

# Dependency-Aware Task Offloading Based on Deep Reinforcement Learning in Mobile Edge Computing Networks

Junnan Li, Zhengyi Yang\*, Kai Chen, Zhao Ming, Xiuhua Li, Qilin Fan, Jinlong Hao, and Luxi Cheng

## Abstract

With the rapid development of innovative applications, lots of computation-intensive and delay-sensitive tasks are emerging. Task offloading, which is regarded as a key technology in the emerging Mobile Edge Computing (MEC) paradigm, aims at offloading the tasks from mobile devices (MDs) to edge servers or the remote cloud to reduce system delay and energy consumption of MDs. However, most of existing task offloading studies either didn't consider the dependencies among tasks or simply designed heuristic schemes to solve dependent task offloading problems. Different from these studies, we propose a deep reinforcement learning based task offloading scheme to jointly offload tasks with dependencies. Specifically, we model the dependencies among tasks by directed acyclic graphs and formulate the task offloading problem as minimizing the average cost of energy and time (CET) of users. To solve this NP-hard problem, we propose a deep Q-network learning based framework that creatively utilizes deep neural network to extract system features. Simulation results show that our proposed scheme outperforms heuristic baselines in reducing the CET of users and can obtain near-optimal offloading strategies.

## Index Terms

Mobile Edge Computing, Deep Reinforcement Learning, Directed Acyclic Graph, Deep Neural Network.

\*Corresponding author: Zhengyi Yang (email: zyyang@cqu.edu.cn). There is no conflict of interest for this paper.

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

J. Li, Z. Yang, X. Li, and Q. Fan are with School of Big Data & Software Engineering, Chongqing University, Chongqing 401331, China (e-mail: lijunnan@stu.cqu.edu.cn, zyyang@cqu.edu.cn, lixiuhua1988@gmail.com, fanqilin@cqu.edu.cn).

K. Chen is with CISDI R&D Co.,Ltd, Chongqing 401122, China (e-mail: Kai.chen@cisdi.com.cn).

Z. Ming is with the Centre for Wireless Communications, University of Oulu, Oulu 90570, Finland. (e-mail: zhao.ming@oulu.fi.)

J. Hao is with School of Big Data & Software Engineering, Chongqing University, Chongqing 401331, China, and also with Chongqing Changan Automobile Software Technology Co. Ltd, Chongqing 400023, China (e-mail: haojl@changan.com.cn).

L. Cheng is with Chongqing Innovation Center of Industrial Big-Data Co. Ltd, Chongqing 400700, China, and also with School of Big Data & Software Engineering, Chongqing University, Chongqing 401331, China (e-mail: chengluxi1818@126.com).

## I. INTRODUCTION

The rapid development of communication technologies and the explosive growth of the number of mobile devices (MDs) accelerate the emergence of innovative applications, such as augmented reality, face recognition, intelligent driving, and smart home [1]. These complex applications generally require intensive computing resources and low response latency, putting great pressure on the core networks. However, it is generally difficult to run these applications at MDs since these MDs are usually limited by computation, storage, and energy resources. At the same time, the remote cloud is far away from the MDs, running these applications in the cloud will result in high transmission latency.

Mobile Edge Computing (MEC), which has been one of the most emerging technologies in recent years, is considered as an effective paradigm to overcome this dilemma [2]. Specifically, as a key technology of MEC, task offloading aims at offloading computation-intensive tasks from MDs to the edge servers (ESs) deployed at the network edge (e.g., base stations (BSs)). By jointly considering the properties of tasks and resource utilization of ESs, task offloading can significantly reduce system delay and energy consumption of MDs and improve the quality of service to users [3]–[5]. Hence, task offloading plays a vital role in satisfying the requirements of these computation-latency-sensitive tasks.

A number of studies investigated task offloading problems in the scenario with a single ES [6], [7]. For instance, the authors in [8] considered a general scenario with only one user and one ES and modeled the task offloading problem as a binary problem, which represented executing the task locally or at the ES. However, this work only considered atomic offloading and cannot process the same task in parallel. In [9], a heuristic algorithm was proposed to solve a mixed-integer linear programming problem in deciding the task offloading strategies in a multi-user scenario. In [10], an approximate dynamic programming scheme was used to solve a dynamic optimization problem. However, these studies didn't consider the dependencies among tasks. Meanwhile, some recent works focused on dependent task offloading problems and adopted Directed Acyclic Graphs (DAGs) to represent the dependencies among tasks [11], [12]. However, the offloading strategies of these papers were obtained by heuristic algorithms that generally fall into local optimal while ignoring the overall performance. Besides, a lot of researchers also utilized the promising technology Deep Reinforcement Learning (DRL) which is the combination of Reinforcement Learning (RL) and Deep Neural Network (DNN) to determine the task offloading policies [13]–[16]. However, these schemes didn't consider the dependencies among tasks.

The gap in the literature as discussed above motivates us to propose a new DRL-based task offloading scheme, which can reduce system latency and energy consumption of the MDs. We model the task offloading decisions as a Markov Decision Process (MDP) and represent the tasks by DAGs. Moreover, we design a DNN-based embedding model to obtain the key features of the MEC system and tasks and propose a Deep Q-network (DQN) based algorithm to interact with the environment for determining the task offloading decisions efficiently. The major contributions of this paper are summarized as follows:

- 1) We design a general End-Edge-Cloud network architecture and propose a novel DRL based dependency-aware task offloading scheme to jointly offload tasks with dependencies in the considered networks.

- 2) We model the dependencies among tasks by DAGs and formulate the problem as minimizing the average CET of all users. To solve this NP-hard problem, we creatively utilize DNN to extract system features and propose a DQN based algorithm to determine task offloading strategies.
- 3) Extensive simulation results demonstrate that our proposed method outperforms heuristic baselines and can obtain near-optimal offloading strategies.

The remainder of this paper is organized as follows. Section II introduces the related work concerning task offloading in recent years. In Section III, we introduce system model and formulate the optimization problem. In Section IV we present the proposed DRL-based task offloading framework. Simulation results are provided in Section V. Finally, Section VI concludes this paper.

## II. RELATED WORK

In recent years, research concerning task offloading has gradually shifted from one-shot task offloading to dependency-aware task offloading. In this section, we review the development of task offloading technology as three-fold.

### A. One-Shot Task Offloading

There exist two main categories in one-shot task offloading paradigm, i.e., application-level task offloading and atomic-level task offloading. On one hand, application-level task offloading has been studied by many researchers in recent years, focusing on offloading entire tasks to the ESs. For instance, in [17], Chen *et al.* developed a general MEC scenario that consists of device, edge, and cloud layer to reduce the overall system overhead by the collaboration between different layers. The idea of game theory was adopted to study the optimization of multimedia services in MEC networks, which provided more efficient service response and ensured the robustness of the system [18]–[20]. Some researchers also used an augmented graph to model the tasks to jointly optimize the computing power and the transmission delay [21]. On the other hand, atomic-level task offloading, also known as partial offloading, aims at splitting the original tasks to be several subtasks for task offloading. In this field, You *et al.* developed a multi-user task offloading system and proposed to split computation data for separate computing [22]. Moreover, Gao *et al.* applied task partitioning into task offloading to effectively minimize the cost of computation delay and energy consumption [23].

### B. Dependency-Aware Task Offloading

At the same time, some researchers focused on dividing tasks into several subtasks according to different running scenarios. [24]–[31]. However, in some special scenarios such as face recognition and speech recognition [32], these generated subtasks are with dependencies, making makes traditional task offloading methods difficult to be applied. To cope with this challenge, many researchers considered designing new schemes that support dependent task offloading. For instance, Mehrabi *et al.* investigated the optimization of task offloading decisions in a device-enhanced MEC system to minimize the energy consumption of MDs with the dependencies among tasks considered [26]. Kao *et al.* modeled the task graphs as serial trees and designed an online algorithm to reduce the system delay

[27]. However, these studies only considered the dependencies among tasks as sequences rather than a more general structure. In [28], a heuristic algorithm was proposed to minimize the cost of energy consumption in ultra-dense networks. However, the heuristic algorithm was not so robust as they may easily fall into local optimal. In [31], Zhao *et al.* designed an efficient convex optimization based algorithm to solve the dependent task offloading and service caching problem. However, these schemes may suffer low convergence speed in complex dynamic wireless environments with lots of tasks.

### C. Deep Reinforcement Learning based Task Offloading

Additionally, many researchers proposed to utilize deep reinforcement learning to tackle the task offloading problem as it can solve complex decision-making problems by dynamically interacting with the environment [33]–[36]. For instance, the authors in [33] introduced blockchain technology to MEC networks and combined it with DRL to address the complex task offloading problem. In [34], Qi *et al.* formulated the task offloading problem as a long term planning problem and proposed a DRL based algorithm to solve it. Moreover, Tang *et al.* considered non-divisible and delay-sensitive tasks and proposed a DQN based algorithm to reduce the system delay and increase the stability of transmitting tasks [35]. However, these researches mainly focused on solving the application-level task offloading problem and didn't consider the dependencies among tasks.

To summarize, the dependent task offloading problem is still unexplored well, designing effective and robust dependency-aware task offloading schemes remains a key topic in MEC networks.

## III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we introduce the system model and formulate the optimization problem, the used key notations and definitions are summarized in Table I.

### A. Network Architecture

We design a general MEC system that consists of three layers from the bottom to the top as shown in Fig. 1, i.e., the device layer, the edge layer, and the cloud layer. The device layer includes lots of MDs, e.g., mobile phones, tablets, and smart Internet of Things/Vehicles (IoT/IoV) devices. These MDs are randomly distributed in different cells covered by different base stations (BSs) and are connected to the BSs through cellular links. In the edge layer, the BSs consist of edge servers (ESs) that are limited by computation, storage, and bandwidth resources. These BSs are also connected to each other via BS-to-BS links, and to the remote cloud server via backhaul links. In the cloud layer, the cloud server has a large amount of computation, calculation, and bandwidth resources. Specifically, in the MEC system, the BS with an edge server can offload the task to another BS or the cloud server. We consider each MD runs an application that consists of a series of tasks. These tasks are partial with dependencies and can be processed at the MDs, offloaded to the edge, or offloaded to the cloud. Only if all the tasks of an application are finished can this application terminate. We assume that all applications are initiated and terminated at MDs.

TABLE I  
KEY MODELING PARAMETERS AND NOTATIONS

Notation	Definition
$\mathcal{N}$	The set of applications
$\mathcal{B}$	The set of BSs
$q_{i_n}$	The $i$ -th sub-task of application $n$
$L_{i_n}$	The required CPU cycles of $q_{i_n}$
$S_{i_n}$	The data size of $q_{i_n}$
$P_{i_n}^e$	The transmission power between MD $_n$ and the ES
$R_{i_n}^e$	The transmission rate between MD $_n$ and the ES
$R_b^{e,c}$	The transmission rate between BS $b$ and the cloud server
$T_{i_n}^e$	The transmission time from MD $_n$ to the directed BS
$T_{i_n}^c$	The transmission time from BS to the cloud server
$T_{i_n}^{b,b'}$	The transmission time between BSs $b$ and $b'$
$f_n^l$	The local computation capability of the MDs
$f_b^e$	The computation capability of the ES $b$
$f^c$	The computation capability of the cloud server
$t_{i_n}^l$	The local execution time of task $q_{i_n}$
$t_{i_n}^e$	The execution time of task $q_{i_n}$ at the ES
$t_{i_n}^c$	The execution time of task $q_{i_n}$ on the cloud server
$pred(q_{i_n})$	The set of immediate predecessors of task $q_{i_n}$
$succ(q_{i_n})$	The set of immediate successors of task $q_{i_n}$

### B. System Modeling

In the designed network, we consider  $B$  BSs and  $N$  MDs, denoted as  $\mathcal{B} = \{1, 2, \dots, B\}$  and  $\{\text{MD}_1, \text{MD}_2, \dots, \text{MD}_N\}$ , respectively. Each MD runs an application and the set of the  $N$  applications is denoted as  $\mathcal{N} = \{1, 2, \dots, N\}$ , where application  $n$  is generated on MD $_n$ ,  $\forall n \in \mathcal{N}$ . We assume that  $n$  consists of a series of tasks with dependencies, denoted as  $\mathcal{I}_n = \{1, 2, \dots, I_n\}$ . The  $i$ -th sub-task of application  $n$  is denoted as  $q_{i_n}$ ,  $i_n \in \mathcal{I}_n$ , which's required CPU cycles and input data size are denoted as  $L_{i_n}$  and  $S_{i_n}$ , respectively.

For application  $n$ , we depict the dependency relationships of its tasks by a directed acyclic graph (DAG) with an entry node and an exit node, as shown in Fig. 2. The entry node and exit node represent the beginning task and the end task of the application, respectively. We use  $\theta_n$  to denote the dependencies of all the tasks of  $n$ , where  $e_{ij} \in \theta_n$  represents that task  $q_{i_n}$  should be processed before task  $q_{j_n}$ ,  $\forall q_{i_n}, q_{j_n} \in \mathcal{N}$ . For instance, in Fig. 2, task 6 can only be processed after task 2, 4, and 5 have been processed. As a result, the modeled DAG can be described by  $G_n = (\mathcal{I}_n, \theta_n)$ . Specifically, for simplicity, we set two auxiliary nodes as node  $0_n$  and node  $(I+1)_n$ , denote the entry task and exit task, respectively. Moreover, the workload of these two tasks is set as  $L_{0_n} = L_{(I+1)_n} = 0$ .

We denote the task offloading policies of  $q_{i_n}$  as  $\alpha_{i_n} \in \{0, 1, \dots, B, B+1\}$ , where 0 means task  $q_{i_n}$  is executed locally,  $1, \dots, B$  means it is offloaded to the corresponding ESs, and  $B+1$  means it is offloaded to the cloud server. Note that, as the speed of download transmission is high, we consider the download transmission time can be ignored [37]. Moreover, we define *ready time* of a task as the time when all its immediate predecessors finish

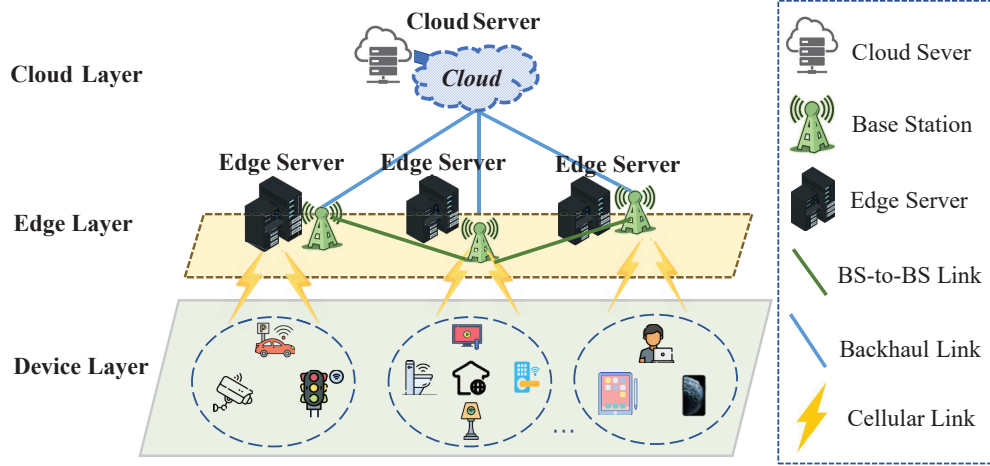


Fig. 1. Illustration of the network architecture of the considered system in MEC networks.

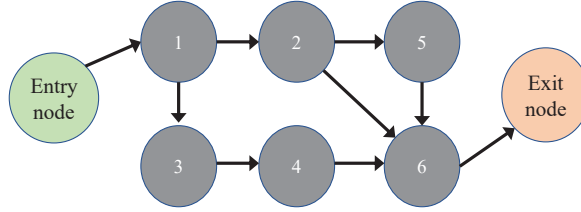


Fig. 2. The modeled DAG structure of the tasks of an application with two auxiliary nodes.

processing, and *finish time* as when the task finishes processing. Based on this, the ready time and finish time of  $q_{i_n}$  that is executed locally, at the edge server, or at the cloud server can be denoted as  $RT_{i_n}^l, RT_{i_n}^e, RT_{i_n}^c$  and  $FT_{i_n}^l, FT_{i_n}^e, FT_{i_n}^c$ , respectively.

### C. Device Layer Processing Modeling

We denote the set of immediate predecessors of task  $q_{i_n}$  as  $pred(q_{i_n})$ . When  $q_{i_n}$  is processed locally, the ready time of  $q_{i_n}$ , i.e.,  $RT_{i_n}^l$ , should be the maximum finish time of  $q_{j_n} \in pred(q_{i_n})$ . As  $q_{j_n}$  can also be processed locally, offloaded to the ESs, or offloaded to the cloud server, the maximum finish time of  $q_{j_n}$  should be the sum of its maximum finish time and local execution time and can be calculated by

$$RT_{i_n}^l = \max_{q_{j_n} \in pred(q_{i_n})} \{max\{FT_{j_n}^l, FT_{j_n}^e, FT_{j_n}^c\}\}. \quad (1)$$

Moreover, we denote the local execution time of  $q_{i_n}$  as  $t_{i_n}^l$ , the computation capacity of MD $_n$  as  $f_n^l$ , which is upper bounded by  $f_n^l < f_{peak}^l$ , thus,  $t_{i_n}^l$  can be calculated by  $t_{i_n}^l = L_{i_n}/f_n^l$ , and the finish time of  $q_{i_n}$  can be

obtained by  $FT_{i_n}^l = RT_{i_n}^l + t_{i_n}^l$ . The energy consumption of task  $q_{i_n}$  performed locally can be obtained as:

$$e_{i_n}^l = \kappa L_{i_n} (f_{i_n}^l)^2, \quad (2)$$

where  $\kappa$  is the switched capacitance [2].

#### D. Edge Layer Processing Modeling

When  $q_{i_n}$  is processed at the edge, it should first be transmitted to the BS  $b$  which servers MD $_n$ . We denote  $h_{i_n}^e$  as the wireless channel gain between MD $_n$  and  $b$ ,  $P_{i_n}^e$  as the transmission power. Thus, the transmission rate between MD $_n$  and BS  $b$  can be given by

$$R_{i_n}^e = W \log_2 \left( 1 + \frac{P_{i_n}^e h_{i_n}^e}{\sigma^2} \right), \forall i_n \in \mathcal{I}_n, \forall n \in \mathcal{N}, \quad (3)$$

where  $\sigma^2$  denotes the Gaussian noise power,  $W$  denotes the channel bandwidth. Thus, the transmission delay for uploading  $q_{i_n}$  to BS  $b$  can be obtained by  $T_{i_n}^e = S_{i_n} / R_{i_n}^e$ , and the energy consumption during the uploading process can be obtained as:

$$e_{i_n}^u = P_{i_n}^e \times T_{i_n}^e. \quad (4)$$

Secondly, BS  $b$  offloads task  $q_{i_n}$  to the target BS  $b'$  through BS-to-BS links. We denote the transmission time of this step as  $T_{i_n}^{b,b'}$ , which can be calculated as  $T_{i_n}^{b,b'} = \frac{S_{i_n}}{R_{b,b'}} \times h$ , where  $R_{b,b'}$  denotes the transmission rate between BSs  $b$  and  $b'$ ,  $h$  represents the number of hops. Particularly  $h = 0$  holds while  $b = b'$ , which means that the target BS where  $q_{i_n}$  should be processed is the connected BS of MD $_n$ .

Last, task  $q_{i_n}$  can be processed at the ES in  $b$ . We denote the CPU frequency of the ES as  $f_b^e$ , and calculate the time for processing task  $q_{i_n}$  as  $t_{i_n}^e = L_{i_n} / f_b^e$ . As a result, the ready time of task  $q_{i_n}$  executed at the edge server can be calculated as

$$RT_{i_n}^e = \max_{q_{j_n} \in \text{pred}(q_{i_n})} \{ \max\{FT_{j_n}^l, FT_{j_n}^e, FT_{j_n}^c\} + T_{i_n}^e + T_{i_n}^{b,b'} \}, \quad (5)$$

and the finish time of task  $q_{i_n}$  executed at the edge server can be obtained by  $FT_{i_n}^e = RT_{i_n}^e + t_{i_n}^e$ .

#### E. Cloud Layer Processing Model

We consider the transmission rate from the BS  $b$  to the cloud server as constant denoted as  $R_b^{e,c}$ , and the transmission time from  $b$  to the cloud server can be calculated as  $T_{i_n}^c = \frac{S_{i_n}}{R_b^{e,c}}$ . We denote the cloud computing capability as  $f^c$ , then the execution time of task  $q_{i_n}$  can be calculated as  $t_{i_n}^c = \frac{L_{i_n}}{f^c}$ . The ready time of task  $q_{i_n}$  in the cloud can be expressed as

$$RT_{i_n}^c = \max_{q_{j_n} \in \text{pred}(q_{i_n})} \{ \max\{FT_{j_n}^l, FT_{j_n}^e, FT_{j_n}^c\} + T_{i_n}^e + T_{i_n}^c \}, \quad (6)$$

and the finish time of task  $q_{i_n}$  in the cloud can be obtained by  $FT_{i_n}^c = RT_{i_n}^c + t_{i_n}^c$ .

### F. Problem Formulation

For each application  $n$  of MDs, we define the CET  $\xi_n$  as

$$\xi_n = w_t \times (FT_{(i+1)_n} - FT_{0_n}) + w_e \times E, \quad (7)$$

where  $E$  is the total energy consumption of each mobile device and can be calculated by

$$E = \sum_{i=1}^I e_{i_n}, \quad (8)$$

where  $e_{i_n}$  represents the energy consumption of locally or uploading the task and can be given by

$$e_{i_n} = \begin{cases} e_{i_n}^l, & \alpha_{i_n} = 0 \\ e_{i_n}^u, & \alpha_{i_n} \neq 0 \end{cases}. \quad (9)$$

Specifically, if a sub-task is performed locally, the energy consumption can be calculated by equation 2, if a sub-task is offloaded to the edge server or the cloud server, the energy consumption can be calculated by equation 4. Note that, we ignore the energy consumption transmitted between ESs and from ES to the cloud server.

Our object is to minimize the average CET of all users. To be specific, we aim to find a near-optimal offloading strategy  $\alpha_n = \{\alpha_{0_n}, \alpha_{1_n}, \dots, \alpha_{(i+1)_n}\} = \{0, |\mathcal{B}| + 1\} \cup \mathcal{B}$  for each user, where  $\alpha_{0_n} = 0$  and  $\alpha_{(i+1)_n} = |\mathcal{B}| + 1$ . Thus, the overall optimization problem can be formulated as

$$\min_{\alpha_n} \frac{\sum_{n=1}^N \xi_n}{N} \quad (10a)$$

$$s.t. \quad RT_{i_n} \leq FT_{i_n}, \quad (10b)$$

$$\max_{q_{j_n} \in \text{pred}(q_{i_n})} FT_{j_n} \leq RT_{i_n}, \quad (10c)$$

$$\alpha_{0_n} = 0, \alpha_{(i+1)_n} = |\mathcal{B}| + 1, \quad (10d)$$

$$\alpha_{i_n} \in \{0, |\mathcal{B}| + 1\} \cup \mathcal{B}, \forall i \in \mathcal{I}_n, \forall n \in \mathcal{N}, \quad (10e)$$

$$w_t + w_e = 1, \quad (10f)$$

where constraints (10b) and (10c) represent the dependency relationships of tasks, indicating that task  $q_{i_n}$  can be processed only after all its immediate predecessors have finished processing. Constraint (10d) means all applications should be initiated and terminated locally. Constraint (10e) specifies that task  $q_{i_n}$  is executed locally, at the ES, or at the cloud server. Constraint (10f) denotes the weights of processing time and energy consumption. Note that this optimization problem is not convex but NP-hard due to the above constraints, we propose DRL based framework to solve it.

## IV. DRL-BASED TASK OFFLOADING ALGORITHM

In this section, we propose a DRL based task offloading scheme to minimize the average CET of all users. Specifically, the proposed scheme can observe and learn from complex data in MEC networks and generate optimal task offloading strategies to maximize the reward in DRL.

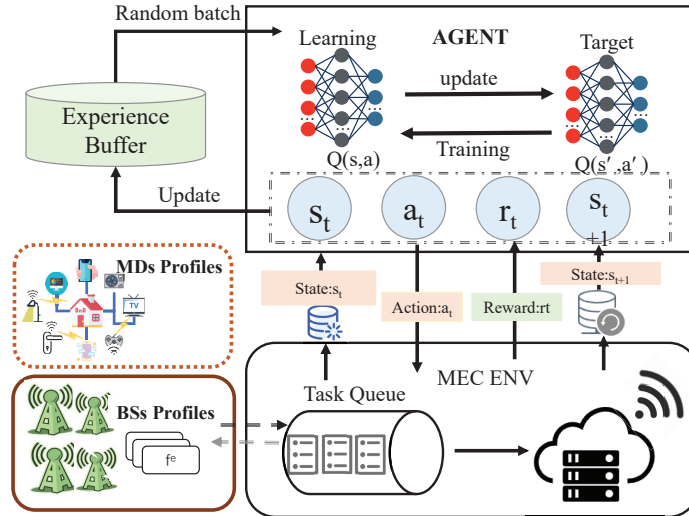


Fig. 3. Illustration of the proposed DRL framework.

#### A. DRL-based Task Offloading Framework Design

We proposed a DRL-based framework as shown in Fig. 3. Specifically, in our designed framework, the ESs collect the configuration information of the tasks and put them into the task processing queue, and the DRL agent generates an action based on the observed state. After that, the ESs process the task offloading strategies (i.e., process the tasks locally or offload to other facilities) and the DRL agent calculates the reward for this action and stores the observed data (state, action, reward, and next state) for further training the model. Then, the agent extracts sample data from the experience replay buffer and trains the learning network through the defined loss function. Finally, the agent updates the target network every  $N$  episodes. We model the dependent task offloading problem as MDP in the MEC networks, where the system state, action, and reward can be described as follows:

1) *System State*: We define the system state vector space as  $S_t = \{S_{i_n}, L_{i_n}, P_{i_n}^e, h_{i_n}^e, f_n^l, f_b^e\}$ , where  $S_{i_n}$  is the input data size of sub-task  $q_{i_n}$ ;  $L_{i_n}$  is the required CPU cycles of  $q_{i_n}$ .  $P_{i_n}^e$  and  $h_{i_n}^e$  are transmission power and the upload wireless channel gain between MD $_n$  and ES  $b$  respectively.  $f_n^l$  and  $f_b^e$  denote the computation capability of MDs and ESs respectively.

2) *System Action*: At each time slot  $t$ , we assume that there is only one task that can be decided. Action  $A_t$  can be expressed as

$$A_t = 0, 1, 2, \dots, |\mathbb{B} + 1|, \quad (11)$$

where  $A_t = 0$  represents that the task is performed locally,  $A_t = \{1, 2, \dots, \mathbb{B}\}$  represents the task is offloaded to the ESs, and  $A_t = \mathbb{B} + 1$  means this task is offloaded to the cloud server.

3) *System Reward*: At each time slot  $t$ , the system selects the most appropriate offloading action  $A_t$  according to the current state  $S_t$ . In order to minimize the average CET of all users, we define the reward at time step  $t$  as

the negative increment of the weighted sum of CET of all users in MEC system. Then the system reward can be calculated as

$$R_t = \sum_{n=1}^N \xi_n^t - \sum_{n=1}^N \xi_n^{t+1}. \quad (12)$$

### B. Deep Q-network Learning Model

The DRL agent adopts a balanced method that consists of exploitation and exploration to learn and interact with the MEC networks. The method of exploitation is to use the experience related to Q-value learned through greedy learning to make use of relevant experience, which can be described as

$$a = \underset{a'}{\operatorname{argmax}} Q(s, a'; w), \quad (13)$$

where  $w$  is the parameters matrix. In addition, exploration agents are allowed to take random actions to obtain MEC environment information. In this paper, to keep the balance between exploitation and exploration, we adopt a method called  $\epsilon$ -greedy, which means that the probability of the model selecting action with greedy algorithm is  $1 - \epsilon$ , and the probability of randomly selecting action is  $\epsilon$ . In the beginning, the agent interacts with an unfamiliar environment, it uses exploration methods to randomly obtain MEC environment information. With the gradual increase of accumulated experience, the agent begins to use the learned experience to explore the environment, so the agent will use exploitation methods more.

Moreover, the goal of the agent is to maximize the long-term cumulative reward by finding the optimal action-values  $Q^*(s, a)$  and finding the optimal decision  $\pi^*$ . The agent can obtain the action-value  $Q(s, a)$  by taking action  $a$  on state  $s$  following the policy  $\pi$ . Then we can obtain the optimal action-value  $Q^*(s, a)$  by selecting the maximum values of  $Q(s, a)$  in all possible values as

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a, \pi]. \quad (14)$$

The optimal policy  $\pi^*$  satisfies the Bellman equation and can optimize action-value  $Q^*(s, a)$  as

$$Q^*(s, a) = \mathbb{E}'_s [R + \gamma \max_{a'} Q(s', a') | s, a]. \quad (15)$$

The step to find the optimal action value is to take action  $a$  from all possible actions for the next step of  $Q(s, a)$  that can maximize  $R + Q(s', a')$ , and repeat this action until it produces the optimal policy. During each round of iteration, the Q value keeps increasing by updating the Bellman equation, and  $Q_t \rightarrow Q^*$  as  $t \rightarrow \infty$ , as given by

$$Q_{t+1}(s, a) = \mathbb{E}'_s [R + \gamma \max_{a'} Q_t(s', a') | s, a], \quad (16)$$

where  $Q_t$  is the Q-value at time slot  $t$ , and  $Q^*$  is the optimal value-function.

In our proposed DQN model, to prevent fluctuations and errors during training, the DQN model introduces a target network, which has a fixed parameter  $w^-$ . The target network parameters can keep unchanged in certain episodes. The other neural network called the primary network with parameters  $w$ , can continuously learn from the experience buffer, every  $N$  episodes, the primary network updates its parameters to the target network to minimize

the loss function. The mean square error between current action-value with  $Q(s', a'; w)$  and optimal  $Q^*(s', a')$  can be substituted with fixation term  $F$  as

$$F = r + \gamma \max_{a'} Q(s', a'; w^-), \quad (17)$$

where  $w^-$  is updated in previous iterations. The loss function can be given by

$$Loss(w) = \mathbb{E}_{s,a,r}[(\mathbb{E}_s[F|s,a] - Q(s', a'; w^-))]. \quad (18)$$

---

**Algorithm 1** DQN-Learning for MEC
 

---

- 1: **Initialize:** replay memory  $D$  with capacity  $N$ , DQN parameters with random  $w$  and  $w^-$
  - 2:  $\epsilon \leftarrow \epsilon_{start}$
  - 3: **for**  $i = 1, 2, \dots, N$  **do**
  - 4:   Input raw data  $G_i$
  - 5:   Preprocess initial state:  $S_t \leftarrow G_i$  by DNN-based model
  - 6:   **for**  $j = 1, 2, \dots, M$  **do**
  - 7:     Select action  $A_t$  from  $S_t$  using:
  - 8:      $\pi \leftarrow \epsilon\text{-Greedy}(Q(S_t, A_t, w))$ ;
  - 9:     Perform action  $A_t$ , Observe reward  $R_t$  and obtain next state  $S_{t+1}$
  - 10:    Preprocess next state:  $S' \leftarrow S_{t+1}$
  - 11:    Store  $(S, A, R, S')$  in memory replay memory;
  - 12:     $S' \leftarrow S$
  - 13:    Sample random minibatch of  $(s_j, a_j, r_j, s_{j+1})$  from replay memory;
  - 14:     $i \leftarrow N, j \leftarrow M$
  - 15:    **if** episode terminates at step  $j + 1$  **then**
  - 16:     Set target  $F \leftarrow r_j$
  - 17:    **else**
  - 18:     Set target  $r + \gamma \max_{a'} Q(s', a', w^-)$
  - 19:    **end if**
  - 20:    Every  $n$  steps, update  $w^- \leftarrow w$  by loss function (18)
  - 21:    **end for**
  - 22: **end for**
- 

### C. Deep Q-network Learning Process

In this subsection, we introduce the learning process of DQN. The main process of implementation is provided in Algorithm 1. Firstly, the DRL agent initializes the parameters including the replay experience buffer and the exploration proportion  $\epsilon$ , the learning network, and the target network. In each episode, the action is selected by policy  $\pi$  with a probability according to  $\epsilon - greedy$  strategy, otherwise, the agent chooses the action randomly. Each interaction produces a new state  $s_t$ , action  $a_t$ , reward  $r_t$ , and  $s_{t+1}$  and stores them into the experience buffer. Then the DRL agent randomly samples a batch  $(s_j, a_j, r_j, s_{j+1})$  from the experience buffer and sends them to

TABLE II  
MAIN SIMULATION PARAMETERS.

Parameters	Value
$f_n^l$	1.2-1.4GHz uniformly
$f_b^e$	2.2-2.3GHz uniformly
$f^c$	3.2GHz
$W$	25MHz
$p_{i_n}^e$	1.8W
$S_{i_n}$	350KB-600KB uniformly
$\kappa$	$10^{-27}$
$\sigma$	$2 \times 10^{-13}$
$\gamma$	0.90
$\alpha(lr)$	$3e^{-4}$

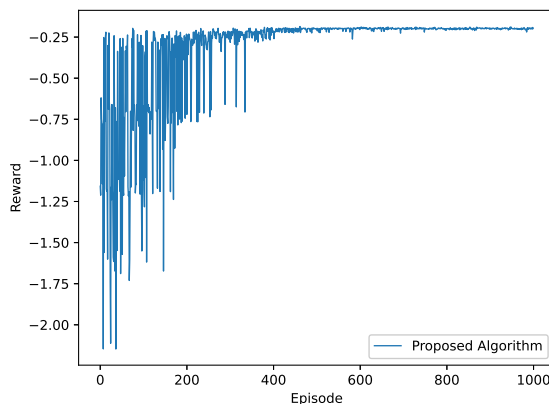


Fig. 4. Convergence Performance for proposed algorithm

learning and target networks. Then the DNNs can be updated according to the loss function (18). In this way, we can update the model parameter  $w$  of DNNs by stochastic gradient descent as

$$\nabla Loss(w) = \mathbb{E}_{s,a,r}[(r + \gamma \max_{a'} Q(s', a'; w^-) - Q(s, a; w)) \nabla Q(s, a; w)]. \quad (19)$$

## V. SIMULATION

### A. Simulation Settings

In this section, we simulate and test our MEC system in a python environment, the neural networks contains two fully connected layers, each of which contains 128 units. And the main parameters are listed in Table II. Other parameters of simulations in this paper are similar to [37]. We use the following baseline schemes to evaluate the performance of our proposed DQN algorithm.

- AAEL: The situation of all tasks are perform locally.
- AOCC: The situation of all tasks of applications are offloaded to the cloud server.

- AOES: The situation of all tasks of applications are offloaded to the ESs.
- Greedy: Each task is greedily performed locally or offloaded remotely based on the weighted sum of estimated time and energy consumption.

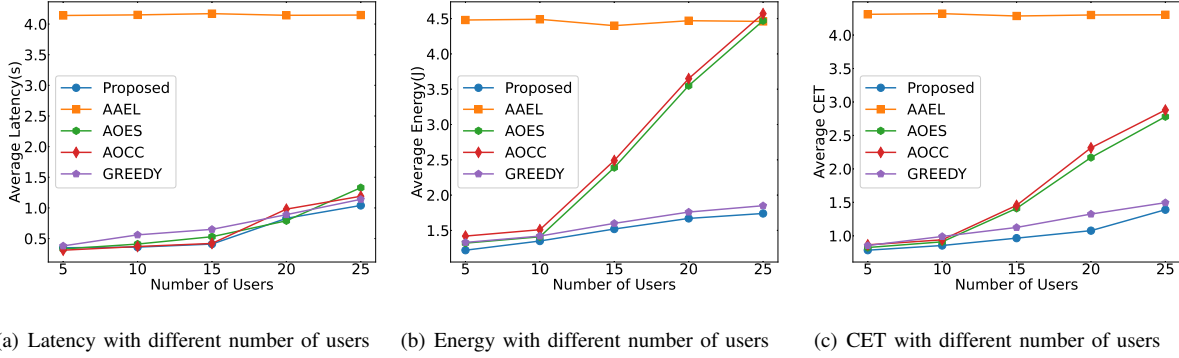


Fig. 5. Average Latency, Energy and CET with different number of users

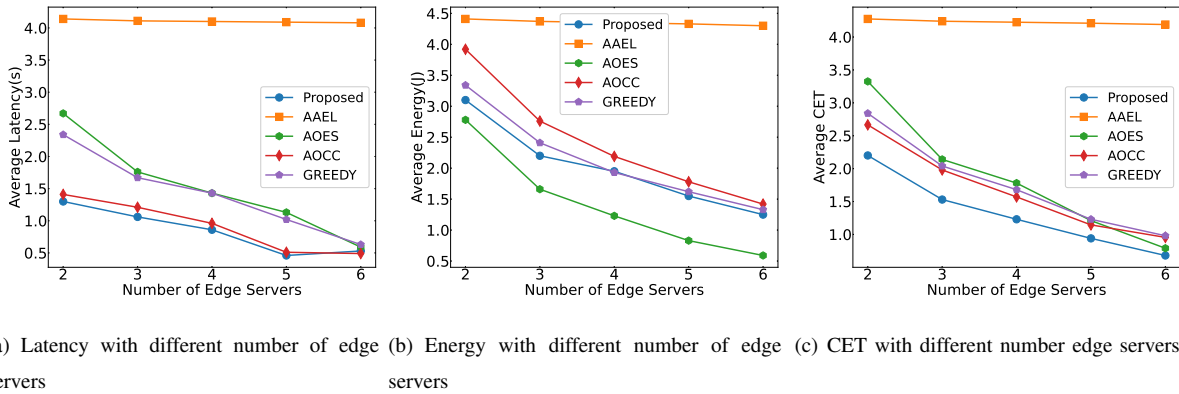


Fig. 6. Average Latency, Energy and CET with different number of edge servers

## B. Simulation Results

1) *Convergence Results*: As shown in Fig. 4, our proposed algorithm begins to converge after 500 episodes, and the convergence value approaches  $-0.25$ . The reason is that we set the reward as the time of time step  $t$  as the negative increment of the sum of task CET of all users.

2) *Performance on Different Numbers of Users*: In this simulation, we assume latency and energy consumption are equally important to the users and we set  $w_t = w_e = 0.5$ . All the users are randomly distributed around three ESs and the number of users varies from 5 to 25. As shown in Fig. 5, the algorithm we proposed can keep the balance between latency and energy consumption, although the performance of the greedy algorithm is similar to the proposed algorithm, the greedy algorithm needs to calculate the whole MEC system state and obtain the reward of actions to choose next action. As the number of users increases, the system cannot withstand the enormous

computational pressure. As shown in Fig. 5(a) and Fig. 5(b), we can find that if all the tasks are executed locally, it can lead to significant delay and energy consumption. In contrast, if all the tasks are offloaded to the cloud server, the system latency will reduce significantly, but as the number of users increases, it will also cause increasing energy consumption. The proposed algorithm in Fig. 5(c) shows that it can find an optimal offloading strategy to keep a balance between latency and energy consumption.

3) *Performance on Different Numbers of ESs*: To evaluate the performance on different numbers of ESs, we set the number of users as 25. Fig. 6 shows the performance of different algorithms as the number of ESs increases from 2 to 6. In Fig. 6(a), as the number of ESs increases, the latency of performing tasks locally remains high, while the latency of other algorithms is gradually decreasing. This is due to the fact that as the number of ESs increases, the number of users corresponding to each ES decreases, which reduces the channel pressure. Fig. 6(b) indicates that the performance of the proposed algorithm is better than most of the baselines (except the AOES algorithm). We can find that offloading tasks to the ESs can reduce energy consumption. Fig. 6(c) shows that the algorithm we propose outperforms other baselines in reducing the average CET.

## VI. CONCLUSION

In this paper, we have designed an edge-cloud computing system to cope with dependent task offloading issue. We have modeled the process of dependent task offloading, and formulated the problem as minimizing the average CET of all applications. To address the problem, we have proposed a DRL-based task offloading scheme to find a near-optimal offloading strategy. Specifically, we have represented the dependent tasks of applications with DAG and extracted the system feature with a DNN-based model. Then we have modeled the task offloading problem as MDP. Finally, we have adopted a DQN based algorithm to solve the above problem. Simulation results have shown that the proposed scheme outperforms other baselines in reducing the average CET and can obtain a near-optimal offloading strategy.

## REFERENCES

- [1] D. Mazza, A. Pagès-Bernaus, D. Tarchi, A. A. J. Perez, and G. E. Corazza, "Supporting mobile cloud computing in smart cities via randomized algorithms," *IEEE Syst. J.*, vol. 12, no. 2, pp. 1598–1609, Oct. 2018.
- [2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 4, pp. 2322–2358, June 2017.
- [3] 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, June 2018.
- [4] M. Du, Y. Wang, K. Ye, and C. Xu, "Algorithmics of cost-driven computation offloading in the edge-cloud environment," *IEEE Trans. Computers*, vol. 69, no. 10, pp. 1519–1532, Oct. 2020.
- [5] M. Song, Y. Lee, and K. Kim, "Reward-oriented task offloading under limited edge server power for multiaccess edge computing," *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13 425–13 438, Mar. 2021.
- [6] L. Huang, S. Bi, and Y. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mob. Comput.*, vol. 19, no. 11, pp. 2581–2593, Sept. 2020.
- [7] 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, Sept. 2020.
- [8] 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, Apr. 2016.

- [9] 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, May 2019.
- [10] 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, May 2019.
- [11] C. Shu, Z. Zhao, Y. Han, and G. Min, "Dependency-aware and latency-optimal computation offloading for multi-user edge computing networks," in *Proc. IEEE International Conference on Sensing, Communication, and Networking (SECON)*, Apr. 2019, pp. 1–9.
- [12] Z. Ming, X. Li, C. Sun, Q. Fan, X. Wang, and V. C. M. Leung, "Dependency-aware hybrid task offloading in mobile edge computing networks," in *Proc. IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, Dec. 2021, pp. 225–232.
- [13] M. Tang and V. W. S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mob. Comput.*, vol. 21, no. 6, pp. 1985–1997, May 2022.
- [14] 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, Jan. 2019.
- [15] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 3, pp. 881–892, Feb. 2021.
- [16] 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 Communications Conference (GLOBECOM)*, Apr. 2019, pp. 1–6.
- [17] M. Chen, M. Dong, and B. Liang, "Resource sharing of a computing access point for multi-user mobile cloud offloading with delay constraints," *IEEE Trans. Mob. Comput.*, vol. 17, no. 12, pp. 2868–2881, Sept. 2018.
- [18] 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, Sept. 2020.
- [19] H. Xiao, C. Xu, T. Cao, L. Zhong, and G. Muntean, "GTTC: A low-expenditure iot multi-task coordinated distributed computing framework with fog computing," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, Mar. 2019, pp. 1–6.
- [20] T. Cao, C. Xu, J. Du, Y. Li, H. Xiao, C. Gong, L. Zhong, and D. Niyato, "Reliable and efficient multimedia service optimization for edge computing-based 5g networks: Game theoretic approaches," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, no. 3, pp. 1610–1625, Sept. 2020.
- [21] X. Chen, C. Xu, M. Wang, Z. Wu, L. Zhong, and L. A. Grieco, "Augmented queue-based transmission and transcoding optimization for livecast services based on cloud-edge-crowd integration," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 11, pp. 4470–4484, Dec. 2021.
- [22] C. You, K. Huang, H. Chae, and B. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wirel. Commun.*, vol. 16, no. 3, pp. 1397–1411, Sept. 2017.
- [23] M. Gao, R. Shen, L. Shi, W. Qi, J. Li, and Y. Li, "Task partitioning and offloading in dnn-task enabled mobile edge computing networks," *IEEE Trans. Mob. Comput.*, Sept. 2021.
- [24] J. Yan, S. Bi, L. Huang, and Y. A. Zhang, "Deep reinforcement learning based offloading for mobile edge computing with general task graph," in *Proc. IEEE International Conference on Communications (ICC)*, Aug. 2020, pp. 1–7.
- [25] J. Yan, S. Bi, and Y. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Wirel. Commun.*, vol. 19, no. 8, pp. 5404–5419, Sept. 2020.
- [26] M. Mehrabi, S. Shen, V. Latzko, Y. Wang, and F. H. P. Fitzek, "Energy-aware cooperative offloading framework for inter-dependent and delay-sensitive tasks," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, Feb. 2020, pp. 1–6.
- [27] Y. Kao, B. Krishnamachari, M. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Trans. Mob. Comput.*, vol. 16, no. 11, pp. 3056–3069, Aug. 2017.
- [28] 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 Communications Conference (GLOBECOM)*, Apr. 2019, pp. 1–6.
- [29] J. Yan, S. Bi, and Y. A. Zhang, "Optimal offloading and resource allocation in mobile-edge computing with inter-user task dependency," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, Oct. 2018, pp. 1–8.
- [30] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, and F. Yang, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4961–4971, Apr. 2020.
- [31] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 11, pp. 2777–2792, June 2021.
- [32] J. Wang, J. Hu, G. Min, W. Zhan, A. Y. Zomaya, and N. Georgalas, "Dependent task offloading for edge computing based on deep reinforcement learning," *IEEE Trans. Computers*, vol. 71, no. 10, pp. 2449–2461, Nov. 2021.

- [33] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 8050–8062, June 2019.
- [34] Q. Qi, J. Wang, Z. Ma, H. Sun, Y. Cao, L. Zhang, and J. Liao, "Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4192–4203, Sept. 2019.
- [35] M. Tang and V. W. S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mob. Comput.*, vol. 21, no. 6, pp. 1985–1997, May 2022.
- [36] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated deep reinforcement learning for internet of things with decentralized cooperative edge caching," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9441–9455, July 2020.
- [37] J. Chen, Y. Yang, C. Wang, H. Zhang, C. Qiu, and X. Wang, "Multi-task offloading strategy optimization based on directed acyclic graphs for edge computing," *IEEE Internet Things J.*, vol. 9, no. 12, pp. 9367 – 9378, Sept. 2021.