Multi-Objective DAG Task Offloading in MEC Environment Based on Federated DQN With Automated Hyperparameter Optimization

Zhao Tong, *Senior Member, IEEE*, Jiaxin Deng, Jing Mei, Yuanyang Zhang, and Keqin Li, *Fellow, IEEE*

***Abstract*—The widespread adoption of the Internet of Things (IoT) has increased demand for task processing via mobile edge computing (MEC). In this study, we designed a directed acyclic graph (DAG) task offloading workflow in MEC. Traditional task offloading often does not simultaneously take into account task upload delay and task communication delay, failing to accurately reflect real-world issues. The constraints between task execution delay, upload delay and communication delay were introduced to model system response time and energy consumption for optimization. To satisfy task dependencies, the edge rank_u sorting (ERS) algorithm is used to generate specific offloading queues. A federated deep q-network (FDQN) algorithm addresses the offloading issue. It is different from the traditional approach of uploading task information data to the edge and facing data privacy risks. FDQN deploies the model locally and only collects model parameters for aggregation to update the local model. The algorithm improves the performance and stability of the model while protecting user privacy. To automatically tune hyperparameters for multiple devices, we used the tree of parzen estimators (TPE) algorithm, and named the whole process federated DQN with automated hyperparameter optimization (FDAHO). Experimental results show that FDAHO outperforms other algorithms in scenarios of different task number, task types, and user numbers, with consideration of benchmarks.**

***Index Terms*—FDAHO, mobile edge computing, multi-objective optimization, task offloading.**

I. Introduction

IN THE past few years, the rapid proliferation of the Internet of Things (IoT) devices has ushered in a transformative era, reshaping the digital landscape [1]. From sensors, actuators to smart home appliances, IoT devices have generated an unprecedented volume of data. It holds the potential to revolutionize various domains, including healthcare, industrial automation, and the development of smart cities. As IoT continues its widespread integration into our lives, addressing the critical challenges posed by the inherent constraints of computational resources and energy resources in these devices becomes imperative.

Mobile edge computing (MEC) has risen as a pivotal technology. It holds the key to unlocking the full potential of IoT. MEC leverages the proximity of edge servers to IoT devices, enabling offloading computationally demanding tasks from local node with finite resources to more potent edge nodes. This paradigm shift offers the potential of optimizing the performance and energy efficiency of IoT applications, playing crucial role in the evolving landscape of 5G and beyond [2]. Moreover, How to allocate resources reasonably and formulate offloading strategies in MEC has been gaining increasing attention.

In addition, complicating the task offloading process is the existence of directed acyclic graph (DAG) tasks. These tasks, which involve dependencies between sub-tasks, are common in IoT applications. For instance, in a video navigation app, the final result depends on the results of graphical rendering, video processing, and face detection [3]. In MEC offloading design, the interdependence between these tasks must be taken into account, unlike the traditional scenario where tasks can be offloaded independently.

However, traditional heuristic algorithms often rely on predefined rules and fixed strategies, which can be inflexible in dynamic and complex environments. They may struggle to adapt to changing conditions and fail to optimize long-term outcomes. In contrast, reinforcement learning (RL) is instrumental in task offloading and scheduling. It can be employed to formulate intelligent decision-making strategies, aiding systems in making optimal decisions for task offloading and scheduling according to current environmental and task requirements. Through learning and optimization, RL algorithms can gradually adapt their strategies to enhance overall system performance [4], [5]. In traditional RL, a single agent learns a policy by interacting with an environment and optimizing its actions based on rewards.

In the context of private MEC, users have private and sensitive data. Users cannot share their personal data with each other. Uploading this data to public servers for processing significantly increases the risk of privacy breaches. Centralized model training fails to meet this requirement. In a heterogeneous MEC scenario, various types of edge computing resources and devices are utilized to provide services. These devices have different

Received 24 July 2024; revised 23 September 2024; accepted 26 September 2024. Date of publication 11 October 2024; date of current version 30 December 2024. This work was supported in part by the Program of National Natural Science Foundation of China under Grant 62372172 and Grant 62072174, in part by Distinguished Youth Science Foundation of Hunan Province China under Grant 2023JJ10030, and in part by Graduate Research InnovationProgram of Hunan Province under Grant CX20240548. *(Corresponding authors: Zhao Tong; Jing Mei.)*

Zhao Tong, Jiaxin Deng, Jing Mei, and Yuanyang Zhang are with the College of Information Science and Engineering, Hunan Normal University in Changsha, Hunan 410081, China (e-mail: tongzhao@hunnu.edu.cn; baal@hunnu.edu.cn; jingmei1988@163.com; 202270294009@hunnu.edu.cn).

Keqin Li is with the College of Information Science and Engineering, National Supercomputing Center in Changsha, Hunan University, Hunan 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TSC.2024.3478841

computing capabilities and bandwidth, etc. The heterogeneity of devices can lead to suboptimal performance in distributed model training. Federated reinforcement learning (FRL) combines reinforcement learning with federated learning (FL), aiming to perform reinforcement learning tasks in a distributed environment while protecting data privacy [6]. FRL achieves this by training models locally and only sharing model updates, thereby safeguarding data privacy. It allows these distributed devices to collaborate, thereby improving model performance.

In the model training process, hyperparameters are adjusted to improve the model's performance. Hyperparameters differ from model parameters in that they are set before the training begins and have a significant impact on the model's performance. The tree-structured parzen estimator (TPE) is a Bayesian optimization method used to efficiently explore and optimize the hyperparameter space [7]. TPE constructs probabilistic models to guide the search process, making it more efficient compared to traditional methods.

Our research is dedicated to addressing the multifaceted challenges associated with task offloading in MEC networks. These challenges encompass:

- Task offloading: In MEC environments, effectively assigning tasks to diverse edge nodes with varying computational capabilities constitutes a complex optimization problem, especially when considering dependencies in the DAG task. This challenge is further compounded by the dynamics and diversity of IoT workloads. Traditional task offloading often does not simultaneously take into account task upload delay and task communication delay, failing to accurately reflect real-world issues.
- Privacy preservation: Most current multi-user DAG offloading algorithms upload data information to a single server for unified management and allocation, which may lead to task information leakage during transmission. Preserving the privacy of users' data is paramount during the offloading of tasks to edge servers. Traditional data-centric method often fall short when faced with sensitive information generated by IoT devices.
- Hyperparameter optimization: Hyperparameter optimization is an indispensable process in RL. It is also a challenge due to the vast number of possible hyperparameters combinations, high computational costs for model evaluation, and complex, non-linear interactions between hyperparameters. However, in previous offloading algorithms, hyperparameters were mostly adjusted manually. This method cannot guarantee the accuracy of hyperparameters and also incurs human labor and time costs in practical applications.

In response to these formidable challenges, our study introduces an innovative multi-objective DAG task offloading algorithm. The main contributions of this paper are summarized as follows:

- In this study, a comprehensive and customized simulation environment for MEC scenarios was constructed. The paper details the intricacies of the model framework, considering various factors such as heterogeneous devices, network dynamics, and resource limitations. This not only enhances the realism of the experiments but also lays the foundation for testing the effectiveness of different algorithms.
- This research introduces a hybrid method called federated deep q-network (FDQN). FDQN combines the knowledge aggregation capability of FL with the decision-making capacity of deep q-network (DQN). The cooperation of two algorithm empowers the task offloading decision-making process in MEC environments. It achieves an enhanced balance between local adaptation and global optimization, ultimately contributing to better task allocation strategies. Additionally, since the training process takes place locally, it effectively safeguards user privacy.
- The application of the TPE optimization algorithm for the automatic fine-tuning of hyperparameters in DQN represents a significant advancement. This method enhances the efficiency of hyperparameter optimization and boosts the performance of the DQN algorithm. The use of TPE mitigates the labor and time costs associated with traditional manual parameter adjustments. This enables more efficient use of computing resources.
- Real data sets are used in the experiments. The efficacy of FDQN with automated hyperparameter optimizatio (FDAHO) algorithm is confirmed through rigorous benchmarking with various cutting-edge algorithms on different benchmark sets. Notably, the algorithm demonstrates good results in optimizing both response time and energy consumption, outperforming traditional techniques in these critical indicators. The experimentation showcases the robustness and applicability of the algorithm across different scenarios and use cases.

The remainder of this paper is structured as follows. Section II is devoted to related work. Sections III and IV outlines the system model and problem formulation. In Section V, the FDAHO algorithm is presented. Numerical simulations are presented in Section VI, with the conclusion provided in Section VII.

II. Related Work

In this section, we first summarize the scheduling and offloading methods for tasks. Second, we discuss study under different architectural characteristic. Last, different DAG Offloading algorithms are summarized.

A. *Task Model*

Generally or typically, in the research on task offloading, tasks are categorized as either dependent tasks or independent tasks. In [8], [9], [10], [11], tasks can be split, with a portion offloaded to other devices. Baek et al. [8] used DRL to solve the task partial offloading problem in fog networks. The authors proposed a method for minimizing energy consumption in fog networks by optimizing the scheduling of CPU resources and computational offloading. Zhao et al. [9] proposed an intelligent partial offloading scheme called IGNITE for vehicle edge computing (VEC) that combines digital twin (DT) technology and DRL algorithm. The goal is to optimize task offloading decisions in VEC, considering large-scale data, low delay constraints, and dynamic network topologies. Khoobkar et al. [10]

discussed optimization of both latency and energy consumption through partial offloading in fog-cloud computing environments. They emphasized scalability issues in traditional game theory models and insufficient consideration of dynamic changes in fog environments. In [12], [13], [14], [15], tasks remain as a distinct whole and cannot be split. Tran et al. [12] discussed the combined problem of task offloading and resource allocation in MEC networks with multiple servers. They proposed a solution to maximize the users' task offloading benefit. Liu et al. [13] discussed task offloading in VEC, addressing intermittent connections and task processing interruptions caused by high mobility of vehicles. They proposed a task offloading solution utilizing mobility analysis of multi-hop vehicle computation resources. Tang et al. [14] solved the task offloading challenge in MEC, and introduced a DRL distributed algorithm to handle the uncertain load dynamics on device. In the above studies, independent tasks were used. In this paper, the type of task we use is a dependent DAG task.

B. The Scenario

Tasks with dependencies can be modeled as DAG tasks. In [16], [17], [18], the focus is on studying the scheduling problems of DAG tasks in the cloud. In order to better utilize the computing resources in the cloud environment, Li et al. [16] employed load balancing for task scheduling. This algorithm can estimate the runtime of tasks in the cloud based on the current environment state, while establishing a sorting model to minimize the overall system latency. Zhang et al. [17] proposed an algorithm named EPRD to address the task processing time of priority-constrained workflow applications in a cloud environment. The algorithm first sorts tasks and then schedules them based on the relative distance of virtual machines. In contrast to centralized cloud computing, MEC places computing resources at edge nodes closer to mobile devices, enabling more efficient support for mobile devices and applications [19]. In [20], [21], [22], research was conducted on issues related to DAG task offloading in the MEC environment. Sahni et al. [21] explored the optimization of multihop offloading for multiple DAG tasks in collaborative edge computing (One user and multiple MEC servers). Guo et al. [22] discussed the transmission power and computation model for mobile devices offloading tasks to wireless access points and cloud computing (Multiple users and single MEC server). It highlights the challenges of interference and low data rates when multiple devices offload computation simultaneously. However, the aforementioned studies did not address the challenge of DAG task offloading among multiple local devices and multiple edge servers.

C. Offloading Strategy

In [21], [23], heuristic algorithms is employed to solve the DAG task offloading problem. Liang et al. [23] discussed the problem of offloading computation-intensive tasks with precedence constraints in a MEC environment with multiple servers. The authors introduced task upload time and optimize maximum span by altering the offloading sequence and dynamically adjusting frequencies. Pan et al. [24] proposed an optimization approach for workflows in MEC, aiming to optimize the cost, energy consumption, and deadline constraints of MEC workflows. The method includes adaptive clustering, dynamic adjustments of crossover and mutation probabilities, and the utilization of historical information. However, these algorithms cannot completely address more complex scenarios of DAG task offloading. Such as dependency constraints between tasks and systems, device heterogeneity, etc. Su et al. [25] proposed a deep reinforcement learning-based algorithm for partial offloading of DAG applications in the Internet of Vehicles, aiming to optimize task execution time and energy consumption. Zhang et al. [26] proposed a graph neural network-Augmented deep reinforcement learning (GA-DRL) scheme for efficient DAG task scheduling in vehicular clouds, optimizing task completion time. However, these studies do not consider privacy protection mechanisms or automated hyperparameter tuning mechanisms.

In this paper, we address the multi-objective optimization problem of DAG task offloading in a scenario with multiple users and edge hosts. To address this problem, we employ federated deep reinforcement learning and automated hyperparameter optimization algorithms as a solution. Table I summarizes the comparisons between our work and the related works.

III. System Model

In this section, we provide an overview of the MEC system architecture model, DAG task model, computation model and communication model.

A. Architecture Model

In this MEC architecture, it consists of two layers of devices. The first layer is the IoT device layer, which consists of n IoT devices, namely local devices. The set of local devices is denoted by $\mathcal{N} = \{1, 2, \ldots, N\}$. The second layer is the edge server layer, which consists of a base station and multiple edge hosts mounted on the base station. The set of edge hosts is denoted by $\mathcal{M} = \{1, 2, \ldots, M\}$. Edge servers are placed on wireless access points and communicate with local devices through wireless channels. Each local device is equipped with a scheduler and a decision maker. The scheduler is used to sorting DAG tasks, ensuring that dependency relationships between tasks are satisfied. The decision maker determines which device should execute the tasks, ensuring the efficiency of task execution. The edge-IoT two-tier architecture model is shown in Fig. 1.

A summary of the main concepts in this model is presented in Table II.

B. Task Model

In this paper, the DAG task model is utilized, which consists of a set of nodes and directed edges connecting them. Each local device generates one or more DAG tasks, described by $G = \{V, E\}$. The vertex set V includes multiple dependent tasks denoted as $V = \{V_1, V_2, \ldots, V_S\}$. The set of edges E represents the communication relationships between tasks, indicating the

TABLE I
COMPARISON OF EXISTING WORKS

	Author	Task Model	Scenario	Optimization objective	Strategy	Task upload time	Task communication time	Hyperparameters setting
1	Baek et al. [8]	Independent	Fog computing	Multiple objective	DRL	Yes	No	Manual
2	Zhao et al. [9]	Independent	MEC	Multiple objective	DRL	Yes	No	Manual
3	Khoobkar et al.[10]	Independent	Fog computing	Multiple objective	Heuristic	Yes	No	No
4	Tran et al. [12]	Independent	MEC	Multiple objective	Heuristic	Yes	No	No
5	Liu et al. [13]	Independent	MEC	Single objective	Heuristic	Yes	No	No
6	Tang et al. [14]	Independent	MEC	Single objective	DRL	Yes	No	Manual
7	Li et al. [16]	Dependent	Cloud computing	Single objective	Heuristic	No	Yes	No
8	Zhang et al. [17]	Dependent	Cloud computing	Single objective	Heuristic	No	Yes	No
9	Sahni et al. [21]	Dependent	MEC	Single objective	Heuristic	Yes	No	No
10	Guo et al. [22]	Dependent	MEC	Multiple objective	Heuristic	No	Yes	No
11	Liang et al. [23]	Dependent	MEC	Single objective	Heuristic	Yes	Yes	No
12	Pan et al. [24]	Dependent	MEC	Multiple objective	Heuristic	No	Yes	No
13	Su et al. [25]	Dependent	MEC	Multiple objective	DRL	Yes	No	Manual
14	Zhang et al. [26]	Dependent	MEC	Single objective	DRL	No	Yes	Manual
14	**Our**	**Dependent**	**Private MEC**	**Multiple objective**	**FDAHO**	**Yes**	**Yes**	**Automic**

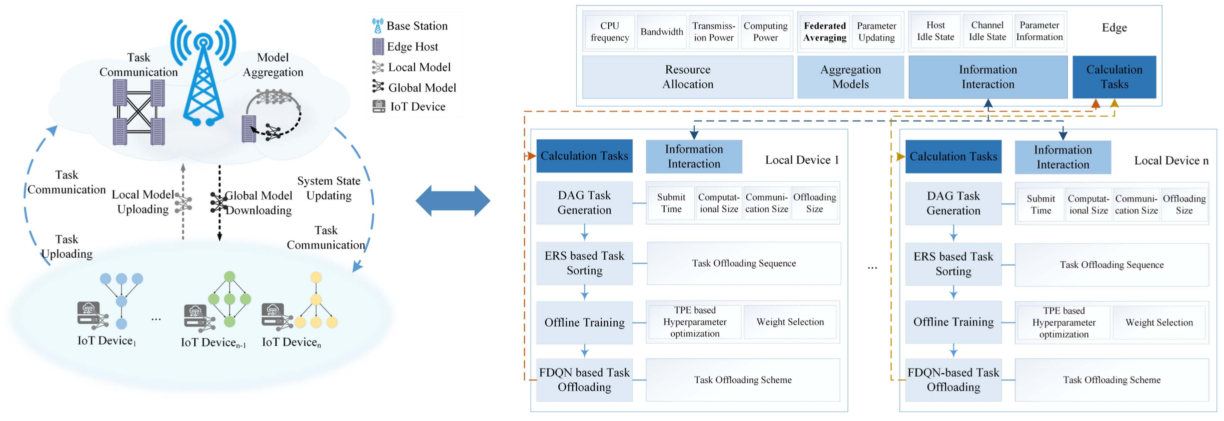


Fig. 1. The edge-IoT two-tier architecture.

TABLE II
SUMMARY OF NOTIONS IN THE MODEL

Notion	Definition
B_l	The allocated bandwidth between the local device and the base station.
$B_m^{m'}$	The allocated bandwidth between the edge host m and the edge host m'.
$C_{s,s'}$	The communication data size between tasks s and s'.
c_s	The computational size of task s.
d_s	The offloading size of task s.
$e_s^{\alpha_s}$	The execution energy consumption required for the device α_s to execute task s.
e_s^{l,α_s}	The upload energy consumption for offloading task s from the local device to the device α_s.
$e_{s,s'}^{\alpha_s,\alpha_{s'}}$	The data transmission energy consumption required between tasks when task s is on device α_s and task s' is on edge host $\alpha_{s'}$.
$t_s^{\alpha_s}$	The execution time required for the device α_s to execute task s.
t_s^{l,α_s}	The upload time for offloading task s from the local device to the device α_s.
$t_{s,s'}^{\alpha_s,\alpha_{s'}}$	The data transmission time required between tasks when task s is on device α_s and task s' is on device $\alpha_{s'}$.

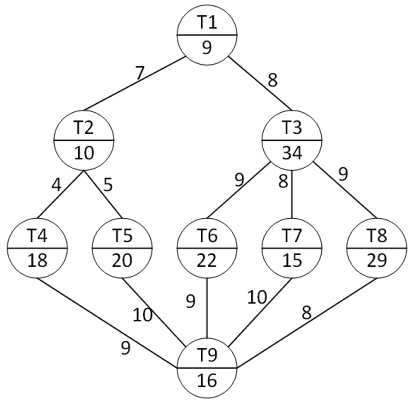


Fig. 2. The DAG task model.

dependency relationship. For instance, an edge $(s, s') \in E$ between task node s and s' signifies that task V_s must be completed before task $V_{s'}$. Tasks without predecessors are enter tasks, and those without successors are exit tasks. Each task is represented by a tuple $V_s =< c_s, d_s >$, where c_s is the computational size, and d_s is the offloading size [23]. The weight of the edge, denoted as $C_{s,s'}$, represents the communication data size from task $V_{s'}$ to V_s. The DAG task is illustrated in Fig. 2.

C. Computation Model

Tasks can be executed either on the local device or upload to an edge host for execution. When the task s is executed locally, its execution time is

$$t_s^l = \frac{c_s}{f_l}, \tag{1}$$

where c_s denotes the computational size of task s, and f_l denotes the computational capacity of the local device. When task s is executed locally, its execution energy consumption is

$$e_s^l = p_l^{comp} \times t_s^l, \tag{2}$$

where p_l^{comp} denotes the local execution power.

When the task s is offloaded to the edge host m for execution, its execution time is

$$t_s^m = \frac{c_s}{f_m}, \tag{3}$$

where f_m represents the computing capacity of edge host m. Similarly, its execution energy consumption is

$$e_s^m = p_m^{comp} \times t_s^m, \tag{4}$$

where p_m^{comp} represents the computational power of edge host m.

D. Communication Model

In the research, the communication model of single base station and multiple users is used. Device communication in this architecture utilizes full-duplex communication technology, allowing both parties to send and receive information at the same time. Local devices and base station communicate wirelessly. The transmission of data between devices consumes energy. In this model, communication between local devices and edge hosts is facilitated through frequency division multiple access (FDMA) technology. Communication between edge hosts through wired connection. Each local device is assigned a transmission bandwidth of B_l. Let H represent the channel gain from the local device to the base station. Let N_0 represent the interference encountered during communication. Therefore, following Shannon's formula, the transmission rate between the local device and edge host m is

$$r = B_l \times \log_2\left(1 + \frac{p_l^{tran} \times H}{N_0 \times B_l}\right), \tag{5}$$

where p^{tran} is the transmission power of the device and N_0 is the noise density [27]. If expanded to a multi-base station scenario, interference from signals of other base stations needs to be added. The interference power I from other base stations is expressed as $I = \sum_{j \neq i} P_j \cdot h_j$, where P_j is the transmission power of the j-th interfering base station, and h_j is the channel gain from that interfering base station to the receiving device. The transmission rate under multi-base station interference is given by $r = B_l \times \log_2 (1 + \frac{p_l^{tran} \times H}{N_0 \times B + I})$.

When Task s is uploaded and executed on the edge host m, the transmission time of its upload task is

$$t_s^{l,m} = \frac{d_s}{r_l}, \tag{6}$$

where d_s is the offloading size of the task and r_l denotes the transfer rate from the local device to the edge host m. In the meantime, the transmission energy consumption of its upload task is

$$e_s^{l,m} = p_l^{tran} \times t_s^{l,m}, \tag{7}$$

where p_l^{tran} is the transmission power of the local device.

If two dependent tasks are offloaded to different devices, there is communication time between them. Task s is the predecessor task of task s'. Task s is offloaded locally and another task s' is offloaded to edge host m. The transmission time for data communication between task s and s' is

$$t_{s,s'}^{l,m} = \frac{C_{s,s'}}{r_l}, \tag{8}$$

where $C_{s,s'}$ denotes the size of the communication data transmitted by task s to task s', and r_l is the transmission rate. The transmission energy consumption for data communication between tasks is

$$e_{s,s'}^{l,m} = p_l^{tran} \times t_{s,s'}^{l,m}. \tag{9}$$

When the predecessor task s is executed on the edge host m and the task s is executed locally, the computational formulas are the same as (8) and (9).

Task s is executed on the edge host m, while another task s' is executed on the edge host m'. The transmission energy consumption for data communication between tasks is

$$t_{s,s'}^{m,m'} = \frac{C_{s,s'}}{B_m^{m'}}, \tag{10}$$

where $B_m^{m'}$ is the transmission bandwidth from edge host m to edge host m'. The transmission energy consumption for data communication between task s and s' is

$$e_{s,s'}^{m,m'} = p_m^{tran} \times t_{s,s'}^{m,m'}, \tag{11}$$

where p_m^{tran} is the transmission power of edge host m.

IV. Problem Formulation

In this section, the optimization indicators are presented, and the optimization objective is elucidated.

A. Optimization Indicators

This paper considers two optimization indicators: the response time and the energy consumption. The decision to offload DAG tasks and determine the device for task execution is made by the local device. Let the offloading decision of task s be α_s, $\alpha_s \in \{0\} \cup \mathcal{M}$. The offloading decision, denoted by $\alpha_s = 0$, signifies that the task is executed on local device. $\alpha_s = m$ indicates that the task is uploaded and executed on edge host m.

The arrival time of task s at device α_s is denoted by $load(\alpha_s)$. The idle time of device α_s is represented as $avail(\alpha_s)$, while ensuring that the available time slots are sufficient to accommodate the execution of tasks. $EST_T(\alpha_s)$ stands for the theoretical earliest start time of the task, indicating the earliest completion time when communication data from all predecessor tasks arrive at the current device α_s. The actual earliest start time of the task s on device α_s is

$$EST_A(\alpha_s) = \max\{load(\alpha_s), avail(\alpha_s), EST_T(\alpha_s)\}. \tag{12}$$

If the task is executed locally, the completion time for task upload is 0; if offloaded on edge host m, the theoretical upload start time is the transmission channel idle time plus the task upload time. Thus, the time when task s arrives at the device α_s is

$$load\left(\alpha_s\right)=\begin{cases}0, & \text{if } \alpha_s=0\\ wait(l,m)+t_s^{l,m}, & \text{if } \alpha_s=m\end{cases}, \tag{13}$$

where $wait(l,m)$ is the transmission channel idle time between local device and edge host m. The theoretical earliest start time of task s is determined by the latest arrival time of all its predecessor tasks. Thus, the theoretical earliest start time of the task is

$$EST_T\left(\alpha_s\right)=\max_{s'\in pred(s)}\{EFT\left(\alpha_s\right)+wait\left(\alpha_{s'},\alpha_s\right) +t_{s',s}^{\alpha_{s'},\alpha_s}\}, \tag{14}$$

where $pred(s)$ denotes the set of predecessor tasks for task s. When task s executes on the same device as its predecessor task s', the transfer time of its data communication is 0, that is

$$wait\left(\alpha_{s'},\alpha_s\right)+t_{s',s}^{\alpha_{s'},\alpha_s,comm}=0.$$

The earliest completion time of task s at device α_s is

$$EFT(\alpha_s)=EST_A(\alpha_s)+t_s^{\alpha_s}. \tag{15}$$

The total response time is

$$t_{total}=\max_{s\in\mathcal{S}}\{EFT(\alpha_s)\}, \tag{16}$$

where $\mathcal{S}=\{1,2...,S\}$ is the set of all tasks.

When a task is executed locally, there is no need to consider the upload energy consumption of the task. The energy consumed by task s is given by

$$EC(\alpha_s)=\begin{cases}e_s^{\alpha_s}+\sum_{s'\in pred(s)}e_{s',s}^{\alpha_{s'},\alpha_s}, & \alpha_s=0\\ e_s^{l,\alpha_s}+e_s^{\alpha_s}+\sum_{s'\in pred(s)}e_{s',s}^{\alpha_{s'},\alpha_s}, & \alpha_s=m\end{cases}. \tag{17}$$

The total energy consumption is

$$e_{total}=\sum_{s=1}^{S}EC(\alpha_s). \tag{18}$$

B. Optimization Objective

The optimization objective in this paper is to minimize both the total response time and energy consumption. The optimization function is represented as

$$U=\min_{s\in\mathcal{S}}\{\lambda_1 t_{total}+\lambda_2 e_{total}\}, \tag{19}$$

$$\text{s.t. } \alpha_s\in\{0\}\cup\mathcal{M}, \tag{19a}$$

$$f_{\alpha_s}\leqslant f_{\alpha_s}^{\max}, \tag{19b}$$

$$\sum^{N}B_l\leqslant B_l^{total}, \tag{19c}$$

$$load(\alpha_s)\leqslant avail(\alpha_s), \tag{19d}$$

$$EFT(\alpha_{s'})\leqslant EST_A(\alpha_s),\ s'\in pre(s), \tag{19e}$$

$$[EST_A(\alpha_s),EFT(\alpha_s)]\bigcap[EST_A(\alpha_{s^*}),EFT(\alpha_{s^*})], \quad s^*\neq s,\ \alpha_{s^*}=\alpha_s,\ s^*\in pro(\alpha_s). \tag{19f}$$

Where the sum of the target weights is 1, that is, $\lambda_1+\lambda_2=1$. Constraint (19a) mandates that a task selects only one offloading location. Constraint (19b) dictates that the computational capacity of the device executing the task does not exceed its maximum limit. Constraint (19c) states that the total allocated bandwidth to devices must not exceed the available total bandwidth. Constraint (19d) mandates that the task's start execution time must be later than the time at which it reaches the device. Constraint (19e) indicates that the processing of a task cannot begin until all its predecessor tasks are completed, thereby ensuring the correct execution order of the tasks. Constraint (19f) indicates that only one task can be executed at a time on the same host, where $pro(\alpha_s)$ denotes the set of tasks that are on the same host as the task s. The task s^* may come from different users.

V. FDAHO Offloading Strategy

In this section, the FDAHO algorithm is proposed. The algorithm comprises three parts. The first part is the edge rank_u sorting (ERS) based sorting algorithm. The second part is the FDQN based offloading algorithm. The third part is the TPE based hyperparameter optimization algorithm.

A. ERS Based Sorting Algorithm

The DAG tasks have constraint relationships between them. Therefore, before task offloading, it is necessary to sort the tasks to ensure the order of task execution. This paper employs the ERS algorithm. ERS draw on the method in [28]. Topcuoglu et al. [28] primarily focus on task scheduling in heterogeneous cloud computing environments without explicitly addressing the complexities introduced by task offloading in MEC. Unlike traditional methods, the ERS algorithm focuses on optimizing task offloading while considering the unique constraints and requirements of MEC.

Initially, compute the rank value for each task, which is composed of the following components: (1) the average execution time of the target task; (2) the maximum value of the sum of the average communication time and rank values of its successor tasks. Given that the exit task lacks successor tasks, its rank value equals the average execution time of the exit task. Subsequently, arrange tasks in descending order based on their rank values. The offloading queue defined after task sorting is $\boldsymbol{\phi}=\{\phi_1,\phi_2,\ldots,\phi_S\}$, $\phi_s\in S$, $\phi_{s'}\in S$, $s\neq s'$, $\phi_s\neq\phi_{s'}$. This measure helps prioritize task nodes based on their positions in the graph. $\overline{t_s}$ is the average of the task's execution times across all devices. $\overline{t_{s,s'}}$ is the average transfer time for data communication between tasks across all channels. The rank value of task s is recursively calculated by

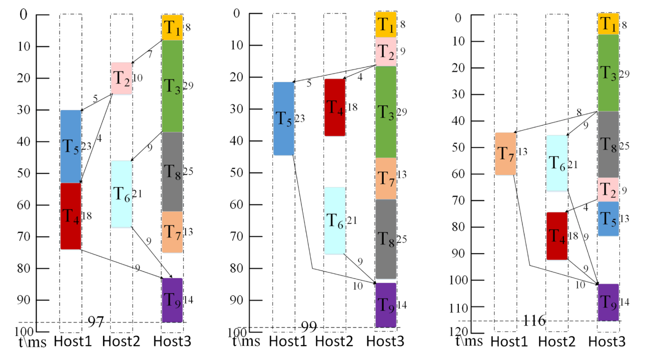
$$Rank(s)=\overline{t_s}+\max_{s'\in succ(s)}\left\{Rank\left(s'\right)+\overline{t_{s,s'}}\right\}, \tag{20}$$

where $succ(s)$ is the set of all successor tasks. The average execution time of task s on the device is

$$\overline{t_s}=\frac{1}{1+M}\times\left(t_s^l+\sum_{m=1}^{M}t_s^m\right). \tag{21}$$

TABLE III
COMPUTATIONAL COSTS

Task	Host1	Host2	Host3
T_1	10	9	8
T_2	12	10	9
T_3	21	18	16
T_4	23	20	18
T_5	39	34	29
T_6	25	21	19
T_7	16	14	13
T_8	33	29	25
T_9	19	16	14



(a) Edge rank_u sorting (b) Edge rank_d sorting (c) Edge critical paths sorting

Fig. 3. Total task response time of different sorting.

The average communication time for data transfer between task s and task s' on the channel is

$$\overline{t_{s,s'}} = \frac{t_{s,s'}^{l,m} + t_{s,s'}^{m,l} + \sum_{m=1}^{M} \sum_{m \neq m'} t_{s,s'}^{m,m'}}{2 + \binom{M}{2}}. \quad (22)$$

Assume that the DAG tasks in Fig. 2 have the computational costs on three different edge host as shown in Table III. Each host possesses varying computational capabilities or characteristics. These capabilities impact task execution efficiency. They also influence overall computational workload distribution strategies. Compare different offloading sorting algorithms. As shown in Fig. 3(a)–(c), they represent edge rank_u sorting, edge rank_d sorting, and edge critical path sorting respectively. The term "rank_u" represents the ranking calculated from the bottom up, commencing from the exit task. Tasks are then sorted in descending order according to their rank values to ascertain the offloading sequence. The resulting task offloading order is $\phi_{rank\_u} = \{T_1, T_3, T_2, T_8, T_5, T_6, T_4, T_7, T_9\}$. The term "rank_d" signifies the ranking calculated from the top down, starting from the enter task. Tasks are then sorted in ascending order according to their rank values to determine the offloading sequence. The resulting task offloading order is $\phi_{rank\_d} = \{T_1, T_2, T_3, T_4, T_5, T_7, T_6, T_8, T_9\}$. The term "critical path" refers to the strategy of summing the rank_u and rank_d values for each task, and prioritizing the offloading of tasks that have the highest combined value on this path. The resulting task offloading order is $\phi_{critical\_path} = \{T_1, T_3, T_8, T_6, T_7, T_2, T_5, T_4, T_9\}$. When each task is offloaded

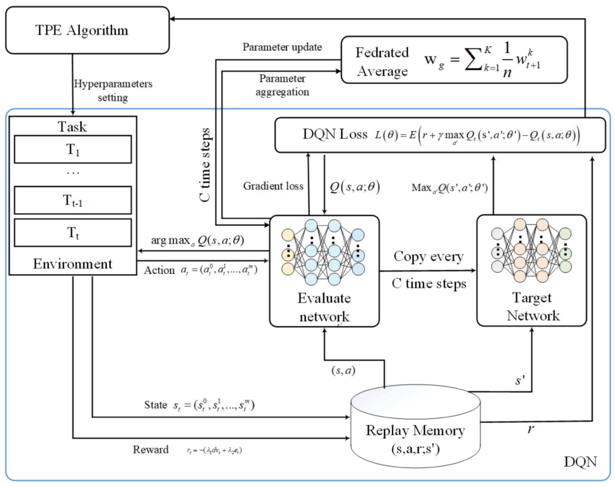

TPE Algorithm
Parameter update
Fedrated Average
$w_g = \sum_{k=1}^{K} \frac{1}{n} w_{t+1}^k$
Hyperparameters setting
Parameter aggregation
Task
T_1
T_{t-1}
T_t
Environment
C time steps
DQN Loss $L(\theta) = E\left(r + \gamma \max_{a'} Q_t(s', a'; \theta') - Q_t(s, a; \theta)\right)$
Gradient loss
$Q(s,a;\theta)$
$\max_{a'} Q(s', a'; \theta')$
$\arg\max_a Q(s,a;\theta)$
Action $a_t = (a_t^0, a_t^1, \ldots, a_t^m)$
Evaluate network
Copy every C time steps
Target Network
(s,a)
s'
State $s_t = (s_t^0, s_t^1, \ldots, s_t^m)$
Replay Memory (s,a,r,s')
r
Reward
DQN

Fig. 4. FDAHO model.

to the device with the shortest response time, the total corresponding time of edge rank_u sorting, edge rank_d sorting, and edge critical path sorting is 97ms, 99ms, and 116ms, respectively [28], [29]. The total time is the shortest with edge rank_u sorting. The reason is that edge rank_u based sorting tends to prioritize scheduling tasks that affect the execution of subsequent tasks, thereby more effectively reducing the overall execution time.

B. FDQN Based Offloading Algorithm

The algorithm flowchart of the FDAHO model is shown in Fig. 4. First, after generating the task offloading queue, tasks are sequentially placed into the DQN environment. The evaluation network is trained based on the state and available actions, selecting the action with the highest Q-value. The next state and reward obtained are stored in the cache. Additionally, every C steps, the evaluation network is copied to the target network to calculate the DQN loss value. The parameters of all device models are aggregated and updated. TPE evaluates the performance of parameters by setting different hyperparameters and assessing the average loss value after DQN execution. The process occurs during the training phase of FDAHO and does not affect system response time and energy consumption during the testing phase. Once the model training is completed, there is no need to update hyperparameters in real-time; instead, we directly use the optimized DQN model for task offloading. This is an offline training and online deployment mechanism.

1) DQN Algorithm: For each DAG task, the optimal device needs to be selected for offloading and execution. However, traditional algorithms struggle to effectively handle the high-dimensional state space of data. Therefore, we introduce the DQN algorithm from DL to address this issue. DQN extends Q-learning by incorporating deep neural networks to handle more complex state spaces. Compared to other DL algorithms, the DQN method is more suitable for problems with discrete action spaces. The model comprises an evaluation network and a target network, both sharing same structures. The evaluation network is responsible for action selection and assessing the current policy. During training, its parameters are updated through gradient descent to estimate the optimal Q-value function. The target network is employed for computing target Q-value, and its parameters remain frozen throughout training, avoiding changes with each update. it periodically obtains the parameters of the evaluation network to ensure stable target Q-value computation over the entire training process. Thus, The Q-value in DQN update formula is:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \times (r + \gamma \times \max_{a'} Q_t(s',a') - Q_t(s,a)), \quad (23)$$

where α represents the learning rate, determining the extent to which the error is learned, while γ serves as the discount factor, denoting the degree to which future rewards are discounted. For action selection, DQN employs an ε-greedy strategy. The formula for this strategy is expressed as follows:

$$a = \begin{cases} random(a), & 0 \le e < \varepsilon \\ \arg\max_a Q(s,a;\theta), & \varepsilon \le e \le 1 \end{cases}, \quad (24)$$

where e $(e \in [0,1])$ represents a random value. ε diminishes with an increase in the iteration count, ensuring the accuracy of the converged model. Finally, the predicted loss value for the current round is calculated:

$$L(\theta) = E(r + \gamma \times \max_{a'} Q_t(s',a';\theta') - Q_t(s,a;\theta)), \quad (25)$$

where θ' indicates the parameters of the evaluate network, while θ indicates the parameters of the target network. The evaluate network is updated based on the feedback of $L(\theta)$. The target network acquires parameters every C iterations. This approach ensures stable updates of the algorithm and reduces errors.

Based on the properties of the algorithm described above, the whole overall procedure can be depicted as a markov decision process (MDP) [30], [31]. Crucial components of MDP are designed as follows:

- State Space: At each time step t, an offloading decision is made for a task. The state space of the DQN consists of response time and energy consumption of tasks on different devices. $i = 0$ represents the local device, and $i \in [1,m]$ represents edge devices. s_t^i represents the response time and energy consumption of a task offloaded to device i. The current state is represented as a vector, defined as

$$s_t = (s_t^0, s_t^1, \ldots, s_t^m),$$
$$s_t^i = \lambda_1 dv_t^i + \lambda_2 e_t^i \quad i \in [0,m]. \quad (26)$$

- Action Space: The action space of the DQN is determined by the location where the task is offloaded. Each task can only be offloaded to a location. We use a vector to ensure uniqueness, such as $a_t = (1, 0, \ldots, 0)$ where $a_t^0 = 1$ and $a_t^1 = a_t^2 = \ldots = a_t^m = 0$, which indicates executing the task locally. The action space is specified as

$$a_t = (a_t^0, a_t^1, \ldots, a_t^m),$$
$$\begin{cases} a_t^i \in \{0,1\} \\ \sum_i^{m+1} a_t^i = 1. \end{cases} \quad (27)$$

Algorithm 1: DQN Algorithm.

```
Require: State set $S$, action set $A$
Ensure: Offloading action $a$
1: Initialize target network $\hat{Q}$ and evaluation network $Q$
2: Set the size of the experience replay buffer $D$ to $C_{ep}$
3: for all $e = 1, E$ do
4:   Initialize $s_1$
5:   for all $t = 1, T$ do
6:     Compute two indicators of current task on all
       devices by (29) and (30)
7:     choose action $a_t$ by (24)
8:     Executing $a_t$ and observing $s_{t+1}$ and $r_t$
9:     Storing experiences $(s_t, a_t, r_t, s_{t+1})$ in $D$
10:    Set $y_j = \begin{cases} r_j, \text{ if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta'), \text{ otherwise} \end{cases}$
11:    Randomly sampling from the experience replay
       buffer and conducting training
12:    Performing one step Q-learning training
13:    Calculate $L(\theta)$ by (25) and record
14:    Every C time steps, transfer the parameters of $Q$ to
       $\hat{Q}$
15:  end for
16: end for
```

- Reward: After choosing an action in the current state, the agent gets an instant reward. The reward function is a measure of the quality of the agent's behaviour in the environment. The reward function is expressed as the weighted sum of response time and energy consumption of the task in the current state. Specifically, the reward is defined as the difference between the total response time of the task in the current state and the previous state, along with the total energy consumption of the task. The reward formula is

$$r_t = -(\lambda_1 dv_t + \lambda_2 e_t). \quad (28)$$

If task ϕ_s is offloaded at time step t, the response time of task ϕ_s is defined as the difference between the response times of all remaining tasks from time step t to time step $t-1$. The response time of the task in the current time step t is denoted by:

$$dv_t = \max\left\{EFT(\alpha_{\phi_s}), \max_{\phi_{s'} \in pre(\phi_s)} \{EFT(\alpha_{\phi_{s'}})\}\right\} - \max_{\phi_{s'} \in pre(\phi_s)} \{EFT(\alpha_{\phi_{s'}})\}, \quad (29)$$

where $pre(\phi_s)$ represents all tasks that has been offloaded before, $pre(\phi_s) = (\phi_1, \phi_2, \ldots, \phi_{s-1})$. The energy consumption of the task at the current time step t is

$$e_t = EC(\alpha_{\phi_s}). \quad (30)$$

The comprehensive procedure of the DQN algorithm is depicted in Algorithm 1, with a time complexity of $O(E \times T)$, where E is the number of episodes and T is the size of the total timestamp.

2) Federated Averaging Algorithm: Each local device has limited and independent task data. However, the DQN algorithm improves its performance based on a large number of samples. In a distributed scenario, FL can have a positive impact on DQN or other deep reinforcement learning models. FL enables local device-based model training without transferring raw data to a central server. This approach helps protect user privacy as individual data does not leave the device directly. Through FL, the central server can coordinate these local models to form a global model. This is beneficial for reducing communication burdens on the central server and enabling model training in MEC. FL allows the sharing of learned knowledge among different devices, and collaborative training can enhance the performance of the global model. For DQN, this collaborative approach can assist the model in better adapting to diverse local environments and user behaviors, thereby improving the model's generalization performance [32], [33].

Next, the algorithm process is introduced. After K devices undergoes C rounds of training, local model parameters $\{W_t^1, W_t^2, \ldots, W_t^K\}$ are transmitted to the edge server. The edge server collects local model parameters and performs aggregation averaging. In FL, the federated averaging algorithm is used to generate global model parameters with the formula represented as

$$W_g = \frac{1}{K} \times \sum_{k=1}^{K} W_t^k, \tag{31}$$

where g is number of rounds of aggregation. Finally, the global model parameters are delivered to each local device. The formula for this is

$$W_{t+1}^k = W_g. \tag{32}$$

The FDQN is derived from the combination of FL and DQN. The comprehensive procedure of the FDQN is depicted in Algorithm 2, with a time complexity of $O(K \times E \times T)$.

Algorithm 2: FDQN Algorithm.

```
Require: Task offloading sequence and device states
Ensure: System response time and energy consumption
1: Edge Server: At the initial time step, the global
   parameter $w_0$ is initialized and issued to the local device
2: Local Device: Obtain $w_0$ from Edge server and run the
   DQN algorithm at the initial time step
3: for all round $g = 1, 2, \ldots, G$ do
4:    Edge Server:
5:    Waiting to receive device parameters $w_t^k$
6:    Calculate $w_g$ by formula (31) and sent to each local
      device at time step $t + 1$
7:    Local Devices :
8:    for all local device $k \in K$ in parallel do
9:       for all time step t = 1,2,... do
10:         Execute lines 6-15 of the Algorithm 1
11:         if $t\%C == 0$ then
12:            Transmit the model parameters $w_t^k$ of the
               current time step $t$ to the edge server
13:            Wait to receive the aggregated model
               parameters $W_g$
14:            Update the model parameters at $t + 1$
15:         end if
16:      end for
17:   end for
18: end for
```

C. TPE Based Hyperparameter Optimization Algorithm

Well-tuned hyperparameters can enhance the performance of the algorithm. Therefore, before running the FDQN model, it is necessary to perform hyperparameter optimization. This paper employs the TPE hyperparameter optimization algorithm. Compared to traditional parameter optimization methods, it can quickly find relatively optimal solutions.

The TPE algorithm comprises the following steps. First, collect and analyze the observational data, denoted as $Z = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(k)}, y^{(k)})\}$. Subsequently, define a probability density function as

$$p(\mathrm{x}|y) = \begin{cases} o_1(x), & y^* \le y \\ o_2(x), & y^* > y \end{cases}, \tag{33}$$

where $o_1(x)$ is formed using the remaining observations. $o_2(x)$ is formed by observed variables $\{x^{(i)}\}$ such that $y^* > y^i$. y^* is chosen to be a quantile η of the remaining observed y values, where $p(y^* > y) = \eta$. All in all, $o_1(x)$ models the density of poorly observed values, while $o_2(x)$ models the density of well-observed values. Next, observations are segmented into sets Z_{o_1} and Z_{o_2} for model construction. This segmentation is accomplished by sorting the observations according to their corresponding y values. The infill criterion utilized by TPE is the following expected improvement (EI) function:

$$\begin{aligned} \mathrm{EI}_{y^*}(x) &= \int_{-\infty}^{\infty} \max(y^* - y, 0) \times p(y|x)dy \\ &= \int_{-\infty}^{y^*} (y^* - y) \times p(y|x)dy \\ &\propto (\eta + (1-\eta) \times \frac{o_1(x)}{o_2(x)})^{-1}. \end{aligned} \tag{34}$$

The final step involves selecting the candidate x^* with the highest EI value [7], [34].

Next, we set the search space for hyperparameters to be

$$\mathrm{x}^* = \begin{cases} B \in \{16, 32, 64, 128\} \\ \gamma \in [0.5, 1] \\ AF \in \{softplus, relu, tanh, sigmoid\} \end{cases}, \tag{35}$$

where B is batch size. In DQN model training, it indicates the number of data samples used in each parameter update. The choice of batch size affects the training speed, memory requirements, optimization stability, and the model's ability to generalize. γ is discount factor. It determines the present value of future rewards in a sequence of actions. The discount factor influences how much importance an agent places on future rewards compared to immediate rewards during decision-making. AF is active function. The activation function in neural networks determines the output of a neuron based on its input. The choice of activation function affects the network's capacity to learn

Algorithm 3: TPE Algorithm.

```
Require: Observational data set $Z$
Ensure: The optimal hyperparameter combination in set $Z$
1: Initialize the probability density function
2: for all $i = 1, n_i$ do
3:    $Z_{o_2} = \{(x, y)|$x with the best- $\lceil \eta \times |Z| \rceil$ $y$ values in $Z\}$
4:    $Z_{o_1} = Z \backslash Z_{o_2}$
5:    Build $o_1(x)$ and $o_2(x)$ separately
6:    $C = \{x^{(i,j)} \sim o_2(x) | j = 1, \ldots, n_c\}$
7:    $x^* = \arg\max_{x \in C} EI_{y^*}(x)$ select the best candidate
8:    Use $x^*$ as the hyperparameter for Algorithm 1
9:    Calculate $y^*$ value by (36)
10:   $Z = Z \bigcup \{(x^*, f(x^*))\}$
11: end for
```

and its ability to model complex relationships within the data. The average loss of DQN is defined as the objective value that needs to be minimized. Therefore, the objective value for hyperparameter optimization is

$$y^* = \frac{1}{E} \times \frac{1}{T} \times \sum_{e=1}^{E} \sum_{t=1}^{T} L(\theta), \tag{36}$$

where E is the number of training rounds for DQN, and T is the number of DAG tasks. The overall process of the TPE algorithm is illustrated in Algorithm 3, with a time complexity of $O(n_i \times E \times T)$, where n_i is the number of iterations.

VI. Numerical Simulation

In this section, we first construct the experimental platform and environment. Subsequent selections are made for the hyperparameters of the experiment. Following this, the weights are determined for optimizing the objectives. Lastly, the performance of the FDHAO is analyzed and compared against the traditional training algorithm.

A. Experimental Settings

This experiment utilizes Python as the simulation platform and employs TensorFlow to construct the neural network. The DAG parsing library is utilized to generate and parse information related to DAG tasks. The data center library is employed to simulate parameters of edge server and local devices, including virtual machines, hosts, base stations, and so forth. The local device is used as follows: (1) Generate DAG tasks. (2) Sort and offload tasks. (3) Execute tasks locally. The role of edge servers is as follows: (1) Provide remote computing capabilities. (2) Collect system information and resource status. (3) Aggregate model parameters. The task library is responsible for defining DAG tasks, subtasks, as well as input and output file information among tasks. The shared library serves as a repository for sharing information between devices. Meanwhile, the offloading library defines various offloading algorithms to simulate the offloading process. The experimental workflow is structured as follows: (1) Create edge and local devices. (2) Generate tasks from local devices. (3) Parse the DAG task information. (4) Choose an appropriate task offloading algorithm. (5) Collate and analyze experimental results and data. The devices are heterogeneous. The relevant parameters of heterogeneous devices for the experiment are presented in Table IV.

TABLE IV
Experimental Simulation Parameters

Parameters	Value
$B_l(MHz)$	$[500, 1000]$
$B_m^{m'}(MHz)$	$[1000, 2500]$
(dBm/Hz)	-174
$f_l(MIPS)$	$[500, 1000]$
$f_m(MIPS)$	$[1000, 2500]$
$p^{comp}(W)$	$[1, 25]$
$p^{tran}(W)$	$[30, 80]$

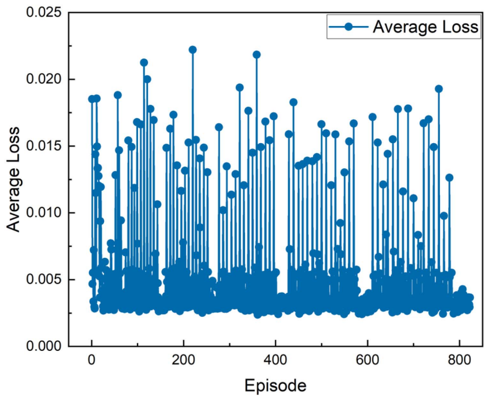


Fig. 5. Convergence of TPE Algorithm.

In the research on DAG task offloading, information such as the computational size of tasks is generally known and can be approximately analyzed. Although there may be some minor errors, they can be ignored. For datasets, we utilize real scientific workflows sourced from the Pegasus workflow management system, including workflows such as Inspiral, Sipht, Montage, and CyberShake and so on [35], [36]. These workflows are provided in DAX format in XML. These workflows have different characteristics and structures, serving as common benchmarks for evaluating the effectiveness of workflow scheduling algorithms [37].

B. Hyperparameter Optimization Experiments

The performance of the DQN algorithm is influenced by several key hyperparameters. The active function, batch size and discount factor are mainly selected as objects for automatic hyperparameter optimization of TPE. The average loss of the DQN algorithm is used as an indicator to evaluate the performance of the selected parameters. A smaller average loss suggests that the selected hyperparameters positively influence the algorithm's performance. The TPE algorithm initially guesses the hyperparameters corresponding to optimal values and subsequently validates them. The relationship between the number of TPE iterations and the objective value is depicted in Fig. 5. The TPE algorithm quickly identifies a relatively optimal set of hyperparameters. Around 170 iterations, it becomes evident that

TABLE V
HYPERPARAMETERS SETTING

Hyperparameters	Value
AF	Relu
B	64
γ	0.906
C_{ep}	100
ε	0.5
$layers$	3
α	1e-3
L	Mean-square error

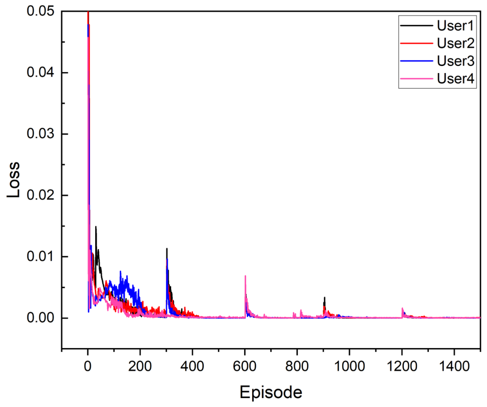


Fig. 6. Convergence of DQN Algorithm.

favorable results emerge when the hyperparameters fall within a specific range. As the iterations progress to 400, 600, and 800, there is an increasing consistency in selecting hyperparameters within a certain range. This is attributed to the TPE algorithm relying on guessing, preventing it from gradually converging to the optimal value as the number of iterations increases. But it prevents leading to local optimal solutions and takes less time to find the optimal solution. After conducting numerous experiments, hyperparameters corresponding to the minimum average loss were chosen. The activation function is “Relu”, the discount factor is 0.906, and the batch size is 64. The settings for all hyperparameters of the DQN are detailed in Table V.

Next, the convergence of the ultimately selected hyperparameters is tested using datasets owned by different users. Taking tasks generated by four different users with varying DAG types as an example, the convergence of the DQN is depicted in Fig. 6. The graph indicates that with an increase in the number of iterations, the loss value for all four users gradually decreases and eventually stabilizes. By the time it reaches 1300 iterations, the loss value no longer exhibits significant fluctuations. This indicates that the selected hyperparameters can provide the algorithm with good stability and performance.

C. Weight Selection Experiments

To measure the significance of optimization objectives in practical MEC scenarios, setting appropriate weighting factors is crucial. Different users may prioritize distinct indicators. Experiments were conducted by varying the weight of response time from 0.1 to 0.9, with response times and energy consumption recorded at each weight value. The average response time and

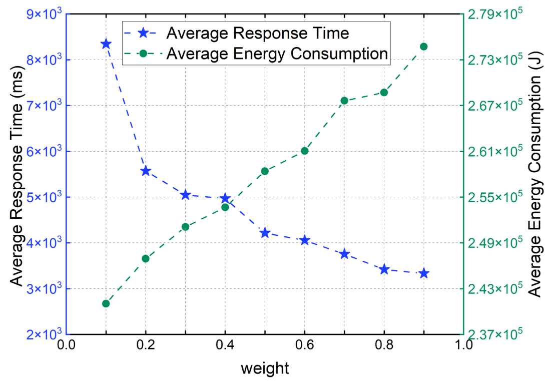


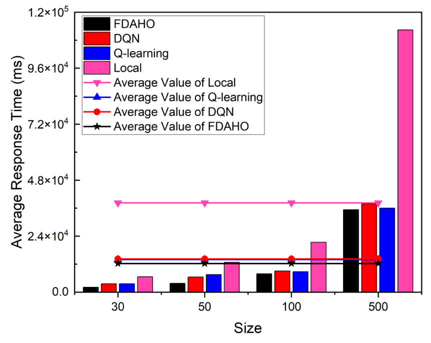
Fig. 7. The response time and energy consumption under different weights.

energy consumption indicators are obtained by experiments with multiple groups of data. The average response time and average energy consumption corresponding to different weightings are depicted in Fig. 7. With an increase in the weight assigned to response time, the response time decreases while the energy consumption increases. When the weight is around 0.4, the average response time and average energy consumption intersect, indicating a balanced proportion between the weightings of these indicators. A weight of 0.4 was selected as the balancing point between response time and energy consumption for subsequent algorithm performance experiments.

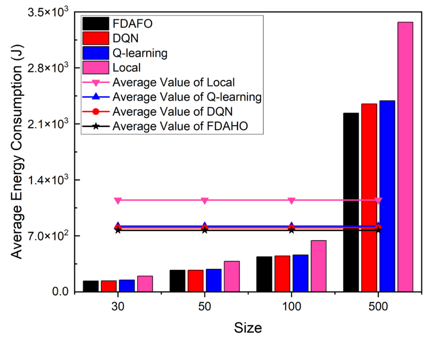
D. Performance Experiments

To evaluate the effectiveness of the FDAHO algorithm, we conducted a comparison with Q-learning and DQN. The primary distinction between these algorithms and FDAHO lies in the training methods. Additionally, we also used the local offloading algorithm as a reference.

By controlling experimental variables, the experiment tested performance indicators in different scenarios. To ensure fairness in task offloading among users, a random strategy is used to select the user offloading order. First, a MEC environment consisting of 4 local devices and 4 edge hosts was constructed. When each local device had distinct types of DAG tasks, the performance of the algorithm was tested under scenarios with average task numbers of 30, 50, 100, and 500, respectively. The performance indicators for all devices were averaged to obtain the average response time and average energy consumption. The experimental results are illustrated in Fig. 8. Whereas Fig. 8(a) compares the average response time of the algorithms for task offloading, Fig. 8(b) contrasts the average energy consumption of the task offloading algorithms. As the number of tasks increases, both indicators also increase accordingly. The line graphs in the figure illustrate the average performance values of the algorithms across the four different task counts. This provides a more intuitive way to compare the performance results of the algorithms. For example, in the scenario where different users execute different types of DAG tasks but with the same number of tasks, the black line represents the average value obtained by summing the corresponding values of FDAHO for task numbers 30, 50, 100, and 500 and then taking the average. In

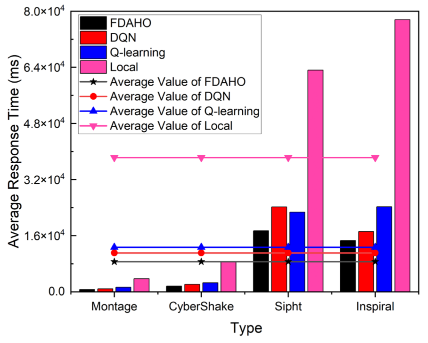


(a) Average response time

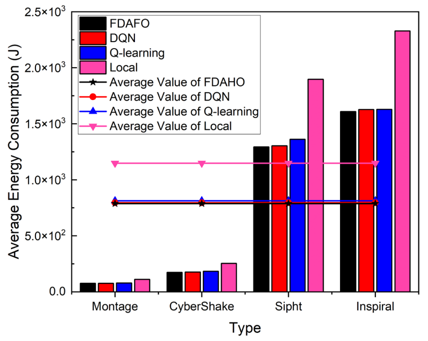


(b) Average energy consumption

Fig. 8. Comparing indicators in scenarios where different users utilize DAG tasks of varying types but with the same number of tasks.



(a) Average response time



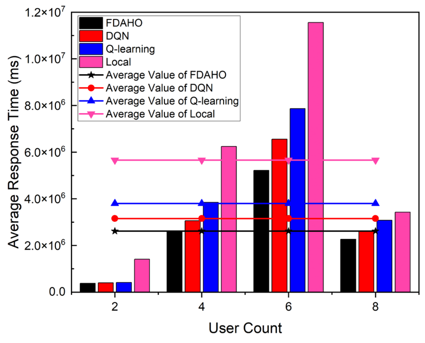
(b) Average energy consumption

Fig. 9. Comparing indicators in scenarios where different users utilize DAG tasks of the same type but with varying numbers of tasks.

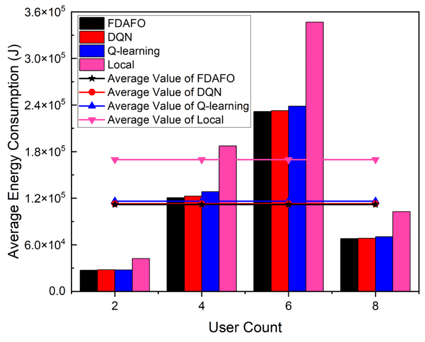
this scenario, FDAHO improves response time by 15.8%, 12.1%, and 67.9% compared to DQN, Q-learning, and Local offloading algorithms, respectively. Energy consumption improvements are 3.9%, 6.2%, and 32.9%, respectively. Testing with various task numbers, the FDAHO algorithm demonstrates a better reduction in both indicators. Simultaneously, since the training data does not need to be uploaded to the edge server, it will not cause data leakage, thus protecting the privacy of users.

Another scenario involves local devices with the same type of tasks but a different number of generated tasks. The experimental results are depicted in Fig. 9. Taking the first horizontal axis value from the graph as an example, it indicates that each local device utilized a "Montage" type dataset. However, the quantity of DAG tasks generated by each local device varied within the range of 30 to 500. The meanings of the other horizontal axis values are similar. We tested datasets such as Montage, CyberShake, Sipht, and Inspirial. Each type of dataset possesses distinct node information and varying graph structures for the DAG tasks. Due to the varying task types, both indicators are illustrated in Fig. 9(a) and (b), respectively. In the scenario where different users execute the same type of DAG tasks but with varying numbers of tasks, the black line represents the average value obtained by summing the corresponding values of FDAHO for task types Montage, CyberShake, Sipht, and Inspiral and then taking the average. In this scenario, FDAHO improves response time by 22.5%, 32.5%, and 77.5% compared to DQN, Q-learning, and Local offloading algorithms, respectively. Energy consumption improvements are 1.0%, 3.1%, and 31.4%, respectively. The FDAHO algorithm demonstrates more pronounced improvements in response time of the task compared to Q-learning and DQN. This is attributed to the similarity in the data types used by local devices, enabling FL to better integrate model parameters and enhance model performance. While the optimization in terms of energy consumption is modest, it still slightly outperforms the other two traditional machine learning algorithms. This is the result of balancing between reducing response time and increasing energy consumption. Therefore, it's challenging to optimize both metrics simultaneously. However, the FDAHO algorithm strives to strike a balance between the two.

The third testing scenario involves each device generating one or multiple DAGs with random numbers and types. The experiments were conducted to measure the response time and energy consumption for user counts of 2, 4, 6, and 8. The results are presented in Fig. 10. In the scenario where different users execute DAG tasks with random numbers and types, the black line represents the average value obtained by summing the corresponding values of FDAHO for user numbers 2, 4, 6, and 8 and then taking the average. In this scenario, the FDAHO improves response time by 17.1%, 31.1%, and 53.8% compared to DQN, Q-learning, and Local offloading algorithms, respectively. Energy consumption improvements are 0.9%, 3.8%, and 34.1%, respectively. Due to the random datasets used by each local device, after multiple experiments, it's not guaranteed that the average response time and energy consumption will increase with the number of users. For instance, with 8 users, there might be instances where some devices generate fewer tasks, leading to a lower response time and energy consumption compared



(a) Average response time



(b) Average energy consumption

Fig. 10. Comparing indicators in scenarios where different users utilize DAG tasks of random numbers and types.
.

to other user counts. However, our algorithm still demonstrates superior experimental performance.

The experimental results above indicate that the FDAHO algorithm performs better in terms of both response time and energy consumption compared to Q-learning and DQN algorithms. Additionally, it exhibits significant advantages over local offloading methods.

VII. Conclusion

The paper introduces a task offloading algorithm named FDAHO. This algorithm is designed to address multi-objective optimization for DAG task offloading in an MEC environment, all while ensuring user privacy. The components of this algorithm include various techniques such as the ERS based sorting algorithm, FDQN based offloading algorithm, and TPE based hyperparameter optimization algorithm. Experiments were conducted under various scenarios, considering different types of tasks, varying task size, and distinct user counts. The experimental outcomes demonstrate that the FDAHO algorithm effectively reduces both response time and energy consumption of the task while safeguarding user privacy.

References

[1] Y. Qi and M. S. Hossain, "Harnessing federated generative learning for green and sustainable Internet of Things," *J. Netw. Comput. Appl.*, vol. 222, 2024, Art. no. 103812.

[2] M. Liyanage, P. Porambage, A. Y. Ding, and A. Kalla, "Driving forces for multi-access edge computing (MEC) IoT integration in 5G," *ICT Express*, vol. 7, no. 2, pp. 127–137, 2021.

[3] F. Liu, J. Huang, and X. Wang, "Joint task offloading and resource allocation for device-edge-cloud collaboration with subtask dependencies," *IEEE Trans. Cloud Comput.*, vol. 11, no. 3, pp. 3027–3039, Third Quarter 2023.

[4] X. Chen and G. Liu, "Energy-efficient task offloading and resource allocation via deep reinforcement learning for augmented reality in mobile edge networks," *IEEE Internet Things J.*, vol. 8, no. 13, pp. 10 843–10 856, Jul. 2021.

[5] J. Yang, X. You, G. Wu, M. M. Hassan, A. Almogren, and J. Guna, "Application of reinforcement learning in UAV cluster task scheduling," *Future Gener. Comput. Syst.*, vol. 95, pp. 140–148, 2019.

[6] B. Liu, L. Wang, and M. Liu, "Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems," *IEEE Trans. Robot. Autom.*, vol. 4, no. 4, pp. 4555–4562, Oct. 2019.

[7] H.-P. Nguyen, J. Liu, and E. Zio, "A long-term prediction approach based on long short-term memory neural networks with automatic parameter optimization by tree-structured parzen estimator and applied to time-series data of NPP steam generators," *Appl. Soft Comput.*, vol. 89, 2020, Art. no. 106116.

[8] J. Baek and G. Kaddoum, "Online partial offloading and task scheduling in SDN-Fog networks with deep recurrent reinforcement learning," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 11 578–11 589, Jul. 2022.

[9] L. Zhao et al., "A digital twin-assisted intelligent partial offloading approach for vehicular edge computing," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 11, pp. 3386–3400, Nov. 2023.

[10] M. H. Khoobkar, M. D. T. Fooladi, M. H. Rezvani, and M. M. G. Sadeghi, "Joint optimization of delay and energy in partial offloading using dual-population replicator dynamics," *Expert Syst. Appl.*, vol. 216, 2023, Art. no. 119417.

[11] Z. Zabihi, A. M. Eftekhari Moghadam, and M. H. Rezvani, "Reinforcement learning methods for computation offloading: A systematic review," *ACM Comput. Surv.*, vol. 56, no. 1, pp. 1–41, 2023.

[12] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[13] L. Liu, M. Zhao, M. Yu, M. A. Jan, D. Lan, and A. Taherkordi, "Mobility-aware multi-hop task offloading for autonomous driving in vehicular edge computing and networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 2, pp. 2169–2182, Feb. 2023.

[14] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.

[15] K. Liu, J. Peng, H. Li, X. Zhang, and W. Liu, "Multi-device task offloading with time-constraints for energy efficiency in mobile cloud computing," *Future Gener. Comput. Syst.*, vol. 64, pp. 1–14, 2016.

[16] C. Li, J. Tang, T. Ma, X. Yang, and Y. Luo, "Load balance based workflow job scheduling algorithm in distributed cloud," *J. Netw. Comput. Appl.*, vol. 152, 2020, Art. no. 102518.

[17] L. Zhang, L. Zhou, and A. Salah, "Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments," *Inf. Sci.*, vol. 531, pp. 31–46, 2020.

[18] R. NoorianTalouki, M. H. Shirvani, and H. Motameni, "A heuristic-based task scheduling algorithm for scientific workflows in heterogeneous cloud computing platforms," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 8, pp. 4902–4913, 2022.

[19] Q. Chen, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "An integrated framework for software defined networking, caching, and computing," *IEEE Netw.*, vol. 31, no. 3, pp. 46–55, May/Jun. 2017.

[20] B. Cheng, "Multi-population cooperative elite algorithm for efficient computation offloading in mobile edge computing," *J. Grid Comput.*, vol. 21, no. 4, 2023, Art. no. 54.

[21] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multihop offloading of multiple DAG tasks in collaborative edge computing," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4893–4905, Mar. 2021.

[22] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[23] J. Liang, K. Li, C. Liu, and K. Li, "Joint offloading and scheduling decisions for DAG applications in mobile edge computing," *Neurocomputing*, vol. 424, pp. 160–171, 2021.

[24] L. Pan, X. Liu, Z. Jia, J. Xu, and X. Li, "A multi-objective clustering evolutionary algorithm for multi-workflow computation offloading in mobile edge computing," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1334–1351, Second Quarter 2023.

[25] S. Su, P. Yuan, and Y. Dai, “Reliable computation offloading of DAG applications in Internet of Vehicles based on deep reinforcement learning,” *IEEE Trans. Veh. Technol*, to be published, doi: 10.1109/TVT.2024.3385108.

[26] Z. Liu, L. Huang, Z. Gao, M. Luo, S. Hosseinalipour, and H. Dai, “GA-DRL: Graph neural network-augmented deep reinforcement learning for DAG task scheduling over dynamic vehicular clouds,” *IEEE Trans. Netw. Service Manag.*, vol. 21, no. 4, pp. 4226–4242, Aug. 2024.

[27] R. Chen and X. Wang, “Maximization of value of service for mobile collaborative computing through situation-aware task offloading,” *IEEE Trans. Mobile Comput.*, vol. 22, no. 2, pp. 1049–1065, Feb. 2023.

[28] H. Topcuoglu, S. Hariri, and M.-Y. Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[29] J. Chen, Y. He, Y. Zhang, P. Han, and C. Du, “Energy-aware scheduling for dependent tasks in heterogeneous multiprocessor systems,” *J. Syst. Archit.*, vol. 129, 2022, Art. no. 102598.

[30] X. Gao, M. C. Ang, and S. A. Althubiti, “Deep reinforcement learning and markov decision problem for task offloading in mobile edge computing,” *J. Grid Comput.*, vol. 21, no. 4, 2023, Art. no. 78.

[31] S. S. Shinde and D. Tarchi, “A Markov decision process solution for energy-saving network selection and computation offloading in vehicular networks,” *IEEE Trans. Veh. Technol*, vol. 72, no. 9, pp. 12 031–12 046, Sep. 2023.

[32] Z. Tong, J. Wang, J. Mei, K. Li, W. Li, and K. Li, “Multi-type task offloading for wireless Internet of Things by federated deep reinforcement learning,” *Future Gener. Comput. Syst.*, vol. 145, pp. 536–549, 2023.

[33] W. Hou, H. Wen, H. Song, W. Lei, and W. Zhang, “Multiagent deep reinforcement learning for task offloading and resource allocation in cybertwin-based networks,” *IEEE Internet Things J.*, vol. 8, no. 22, pp. 16 256–16 268, Nov. 2021.

[34] S. Tao, P. Peng, Y. Li, H. Sun, Q. Li, and H. Wang, “Supervised contrastive representation learning with tree-structured parzen estimator Bayesian optimization for imbalanced tabular data,” *Expert Syst. Appl.*, vol. 237, 2024, Art. no. 121294.

[35] R. F. Da Silva, W. Chen, G. Juve, K. Vahi, and E. Deelman, “Community resources for enabling research in distributed scientific workflows,” in *Proc. IEEE 10th Int. Conf. E-Sci.*, 2014, pp. 177–184.

[36] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, “Characterization of scientific workflows,” in *Proc. IEEE 3rd Workshop Workflows Support Large-Scale Sci.*, 2008, pp. 1–10.

[37] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, “Characterizing and profiling scientific workflows,” *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 682–692, 2013.

Zhao Tong (Senior Member, IEEE) received the PhD degree in computer science from Hunan University, Changsha, China in 2014. He was a visiting scholar with the Georgia State University from 2017 to 2018. He is currently an professor with the College of Information Science and Engineering of Hunan Normal University, the young backbone teacher of Hunan Province, China. His research interests include parallel and distributed computing systems, resource management, Big Data and machine learning algorithm. He has published more than twenty five research papers in international conferences and journals, such as *IEEE Transactions on Parallel and Distributed Systems*, *Information Sciences*, *Future Generation Computer Systems*, *Neural Computing and Applications*, and *Journal of Parallel and Distributed Computing*, PDCAT, etc. He is a senior member of the China Computer Federation (CCF).

Jiaxin Deng received the BS degree in computer science and technology from Hunan City University, Yiyang, China, in 2022. He is currently working toward the MS degree with the College of Information Science and Engineering, Hunan Normal University, Changsha, China. His research focuses on resource scheduling and task offloading in mobile edge computing.

Jing Mei received the PhD degree in computer science from Hunan University, China, in 2015. She is currently an associate professor with the College of Information Science and Engineering in Hunan Normal University. Her research interests include cloud computing, fog computing and mobile edge computing, high performance computing, task scheduling and resource management, etc. She has published more than thirty research articles in international conference and journals, such as *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed System*, *IEEE Transactions on Service Computing*, *Cluster Computing*, *Journal of Grid Computing*, *Journal of Supercomputing*.

Yuanyang Zhang is currently working toward the master’s degree with the College of Information Science and Engineering, Hunan Normal University, located in Changsha, China. Her research interests mainly revolve around the areas of mobile edge computing and game theory.

Keqin Li (Fellow, IEEE) is a SUNY distinguished professor of computer science with the State University of New York. He is also a National distinguished professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy–efficient computing and communication, embedded systems and cyber–physical systems, heterogeneous computing systems, Big Data computing, high–performance computing, CPU–GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored more than eight hundred fifty journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds more than seventy patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world’s top five most influential scientists in parallel and distributed computing in terms of both single–year impact and career–long impact based on a composite indicator of Scopus citation database. He has chaired many international conferences. He is currently an associate editor of *ACM Computing Surveys* and *CCF Transactions on High Performance Computing*. He has served on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Sustainable Computing*. He is an AAIA Fellow. He is also a member of Academia Europaea (Academician of the Academy of Europe).