample data $j$ in the training process, where $x_j$ and $y_j$ are respectively feature vector and label of the sample data $j$. It is assumed that data on $N$ MDs form

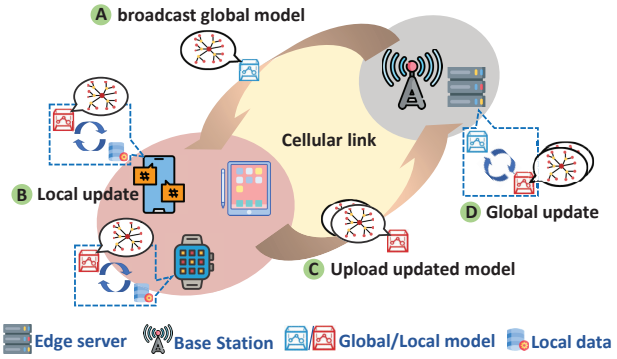



Fig. 1. Illustration of federated learning in MEC networks.

local datasets $\mathcal{D} = \{D_1, D_2, \ldots, D_i, \ldots, D_N\}$. Hence, the loss function value for the dataset $D_i$ on the MD $i$ is

$$F_i(w) = \frac{1}{|D_i|} \sum_{j \in D_i} f_j(w), \forall i \in \mathcal{N}, \tag{1}$$

where $|\cdot|$ denotes the size of the set. The global loss function value for all datasets is:

$$F(w) = \frac{\sum_{i \in \mathcal{N}} |D_i| F_i(w)}{\sum_{i \in \mathcal{N}} |D_i|}. \tag{2}$$

### B. Dynamic Asynchronous Federated Learning Model

When deploying the FL model in MEC networks, in order to improve the accuracy of the model quickly and control energy consumption, we should consider the difference in computing resources of different MDs. MDs with strong computing power and low energy consumption should participate in global updates more frequently. In addition, there are other unstable factors (e.g., network congestion and equipment offline) in MEC networks, we need to dynamically adjust the strategy for ES performing global updates according to the state of the network. In this section, we design a dynamic asynchronous federated learning (DAFL) model based on FedAsync.

Firstly, Each MD $i$ updates local model parameters $w_i(k)$ at local training epoch $k \in \{1, 2, \ldots\}$, the specific update rules are as follows:

$$w_i(k) = w_i(k-1) - \alpha \nabla F_i(w_i(k-1)), \forall i \in \mathcal{N}, \tag{3}$$

where $\alpha$ is the learning rate. Then, the MD $i$ will upload the updated model to the ES. Let variable $n_t \in \{1, 2, \ldots, N\}$ denotes the number of MDs participating in aggregation at global update epoch $t \in \{1, 2, \ldots, T\}$, where $T$ is the total number of global updates until the model fully converges. Thus the ES will perform a weighted average summation of the first $n_t$ received model parameters at epochs $t$ according to the size of the dataset $D_i$ to update the global model parameters by

$$\tilde{w}(t) = \frac{\sum_{i \in \mathcal{N}} x_{t,i} |D_i| w_i(k_{t,i})}{\sum_{i \in \mathcal{N}} x_{t,i} |D_i|}, \tag{4}$$

where $k_{t,i}$ denotes the index of local update epoch of MD $i$ at global update epoch $t$ and $x_{t,i} \in \{0, 1\}$ is a binary variable to denote whether MD $i$ participates in the global update or not at epoch $t$, it follows $\sum_{i \in \mathcal{N}} x_{t,i} \geq n_t, \forall i \in \mathcal{N}$. Lastly, the

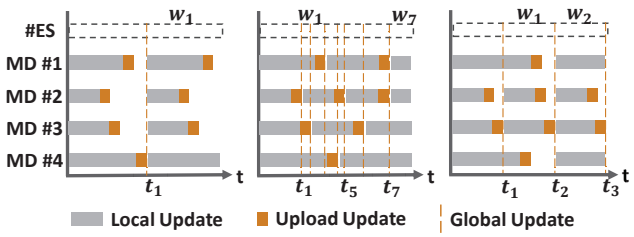



Fig. 2. Illustration of synchronous FL, asynchronous FL, and DAFL.

ES will broadcast the updated model to all MDs, which will do model update by $w_i(k_{t,i}+1)=\tilde{w}(t), i\in\mathcal{N}$. The global loss function value $F(\tilde{w}(t))$ at epochs $t$ can be obtained as

$$F(\tilde{w}(t))=\frac{\sum_{i\in\mathcal{N}} x_{t,i}|D_i|F_i(\tilde{w}(t))}{\sum_{i\in\mathcal{N}} x_{t,i}|D_i|}. \tag{5}$$

To better explain DAFL, Fig. 2 describes synchronous FL (left subplot), asynchronous FL (middle subplot), and DAFL (right subplot), respectively. We assume that there are 4 MDs in the considered MEC network. In synchronous FL, the ES must receive the updated models from all 4 MDs for the global update. The ES will perform a global update once receiving updated models from any MDs in asynchronous FL. In DAFL, the ES will perform the global updates when receiving updated models from $n_t$ MDs, where the value of $n_t$ varies with epoch to adapt to dynamic MEC networks. For example, there are 2 MDs and 4 MDs participating in the global update at $t_1$-th and $t_2$-th epoch, respectively.

### C. Energy Consumption Model

According to the logic of DAFL, we can divide its energy consumption into two aspects: local energy consumption and upload energy consumption.

*1) Local Energy Consumption:* the computation capability of MD $i$ can be measured by the number of CPU cycles per second, denoted as $f_i$ and it takes $C$ CPU cycles to compute one sample data. With the reference to[17], the time required for one local update epoch on MD $i$ with batch size $\varpi_i$ is

$$H_i^{cmp}=\frac{C\varpi_i}{f_i}, \forall i\in\mathcal{N}. \tag{6}$$

Similarly, the energy consumption for one local update epoch on MD $i$ is calculated by

$$E_i^{cmp}=\kappa C\varpi_i f_i^2, \tag{7}$$

where $\kappa$ is the effective switched capacitance that depends on the chip architecture.

*2) Upload Energy Consumption:* the MDs need to upload the model parameters to the ES. We consider upload energy consumption from the MDs to the ES. According to the Shannon-Hartley formula, the achievable rate of MD $i$ is

$$r_i=b_i\log_2(1+\frac{g_i p_i}{N_0 b_i}), \tag{8}$$

where $b_i$ is the bandwidth allocated to MD $i$, $g_i$ is the channel gain between MD $i$ and ES, $p_i$ is the average transmit power of MD $i$, and $N_0$ is the power spectral density of the Gaussian noise. Denote the size uploaded model as $s$, thus, we can obtain the transmission time $H_i^{com}$ for the global update by

$$H_i^{com}=\frac{s}{r_i}=\frac{s}{b_i\log_2(1+\frac{g_i p_i}{N_0 b_i})}. \tag{9}$$

To transmit the model parameters once, the energy consumption of MD $i$ will be

$$E_i^{com}=H_i^{com}p_i=\frac{sp_i}{b_i\log_2(1+\frac{g_i p_i}{N_0 b_i})}. \tag{10}$$

After that, the ES will broadcast the updated model parameters to each MD via the downlink. Since both the transmit power of the ES and the broadcast bandwidth are large enough, the consumption it takes to broadcast is negligible. Therefore, the energy consumption of all MDs at epoch $t$ is calculated as

$$E_t=\sum_{i\in\mathcal{N}} x_{t,i}E_i^{cmp}+x_{t,i}E_i^{com}. \tag{11}$$

The time spent at epoch $t$ to perform local updates and upload the updated models can be obtained by

$$H_t=\max_{i\in\mathcal{N}}\left(x_{t,i}H_i^{cmp}+x_{t,i}H_i^{com}\right). \tag{12}$$

### D. Problem Formulation

In MEC networks, the energy of MDs are limited and precious, thus it is necessary to make the energy consumption of the training process as low as possible. Our goal is to minimize the energy consumption under the given constraints by determining a reasonable sequence of $n_t$. Note that we dynamically choose a certain number of MDs by their arrival order to participate in the global aggregation at each epoch. It means when we determine the decision $n_t$, the variable $x_{t,i}$ of whether MD $i$ participates in the global update or not at epoch $t$ is also determined. Hence, the corresponding optimization problem can be formulated as

$$\min_{\{n_t\}}\sum_{t=1}^{T}E_t, \tag{13a}$$

$$s.t. F(\tilde{w}(T))\le\mathcal{F}, \tag{13b}$$

$$\sum\nolimits_{i\in\mathcal{N}} x_{t,i}\ge n_t, \tag{13c}$$

$$n_t\in\{1,2,\ldots,N\}, \forall t\in\{1,2,\ldots,T\}. \tag{13d}$$

The goal of the above optimization problem is to minimize energy consumption. The constraint in (13b) describes that the final loss function value of the model must reach the expected convergence value $\mathcal{F}$. The constraint in (13c) requires ES to receive at least $n_t$ models for a global update. The above optimization problem is a dynamic sequential decision problem, which is NP-hard. It is challenging to solve it due to the extremely large state space and dynamically changing network environment. Therefore, we propose an energy-efficient DRL-based algorithm for learning a competitive suboptimal sequential decision in the considered MEC network.

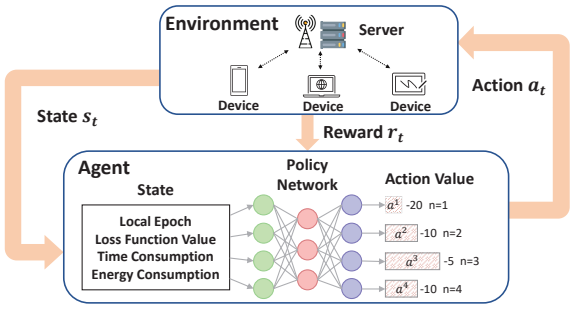



Fig. 3. Illustration of DRL architecture.

## III. Proposed Algorithm Design

### A. DRL Formulation

To minimize energy consumption during the training process, we use DRL to determine the value of $n_t$. Fig. 3 shows the architecture of the DRL, in which the agent perceives a state $s_t$ from the environment, and then outputs the value of each action in the corresponding state. According to this strategy $\pi$, the agent selects an action $a_t$ with the greatest value to execute and receives a reward $r_t$. The actual value of action $a_t$ in state $s_t$ is $q_\pi(s_t, a_t) = \sum_{d=0}^{T-t} \gamma^d r_{t+d}$, which contains immediate reward and subsequent reward with a discount factor $\gamma \in (0, 1]$. The agent's goal is to take an action $a_t$ in each state $s_t$ to maximize the reward $r_t$. The detailed description of the proposed DRL algorithm is as follows.

*1) System State:* The vector $s_t = (t, E_t, H_t, \tilde{H}_t, F_t, \Delta F_t)$ denotes the state of epoch $t$. $E_t$ and $H_t$ are the calculated energy and time consumption of training at epoch $t$, respectively. $\tilde{H}_t$ and $F_t$ denote the actual time consumption and loss function at epoch $t$, respectively. $\Delta F_t$ is the difference between loss value after epoch $t-1$ and loss value after epoch $t$, i.e., $\Delta F_t = F_t - F_{t-1}$.

*2) Action Space:* The action is defined by a positive integer $a_t = n_t \in \{1, 2, \ldots, N\}$. At each epoch $t$, the DRL agent selects an action according to the policy based on the current state to perform the global update.

*3) Policy:* Policy $\pi$ is the mapping of state space to action space, i.e., $a_t = \pi(s_t)$. We use a neural network to represent the policy $\pi(a|s)$ which is a probability distribution over the entire action space.

*4) Reward:* When an action is performed, the agent gets a reward $r_t$ considered as a combination of the loss function value and the energy consumed at epoch $t$, i.e.,

$$r_t = -\Phi^{\frac{E_t - \overline{E}_{t-1}}{\overline{E}_{t-1}}} + \frac{\Delta F_t}{\mathcal{F}}, t \in \{1, 2, \ldots, T\}, \qquad (14)$$

where $\Phi$ is a positive constant to enables $r_t$ to increase exponentially with decreasing energy consumption $E_t$. $\overline{E}_{t-1}$ is the moving average energy consumption and follows $\overline{E}_t = \lambda E_t + (1-\lambda)\overline{E}_{t-1}, \lambda \in (0, 1)$. At epoch $t$, the less energy is consumed, the closer the loss function value is to convergence, and the more reward the agent gets.

**Algorithm 1** Energy-Efficient DRL-based DAFL (DDAFL)

1: **Input: dataset, all MDs and BS information.**
2: Initialize DQN parameters $\theta$.
3: **for** each episode $i \in \{1, 2, \ldots, M\}$ **do**
4: **—[Processing on the ES].**
5: Initialize environment parameters.
6: **for** $t \in \{1, 2, \ldots, T\}$ **do**
7: Select a random action $a_t$ with probability $\epsilon$ otherwise select $a_t = \max_a \mathcal{Q}(s_t, a; \theta)$.
8: **while** number of received updated models $< a_t$ **do**
9: Waiting for updated models from MDs.
10: **end while**
11: Perform the global update according to (4).
12: Compute the global loss function, energy and time consumption by (5) (11) (12).
13: Broadcast the updated model to all MDs.
14: Observe reward $r_t$ with (14) and update state $s_{t+1}$.
15: Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay memory.
16: Sample random minibatch $\mathcal{G}$ of transitions $(s_j, a_j, r_j, s_{j+1})$ from replay memory.
17: Perform a gradient descent step on $Loss(\theta)$ by (17).
18: **end for**
19: **—[Processing on each MD].**
20: Receive the globally updated model from ES.
21: **for** each local update **do**
22: Perform the local update by stochastic gradient descent.
23: **end for**
24: Upload the updated model to the ES.
25: **end for**
26: **Output: the decision of $n_t$ and the final FL model.**

### B. Training Method of DDAFL

We train DRL the agent using DQN, in which DNNs are designed to estimate the value function approximation $\mathcal{Q}(s_t, a; \theta)$, where $\theta$ is the parameter of DNNs. In fact, DNNs can be seen as a policy $\pi(s_t|a; \theta) = \max_a \mathcal{Q}(s_t, a; \theta)$. The policy $\pi$ is continuously updated so that the agent can take more valuable actions $a_t$ in state $s_t$.

The DDAFL algorithm is described in Algorithm 1. DDAFL algorithm deployed on the ES initializes DQN parameter $\theta$ (Line 1). Agent initializes the environment parameters (e.g., task model $w(0)$) for each episode (Line 4). In each epoch, the action is selected by policy $\pi$ with a probability according to $\epsilon-$greedy strategy, otherwise, the action is randomly selected. The global aggregation starts when the agent receives the corresponding number of models and broadcasts the updated model to all MDs (Lines 7-13). After updating the reward and next state, the current state, selected action, reward, and the next state are stored in replay memory. Then agent randomly selects minibatch of sample $(s_j, a_j, r_j, s_{j+1})$ from replay memory for training DNNs according to loss function as follows:

$$Loss(\theta) = \mathbb{E}\left[(y_j - \mathcal{Q}(s_j, a; \theta))^2\right], \qquad (15)$$

where $y_j$ is the target value function including immediate reward and subsequent reward as follows:

$$y_j = r_j + \gamma \max_{a'} \mathcal{Q}\left(s_{j+1}, a'; \theta\right), \tag{16}$$

where $a'$ is the action space of state $s_{j+1}$. In this way, we can update the model parameter $\theta$ of DNNs by stochastic gradient descent as:

$$\nabla_\theta Loss\left(\theta\right) = \mathbb{E}\Big[\Big(r_j + \gamma \max_{a'} \mathcal{Q}\left(s_{j+1}, a'; \theta\right) - \mathcal{Q}\left(s_j, a; \theta\right)\Big) \nabla_\theta \mathcal{Q}\left(s_j, a; \theta\right)\Big]. \tag{17}$$

For MDs, they need to wait to receive a globally updated model from ES, then perform local updates and upload the updated models (Lines 19-23).

## IV. Evaluation Results

### A. Setup

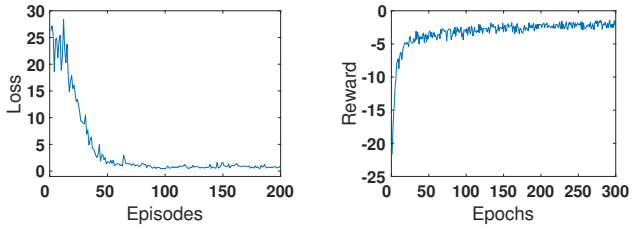
To evaluate the proposed dynamic asynchronous federated learning scheme, we design multi-process concurrent programs based on TensorFlow conducted on the server (CPU:Intel(R) Xeon(R) Silver 4214, GPU:NVIDIA GeForce RTX 3090) to simulate the training process. The child process simulates MDs and ES, which can communicate with each other and have independent datasets for model training.

*1) Evaluation Settings:* In simulations, we mainly consider the energy consumption of MDs. Similar to[18], the related parameters are set as follows: transmission bandwidth $b_i$ and computing ability $f_i$ are randomly distributed in $[5, 10]$ MHz and $[2, 8]$ GHz, respectively. For local training, computing one sample data needs $C = 300$ CPU cycles and the size of mode $s = 100$ KB. The Gaussian noise power $N_0$ is set as $2 \times 10^{-13}$ W. Besides, we set the value of parameters $\alpha = 0.001, \gamma = 0.99, \Phi = 1.5, \lambda = 0.2$ for better convergence of the algorithm.

*2) Dataset and Model:* We evaluate the proposed algorithm with the typical dataset, i.e., Fashion-MNIST (referred to as FMNIST) and the logistic regression (referred to as LR) model for image classification. The FMNIST dataset contains 70,000 images of handwritten digits 0-9 (60,000 for training and 10,000 for testing). The hidden layer of the LR model contains two fully connected layers, each of which contains 128 units.

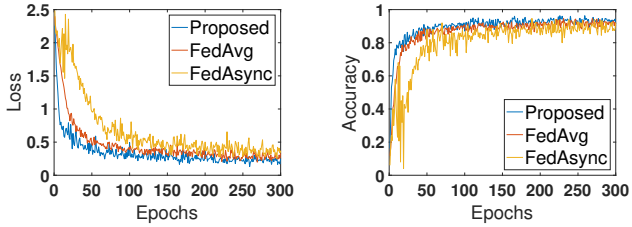
*3) Performance Metrics:* We use the following metrics to measure the efficiency and performance of the proposed algorithm. (1) **Loss function** shows the difference between the prediction and the actual data and reflects the quality of the model. (2) **Reward** is the return from the environment after the agent performs an action. (3) **Accuracy** is the ratio of the number of samples correctly classified by the classification model to the total number of samples. (4) **Energy consumption** refers to the energy consumed by all MDs to complete the federated model training.

*4) Baselines:* In order to compare the superiority of the proposed algorithm, there are two baselines for comparison as follows: (1) **FedAvg**[12], the ES does not perform global update until it receives the models uploaded by all MDs. (2) **FedAsync**[14], the ES performs a global update once it receives the models uploaded by any MDs.




(a) Loss vs. Number of Episodes (b) Reward vs. Number of Epochs

Fig. 4. Training Convergence of DRL Agent.




(a) Loss vs. Number of Epochs (b) Accuracy vs. Number of Epochs

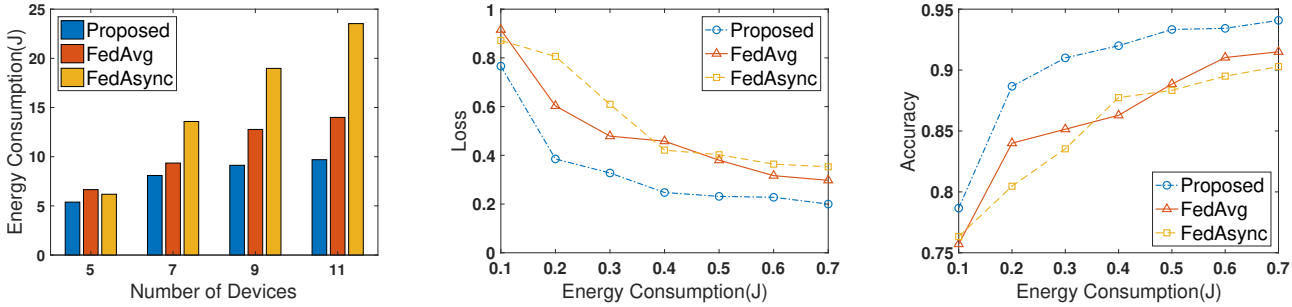
Fig. 5. Training Performance of Model.

### B. Results and Analysis

*1) Training the DRL agent:* We first consider the convergence performance of training the agent, which is conducted with 7 MDs for 200 episodes. Fig. 4 shows the loss function and reward of the training process, respectively. The loss at the beginning of the training is large but it decreases rapidly after several rounds of training in Fig. 4(a). This is because the agent has learned the experience from the training process. After about 50 episodes, the loss starts to converge, which indicates that the agent adapts to the FL environment. Fig. 4(b) records the change of reward in each episode. The reward increases rapidly with epochs and tends to be stable when reaching 150 episodes. That's because the agent learns a better policy to make it choose the most valuable action in each state. Therefore, the proposed algorithm has excellent convergence performance for the FL in the MEC networks.

*2) Model training:* Fig. 5 compares the training performance of the model in terms of loss and accuracy on FedAvg, FedAsync, and the proposed algorithm, respectively. DAFL can achieve a training performance similar to FedAvg. Meanwhile, the loss converges even faster than the FedAvg, and the accuracy rate also rises faster. It means DAFL can achieve better training performance than FedAsync. Besides, the loss converges faster than FedAsync, the accuracy rate rises faster, and the stability is better. Specifically, when the LR model training reaches 100 epochs, the accuracy of the model by DAFL reaches 90.8%, while FedAvg and FedAsync are 87% and 84%, respectively. We can conclude that the proposed DAFL can accelerate the convergence of FL model training.

Fig. 6 shows the energy consumption of the model training process. Fig. 6(a) shows the energy consumption with different numbers (e.g., 5,7,9, and 11) of MDs for considered schemes. In comparison, DAFL needs less energy consumption than FedAvg and FedAsync. Specifically, when the number of MDs is set as 7 in the training, the energy consumption of DAFL




(a) Energy Consumption vs. Number of MDs (b) Loss with Energy Consumption Constraint (c) Accuracy with Energy Consumption Constraint

Fig. 6. Energy Consumption of Model Training Process.

is only 8.09$J$, while FedAvg and FedAsync are 9.35$J$ and 13.58$J$, respectively. This shows that DAFL reduces energy consumption by 13.5% and 40.4% compared with FedAvg and FedAsync, respectively. With the increase in the number of MDs, the rate of increase of energy consumption by DAFL is much smaller than FedAvg and FedAsync. This means that DAFL can reduce much more energy consumption when a large number of MDs participate in training. As shown in Fig. 6(b) and 6(c), the model can achieve smaller loss and higher accuracy with a certain energy consumption limit. Specifically, the accuracy can reach 94.1%, while FedAvg and FedAsync are 91.5% and 90.3%, respectively. We can conclude that the proposed DAFL can greatly reduce energy consumption, especially in scenarios with a large number of MDs.

## V. Conclusion

In this paper, we have designed a DAFL model in MEC networks for effectively improving the efficiency of FL. Specifically, we have analyzed the energy consumption in terms of the local update and upload update and formulated the optimization problem as a dynamic sequential decision problem to minimize the energy consumption. To address the formulated problem, we have further proposed an energy-efficient DRL-based algorithm DDAFL to intelligently determine the number of MDs participating in global aggregation according to the state of MEC networks at each epoch. Simulation results have demonstrated that the proposed algorithm can significantly reduce energy consumption and accelerate model convergence for FL in MEC networks compared with the considered baseline schemes. In the future, we will further exploit the hierarchical aggregation issue in MEC networks.

## Acknowledgment

This work is supported in part by National Key R & D Program of China (Grants No. 2022YFE0125400), National NSFC (Grants No. 61902044, 62072060, 62072332, and 62102053), Chongqing Research Program of Basic Research and Frontier Technology (Grant No. cstc2022ycjh-bgzxm0058), Key Research Program of Chongqing Science & Technology Commission (Grants No. cstc2021jscx-dxwtBX0019), Haihe Lab of ITAI (Grant No. 22HHXCJC00002), and Guangdong Pearl River Talent Recruitment Program (Grants No. 2019ZT08X603 and 2019JC01X235).

## References

[1] A. P. James, "The Why, What, and How of Artificial General Intelligence Chip Development," *IEEE Trans. Cogn. Develop. Syst.*, vol. 14, no. 2, pp. 333-347, Jun. 2022.
[2] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo and J. Zhang, "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing," *Proc. IEEE Inst. Electr. Electron. Eng.*, vol. 107, no. 8, pp. 1738-1762, Aug. 2019.
[3] X. Li, X. Wang, P. -J. Wan, Z. Han and V. C. M. Leung, "Hierarchical Edge Caching in Device-to-Device Aided Mobile Networks: Modeling, Optimization, and Design," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 8, pp. 1768-1785, Aug. 2018.
[4] T. Nishio and R. Yonetani, "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge," in *Proc. IEEE ICC*, Jul. 2019, pp. 1-7.
[5] Q. Yang, Y. Liu, et al, "Federated learning," *Synth. Lect. Artif. Intell. Mach. Learn.*, vol. 13, no. 3, pp. 1–207, Dec. 2019.
[6] Z. Du, C. Wu, T. Yoshinaga, K. -L. A. Yau, Y. Ji and J. Li, "Federated Learning for Vehicular Internet of Things: Recent Advances and Open Issues," *IEEE Open J. Comput. Soc.*, vol. 1, pp. 45-61, May 2020.
[7] V. Turina, Z. Zhang, F. Esposito and I. Matta, "Federated or Split? A Performance and Privacy Analysis of Hybrid Split and Federated Learning Architectures," in *Proc. IEEE ICCC*, Nov. 2021, pp. 250-260.
[8] Z. H. Nasralla, T. E. H. Elgorashi, A. Hammadi, M. O. I. Musa and J. M. H. Elmirghani, "Blackout Resilient Optical Core Network," *IEEE ACM Trans. Netw.*, vol. 30, no. 4, pp. 1795-1806, Aug. 2022.
[9] S. Tuli, S. Ilager, K. Ramamohanarao and R. Buyya, "Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments Using A3C Learning and Residual Recurrent Neural Networks," *IEEE Trans. Mobile Comput.*, vol. 21, no. 3, pp. 940-954, Mar. 2022.
[10] P. Tam, S. Math, et al, "Adaptive Resource Optimized Edge Federated Learning in Real-Time Image Sensing Classifications," *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 14, pp. 10929-10940, Oct. 2021.
[11] J. Liu *et al*., "Adaptive Asynchronous Federated Learning in Resource-Constrained Edge Computing," *IEEE Trans. Mobile Comput.*, Dec. 2020.
[12] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *AISTATS*, Feb. 2017, pp. 1273–1282.
[13] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, 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.
[14] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, Mar. 2019.
[15] Q. Ma, Y. Xu, H. Xu, Z. Jiang, L. Huang and H. Huang, "FedSA: A Semi-Asynchronous Federated Learning Mechanism in Heterogeneous Edge Computing," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3654-3672, Dec. 2021.
[16] P. Wang, W. Ma, H. Zhang, W. Sun and L. Xu, "A Dynamic Contribution Measurement and Incentive Mechanism for Energy-Efficient Federated Learning in 6G," in *Proc. IEEE ICC*, Sep. 2022, pp. 1-6.
[17] Y. Mao, J. Zhang and K. B. Letaief, "Dynamic Computation Offloading for Mobile-Edge Computing With Energy Harvesting Devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590-3605, Dec. 2016.
[18] P. Qin, Y. Fu, G. Tang, X. Zhao and S. Geng, "Learning Based Energy Efficient Task Offloading for Vehicular Collaborative Edge Computing," *IEEE Trans. Veh. Technol.*, vol. 71, no. 8, pp. 8398-8413, Aug. 2022.