Multitask Offloading Strategy Optimization Based on Directed Acyclic Graphs for Edge Computing

Jiawen Chen, Yajun Yang, *Member, IEEE*, Chenyang Wang, *Member, IEEE*, Heng Zhang, Chao Qiu, *Member, IEEE*, and Xiaofei Wang, *Senior Member, IEEE*

***Abstract*—With the advancement of the user application service demands, the IoT system tends to offload the tasks to the edge server for execution. Most of the current studies on edge computation offloading ignore the dependencies between components of the application. The few pieces of research on edge computing offloading which focus on the topology of application are primarily applied in single-user scenarios. Unlike previous work, our work mainly solves dependent task offloading with edge computing in multiuser scenarios, which is more in line with reality. In this article, the dependent task offloading problem is modeled as a Markov decision process (MDP) first. Then, we propose an actor–critic mechanism with two embedding layers for directed acyclic graphs (DAGs)-based multiple dependent tasks computation offloading, namely, ACED, by jointly considering the topology of the application and the channel interference between several users. Finally, the results of simulations also show the priorities of the proposed ACED algorithm.**

***Index Terms*—Dependent task offloading, directed acyclic graphs (DAGs), graph convolutional neural network (GCN), multi-access edge computing (MEC).**

I. Introduction

THE ACCELERATED improvement of mobile communication technology has promoted the emergence of new services and applications that require high performance, e.g., smart home, intelligent driving, and face recognition [1], [2]. Although the computing power of smart devices has made a qualitative leap in recent years, they still cannot support the execution of applications with the complex structure and a large amount of computation due to resource limitations, such as battery capacity and CPU computing power. Cloud computing contains abundant computing resources, but transferring massive data of mobile devices (MDs) to the cloud brings enormous transmission pressure and significant transmission delay to the whole network. Multi-access edge computing (MEC) has become a potential direction in the 5G scenario, which can reduce the pressure on the client-side [3], [4]. Compared with the long-distance transmission delay caused by cloud computing, MEC can quickly and efficiently provide computing services to users, relieving the computing pressure on the core network [5]. It provides computing and storage resources to execute user applications, reducing data exchange with the core network, energy consumption, and network latency. Thus, it can reduce the delay and enable real-time data processing and analysis [6]. Therefore, tasks can be encouraged to be performed on the MEC server to improve the Quality of Service (QoS).

With the advent of the 5G era, people have higher expectations for service quality. Computationally intensive applications with large computing resource requirements are required to be completed in a short time. MEC servers can provide substantial computing resources and interact with users to enhance the user experience. However, offloading the task to the MEC server causes additional communication delays, leading to the performance penalty. Thus, it is necessary to formulate a revolutionary end-edge-cloud cooperative offloading strategy by considering the delay and energy consumption. Recently, many researchers have made great efforts in task offloading decision making for multiusers [7]–[10] and single user [11], [12] in the MEC system.

All of the work mentioned before treats the user's tasks as atomic, performed locally or at the MEC servers. On the contrary, the parallel execution of multiple tasks can productively reduce the completion time of the application, making it more satisfying the people's needs. Increasingly complex IoT applications comprise a series of tasks that were initially designed for multithreaded parallel processing [13]–[15]. The dependencies between tasks can be modeled as a directed acyclic graph (DAG). Some studies put forward the offloading strategies of dependent tasks. The dependent task offloading problem of MEC is NP-hard, but most of the existing work used heuristic algorithms to get the offloading strategy [16]–[19]. However, the heuristic algorithm has many disadvantages, such as falling into the locally optimal solution. When the state and action space is too large, the algorithm's efficiency is very low. So it does not apply to increasingly complex MEC scenarios. Wang *et al.* [20] proposed a sequence to sequence offloading framework to study the offloading strategy of DAG application of a single user with one server, which is not practical in the real world. The encoder–decoder model makes training complex, and as the state space increases, the amount of calculations used in training has shown a substantial increase.

Manuscript received May 4, 2021; revised July 27, 2021; accepted August 19, 2021. Date of publication September 9, 2021; date of current version June 7, 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB2101901 and Grant 2019YFB2101903; in part by the National Science Foundation of China (Youth) under Grant 62072332 and Grant 62002260; in part by the China Postdoctoral Science Foundation under Grant 2020M670654; and in part by the State Key Laboratory of Communication Content Cognition Funded Project under Grant A32003. *(Corresponding author: Xiaofei Wang.)*

The authors are with the College of Intelligence and Computing, Tianjin University, Tianjin 300072, China (e-mail: tankcjw@tju.edu.cn; yjyang@tju.edu.cn; chenyangwang@tju.edu.cn; hengzhang@tju.edu.cn; chao.qiu@tju.edu.cn; xiaofeiwang@tju.edu.cn).

Digital Object Identifier 10.1109/JIOT.2021.3110412

In addition, they generated offloading decisions for all tasks of an application at a time without considering changes in the MEC environment during offloading.

Recently, graph neural networks (GNNs) have made meaningful progress in various research fields. In particular, it demonstrates its strong learning ability in the area of graph representation. The graph convolutional neural network (GCN) can effectively transmit node information through dependencies to better extract the depth features of the task for decision making. Besides, the MEC environment (MEC servers and user information) are also embedded into vectors to decide whether to offload the task. This article proposes an actor–critic mechanism with two embeddings for DAGs-based multiple dependent tasks computation offloading (ACED) to get the optimal computation offloading strategy of multiusers by jointly considering the dependency of tasks and the wireless interference of users.

The contributions of this article are placed in the following.

1) We establish multiple dependent tasks offloading problem with end-edge-cloud collaboration, which considers the computing workload and the dependency among tasks in the application. Tasks with dependencies are executed either locally or on the servers. Then, we model the dependent offloading problem in the MEC system with multiple users and multiple MEC servers as a Markov decision processing (MDP).
2) Considering the channel interference of multiple users and the dependency of tasks, a deep reinforcement learning (DRL)-based algorithm is proposed to reduce the average energy-time cost (ETC) of all users.
3) To capture the structure of the applications and the task features which are passed through the dependency, we use GCN to reinforce the proposed DRL model. Furthermore, we use a multilayer perception (MLP) to embed the MEC environment and user information into a vector to represent the state of the MEC server and user. The output of the model is the decision of the task to be decided at the current state. The results show the superiority of our algorithm in reducing average ETC.

The organization of our work is demonstrated as follows. The related work is introduced in Section II. Section III gives the system model and problem formulation. Section IV introduces the proposed DRL-based multitasks offloading scheme. Section V shows the simulation results.

II. Related Work

In the past few years, MEC was promoted due to MDs' popularity and user service requirements. We hereby review the studies of task offloading from three aspects in the following: 1) single user with single server; 2) multiusers with single server; and 3) multiusers with multiservers. Many studies investigated the offloading strategy of single server MEC system [21]–[25]. Mao *et al.* [11] and Liu *et al.* [26] modeled the computation offloading problem of one user with one MEC server as a binary problem and considered the choice of execution mode (execute locally or execute in the edge server). However, the work mentioned above only considered atomic offloading, so it cannot process different parts of the same task in parallel, which cannot effectively reduce the ETC. Additionally, many scholars studied the partial offloading problem in edge computing scenarios. For multiusers with a single server, Wang *et al.* [27] proposed a multiuser partial computation offloading algorithm (MPCO) to reduce energy consumption. In their article, tasks on a user device can be offloaded to a nearby user device for execution or to a MEC server to relieve the computing load on that user device. The multiuser offloading problem is modeled as a mixed-integer linear programming problem in [28], which is NP-hard, and a heuristic algorithm is proposed to solve the problem by combining edge computing with cloud computing. In [29], approximate dynamic programming techniques were used to solve the dynamic optimization problem. However, these work took no account of the topology of the task. Practically, an application is often made up of multiple tasks, and the output of some tasks is the input of others, so we cannot ignore the dependencies between tasks.

Some recent works focused on the dependent task offloading problem of the DAG structure. Yan *et al.* [30] proposed a DRL-based algorithm by jointly considering the offloading strategy and the resource optimization under dynamic wireless transmission channels. Yu *et al.* [31] proposed a model based on deep imitation learning by considering the collaboration of edge servers and cloud and behavioral cloning to minimize the cost. In the above studies, neural networks were applied to edge computing, but their usage scenarios and offloading problems differed from those in this article.

For the multiuser edge collaboration offloading problem, different from [32], [18] and [33] considered the dependencies between tasks. Han *et al.* [18] proposed a task offloading algorithm with dependencies by jointly optimizing latency and energy cost, a heuristic algorithm that considers both dependencies of tasks and priority. Shu *et al.* [33] proposed a collaborative earliest finish-time offloading (EFO) algorithm to coordinate task offloading strategy, which considers the competition of multiple users in wireless communication and computing resources. Most existing strategies were obtained through heuristic algorithms that easily fall into local optimally and fail to guarantee overall performance. This article proposes a DRL-based scheme to make offloading decisions, which can approach the optimal solution through continuous training and reduce training time.

III. Model

A. System Model

Fig. 1 shows the MEC system with multiple heterogeneous edge servers (containing heterogeneous resources). The main notations of this article are listed in Table I. The MEC servers are considered devices with sufficient computing power installed on a wireless access station (WAS) with solid storage computing capacity. The framework of this work consists of three layers, namely, the cloud layer, the edge layer, and the user layer.

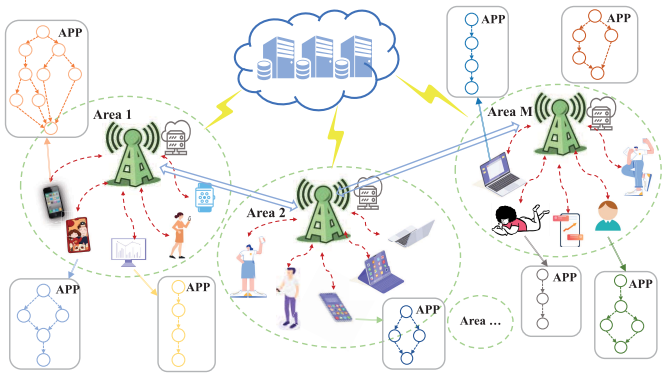


Fig. 1. System model of MEC.

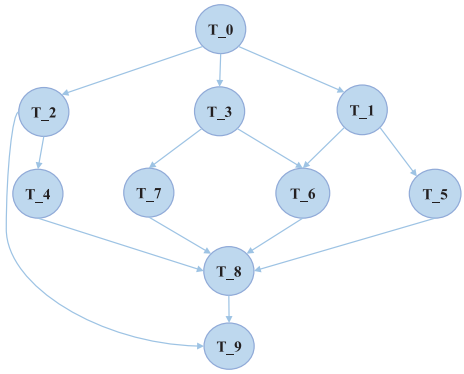


Fig. 2. Structure of an application.

TABLE I
NOTATIONS

Notation	Description
$\mathbb{S}$	The series of MEC servers
$\mathbb{N}$	The series of user devices
$\mathcal{V}_n$	The series of tasks of MD n
$v_{n,i}$	i-th task of MD n
$C_{n,i}$	The required CPU cycles of $v_{n,i}$
$P_{n,i}$	Data size of $v_{n,i}$
f_n^l	The local computation capability per core of MD n
f_s^e	The computation capability per core of edge server s
f_s^c	The computation capability per core of cloud
$T_{n,i}^l$	The local execution time of task $v_{n,i}$.
$e_{n,i}^l$	Energy cost of task processed locally.
$pred(v_{n,i})$	Predecessors of task $v_{n,i}$.
$succ(v_{n,i})$	Successors of task $v_{n,i}$.
$R_{n,i}^u$	Transmission rate of uploading the task $v_{n,i}$.
$T_{n,i}^{s,s'}$	The time of delivering the task $v_{n,i}$ from edge server s to s'.
$e_{n,i}^{s,s'}$	Transmission energy of delivering the task $v_{n,i}$ from edge server s to s'.
W	Bandwidth
$T_{n,i}^{s'}$	Time of task $v_{n,i}$ executing in the edge server s'.
$e_{n,i}^{s'}$	Energy cost of task $v_{n,i}$ executing in the edge server s'.
$T_{n,i}^c$	Time of task $v_{n,i}$ executing in the cloud.
$e_{n,i}^c$	Energy cost of task $v_{n,i}$ executing in the cloud.

The set of MD and MEC servers in the MEC system are denoted by $\mathbb{N} = \{1, 2, \ldots, n, \ldots, N\}$ and $\mathbb{S} = \{1, 2, \ldots, s, \ldots, S\}$, respectively. A set of mobile users are connected to the same MEC server. The MEC server can offload the task to another MEC server or the cloud center via wired transmission. Each mobile user has a computationally intensive application and decides on which devices to perform its tasks. Users can communicate directly with only one MEC server during execution. The edge servers cooperate through optical fiber transmission. Due to the competition between users for computing resources and communication resources, the state changes of the MEC system at different times should be considered during computing offloading.

B. Application Model

An application generated by user n can be modeled as a DAG $G_n = (\mathcal{V}_n, \mathcal{E}_n)$ which is partitioned into I tasks, where $\mathcal{V}_n = \{v_{n,i} | i = 1, 2, \ldots, I\}$ denotes the tasks of application and $\mathcal{E}_n = \{e_{v_{n,i}, v_{n,j}} | (i, j) \in \{1, 2, \ldots, I\} \times \{1, 2, \ldots, I\}\}$, $|\mathcal{E}_n| = e$ is the dependencies between tasks such that task $v_{n,i}$ should complete before task $v_{n,j}$ begins. As shown in Fig. 2, the task T_4 must start executing after T_2 finished. Each task $v_{n,i}$ is associated with a triple $(C_{n,i}, P_{n,i})$, where $C_{n,i}$ indicates the computing workload for accomplishing the task and $P_{n,i}$ represents the processing data (e.g., source codes and parameters) size (e.g., source codes) of the task $v_{n,i}$. Especially, tasks 0 and $I+1$ are defined as entry and exit tasks. Note that the workload of these two virtual tasks can be set as $C_{n,0} = C_{n,I+1} = 0$.

We use $\alpha_{n,i} \in \{0\} \cup \mathbb{S} \cup \{|\mathbb{S}| + 1\}$ to indicate the offloading strategy of task $v_{n,i}$. $\alpha_{n,i} = 0$ indicates the MD n performs the task $v_{n,i}$ locally. Furthermore, $\alpha_{n,i} \in \mathbb{S}$ and $\alpha_{n,i} = S + 1$ represent task $v_{n,i}$ is executed in MEC server s and cloud, respectively. For each user n, $\alpha_{n,0} = \alpha_{n,I+1} = 0$.

To better represent the computing and communication model, some definitions are proposed as follows.

Definition 1: $RT_{n,i}^l$, $RT_{n,i}^s$, $s \in \{1, 2, \ldots, S\}$ and $RT_{n,i}^c$ represent the ready time of task $v_{n,i}$ when it is executed locally, at the server s and at cloud center, respectively.

Definition 2: $FT_{n,i}^l$, $FT_{n,i}^s$, $s \in \{1, 2, \ldots, S\}$ and $FT_{n,i}^c$ represent the finish time of task $v_{n,i}$ when it is executed locally, at the server s and at cloud center, respectively.

Since the downlink transmission speed is high, the downlink transmission time and energy in this article can be ignored.

C. Local Execution Model

Assumed that MD is equipped with ρ_l^n CPU cores, and each CPU core can only serve one task simultaneously. The minimum completion time of tasks are executed in user device n at a time is $FT\_\rho_l^n$. If there is an idle core in MD n, $FT\_\rho_l^n = 0$.

The ready time of $v_{n,i}$ is obtained by

$$RT_{n,i}^l = \max_{v_{n,j} \in \text{pred}(v_{n,i})} \max \left\{ FT_{n,j}^l, FT_{n,j}^s, FT_{n,j}^c, FT\_\rho_l^n \right\} \quad (1)$$

the pred$(v_{n,i})$ is the predecessors of task $v_{n,i}$. Task $v_{n,i}$ do not begin until all of its immediate predecessors $v_{n,j} \in \text{pred}(v_{n,i})$ have been completed. The processing time $T_{n,i}^l$ of task $v_{n,i}$ on the local CPU core depends on the actual operating frequency f_n^l by $T_{n,i}^l = (C_{n,i}/f_n^l)$. All needed data is available at $RT_{n,i}^l$, so that the finish time of task $v_{n,i}$ is obtained as $FT_{n,i}^l = RT_{n,i}^l + T_{n,i}^l$ by the execution time. The energy cost of task $v_{n,i}$ is $e_{n,i}^l = \kappa_n C_{n,i} (f_n^l)^2$, where κ_n is the switched capacitance [4], [34].

D. Edge Execution Model

Assumed that each MEC server is equipped with ρ_s CPU cores. This article uses decision order to approximate the server execution order. First, let a_n denote whether the user device n offload the task to edge server or cloud server. a_n is obtained by

$$a_n = \begin{cases} 0, & \text{the application executed locally} \\ k, & \text{at least one task is offloaded} \end{cases} \tag{2}$$

where the server k is the server which the user n directly connected with.

The process of task offloading in this model can be divided into three stages.

1) The task $v_{n,i}$ is first transmitted through the channel to the server directly connected to user n. The transmission time in this stage is denoted as $T^{\text{trans}}_{n,i}$, which is calculated as $T^{\text{trans}}_{n,i} = (P_{n,i}/R^u_{n,i})$. Where $R^u_{n,i}$ is the uplink rate, and it is calculated as

$$R^u_{n,i} = W \log_2\left(1 + \frac{\mathcal{P}_{n,i} g_{n,i}}{\sigma^2 + \sum_{n' \in \mathbb{N}, a_{n'} = a_n} \mathcal{P}_{n',i'} g_{n',i'}}\right) \tag{3}$$

where W is the channel bandwidth, and the $\mathcal{P}_{n,i}$ is the transmission power of user device n to upload the task $v_{n,i}$. $g_{n,i}$ represent the channel gain between user n and the corresponding edge server when transmit the task $v_{n,i}$. This article sets the channel gain as the -4th power of the distance between the user device and the related edge server. The energy consumption of this stage is $e^s_{n,i} = \mathcal{P}_{n,i} \times T^{\text{trans}}_{n,i}$.

2) MEC server s should transmit the task $v_{n,i}$ to the target MEC server s' through fiber, and the time of this stage is represented as $T^{s,s'}_{n,i}$, which can be obtained by $T^{s,s'}_{n,i} = (P_{n,i}/R_{s,s'}) \times h$. $R_{s,s'}$ is the transmission rate from server s to server s' and h is the number of hops between s and s'. Thus, the energy of this stage is calculated as $e^{s,s'}_{n,i} = \mathcal{P}_{n,i} \times T^{s,s'}_{n,i}$. Especially, $h = 0$ if $s = s'$.

3) Task $v_{i,j}$ is performed in the target server when the task reaches and has an idle CPU core to process it. So the ready time of task $v_{n,i}$ performed in this situation is calculated as $RT^{s'}_{n,i} = \max\{\max_{v_{n,j} \in \text{pred}(v_{n,i})} \max\{FT^l_{n,j}, FT^s_{n,j}, FT^c_{n,j}\} + T^{\text{trans}}_{n,i} + T^{s,s'}_{n,i}, \rho_s\}$. The actual execution time for the MEC server to execute task $v_{n,i}$ is calculated as $T^{s'}_{n,i} = (C_{n,i}/f^e_{s'})$, thus $FT^{s'}_{n,i}$ can be obtained by $FT^{s'}_{n,i} = RT^{s'}_{n,i} + T^{s'}_{n,i}$. In this case, the energy consumed by user device n can be obtained by $e^{s'}_{n,i} = \mathcal{P}^{\text{wait}}_n \times T^{s'}_{n,i}$, where the $\mathcal{P}^{\text{wait}}_n$ is the waiting power of a user device while performing a task on the edge server.

E. Cloud Execution Model

Different from the execution on edge, the time of transferring the task $v_{n,i}$ from the edge server s to the cloud center can be calculated as $T^{s,c}_{n,i} = (P_{n,i}/R_{s,c})$. Assumed that the computation capability of the cloud center has much more robust than that of the edge server, so the tasks can be executed immediately when they arrive in the cloud center.

The ready time of execution in the cloud can be calculated as

$$RT^c_{n,i} = \max_{v_{n,j} \in \text{pred}(v_{n,i})} \max\left\{FT^l_{n,j}, FT^s_{n,j}, FT^c_{n,j}\right\} + T^{\text{trans}}_{n,i} + T^{s,c}_{n,i}. \tag{4}$$

Denote the CPU capable of the cloud as f^c, so that the execution time in the cloud is $T^c_{n,i} = (C_{n,i}/f^c)$. Thus, the finish time when the task $v_{n,i}$ executed in the cloud center is $FT^c_{n,i} = RT^c_{n,i} + T^c_{n,i}$. The energy consumed by performing tasks in the cloud is $e^c_{n,i} = \mathcal{P}^{\text{wait}}_n \times T^c_{n,i}$.

F. Problem Formulation

In this article, we aim to find the near-optimal offloading strategy $\alpha_n = [\alpha_{n,0}, \alpha_{n,1}, \ldots, \alpha_{n,I+1}]$ for each MD n to minimize the average ETC of all users, where $\alpha_{n,i} \in \{0, 1, \ldots, S+1\}$. The ETC of each user n is defined as

$$\text{ETC}_n = w_t \times \left(FT^l_{n,I+1} - FT^l_{n,0}\right) + w_e \times E. \tag{5}$$

The objective of this article is as follows:

$$\min_{\alpha_n} \frac{\sum_{n=1}^{N} \text{ETC}_n}{\mathbb{N}} \tag{6}$$

where $|\mathbb{N}|$ is the number of MD in this MEC system, and E is the total energy consumption of all users.

IV. DRL-Based Multitasks Offloading Algorithm

In this section, the DRL-based multitasks offloading algorithms and GNN-based task embedding method are introduced.

A. DRL-Based Framework

We first proposed a framework based on DRL and GNN to get an approximately optimal offloading strategy for dependent tasks in the multiple users with multiple MEC servers scenarios. The framework is displayed in Fig. 3. At each decision-making time t, the scheduling agent observes the MEC system state S_N as well as the state of current decision-making task $S_{v_{n,i}}$ to select an offloading action and get a reward based on the custom objective. In this article, GNN layers are used to get the embedding of the task to be decided in time t, and MLP layers are used to obtain the embedding of the current MEC system. The agent uses MEC embedding and task embedding for the policy network, which outputs actions. Performing an action A_t in the current state S_t results in a new state S_{t+1}.

B. State Embedding

The MEC environment state is based on the calculation of task and energy overhead, and it is only related to the features of edge servers, the state of tasks, and the status of MD. To extract adequate information, we first embed the MEC environment state and the task information that needs to be decided, respectively.

MEC Embedding: At each time, we extract the information of the MEC environment to make the offloading decision for the current task. The MEC environment status information

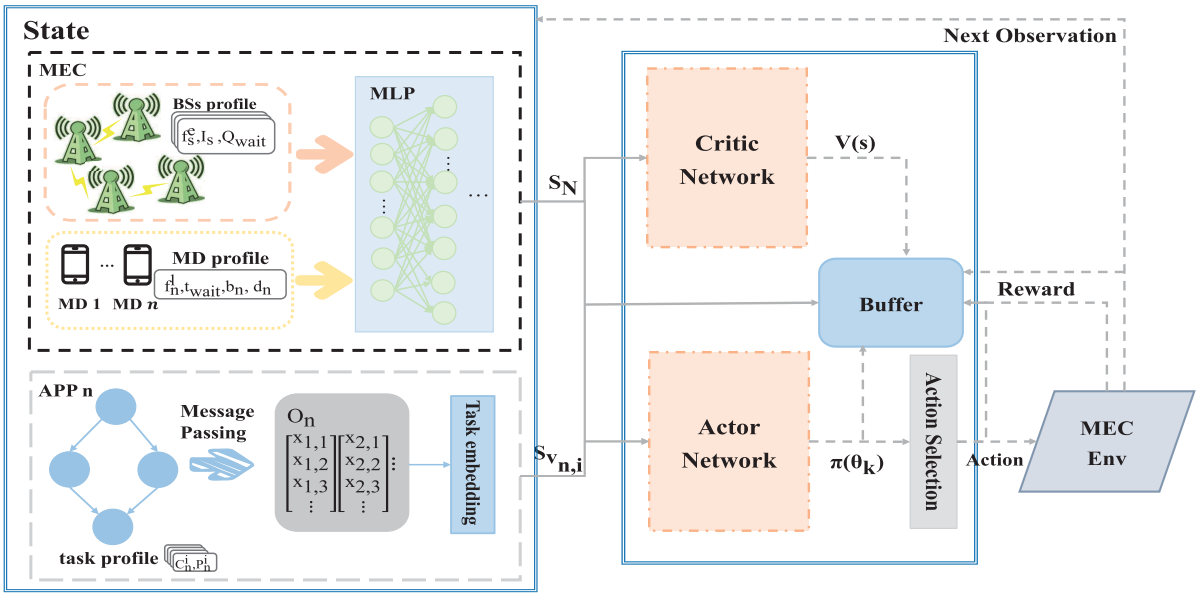


Fig. 3. DRL-based edge computation offloading framework.

includes two parts: 1) edge server status and 2) user status. For edge server s, the CPU computing capable, whether it is the server that the current user is directly connected to, bandwidth and task waiting queue are necessary to make decisions. Likewise, if the task is offloaded to the MEC server, the channel information should be considered. Therefore, the current overall channel gain of the corresponding edge server also belongs to the MEC environment state information. Besides, user information also has a significant influence on the offloading decision. The makespan of the tasks that the current user has made decisions, the computing capabilities of the user device, the distance of the current user from the corresponding server, and the estimated completion time of the task being performed by the current user constitute the user information. The MEC servers' information and user's information compose the necessary information for the MEC environment. The features extracted from the MEC servers and users are fed into the MLP to learn the embedding of the whole MEC environment. In order to better learn the embedding of the MEC system, the MLP we used contains two hidden layers and an output layer, and each layer contains 512 neurons.

Task Embedding: Considering the dependencies between the tasks, we use a GNN network, to efficiently capture the overall structure of the application and better extract the information needed to make the offloading decision for task $v_{n,i}$. Our method is based on GCN [35]–[37], which is used for task embedding. That is to say, the embedding learned through the GNN can be considered as the feature of the task, and this feature extraction does not require manual feature engineering. Aforementioned, given a DAG $G_n = (\mathcal{V}_n, \mathcal{E}_n)$ of an application, where $\mathcal{V}_n$ is a set of tasks and $\mathcal{E}_n$ represents the dependencies among the application. For each task of application G_n, we use $v_{n,i}.f$ to represent the feature of task $v_{n,i}$. So the features of all tasks in the application G_n can be represented as

$$F_n = \left[v_{n,1}.f, v_{n,2}.f, \ldots, v_{n,I}.f\right]. \tag{7}$$

For each task $v_{n,i}$, the number of feature dimensions is 2 ($d = 2$). We use an adjacency matrix A_n to represent adjacency relations of G_n. The state of task $v_{n,i}$ can be represented as $S_{v_{n,i}} = [v_{n,i}.f, A_n]$.

In this part, our target is to output a task-level embedding $O_n \in \mathbb{R}^{I \times D}$, where D is the dimensions of embedding per task, and I is the task number generated by user n, taking F_n and A_n as the input. O_n contains all the information for the application requested by user n, and each row of $O_{n,i}$ represents the embedding vector of the tasks $v_{n,i}$.

The GNN based on a spectral-domain method cannot be used to solve the node representation of the digraph because the Laplace matrix is an asymmetric matrix. So we can only use spatial domain-based GNN to learn the embedding vector of the task. The network layer-wise propagation rules of GCN are represented as

$$H_{n,i}^{l+1} = \sigma\left(\sum_{v_{n,j} \in \mathcal{N}(v_{n,i})} H_{n,j}^{l} W^{l} + H_{n,i}^{l}\right). \tag{8}$$

Note that $H_{n,i}^{0} = F_{n,i}$, $O_n^i = H_{n,i}^{L}$, L is the GCN layers' number. W^l is the weight parameter matrix of the lth network layer and $\sigma(\cdot)$ is a nonlinear activation function, e.g., Sigmoid and Tanh.

Next, the task $v_{n,i}$'s embedding is calculated as

$$E_{S_{v_{n,i}}} = \sigma\left(\sum_{v_{n,j} \in \mathrm{succ}(v_{n,i})} O_n^j\right) \tag{9}$$

where $\mathrm{succ}(v_{n,i})$ denote the set of successor tasks of task $v_{n,i}$.

The flow of task embedding is shown in Fig. 4. As shown in Fig. 4, given the structure of the application G_n and the features F_n of all tasks, tasks can pass information between each other through interdependencies. If we want to get the task embedding of task $v_{n,0}$, we first transfer the information from task 4 to tasks 2 and 3, and then the information of tasks 2 and 3 is delivered to task 1 to get the embedding vector of task 1. Finally, task 1 passes the information to task 0 to get

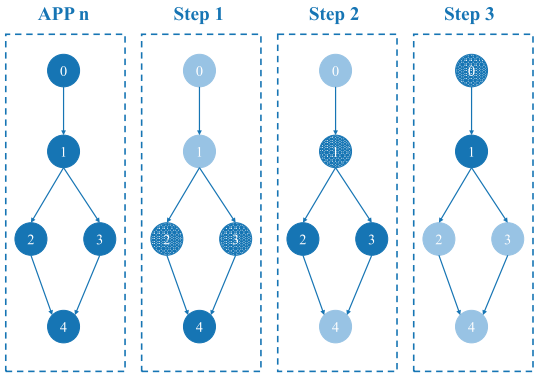


Fig. 4. Task embedding.

Algorithm 1 Task Embedding Algorithm

Input: $\mathcal{G}_n = (\mathcal{V}_n, \mathcal{E}_n)$, task feature F_n, A_n, current task $v_{n,i}$.
Output: task embedding vector $E_{S_{v_{n,i}}}$.
$H_n^0 = F_n$
for $1 =, 1, 2, \ldots$ **do**
  $H_n^{l+1} = \sigma(\sum_{v_{n,j} \in succ(v_{n,i})} H_j^l W^l + H_j^l)$
end for
$O_n = H_n^L$
for $v_{n,i} \in pred(v_{n,j})$ **do**
  calculate the task embedding of $v_{n,i}$ using (9)
end for
Output $E_{S_{v_{n,i}}}$

the final embedding of $v_{n,0}$. The state at decision step t can be denoted as

$$S_t = \left[S_N, S_{v_{n,i}}\right] \tag{10}$$

where the state of the MEC system is represented as S_N. The state embedding can be denoted as $E_{S_t} = [E_{S_N}, E_{S_{v_{n,i}}}]$. Input the embedding vector at time t into the policy network can make the offloading action selections accordingly.

Algorithm 1 shows the specific task embedding algorithm.

C. Action Space

Within each decision step t, there is one and only one task that can be decided. The action A_t at current time t is defined as follows:

$$A_t = 0, 1, 2, \ldots, |\mathbb{S}| + 1. \tag{11}$$

According to S_t, offloading decisions need to be made for the currently decisional task. $A_t = 0$ means that this task is performed locally and $A_t = \{1, 2, s, \ldots, |\mathbb{S}| + 1\}$ represent the corresponding user offloads task to the edge server s or cloud center.

D. Reward

At each decision step, there is a task that needs to be decided. Given any state S_t, the ETC of all users at time step t in this MEC system can be obtained. Take the action A_t, the state of the MEC system is changed from S_t to S_{t+1}. The reward at the time step t is defined as the negative increment of the sum of ETC of all users in the MEC system, which is calculated as

$$r_t = \sum_{n=1}^{N} \mathrm{ETC}_n^t - \sum_{n=1}^{N} \mathrm{ETC}_n^{t+1}. \tag{12}$$

E. Actor–Critic Framework

In this work, the ACED algorithm is used to train the model, which is the basic framework of many DRL-based algorithms, such as proximal policy optimization (PPO) [38]. First, the following is how the ACED exploits the actor–critic framework to deal with the multitasks computation offloading problem.

As shown in Fig. 5, the actor–critic framework consists of two parts: 1) an actor–network for action selection and 2) a critic-network for evaluating the value of state. The overall goal of our model is to find the best strategy to maximize the sum of discount rewards $\hat{R}_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$. Using the discount factor to calculate the reward can reduce the loss of the execution of the task that arrives at the target server first and waits for the task which is decided first but arrives later. In this case, there is little effect on the overall situation. Therefore, the global impact of asynchrony can be approximately eliminated.

The purpose of the actor network is to sample an action for the task that can be selected at the current decision step t. In this model, the input of the actor network includes the information of edge nodes, user's profile, and the task's information to be decided. According to the state of the environment, the actor network outputs the possibilities of all actions. Sample an action, and apply the action to the MEC environment can get the reward r_t and the state of the next step s_{t+1}. The critic network is similar to the actor network, which contains a GCN layer to capture the DAG structure of the application and a DNN to aware of the MEC environment, then we use MLP with one node output to evaluate. This work uses Tanh as an activation function. The state S_t of the current task at current decision step t is used as the input of the critic network, and the output is the value of the current state.

Next, we introduce the process of updating the network. This work uses a mean-square objective, $\mathcal{L}^{MSE} = \mathbb{E}_{\pi_{\theta_{\text{old}}}}[V(t) - \hat{R}_t]^2$, to measure the difference between the value of State t and $\hat{R}_t$. Compared with other policy networks, PPO algorithms use clipping parameters to limit the changing of each network update. It allows one to perform small batches of training for multiple epochs using the collected experience of the current iteration. So the loss function of the Policy (Actor) network is defined as follows:

$$\mathcal{L}^{\text{loss}} = \mathbb{E}_{\pi_{\theta_{\text{old}}}}\left[\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t, g\left(\epsilon, \hat{A}_t\right)\right)\right] \tag{13}$$

$$g\left(\epsilon, \hat{A}_t\right) = \begin{cases} (1+\epsilon)\hat{A}_t, & \hat{A}_t \geq 0 \\ (1-\epsilon)\hat{A}_t, & \hat{A}_t \leq 0 \end{cases} \tag{14}$$

$$\hat{A}_t = Q_\theta(s_t, a_t) - V(t). \tag{15}$$

The proposed ACED algorithm is shown in Algorithm 2. The time complexity analysis is shown as follows. According

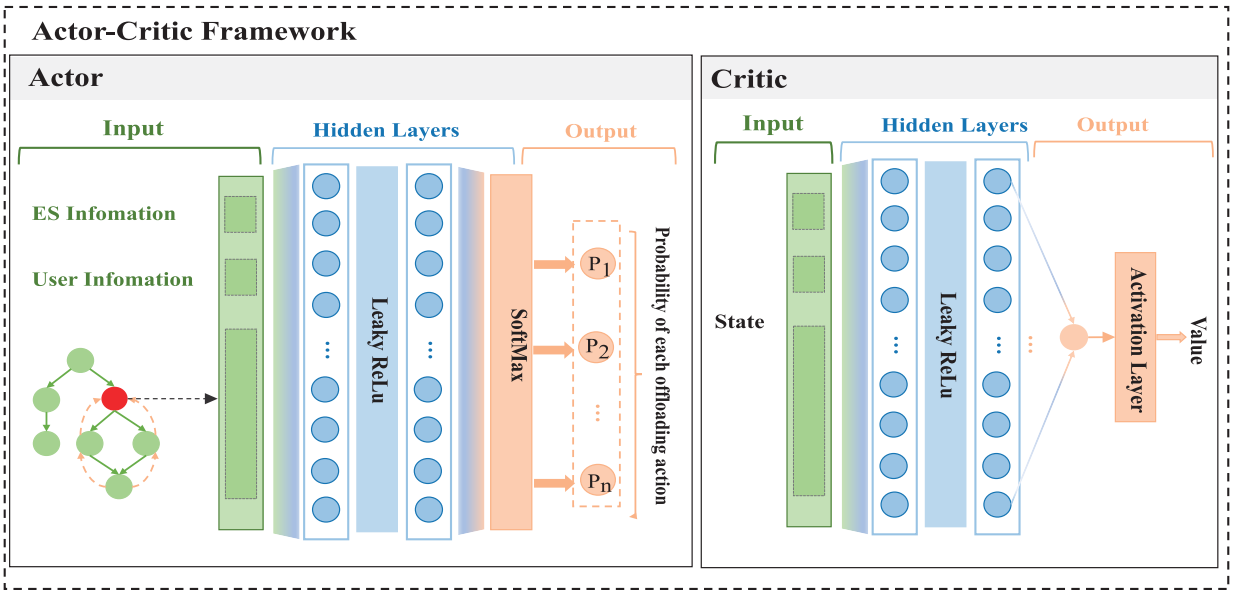


Fig. 5. Actor–critic framework.

Algorithm 2 ACED Algorithm

1: **Input:** $\mathcal{G}_n = (\mathcal{V}_n, \mathcal{E}_n)$, initial actor parameters θ_0, initial critic parameters ϕ_0 .
2: **for** k = 1, 2, . . . , K **do**
3: **while** not all tasks have been decided **do**
4: Get state S_t by (10)
5: Get action by running policy $\pi_k = \pi(\theta_k)$
6: Get reward R_t from MEC environment
7: Store (S_t, A_t, R_t) into memory
8: **end while**
9: Compute discounted rewards $\hat{R}_t$
10: Compute advantage estimate, $\hat{A}_t$, by Eq. (15)
11: Update Actor network by Eq. (13)
12: Update Critic network by $\mathcal{L}^{MSE}$
13: **end for**

to Algorithm 2, there are two loops (lines 2 and 3). The outer loop contains K episodes, and the inner loop makes decisions about all the tasks in that episode. Assume n_t is the number of time steps per episode, so the time complexity of the ACED algorithm is $O(Kn_t)$.

V. Simulation

In this section, we carry out a lot of simulations to evaluate the performance of the ACED algorithm. First, the simulation settings are given, and then the experimental results are presented to verify the strength of the ACED algorithm in minimizing the average ETC of all users in the MEC system.

A. Simulation Setting

First, a MEC system with end-edge-cloud collaboration is simulated in the python environment. The MEC system of this work consists of multiple users, multiple WASs which are equipped with MEC servers and a cloud center. The main parameters of simulations in this work are listed in Table II.

TABLE II
Main Simulation Parameters

Parameters	Value
f_n^l	1.0-1.2GHz uniformly
f_s^e	2.4-2.5GHz uniformly
f_s^c	3.0GHz
W	20MHz
κ_n	10^{-27}
$\mathcal{P}_{n,i}$	1.5W
$\mathcal{P}_n^{wait}$	0.3W
$P_{n,i}$	300KB-500KB uniformly
D_n	30-80m uniformly
γ	0.85

Similar to [39], we set the following baseline algorithms to measure the performance of the ACED algorithm.

1) *OLNA:* All tasks of all users are executed locally.
2) *CFOA:* All tasks of all users are offloaded to the cloud.
3) *EFOA:* All tasks of all users are offloaded to edge servers.
4) *Random:* Random selection of offloading strategies for each task.
5) *Greedy:* Each task is greedily selected to be executed locally, the MEC server or cloud center based on the weighted sum of its estimated finish time and energy consumption.

B. Simulation Results

1) Convergence Performance: Fig. 6 shows the change curve of average ETC consumed by all users of the current episode to complete the application with the increase of training episode in different learning rates ($lr = 5e-6, 8e-6, 1e-5, 2e-5, 5e-5, 8e-5$). When the learning rate is too low, the convergence speed is plodding, and the convergence value

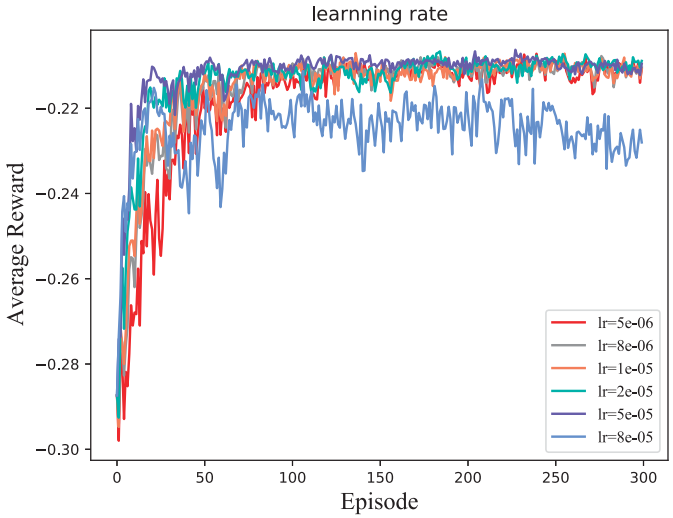


Fig. 6. Reward with different learning rate.

is a little small compared to the high learning rate. A high learning rate makes the reward reach the convergence value quickly. However, the reward is difficult to converge when the learning rate is too high, and the optimal solution may be skipped because of the high learning rate. Comparing of different learning rates shows that the best convergence performance with the learning rate is $5e-5$. Thus, the learning rate is set to $5e-5$ in the following simulation for better performance demonstration.

2) Performance of Different User Number: Assuming that the energy cost and the latency are equally crucial to the user experience, so the weights of latency and energy are both set to 0.5. In this experiment, all users are randomly distributed around three MEC servers. Each user has an application to execute, and each app has 8–10 tasks. The number of users grows from 15 to 35, and the application's structure is randomly generated, but the depth of DAGs is limited to 5. As shown in Fig. 7, the effect of ACED is better than the baselines, and it can balance delay and energy consumption well. Even though the impact of the greedy algorithm is similar to that of ACED, all states of the entire MEC system must be obtained when the greedy algorithm is used for each step of the calculation, and the reward of all actions need to be calculated when selecting action which brings much extra calculation. With the increase of users and MEC servers, the amount of computation has exploded. Therefore, it is unrealistic in actual applications.

We can observe that the ETC of the ACED algorithm increases when the number of users increases. It is mainly due to the rise in the number of users served by each MEC server, which leads to more intense channel competition, resulting in longer transmission time and greater transmission energy consumption. ETC is the weighted sum of delay and energy consumption so that the average ETC of all users increases. When the number of users increases to 30 due to the limited channel bandwidth of each base station, the vast transmission pressure causes a massive increase in the average ETC when tasks need to be offloaded to the edge server. As shown in Fig. 7(a) and (b), we can see that performing tasks locally can cause significant computing latency and consume more energy. Offloading the tasks to the cloud center can reduce latency because of powerful computing capacity. However, the transmission energy consumption of users increases when the number of users growing. So, the ETC of all users is still at a high level. As shown in Fig. 7(c), the proposed ACED algorithm effectively balances latency and energy cost.

3) Performance of Different MEC Server Number: Same as the section above, the weight of latency and energy is set as 0.5 to calculate ETC. Fig. 8 shows the performance of different algorithms when the number of edge servers changes from 2 to 6. There are 35 users in the MEC system. From Fig. 8(c), we can see that the average ETC of the ACED algorithm decreases with the growth of edge servers. It can be observed that ACED can perform best in these algorithms. As shown in Fig. 8(a), we can see that the latency of local execution far exceeds that of at the edge, and with the increase of MEC servers, the latency gap is growing. The cloud center has powerful computing capabilities, wherein the offloaded tasks are executed fast in a relative period, leading to the low-latency performance, which is close to the ACED algorithm. When the number of MEC servers increases, the number of users per MEC service decreases, relieving the communication pressure, and the application's latency drops rapidly. Fig. 8(b) shows that the ACED algorithm is better than CFOA, OLNA, and Random algorithm in energy but worse than the EFNA algorithm. It can be found that offloading the tasks to the edge server can reduce the energy consumed by the user device.

4) Performance of Different Task Number: First, the depth of application is set as 5 in this experiment. The greater the depth, the longer the maximum serial link. Fig. 9 shows the impact of varying task sizes on performance. In this MEC system, there are 3 edge servers and 20 users. With tasks of single application increases, the average ETC, energy, and latency of all algorithms are increasing. In the case of a certain depth, the greater the number of tasks for each application, the greater the degree of parallelism of the tasks, so the time to wait for the finish time of the predecessor task becomes longer. Also, we can find that, compared with other algorithms, the ETC of ACED algorithms grows more slowly as the number of tasks grows. The results approve that the ACED algorithm shows superiority compared to other algorithms, which can get near-optimal results.

5) Performance of Different Depth of Each Application: This experiment mainly reflects the impact of the depth of each application on the average ETC of each user in the MEC environment. The number of users in the MEC environment and the number of tasks for each application are 20 and 18–20, respectively. As shown in Fig. 10, the ACED algorithm generally outperforms several other algorithms, and the average ETC executed by each application in the MEC system is the smallest. As the depth increases, the average ETC of ACED, CFOA, EFOA, Random, and Greedy algorithms increases. When the total number of tasks remains the

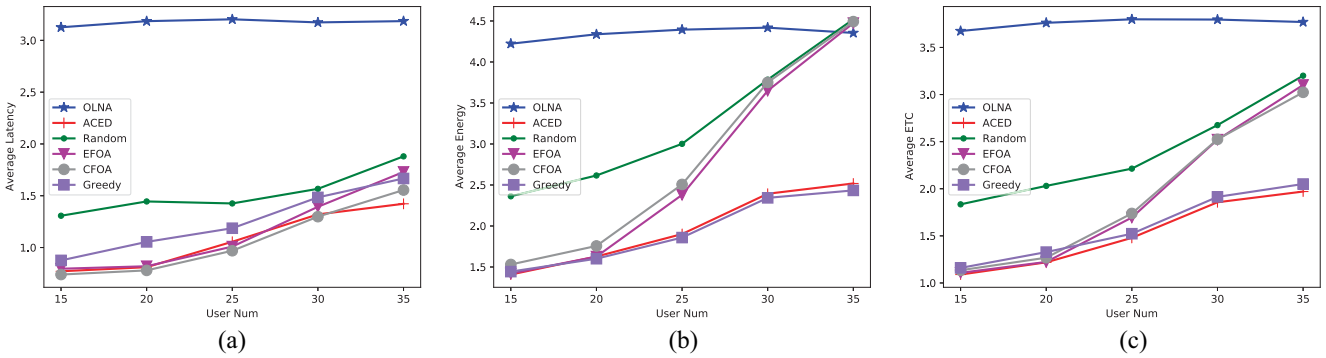


Fig. 7. Average latency, energy, and ETC with different user number. (a) Latency with different user number. (b) Energy with different user number. (c) ETC with different user number.

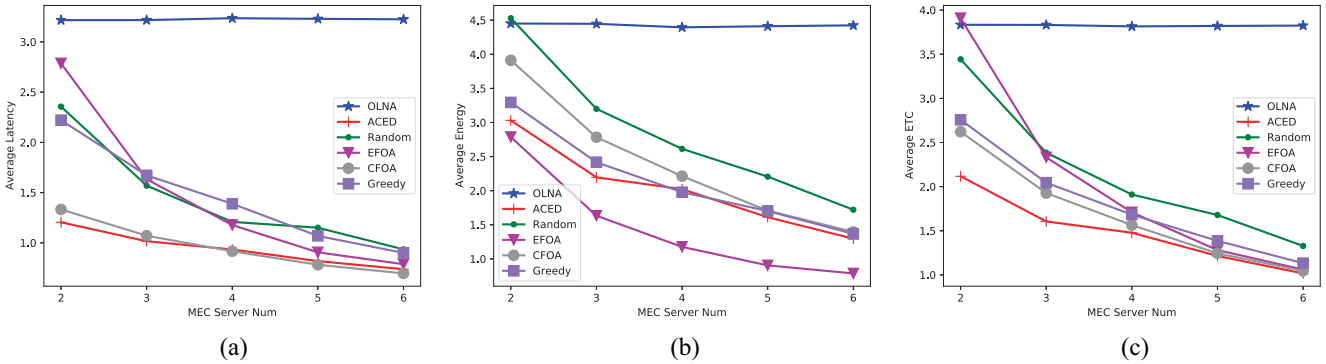


Fig. 8. Average latency, energy, and ETC with different server number. (a) Latency with different server number. (b) Energy with different server number. (c) ETC with different server number.

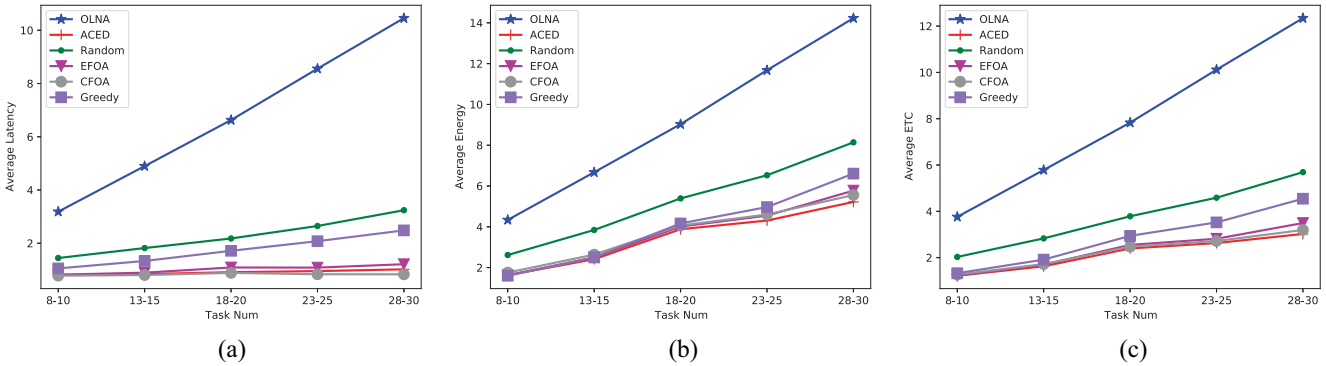


Fig. 9. Average latency, energy, and ETC with different task number. (a) Latency with different task number. (b) Energy with different task number. (c) ETC with different task number.

same, the growth in depth means that the degree of parallelism decreases, and the waiting time and energy required for task execution increase. However, as the depth changes, the average ETC of each user calculated by the OLNA algorithm is basically unchanged. This is mainly because the user device can only execute one task simultaneously, so all tasks must be completed sequentially, regardless of depth.

6) Performance of Different Weight of Latency and Energy: In this simulation, the MEC system includes 25 users and 3 MEC servers, and each application has 8–10 tasks. w_t indicates the importance of latency when generating the offloading strategy. The larger the w_t is, the action should be chosen to reduce latency. As shown in Fig. 11(a) and (b), it proves that as the weight of latency increases, the average latency of all users decreases while the average energy consumption increases. It can be found that as the weight of latency increases, the number of tasks performed locally decreases, and the number of tasks offloaded to the cloud center increases from Fig. 11(c). This is mainly because the computing power in user devices is much smaller than that of edge servers and the cloud. The cloud center has a solid ability to execute tasks, but offloading more tasks to the cloud causes more energy consumption on user devices. Therefore, using end-edge-cloud collaboration for application execution can effectively balance latency and energy consumption.

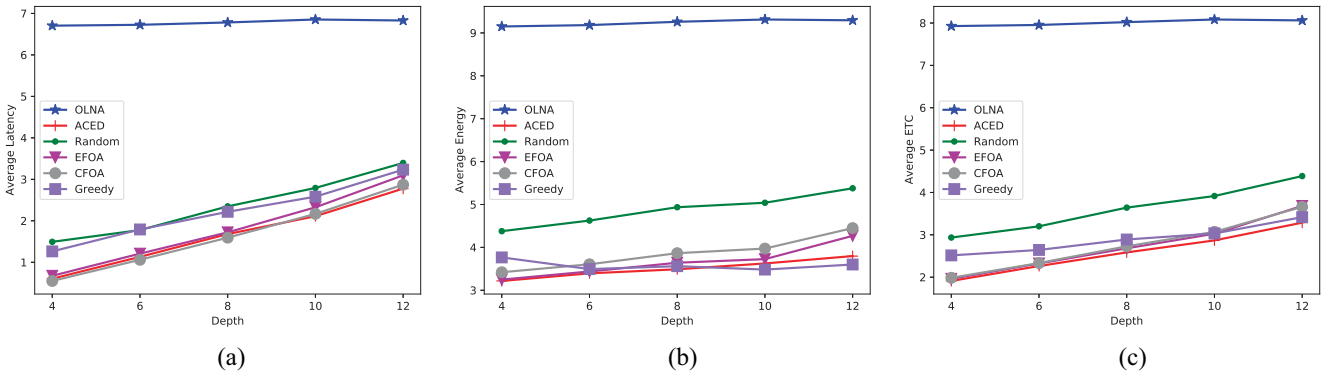


Fig. 10. Average latency, energy, and ETC with different depth. (a) Latency with different depth. (b) Energy with different depth. (c) ETC with different depth.

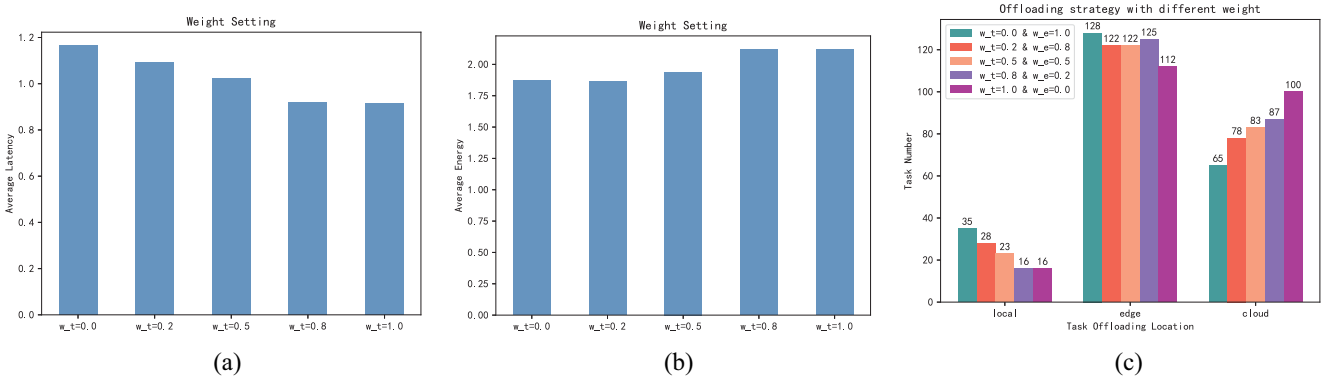


Fig. 11. Average latency, energy, and offloading strategy with different weight. (a) Latency with different weight. (b) Energy with different weight. (c) Offloading strategy with different weight.

VI. Conclusion

This article built an actor–critic mechanism with two embedding layers for DAG-based multitasks computation offloading strategy in the MEC system. A DRL-based algorithm is proposed to reduce the average ETC of all users by jointly considering the structure of the application and the wireless interference of user transmission. We conducted a simulation experiment to measure the network performance, which proves the priorities of our proposed ACED algorithm. The results of simulations show the superiority of the ACED algorithm in reducing the average ETC of users compared with existing work and can get an approximate optimal solution.

References

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

[2] D. Mazza, A. Pagès-Bernaus, D. Tarchi, A. A. Juan, and G. E. Corazza, "Supporting mobile cloud computing in smart cities via randomized algorithms," *IEEE Syst. J.*, vol. 12, no. 2, pp. 1598–1609, Jun. 2018.

[3] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," ETSI, Sophia Antipolis, France, White Paper, 2015.

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

[5] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[6] C. Avasalcai, I. Murturi, and S. Dustdar, "Edge and fog: A survey, use cases, and future challenges," in *Fog Computing: Theory and Practice*. Hoboken, NJ, USA: Wiley, 2020, pp. 43–65.

[7] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[8] C. You and K. Huang, "Multiuser resource allocation for mobile-edge computation offloading," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Washington, DC, USA, 2016, pp. 1–6.

[9] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 726–738, Sep./Oct. 2019.

[10] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.

[11] 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.

[12] X. Gu, C. Ji, and G. Zhang, "Energy-optimal latency-constrained application offloading in mobile-edge computing," *Sensors*, vol. 20, no. 11, p. 3064, 2020.

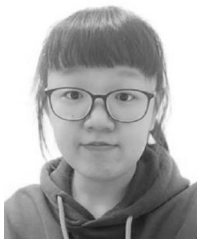
[13] Z. Wang, Z. Zhao, G. Min, X. Huang, Q. Ni, and R. Wang, "User mobility aware task assignment for mobile edge computing," *Future Gener. Comput. Syst.*, vol. 85, pp. 1–8, Aug. 2018.

[14] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[15] W. Sun, J. Liu, and H. Zhang, "When smart wearables meet intelligent vehicles: Challenges and future directions," *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 58–65, Jun. 2017.

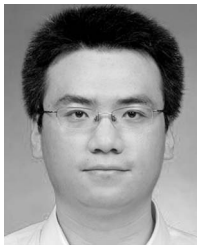
[16] J. Lee, H. Ko, J. Kim, and S. Pack, "Data: Dependency-aware task allocation scheme in distributed edge clouds," *IEEE Trans. Ind. Informat.*, vol. 16, no. 12, pp. 7782–7790, Dec. 2020.

[17] V. De Maio and I. Brandic, "First hop mobile offloading of DAG computations," in *Proc. 18th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput. (CCGRID)*, Washington, DC, USA, 2018, pp. 83–92.
[18] Y. Han, Z. Zhao, J. Mo, C. Shu, and G. Min, "Efficient task offloading with dependency guarantees in ultra-dense edge networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Waikoloa, HI, USA, 2019, pp. 1–6.
[19] C. Shu, Z. Zhao, Y. Han, G. Min, and H. Duan, "Multi-User offloading for edge computing networks: A dependency-aware and latency-optimal approach," *IEEE Internet Things J.*, vol. 7, no. 3, pp. 1678–1689, Mar. 2020.
[20] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, "Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning," *IEEE Commun. Mag.*, vol. 57, no. 5, pp. 64–69, May 2019.
[21] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020.
[22] M. Zeng, W. Hao, O. A. Dobre, Z. Ding, and H. V. Poor, "Massive MIMO-assisted mobile edge computing: Exciting possibilities for computation offloading," *IEEE Veh. Technol. Mag.*, vol. 15, no. 2, pp. 31–38, Jun. 2020.
[23] W. Sun, H. Zhang, L. Wang, S. Guo, and D. Yuan, "Profit maximization task offloading mechanism with D2D collaboration in MEC networks," in *Proc. 11th Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Xi'an, China, 2019, pp. 1–6.
[24] F. Song, H. Xing, S. Luo, D. Zhan, P. Dai, and R. Qu, "A multiobjective computation offloading algorithm for mobile-edge computing," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8780–8799, Sep. 2020.
[25] J. Zhang *et al.*, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2633–2645, Aug. 2018.
[26] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Barcelona, Spain, 2016, pp. 1451–1455.
[27] Y. Wang, M. Sheng, X. Wang, and J. Li, "Cooperative dynamic voltage scaling and radio resource allocation for energy-efficient multiuser mobile edge computing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kansas City, MO, USA, 2018, pp. 1–6.
[28] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4804–4814, Jun. 2019.
[29] L. Lei, H. Xu, X. Xiong, K. Zheng, and W. Xiang, "Joint computation offloading and multiuser scheduling using approximate dynamic programming in NB-IoT edge computing system," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5345–5362, Jun. 2019.
[30] J. Yan, S. Bi, and Y.-J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5404–5419, Aug. 2020.
[31] S. Yu, X. Chen, L. Yang, D. Wu, M. Bennis, and J. Zhang, "Intelligent edge: Leveraging deep imitation learning for mobile edge computation offloading," *IEEE Wireless Commun.*, vol. 27, no. 1, pp. 92–99, Feb. 2020.
[32] U. Saleem, Y. Liu, S. Jangsher, X. Tao, and Y. Li, "Latency minimization for D2D-enabled partial computation offloading in mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 4, pp. 4472–4486, Apr. 2020.
[33] C. Shu, Z. Zhao, Y. Han, and G. Min, "Dependency-aware and latency-optimal computation offloading for multi-user edge computing networks," in *Proc. 16th Annu. IEEE Int. Conf. Sens. Commun. Netw. (SECON)*, Boston, MA, USA, 2019, pp. 1–9.
[34] K. De Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho, "The energy/frequency convexity rule: Modeling and experimental validation on mobile devices," in *Proc. Int. Conf. Parallel Process. Appl. Math.*, 2013, pp. 793–803.
[35] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," 2017. [Online]. Available: arXiv:1704.01665.
[36] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016. [Online]. Available: arXiv:1609.02907.
[37] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. ACM Special Interest Group Data Commun.*, 2019, pp. 270–288.
[38] Y. Wei, F. R. Yu, M. Song, and Z. Han, "Joint optimization of caching, computing, and radio resources for fog-enabled IoT using natural actor–critic deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2061–2073, Apr. 2019.
[39] F. Liu, Z. Huang, and L. Wang, "Energy-efficient collaborative task computation offloading in cloud-assisted edge computing for IoT sensors," *Sensors*, vol. 19, no. 5, p. 1105, 2019.



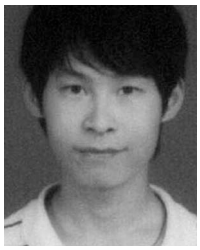
Jiawen Chen received the B.S. degree in computer science and technology from Northeastern University at Qinhuangdao, Qinhuangdao, China, in 2015. She is currently pursuing the M.S. degree with the School of Computer Science and Technology, College of Intelligence and Computing, Tianjin University, Tianjin, China.

Her current research interests include edge computing offloading, reinforcement learning, and deep learning.



Yajun Yang (Member, IEEE) received the B.S. and Ph.D. degrees from Harbin Institute and Technology, Harbin, China, in 2006 and 2013, respectively.

He is currently an Associate Professor with the College of Intelligence and Computing, Tianjin University, Tianjin, China. His current research interests include massive data management, graph mining, and machine learning.



Chenyang Wang (Member, IEEE) received the B.S. and M.S. degrees in computer science and technology from Henan Normal University, Xinxiang, China, in 2013 and 2017, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, College of Intelligence and Computing, Tianjin University, Tianjin, China.

He has been a visiting Ph.D. student under the support of China Scholarship Council with the School of Electrical Engineering, Aalto University, Espoo, Finland, since May 15, 2021. His current research interests include edge computing, big data analytics, reinforcement learning, and deep learning.

Mr. Wang received the Best Student Paper Award of the 24th International Conference on Parallel and Distributed Systems by IEEE Computer Society in 2018 and the Best Paper Award of IEEE International Conference on Communications in 2021.

Heng Zhang is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, College of Intelligence and Computing, Tianjin University, Tianjin, China.

His current research interests include D2D content propagation, recommend system, and edge computing.

Chao Qiu (Member, IEEE) received the B.S. degree in communication engineering from China Agricultural University, Beijing, China, in 2013, and the Ph.D. degree in information and communication engineering from Beijing University of Posts and Telecommunications, Beijing, in 2019.

She is currently a Lecturer with the School of Computer Science and Technology, College of Intelligence and Computing, Tianjin University, Tianjin, China. From September 2017 to September 2018, she visited Carleton University, Ottawa, ON, Canada, as a Visiting Scholar. Her current research interests include machine learning, software-defined networking, and blockchain.

Xiaofei Wang (Senior Member, IEEE) received the master's and Doctoral degrees from Seoul National University, Seoul, South Korea, in 2013.

He was a Postdoctoral Fellow with The University of British Columbia, Vancouver, BC, Canada, from 2014 to 2016. He is currently a Professor with the Tianjin Key Laboratory of Advanced Networking, School of Computer Science and Technology, Tianjin University, Tianjin, China. Focusing on the research of social-aware cloud computing, cooperative cell caching, and mobile traffic offloading, he has authored over 130 technical papers in the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, the IEEE WIRELESS COMMUNICATIONS, the *IEEE Communications Magazine*, the IEEE TRANSACTIONS ON MULTIMEDIA, the IEEE INFOCOM, and the IEEE SECON.

Prof. Wang received the Scholarship for Excellent Foreign Students in IT Field by NIPA of South Korea from 2008 to 2011, the Global Outstanding Chinese Ph.D. Student Award by the Ministry of Education of China in 2012, and the Peiyang Scholar from Tianjin University. In 2017, he received the Fred W. Ellersick Prize from the IEEE Communication Society. He was a recipient of the National Thousand Talents Plan (Youth) of China.