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

Juan Fang, *Member, IEEE*, Dezheng Qu, *Student Member, IEEE*, Huijie Chen, *Member, IEEE*, and Yaqi Liu, *Student Member, IEEE*

***Abstract*—The rapid advancement of mobile edge computing (MEC) networks has enabled the augmentation of the computational power of mobile devices (MDs) by offloading computationally intensive tasks to resource-rich edge nodes. This paper discusses the decision-making process for task offloading and resource allocation among multiple mobile devices connected to a base station. The primary objective is to minimize the time taken to complete tasks while simultaneously reducing energy consumption on the device under a time-varying wireless fading channel. This objective is formulated as an energy-efficiency cost (EEC) minimization problem, which cannot be solved by conventional methods. To address this challenge, we propose a dynamic offloading decision algorithm of dependent tasks (DODA-DT) that adjusts local task execution based on edge node status. The proposed algorithm facilitates fair competition among all devices for edge resources. Additionally, we use a deep reinforcement learning (DRL) algorithm based on an actor-critic learning structure to train the system to quickly identify near-optimal solutions. Numerical simulations demonstrate that the proposed algorithm effectively reduces the total cost of the task in comparison to previous algorithms.**

***Index Terms*—Mobile edge computing, task offloading, optimization algorithm, deep reinforcement learning.**

I. Introduction

IN RECENT years, the rapid progress of mobile communication technology has facilitated the growth of the Internet of Things (IoT). IoT refers to a network of interconnected devices on the cloud or at the edge of the network that shares data, functions or enhances operations through standardized interoperable communication protocols. IoT devices are becoming increasingly popular, especially smartphones and wearable devices. According to Cisco's prediction, the number of connected devices will reach 500 billion by 2030 [1]. Furthermore, IoT devices widely use 5G technology, which is a cellular network that achieves a significant improvement in quality of service (QoS), such as higher throughput and lower latency. Ericsson mobility report points out that the global 5G subscriptions close to 1.3 billion in Q2 2023 [2]. This growth in IoT indicates a greater demand for communication and computing power. However, as mobile devices are designed to be compact and affordable, their computing power and battery power are typically limited. In many situations, users may not have ready access to a power source. This problem makes battery life a crucial factor in the design and use of mobile devices.

To address the aforementioned issues, mobile devices begin to use cloud computing. The traditional cloud computing requires transfer of large amounts of data between mobile devices and cloud servers, leading to susceptibility to network bandwidth, wireless channels, and long distances. These factors result in high communication latency. In response, mobile edge computing (MEC) technology emerges as a promising solution. The MEC system consists of edge servers that are deployed at the edge of the network alongside cellular base stations (BS) or local wireless access points (AP) [3]. The edge servers perform intensive computing tasks instead of mobile devices by offloading tasks to the MEC server. This allows the resource-constrained mobile devices to benefit from the powerful computing capabilities of the resource-rich MEC server. Through task offloading, the MEC system provides a better quality of service and extends the battery life of mobile devices. Compared to the traditional cloud computing, mobile edge computing can significantly reduce both cost and transmission latency.

Nonetheless, it may not always be beneficial to offload all applications to the MEC server because the performance of the system can be affected by factors such as task transfer speed, task data, and network latency. Offloading decisions can have a significant impact on the performance and the price of the system. Therefore, the decision to offload applications to the MEC server should be considered carefully, especially when channel conditions are time-varying, and task data is involved. To tackle these issues, various methods have been studied to make task offloading decisions. In [4], a study was conducted on the joint exploration of user behavior characteristics and MEC server pricing strategies, examining their interaction and impact on determining the optimal user data offloading strategy. Heuristic algorithms have been widely used to alleviate such problems [5], [6], [7], [8], [9], [10], [11],

Manuscript received 5 July 2023; revised 1 September 2023; accepted 11 September 2023. Date of publication 26 September 2023; date of current version 15 April 2024. This work is supported by National Natural Science Foundation of China (62202019, 62276011, 61202076), and supported by Beijing Municipal Natural Science Foundation (4192007), along with other government sponsors. The associate editor coordinating the review of this article and approving it for publication was T. Inoue. *(Corresponding author: Juan Fang.)*

The authors are with the Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China (e-mail: fangjuan@bjut.edu.cn; qudz@emails.bjut.edu.cn; chenhuijie@bjut.edu.cn; liuyq617@emails.bjut.edu.cn).

Digital Object Identifier 10.1109/TNSM.2023.3319294

but they may fall into a local optimum and require frequent recalculations when the wireless channel changes. Deep neural networks (DNNs) have shown potential to yield better results. For instance, [12] discussed MECs that support reinforcement learning (RL) and emphasized the challenges that different RL algorithms can address. By applying RL on top of deep learning, the model convergence can be sped up. Currently, many studies address deep reinforcement learning (DRL) algorithms [13], [14], [15], [16], [17], [18]. However, these studies have not fully considered the amount of task data in the MEC environment. Besides, by leveraging the advantages of value estimation and policy optimization, the actor-critic network offers a versatile and robust framework for reinforcement learning applications [19]. Reference [20] utilized the actor-critic algorithm, which is a policy-based one. In contrast to their earlier work [21] that utilized the DQN algorithm, the actor-critic network eliminates the need for extensive memory for experience replay. This distinction highlights the advantage of the actor-critic algorithm in terms of memory efficiency, making it a more streamlined and practical choice for reinforcement learning tasks. Reference [22] utilized actor-critic networks to effectively reduce content duplication, largely improving global cache hit rates.

In addition, the tasks of mobile devices can be rigidly constrained by a sequential order [23], [24]. The execution of a task may depend on the output of other tasks. Neglecting the interdependencies between tasks can lead to a decrease in performance. To address this issue, directed acyclic graphs are commonly used to represent dependencies between tasks [23]. When a large number of task data arrive at the edge and the local devices have completed small tasks, this may result in waiting time between tasks. This in turn leads to a decrease in computing power in the system. Motivated by this limitation, we can reduce the waiting time by adjusting the devices' frequency. In this way, computing resources can be utilized more efficiently.

For the optimization problem of making offloading decisions, there are several crucial questions that need to be addressed. These questions include determining the most suitable applications for offloading to the edge server, devising effective collaboration strategies for different devices to achieve low energy costs in the system, and allocating computing resources appropriately for tasks given the limited resources of the edge server. Our approach formulates the problem as an energy-efficiency cost (EEC) minimization problem. We leverage the power of deep reinforcement learning (DRL) to formulate the offloading decision. The dynamic parameters in MEC environments make DRL an attractive solution due to its ability to map input system parameters directly to output offloading decisions using a deep neural network (DNN). Compared to other papers, the main contributions are:

- We consider the problem of minimizing EEC in a single base station environment. In this system, we make offloading decisions and allocate resources between multiple mobile devices;
- By reducing the waiting time of local devices, we are able to improve the computational power of the entire system. Furthermore, we have taken into account the interdependence between tasks, ensuring that the optimization process is comprehensive and effective;
- We believe that the computing power at the edge is not unlimited. We allocate computing resources reasonably based on the specific situation of each time slot.

The remainder of this paper is structured as follows. In Section II, we provide a review of relevant literature. In Section III, we introduce our system model, computational model, and the energy-efficiency cost minimization problem. Section IV outlines our proposed dynamic offloading decision algorithm for dependent tasks (DODA-DT). In Section V, we present the results of our experiments conducted on a relevant platform. In Section VI, we conclude this paper.

II. Related Work

Recent research has focused on developing task offloading strategies for MEC. Efficient one-dimensional search algorithms have been proposed to find the optimal task scheduling policy under power constraints [25], while others have defined the problem of offloading dependent tasks using a service cache to reduce execution latency [26]. Resource allocation has also been considered by deriving offloading priorities [27]. However, none of these studies considered both latency and energy consumption simultaneously.

Multiple studies have explored the multi-user scenario after factoring in energy consumption and latency. Reference [8] introduced a mapping scheme between basic resource equipment and application modules, followed by a heuristic dynamic task processing algorithm that reduces task latency time and system power consumption. In a scenario where there are multiple users and servers, [9] designed a two-stage heuristic algorithm based on a genetic algorithm to find a stable convergent solution to the joint optimization problem of task offloading and resource allocation. By weighing task latency and energy consumption between a base station service and multiple users, [11] used a heuristic algorithm to make the approximate optimal offloading decision and resource allocation. In a multi-user MEC network with multiple computational access points, where computational power varied over time, [13] used a deep Q network to obtain the task offloading decision and negotiate the time and energy consumption. In a network environment with multiple non-cooperative mobile devices and multiple edge devices, where tasks were constrained by mobile device power, [14] used deep reinforcement learning to design offloading policies that minimized task discard rate, execution latency, and energy consumption. Additionally, the data queue formed by tasks generated by the application impacts users' offloading decisions and MEC performance. To optimize the offloading threshold of multiple intelligences with respect to task queues at the application layer, along with wireless interference at the physical layer, [28] proposes an approach to improve the expected offloading rate for MECs.

The studies mentioned in the previous text have attempted to address the challenge of task dependencies in mobile edge computing (MEC) to improve the performance of the system.

One approach is to use heuristic algorithms to optimize task scheduling and offloading decisions while taking into account the limited computing power of the edge server. For example, [5] proposed a heuristic algorithm to schedule dynamic tasks with dependencies, while considering the computing power of the edge server. Reference [6] partitioned tasks into sub-tasks and used an analytical offloading decision for each sub-task to optimize the critical path of a weighted directed acyclic graph. Another approach is to consider the heterogeneity of offloading tasks and dynamic task offloading and scheduling. Reference [29] divided the problem into sub-problems to produce the best solution when the main problem and the yielded sub-problems converged, thereby reducing task execution latency while regarding energy as a constraint. In addition, parallelism of tasks at the same level of dependencies can also be used to improve system performance. Reference [7] proposed a parallel transfer and execution scheme for tasks on the mobile device side, while using a heuristic algorithm to design offloading algorithms between tasks with different priorities. Reference [28] integrated the synergy of cloud and mobile edge computing to design an iterative heuristic algorithm for dynamic offloading decisions. Therefore, nowadays researchers are exploring various approaches to address the challenge of task dependencies in MEC to improve system performance, and parallelism of tasks at the same level of dependencies is a potential area for future research.

DRL algorithms based on actor-critic learning structures have also been proposed to reduce energy and time overheads, as shown in [15], [16], [17]. Reference [15] focused on a single mobile device using an MEC server and proposed the idea of lead time, which considered more detailed dependencies between tasks. In contrast, our paper mainly focuses on relationships between mobile devices. References [16], [17] dealt with random task arrival in a certain time, which may lead to uneven allocations of resources if a mobile device generated a large number of tasks with dependencies in a short period of time, except for the first time when all tasks arrived at the edge server at the same time.

Considering the above factors and the advantages of actor-critic network [17], [18], we adopt a DRL algorithm based on an actor-critic learning structure for making offloading decisions to optimize the energy-efficiency cost by taking into account both energy consumption and latency. Additionally, we introduce a time slot mechanism by setting the execution return time of the last executed task at the edge for each round, and adjusting the CPU frequency of mobile devices for local computing. Each device will prepare for the computing resources of the edge server in each time slot, thereby enhancing the overall computing capability of the system. There are also many papers on the three-layer architecture for cloud edge collaboration currently [30], [31]. In this paper, we mainly focus on the optimization relationship between the edge server and mobile devices. In future work, we will also consider cloud computing.

III. System Model and Problem Formulation

This section includes system model, communication model, computation model and problem formulation with task dependency. The system model represents the MEC environment in this paper. Communication model defines how mobile devices and edge servers communicate with each other. The calculation model characterizes the latency and energy cost when performing tasks at the edge and local. Problem formulation refers to the process of optimizing the energy-efficiency cost in the whole system.

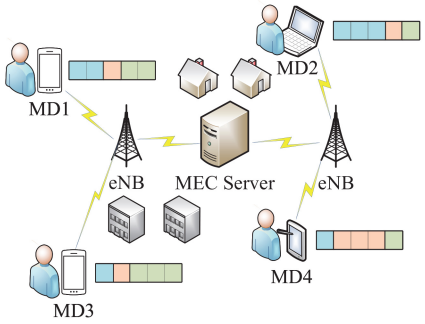
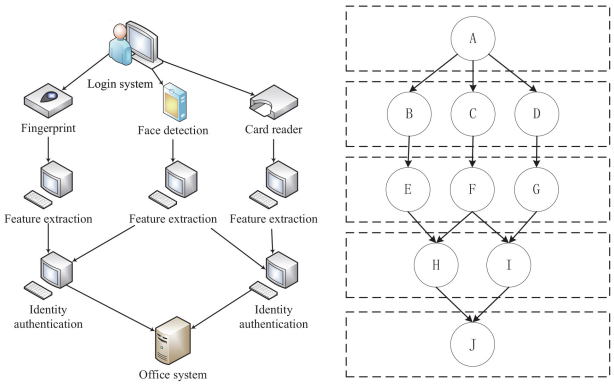


Fig. 1. Illustration of mobile edge computing with multiple mobile devices.

A. System Model

We consider that the MEC network has a single MEC server and a set of cellular base stations. As shown in Fig. 1, these cellular base stations can be represented by either a macro cell (eNB) or a small cell (SCeNB). For simplicity and generality, we will use *eNB* to represent any type of base station. In order to achieve flexible routing and communication between eNBs, we believe that the core cellular network is implemented through Software Defined Network (SDN) technology. SDN simplifies network management by centralizing control logic in a centralized entity called an SDN controller. The MEC server is able to perform multiple tasks simultaneously with a stable power supply. In this model, there are N mobile devices, the grid queue next to the devices denotes a set of computationally intensive tasks with dependencies. The grid with the same color indicates that the tasks are at the same layer. Different layers mean that one task may require the output data of the other tasks. In each time slot, all devices take out a task to participate in the offloading decision. These tasks do not have additional weights assigned by priority. To mitigate the impact of the fluctuating task data on the accuracy of energy efficiency cost changes, we establish an iterative approach by designating consecutive M time slots as an iteration. We compute the total energy efficiency cost once within each iteration. This means that each device has M tasks to be performed in each iteration.

Fig. 2 illustrates a concrete example of specific usage scenarios and representation methods used for task dependencies. Fig. 2(a) shows an office identity authentication system, which includes fingerprint, ID card authentication and face detection. The tasks in the authentication process are dependent but these three authentication actions are independent of each other. We express the system as a directed acyclic graph in Fig. 2(b). Each node i corresponds to a task and a directed edge $e(i, j)$ represents the constraint that task j cannot be executed until its preceding task i is completed. A specific example is that task F needs to be executed after task C. Meanwhile, Task H



(a) (b)

Fig. 2. Description of dependencies between tasks (a) An authentication process of office system; (b) The office system represented by directed acyclic graphs.

and Task *I* require the execution results of Task *F*. In addition, there are many applications with task dependencies (i.e., Chess or online games). These applications can also be represented by directed acyclic graphs.

When we utilize a directed acyclic graph to represent the input-output relationships of tasks, we ignore cyclic graphs since their tasks can be called recursively and we treat them as a single task. There are two commonly used offloading models for MEC, namely, the partial offloading model and the binary offloading model, depending on the nature of the task [32]. We choose to use the binary offloading model for our study since it is a commonly used model for MEC environments. This model categorizes tasks into two types: offloaded to the edge server and those executed locally. We assumed that all tasks in our study were fine-grained, meaning that they were composed of small, independent tasks that could be executed within a short time frame. This assumption allowed us to simplify the problem and focus on the offloading decision-making process without considering the impact of task partitioning. Task granularity can affect the offloading decision and impact the system's overall energy efficiency. More specifically, each mobile device generates a set of tasks with dependencies, which are then grouped into an execution stage. This grouping can be indicated by the dotted box in Fig. 2(b). Tasks from a particular stage are eligible for making task offloading decisions in predetermined time slots. However, if the previous group of tasks is not yet completed, the next group cannot be selected. This design choice eliminates the need for additional scheduling requirements, allowing for more efficient execution of tasks at the server. Notably, the last task in each group requires additional consideration due to its dependencies. We address this issue in our proposed system and present a solution which will be discussed later.

B. Communication Model

This subsection introduces the wireless communication model between the edge server and mobile devices. We use $a_{t,i}$ in $\{0,1\}$ to denote whether or not the *i*th mobile device (MD_i) is going to offload the task in time slot *t*. Specifically, the $a_{t,i} = 1$ means that the task is offloaded to the edge via the wireless network. Otherwise, the task is executed locally.

When task offloading occurs in an MEC environment, the bandwidth is shared orthogonally by all mobile devices where offloading occurs, e.g., via single carrier frequency division multiple access (SC-FDMA). In one iteration, the channel bandwidth W_t occupied by each mobile device depends on the number of devices that executed for time slot *t* offloading:

$$\begin{aligned} W_t &= \frac{B}{\sum_{i\in N} a_{t,i}} \\ s.t. \quad & 0 \le t \le M, \\ & 0 \le i \le N. \end{aligned} \tag{1}$$

where *B* denotes the bandwidth between the mobile devices and the edge server. The channel transmission rate follows:

$$R_{t,i} = W_t log_2\left(1 + \frac{p^{send} h_{t,i}}{\sigma^2}\right) \tag{2}$$

where $h_{t,i}$ denotes the channel gain between the *i*th mobile device and the edge server in time slot *t*. Mobility of the devices is also taken into consideration. We assume that the service area is fixed and mobile devices remain within it, the channel gain varies with the time slot. In the block fading assumption, the channel gain $h_{t,i}$ remains constant for a short duration while uploading a computation-intensive task, but it varies across different task executions. In each time slot, we denote p^{send} as the transmit power of the mobile device during the task upload to the edge server, while σ^2 represents the channel noise power density.

Besides, the location of the devices is a crucial issue that needs factoring. It is important to note that the devices will stay within the service area of the MEC server. Due to the differences in distance and obstacles between each device and the MEC server, their channel gain will fluctuate as they move. To address this issue, we adopt the Rayleigh fading channel model, which is commonly used to address channel fading in urban environments. Integrating this model ensures that our results are more accurate and reliable. Therefore, considering the interferences in the network, we use d_i to denote the distance between the *i*th mobile device and the edge sever for wireless access. We assume the average channel gain h_u to follow the path consumption model. To generate the time-varying wireless channel gain $h_{t,i} = h_{t,1}, h_{t,2} \ldots h_{t,n}$, we use the Rayleigh fading channel model as $h_{t,i} = \alpha_{t,i} h_u$, where $\alpha_{t,i}$ is an independent random channel fading factor following the unit average exponential distribution. Then $h_{t,i}$ is

$$h_{t,i} = \alpha_{t,i} h_u = \alpha_{t,i} A_d \left(\frac{3 * 10^8}{4\pi f_c d_i}\right)^{d_e}, \tag{3}$$

where A_d denotes the antenna gain, f_c denotes the carrier frequency, and d_e denotes the path loss index. We assume that the channel gain remains constant in each time slot, and that the channel gain changes independently with the time slot.

C. Computation Model

In this subsection, we will present the computational models in local devices and edge computing. We assume that after the

offloading tasks are executed, the edge server returns them to the mobile devices as a whole. In time slot *t*, if $a_{t,i} = 1$, we will use $D_{t,i}$ to represent the *i*th device's task data or we will use $L_{t,i}$. They are presented as the number of CPU cycles required to accomplish the tasks. It is worth noting that we use $D_{t,i}$ and $L_{t,i}$ to represent different states of task data, although their values are the same. It is important to highlight that these values do not necessarily indicate the actual amount of data executed, as we will explain later.

1) Edge Computing: In our analysis of task offloading on the edge side, we take into consideration both the delay of task offloading and the delay of task execution. They form the total delay together.

In alignment with prior research [11], [23], the exclusion of the time and energy consumption related to transmitting computation results back to the mobile device is warranted. This decision arises from the observation that input data, such as images or videos, tends to possess considerable size, demanding substantial computational resources for processing. Conversely, the resulting output data typically assumes a much smaller scale, exemplified by a solitary classification label for an image or a concise summary of a video clip. Consequently, the time and energy entailed in transmitting the output data back to the mobile device can be considered inconsequential when compared with the time and energy expended in processing the input data. This is an important consideration when optimizing task offloading strategies in edge computing systems. It allows us to focus primarily on reducing the delay and energy consumption of task offloading and execution, without being overly concerned with the transmission of the results back to the mobile device. Then, the task transmission latency for device *i* can be obtained from

$$T^u_{t,i} = \frac{\phi D_{t,i}}{R_{t,i}} = \frac{\phi D_{t,i}}{W_t log_2\left(1 + p^{send} h_{t,i}/\sigma^2\right)}. \tag{4}$$

Parameter $\phi > 0$ denotes the number of computation cycles needed to process one bit of raw data. We consider that F^e denotes the maximum computational frequency that the edge can provide. The optimal utilization of computational resources at the edge entails the allocation of said resources in accordance with the task data. Through this method, a majority of tasks executed at the edge yield outcomes simultaneously, narrowing the gaps stemming from the time it takes for devices to transmit their respective tasks. The purpose of setting up the edge end in this way is to enhance fairness between devices. At the beginning of each time slot, all devices have the ability to compete for computing resources at the edge. Therefore, the total latency of a device whose offloading task data is $L_{t,i}$ in time slot *t* is

$$\begin{aligned} T^e_{t,i} &= \frac{\phi D_{t,i}}{R_{t,i}} + \frac{\sum_{i\in N} D_{t,i}}{F^e} \\ &= \frac{\phi D_{t,i}}{W_t log_2\left(1 + p^{send} h_{t,i}/\sigma^2\right)} + \frac{\sum_{i\in N} D_{t,i}}{F^e}. \end{aligned} \tag{5}$$

We consider energy consumption as a significant factor in the performance of the system. We divide the energy consumption into two parts: the energy consumption of local task execution and the energy consumption of task offloading. We do not consider the energy consumption for executing tasks on the edge server because it is typically equipped with a power supply and has sufficient energy for task execution. Thus, the energy consumption of task offloading is

$$\begin{aligned} E^u_{t,i} &= p^{send} \frac{\phi D_{t,i}}{R_{t,i}} \\ &= p^{send} \frac{\phi D_{t,i}}{W_t log_2\left(1 + p^{send} h_{t,i}/\sigma^2\right)}. \end{aligned} \tag{6}$$

2) Local Computing: If a task on MD_i is computed locally, the execution on the local device will be influenced by the edge device. The edge sever returns the expected time slot T_s to the local device, as follows:

$$T_s = \frac{\sum_{i\in N} D_{t,i}}{F^e} + T^u_{t,m} \tag{7}$$

where $T^u_{t,m}$ denotes the maximum transmission latency among all devices in which the offloading execution occurs during this time slot. $T^u_{t,m}$ can be obtained by $max\{T^u_{t,i}\}$ and $T^u_{t,i}$ is calculated by equation (4). It is worth noting that T_s is dynamically changing according to the environment in different time slots. MD_i uses dynamic voltage and frequency scaling (DVFS) to adjust the chip voltage according to T_s and its own task data. By changing its own frequency, the power of MD_i can be adjusted to change the energy consumption. The CPU frequency $f^l_{t,i}$ calculated locally by MD_i can be written as

$$f^l_{t,i} = \left\lceil \frac{L^r_{t,i}}{T_s} \right\rceil = \left\lceil \frac{L^r_{t,i}}{\sum_{i\in N} D_{t,i}/F^e + T^u_{t,m}} \right\rceil \tag{8}$$

where $L^r_{t,i}$ represents the actual task data performed locally by the device. The CPU frequency $f^l_{t,i}$ is the frequency supported by the system by searching from the table (voltage-frequency table). $f^l_{t,i}$ has a limitation $f^l_{t,i} \in (0, F^l]$, where F^l denotes the maximum frequency of executing tasks locally. The maximum processing capacity of task data will also change with the change of frequency. Maximum data processed by the local device in time T_s can be obtained from

$$L^{max}_{t,i} = \left(\frac{\sum_{i\in N} D_{t,i}}{F^e} + T^u_{t,m} \right) * F^l. \tag{9}$$

If the local task data $L_{t,i} \le L^{max}_{t,i}$, then we adjust the execution frequency to make $L^r_{t,i} = L_{t,i}$. If $L_{t,i} > L^{max}_{t,i}$, then $L^r_{t,i} = L^{max}_{t,i}$. The remaining task data that cannot be completed with the maximum frequency F^l in this time slot is merged with the task data in the next time slot. However, the task here need to be at the same layer in directed acyclic graph. If this is the last task at this layer, the task will be forced to complete before proceeding to the next time slot. That means the latency T_s of the last task of each directed acyclic graph needs to be executed is $(L_{t,i} + L_{left})/F^l$, where L_{left} represents the remaining data from the previous task.

For example, in Fig. 2(b), task *D*, task *G* and task *I* need particularly considering. If they are selected to execute locally during this time slot and are not completed with the time returned at the edge, we will need to wait for their task execution to complete before next time slot. For other tasks, if there is an unfinished part of the local execution in the time

slot returned at the edge, we will merge it with the next task. This also potentially increases the data of this task in next time slot, thereby enhancing its competitiveness against edge resources. We derive the power to show the process of adjusting the frequency of mobile devices and thus affecting the energy consumption of devices. Local device's power $p_{t,i}^{l}$ can be measured by

$$p_{t,i}^{l} = k\left(f_{t,i}^{l}\right)^{3} \tag{10}$$

where k is the effective exchange capacitance value, limited by the chip architecture. If the local energy is sufficient, the energy consumption for device i will be:

$$E_{t,i}^{l} = p_{t,i}^{l} * T_s = k\left(f_{t,i}^{l}\right)^{3} T_s. \tag{11}$$

D. Problem Formulation With Task Dependency

To minimize the energy-efficiency cost of all devices, the weighted sum of the latency and energy consumption is used by the system. The total energy-efficiency cost formula in the system in one time slot is:

$$Z_{evl} = \beta_t \left(\sum_{\substack{a_{t,i}=1,\\ i=1}}^{N} T_{t,i}^{e} + \sum_{\substack{a_{t,i}=0,\\ i=1}}^{N} T_s \right) + \beta_e \left(\sum_{\substack{a_{t,i}=1,\\ i=1}}^{N} E_{t,i}^{u} + \sum_{\substack{a_{t,i}=0,\\ i=1}}^{N} k\frac{{L_{t,i}}^{3}}{(T_s)^2} \right), \tag{12}$$

where for the weighting method of the multi-objective optimization problem. As researchers in this field, we adopt the practice of setting $0 < \beta_t < 1$, and $0 < \beta_e < 1$ similar to existing works [15]. Here, β_t and β_e denote the weights given to computation completion time and energy consumption, respectively. We assume that these weights are related by $\beta_t+\beta_e=1$. All mobile devices can select different weights based on specific needs together. For instance, all devices with low battery energy would be more inclined towards low energy consumption, while latency-sensitive devices would prefer a higher β_t to reduce execution time. It is worth noting that Z_{evl} does not represent the actual energy-efficiency cost in the system. When calculating energy consumption at the local, we use the total amount of task data for device i. This approach effectively prevents the false low energy consumption in time slot t, where only a small amount of data is offloaded to the edge in the system, resulting in a large amount of data left locally until the next slot. After making the offloading decision, we calculate the actual EEC of the system as

$$Cost_{sys,t} = \beta_t \left(\sum_{\substack{a_{t,i}=1,\\ i=1}}^{N} T_{t,i}^{e} + \sum_{\substack{a_{t,i}=0,\\ i=1}}^{N} T_s \right) + \beta_e \left(\sum_{\substack{a_{t,i}=1,\\ i=1}}^{N} E_{t,i}^{u} + \sum_{\substack{a_{t,i}=0,\\ i=1}}^{N} E_{t,i}^{l} \right). \tag{13}$$

Then, we will give the total energy-efficiency cost for one iteration. The EEC formula of the system at the edge is calculated in the same way as the traditional method:

$$Cost_{edge} = \beta_t \sum_{t=1}^{M} \sum_{\substack{a_{t,i}=1,\\ i=1}}^{N} T_{t,i}^{e} + \beta_e \sum_{t=1}^{M} \sum_{\substack{a_{t,i}=1,\\ i=1}}^{N} E_{t,i}^{u}. \tag{14}$$

When we determine the offloading decision, the optimized algorithm results in a distinction while performing tasks on the local side. In our optimization algorithm, the overall energy-efficiency cost of the system is

$$Cost_{sys} = Cost_{edge} + \beta_t \sum_{t=1}^{M} \sum_{\substack{a_{t,i}=0,\\ i=1}}^{N} T_s + \beta_e \sum_{t=1}^{M} \sum_{\substack{a_{t,i}=0,\\ i=1}}^{N} k\frac{{L_{t,i}^{r}}^{3}}{(T_s)^2}. \tag{15}$$

According to equation (15), the local system's EEC is a quadratic function of the latency on edge. The more tasks from less devices are offloaded at the edge server, the more local energy consumption can be reduced. If the traditional algorithm is used to process these task data, it is not necessary to consider the frequency adjustment of devices performing tasks locally. The latency and energy consumption generated by traditional resource allocation formula is

$$Cost_{tro} = Cost_{edge} + \beta_t \sum_{t=1}^{M} \sum_{\substack{a_s=0,\\ i=1}}^{N} \frac{L_{t,i}}{F^{l}} + \beta_e \sum_{t=1}^{M} \sum_{\substack{a_s=0,\\ i=1}}^{N} kL_{t,i}{F^{l}}^{2}. \tag{16}$$

Equation (16) represents the energy efficiency cost of the traditional algorithm, which is an important parameter to consider in our later comparison of algorithms. Due to the absence of frequency adjustment for mobile devices, the traditional algorithms may consume more energy than the optimized ones. This makes the traditional algorithm less suitable for resource-constrained equipment with limited battery life. Later in our comparison, we will calculate energy efficiency cost of DQN algorithm through this equation. It will be used to evaluate the performance of our proposed algorithm, DODA-DT, against other algorithms. The comparison will shed light on the advantages of our algorithm over traditional cloud computing models. In Table I, we have summarized all the notation that appear in this chapter and their meanings.

IV. Optimization of Dependent Task Offloading

In this section, a DRL-based algorithm is proposed to solve the combined optimization problem for various task data and wireless channels. The aim is to generate an offloading decision in a multi-device mobile edge network swiftly. The algorithm makes decisions in an environment where both channel gain and device data are dynamically changing.

TABLE I
NOTATIONS

Communication Model	
$a_{t,i}$	Whether or not the ith mobile device (MD_i) is going to offload the task in time slot t
MD_i	The ith mobile device
W_t	Channel bandwidth for time slot t
B	Total bandwidth
$R_{t,i}$	Channel transmission rate
$h_{t,i}$	Time-varying wireless channel gain
p^{send}	Transmit power of the mobile device during the task upload to the edge server
σ^2	Channel noise power density
$\alpha_{t,i}$	Independent random channel fading factor following the unit average exponential distribution
A_d	Antenna gain
f_c	Carrier frequency
d_e	Path loss index
Computation Model	
$D_{t,i}$	The offloading task executed by the ith device in time slot t
$L_{t,i}$	The local task executed by the ith device in time slot t
Edge Computing	
$T^u_{t,i}$	Task transmission latency
ϕ	The number of computation cycles needed to process one bit of raw data
F^e	Maximum computational frequency that the edge can provide
$T^e_{t,i}$	Total latency of a device whose offloading task
$E^u_{t,i}$	Energy consumption of task offloading
Local Computing	
T_s	The edge sever returns the expected time slot
$T^u_{t,m}$	maximum transmission latency among $T^u_{t,i}$
$L^r_{t,i}$	Actual task data performed locally by the ith device in time slot t
$f^l_{t,i}$	actually CPU frequency
F^l	maximum frequency of executing tasks locally
$L^{max}_{t,i}$	actually task data executed locally
L_{left}	the remaining data from the previous task
$p^l_{t,i}$	local execution power
k	effective exchange capacitance value
$E^l_{t,i}$	energy consumption of the ith device locally
Problem Formulation with Task Dependency	
Z_{evl}	the weighted cost used by the system in experiment
$Cost_{sys,t}$	the actual EEC of the whole system in a training time
$Cost_{edge}$	cost of the whole system at the edge
$Cost_{sys}$	the overall energy-efficiency cost of the whole system
$Cost_{tro}$	cost generated by traditional resource allocation

The optimal offloading decision theoretically can be selected among 2^N offloading decisions. However, in the multi-device scenario, as the number of mobile devices increases, the search space grows exponentially. In a mobile edge computing environment where parameters are ever-changing, the existing offloading decisions necessitate frequent iterations. This leads to slow convergence and high computational effort.

Algorithm 1 Offloading Decision of Dependency Tasks

```
Initialize the DNN;
Initialize the experience replay memory;
Set memory size, Mem ← 3N;
Initialize the EEC queue Q_s;
for t = 1, 2, ..., M do
    Empty S_ac;
    Input parameters into the DNN;
    Get the offloading parameters X_s;
    Convert X_s to a set of candidate binary offloading actions S_ac with algorithm 2;
    for A_s in S_ac do
        Update the EEC queue Q_t with equation (12);
    end for
    Select the best action Â_s with the lowest cost in Q_t;
    Execute the offloading decision Â_s
    Save the cost in the queue Q_s with equation (13);
    Update experience replay memory with {L, h, Â_s};
    if mod(t, δ) = 0 then
        Randomly select batches from memory;
        Train the DNN with the batches;
        Update gradient LS(θ^t) using Adam algorithm;
    end if
end for
Save the sum of Q_s as the result of one iteration;
```

The algorithm's structure is illustrated in Fig. 3, where we have implemented an actor-critic based network to determine offloading decisions. Our algorithm has three major modules: the actor module, which receives input parameters and produces a set of potential offloading actions; the critic module, which identifies the optimal offloading action; and the policy update module, which updates the system's states following the execution of offloading actions. These modules function in a sequential and iterative manner. The overall process of the algorithm is shown in Algorithm 1.

A. Actor Module

The actor module is used to generate offloading parameters x_t. In time slot t, we take the task data and channel gain of the mobile device as input parameters for the DNN. Then the DNN outputs a set of candidate offloading parameters $X_s = \{x_{t,i} \in (0,1), i = 1, 2, \ldots, N\}$. While constructing the model, we adopt a sigmoid function where each offloading parameter is a value between 0 and 1. By utilizing a threshold of 0.5, we generate an alternative set of offloading actions using the binary representation {0,1} for these offloading parameters. $A_s = \{a_{t,i} \in \{0,1\}, i = 1, 2, \ldots, N\}$ denotes the alternative offloading policy. In the subsequent algorithm, each time slot will generate $2N + 1$ kinds of alternative strategies. This collection uses S_{ac}.

When generating alternative strategies from DNN output parameters, the accuracy of the results improves with an

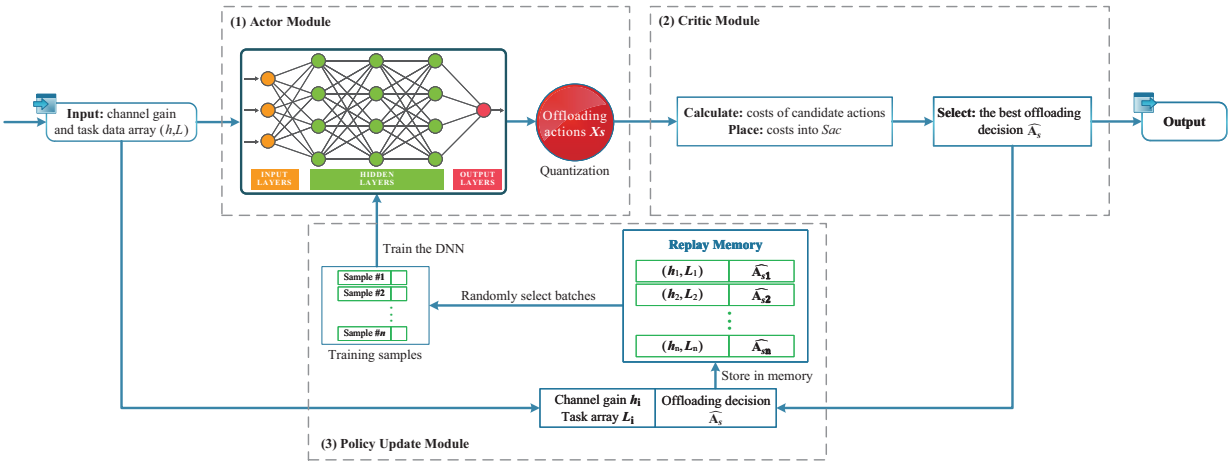


Fig. 3. The schematics of the proposed offloading decision algorithm.

increase in the number of alternative strategies. However, such increment may potentially negate the advantage of DNN's quick response time. To leverage the generated data features fully, we propose a strategy that generates alternative strategies by comparing the data itself. DODA-DT algorithm produces more diverse results than traditional KNN clustering and ensures the order of DNN output parameters. The order-preserving quantization method, initially introduced in [33], was applied to explore the output of the DNN. The method is designed to maintain the order of all elements in a vector both before and after quantization. Specifically, for each alternative set of offloading actions A_s in WD_i the first offloading policy is determined as

$$a_{t,i} = \begin{cases} 1, & x_{t,i} > 0.5 \\ 0, & x_{t,i} \leq 0.5 \end{cases} \tag{17}$$

In our approach, we generate the first offloading policy and then proceed to compare all offloading actions with each parameter of its own. To do this, we take each parameter in the set as a standard, resulting in $N + 1$ alternative offloading parameters X_t. The new alternative offloading decision is obtained using equation below

$$a_{t,i} = \begin{cases} 1, & x_{t,i} > x_{t,j}, \\ 1, & x_{t,i} = x_{t,j}, \quad x_{t,j} \leq 0.5, \\ 0, & x_{t,i} < x_{t,j}, \\ 0, & x_{t,i} = x_{t,j}, \quad x_{t,j} > 0.5. \end{cases} \tag{18}$$

where $x_{t,i}$ is the parameters for comparison and $x_{t,j}$ is the standard. In order to explore more possible results and prevent the model from converging too fast and falling into a local optimum solution, we design two types of Gaussian noise $noise_b \sim \mathcal{N}(0,1)$, $noise_l \sim \mathcal{N}(0,0.5)$, and different approaches to use these two types of noise. Specifically, the Gaussian noise $noise_b$ is added into each round along with the original DNN output parameters for the above binary quantization execution, generating a total of $2N + 1$ alternatives in each time slot. In each cycle φ, the Gaussian noise $noise_l \sim \mathcal{N}(0,0.5)$ no longer uses $noise_b$, and $noise_l$ is added to the DNN-generated parameters N times. Said parameters are quantised together with the original DNN output parameters, generating $2N + 1$ alternatives, as seen in Algorithm 2. Then the actor network will send these candidate offloading actions to the critic network.

Algorithm 2 Generating Offloading Action Candidates

```
Initialize offloading parameters from DNN relaxation X_s;
Initialize and empty the list S_ac;
Convert X_s to a set of candidate binary offloading actions
  A_s with equation (17) (18);
Add A_s to S_ac;
if mod(t, φ) = 0 then
    for i = 1, 2, ..., N do
        X_a1 = X_s + noise_b + noise_l;
        Convert X_a1 to A_s1 with equation (17);
        Add A_s1 to S_ac;
    end for
else
    X_a2 = X_s + noise_b;
    for i = 1, 2, ..., N do
        Convert X_a2 to A_s2 with equation (18);
        Add A_s2 to S_ac;
    end for
end if
return Candidate offloading actions S_ac;
```

B. Critic Module

After the actor module generates a set of candidate offloading actions, the critic module in our algorithm evaluates these actions and selects the best one, denoted as $\widehat{A_s}$, based on the model information.

Specifically, the critic network calculates the possible running time required based on the current channel conditions of the task selected for offloading, from which the decision with the lowest energy-efficiency cost is chosen as the offloading solution. The critic network uses the Z_{evl} function to process the offloading decisions set S_{ac} and then select the best offloading decision $\widehat{A_s}$.

When executing the offloading decision $\widehat{A_s}$, the local devices are adjusted according to the $\widehat{A_s}$, adjusting the

frequency according to equation (13) and recording the energy-efficiency cost in time slot *t*. We can observe the actual energy efficiency cost denoted by $Cost_{sys,t}$ (the calculation method is in Section III). Then, we utilize $Cost_{sys,t}$ to accurately calculate the energy-efficiency cost in the system and store it.

C. Policy Update Module

We update the parameters in the DNN by updating the offloading policy. We design an experience replay memory to store the past state-action pairs. In each time slot, the task data collection of all devices *L*, channel gain set of all devices *h*, and the offloading decision $\widehat{A_s}$ generated in the actor-critic network at *t*-th epoch as a whole $\{L, h, \widehat{A_s}\}$, are added to the memory as new training data.

In our implementation, we begin training the DNN only after collecting the data samples more than half of the memory size. However, new data is constantly added to the memory and will replace the old data once the memory is full. The memory is used to train the DNN within the update interval δ. Specifically, at intervals of *t* time slots, we engage in regular training sessions with the DNN to prevent excessive fitting of the model. When $mod(t, \varphi) = 0$, a random batch of data samples (θ^t, A^τ), $\tau \in S^t$ is selected. The set S^t represents the set of time indices for the chosen samples. Subsequently, we proceed to update the parameters of the DNN by using the Adam algorithm, which aims to minimize the average cross-entropy loss function $LS(\theta^t)$ over the aforementioned data samples. The loss function $LS(\theta^t)$ can be represented as

$$LS(\theta^t) = -1/|S^t| \sum_{\tau \in S^t} \Big[(A^\tau)^T \log f_{\theta^t}(\xi^\tau) + (1 - A^\tau)^T \log(1 - f_{\theta^t}(\xi^\tau)) \Big], \quad (19)$$

where $|S^t|$ denotes the size of the sample batch, $(.)^T$ represents the transpose operator, which allows for the manipulation of vectors and matrices, and *log* refers to the element-wise logarithm operation performed on a vector.

The complexity of DODA-DT hinges on two crucial operations. Algorithm 1, with a complexity of *O*(*M*), contributes to the complexity in one time slot. Secondly, the complexity of invoking Algorithm 2 is $O(N^2)$. As the value of *N* increases, the running time of DODA-DT increases as well. Worst-case scenario, the algorithm exhibits a complexity of $O(N^2 + 2N)$ within a single time slot. Notably, in our experiment, where each iteration comprises *M* time slots, the total time complexity for one iteration amounts to $O(N^2 \times M)$. This time complexity configuration in practical scenarios ensures nearly simultaneous execution, assuring a high level of efficiency.

V. Performance Evaluation

In this section, our team evaluates the performance of our algorithm through numerical simulations. We implement and test the algorithm on PyTorch. As equation (3), We assume d_i to be uniformly distributed between 8 meters and 12 meters, and the average channel gain h_u to follow the path consumption model. $A_d = 3$ denotes the antenna gain, $f_c = 300$ MHz denotes the carrier frequency, and d_e=2.8 denotes the path loss index. We set the wireless bandwidth W to 16 MHz, the channel noise power σ^2 to 10^{-26} W, and the power of the local transmitting task p^{send} to $2\,W$. Additionally, we set the training interval δ to 20, memory size M to 1024 MB, batch size to 128 MB, and the training frequency φ of $noise_l$ to 20. Each layer's tasks are randomly assigned between 20 and 100, with the amount of data per task L_i randomly distributed between 10 MB and 120MB. Each iteration has 400 time slots and we consider 400 iterations. The maximum task execution frequency for mobile devices F^l to 0.5 GHz. The total task execution frequency for edge servers F^e is 10 GHz, and the weighting factor for latency and power consumption is 0.5. The parameters used in PyTorch are listed in Table II. The proposed actor network uses a fully connected multilayer perceptron with two hidden layers containing 120 and 80 hidden neurons, respectively. We set the learning rate of the Adam optimizer to 0.01.

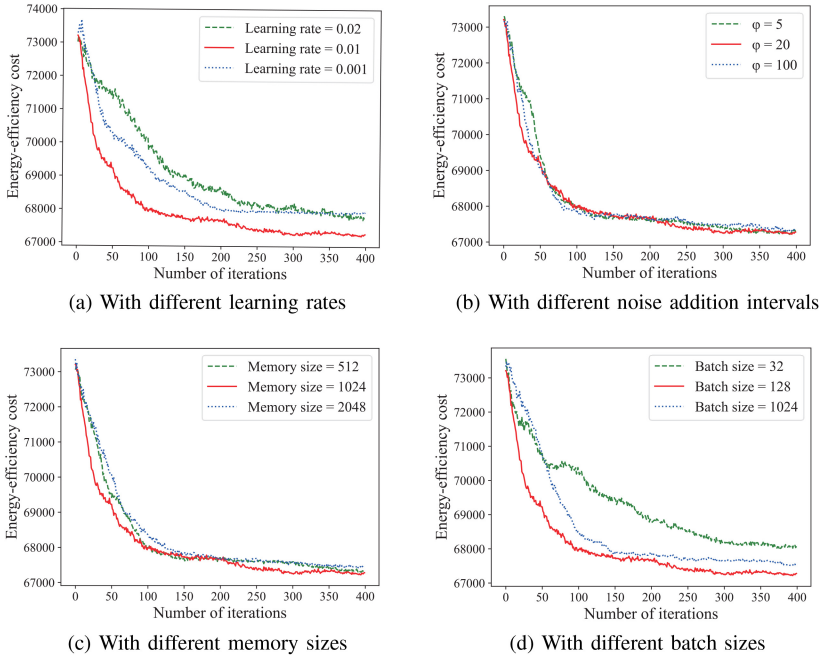
TABLE II
Simulation Parameters

Parameter	Value
Distance of mobile devices to the edge sever d_i	$\mathcal{U}(8, 12)\, meters$
Antenna gain A_d	3
Carrier frequency f_c	300 MHz
Path loss index d_e	2.8
Wireless bandwidth W	$16MHz$
Channel noise power σ^2	10^{-26} $Watt$
Power of the local transmitting task p^{send}	2 $Watt$
Training interval δ	20
Amount of data per task L_i	$\mathcal{U}(10, 120)\, MB$
Maximum frequency for mobile devices F^l	0.5 GHz
Frequency for edge servers F^e	$10GHz$

During the simulation, we generate random data and dependency tasks for mobile devices and proceeded to group the tasks using the breadth-first search approach. To reduce the error caused by random task data, we will evaluate the energy-efficiency cost every 400 time slots, which constitute one iteration. In terms of balancing coefficients, since our algorithm can optimize both delay and energy consumption, we represent the coefficients as the ratio of the optimization results of individual energy consumption to the optimization results of individual delay. This allows for a comprehensive evaluation and effective trade-off between these two essential factors. Next, we evaluate the influence of various parameters in the algorithm.

A. Influence of Parameters in the Algorithm

1) Impact of the Learning Rate: We examine the impact of different learning rates on the performance of our proposed model in Fig. 4(a). We observe that after 400 iterations, the learning rate of 0.001 and 0.02 result in the premature convergence of the model. A low learning rate may result in the



(a) With different learning rates

(b) With different noise addition intervals

(c) With different memory sizes

(d) With different batch sizes

Fig. 4. The energy-efficiency cost for the task graph with different parameters.

model being unable to jump out after entering local optima, while a high one may cause a miss-out on an excellent offloading action. Our results indicate that learning rate of 0.01 in the Adam optimizer is more effective than the ones either too high (0.02) or too low (0.001).

2) Impact of the Noise Addition Frequency: As shown in Fig. 4(b), it is observed that adjusting the addition interval of $noise_l$ does not generally have a significant impact. However, appropriate noise additions allow the exploration of more operational results and can sequentialize the parameters output by the DNN. When $\varphi = 5$, much noise addition may make the original DNN parameters unstable, resulting in chaotic judgments. Too little noise addition $(\varphi = 100)$ may limit the exploration of new states. In our deep reinforcement learning model, the generated action state is used as the label, so insufficient exploration of states can negatively impact model convergence.

3) Impact of the Memory Size: We vary the memory size as a parameter to evaluate its impact on the overall performance of the system. Unlike batch size, the size of memory directly affects the amount of training data available for model learning. A larger memory size enables storing more historical records, so that the training of the model has more options. However, the memory size should not be set too high as it may lead to overfitting, where the model becomes too specific to the training data and performs poorly. On the other hand, a smaller memory size spells that the memory needs frequent updating. Even though it is possible to update the parameters of DNN in a timely manner, we will yet be largely confined when selecting fewer records. As depicted in Fig. 4(c), we observe that a appropriate memory size can accelerate the rate of convergence of the model and make sufficient choices for model training. Adequate model training can further approach the optimal offloading action.

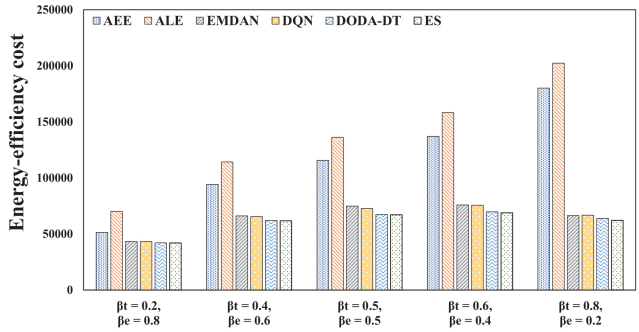
4) Impact of the Batch Size: To ensure that the DNN model learns effectively, we conduct experiments with different batch sizes and analyzed their impact on the model's performance. As shown in Fig. 4(d), we observe that the batch size has a significant impact on the overall energy-efficiency of the system. When setting that size to a smaller value such as 32, DNN may not learn enough to produce accurate results and lead to slow convergence. When setting to 1024, the model becomes over-reliant on history records and fails to discover new action states, leading to premature convergences and local optima. Consequently, reasonable selection of batch size is a requisite for the model to fully fathom out the aforementioned states and strike a balance with the help of past experience.

Through the above experiments, in the subsequent program, we will use the parameters that indicate the best performance, namely, a learning rate of 0.01, $\varphi = 20$, memory size of 1024 MB, and a batch size of 128 MB.

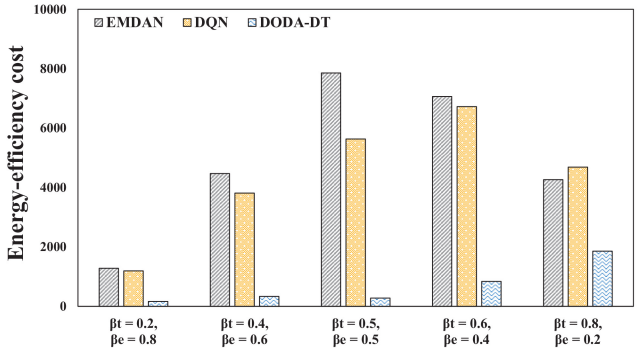
B. Comparison With Other Algorithms

Finally, we compare the DADO-DT algorithm with the other algorithms.

- All Edge Execution (AEE): All UEs offload their tasks and they need to compete for limited communication and computing resources.
- All Local Execution (ALE): In contrast to AEE, it does not allow offloading and all tasks to be executed locally.
- Exhaustive Search (ES): We perform all possible operations on the system, find the lowest offloading decision, and execute it.
- Execution Method without Different Noise Addition (EMDAN): DODA-DT without different noise addition.
- Deep Q-Network Execution (DQN): In the context of the DQN framework, the Q function is represented by a deep neural network, serving as our modeling tool. As



(a) Comparison of EEC between six algorithms



(b) Three algorithms exceed the EEC of the ES algorithm

Fig. 5. Comparisons of EEC for different weights between execution time and energy consumption.

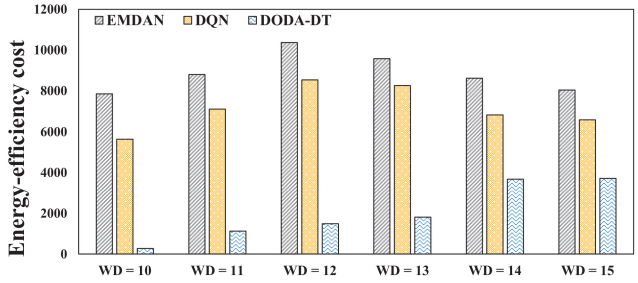


Fig. 6. Comparisons of EEC for different numbers of WDs.

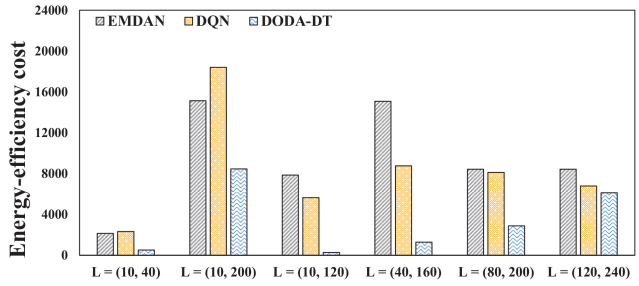


Fig. 7. Comparisons of EEC for different task data.

each time slot t unfolds, the offloading decision process regarding the next action hinges upon the evaluation of the prevailing state. The current state, the chosen action, the subsequent state, and the corresponding reward, are recorded and stored within the memory pool. To ensure continuous improvement, we extract batches of data on a regular basis from the memory pool and utilize for training purposes.

1) Influence of Different Weights: We investigate the effects of adjusting the weights between execution time and energy consumption in optimizing the performance of the system. After adjusting the initial proportion of parameters based on the optimization results of single optimization objective (i.e., single latency and single energy consumption), we reflect the different focuses of multi-objective optimization through appropriate weighting. As shown in Fig. 5(a), we observe that the AEE algorithm and ALE algorithm generate extremely high EEC. Due to the limitations of channel bandwidth and network environment, offloading all tasks to the edge will significantly reduce resource utilization. Executing all tasks locally has also resulted in low usage of the edge. These two algorithms have significantly higher EEC than the other four, which also confirms the importance of offloading decision.

In Fig. 5(b), we compare the portions of EMDAN, DQN, and DODA-DT algorithms that exceed the ES algorithm. These excess EEC effectively reflect the gap with the best solution. Though the process of approaching a single optimization goal may reduce this optimization effect, we find that the DODA-DT algorithm has excellent performance under different latency and energy consumption weights. Compared with the EMDAN algorithm, DODA-DT has a large number of higher quality labels, so it can train the model faster and more smoothly. Compared with the DQN algorithm, the DODA-DT algorithm also has obvious advantages, and the performance of the DODA-DT algorithm is comparable to that of the ES algorithm in most cases.

2) Influence of the Number of Devices: Different numbers of devices can lead to an increase in the amount of task data in a time slot. At the same time, the increase in devices also brings pressure to the communication network. The EEC of AEE and ALE significantly increases with the increase of device quantity. In Fig. 6, we compare the optimization EEC of DQN, EMDAN, and DODA-DT algorithms that are relative to the ES algorithm. As the number of devices increases, the ES algorithm increases by 2^N, meaning it becomes harder to obtain the best offloading decision increases. When the number of devices increases to a certain level, the main bottleneck of the system may become competition for network channels. The limited bandwidth and quality of wireless channels may gradually weaken the advantages of the DODA-DT algorithm. This leads to EMDAN and DQN algorithms gradually approaching the results of DODA-DT algorithms.

3) Influence of the Task Data: In Fig. 7, we present the impact of task data on the system performance. Specifically, we vary the amount of task data for each device within a given range L in one iteration. Our results indicate that the proposed DODA-DT is robust and performs well under different amounts of task data. Moreover, when we expand the range of task data, the advantages of our algorithm are even more evident. In Fig. 7, we also find that the most effective scenario of DODA-DT algorithm is that there are both tasks with Big data and tasks with small data, which are more like realistic situations. When the data of tasks maintains at a high level, the gap between the three algorithms decreases due to

the network resource constraints. In the future work, we will consider using multiple edge servers to solve the problem of resource constraints at the edge.

VI. Conclusion

This paper proposes a deep reinforcement learning-based offloading decision strategy that maximizes the system's computing power.The DODA-DT algorithm enables all devices to compete for the computing resources available in each time slot, ensuring the fairness of all mobile devices to the maximum extent. Additionally, an order-preserving quantization algorithm is used to deal with the output parameters of the DNN, accelerating the convergence of the model. The experimental results indicate that DODA-DT effectively enhances the energy efficiency of the system.

Acknowledgment

The authors would like to thank the reviewers for their efforts and for providing helpful suggestions that have led to several important improvements in their work. They would also like to thank all teachers and students in their laboratory for helpful discussions.

References

[1] "Cisco and SAS edge-to-enterprise IoT analytics platform." Accessed: Feb. 2017. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/unified_computing/ucs/UCS_CVDs/Cisco_SAS_Edge_to_Enterprise_IoT_Analytics_Platform.html

[2] "Ericsson mobility report update: Global 5G subscriptions close to 1.3 billion in Q2." Accessed: Aug. 2023. [Online]. Available: https://www.ericsson.com/en/digitalservi

[3] T. X. Tran and D. Pompili. "Joint task offloading and resource allocation for multi-server mobile-edge computing networks." 2017. [Online]. Available: https://arxiv.org/abs/1705.00704.

[4] G. Mitsis, E. E. Tsiropoulou, and S. Papavassiliou, "Price and risk awareness for data offloading decision-making in edge computing systems," *IEEE Syst. J.*, vol. 16, no. 4, pp. 6546–6557, Dec. 2022.

[5] M. Wang, T. Ma, T. Wu, C. Chang, F. Yang, and H. Wang, "Dependency-aware dynamic task scheduling in mobile-edge computing," in *Proc. 16th Int. Conf. Mobil. Sens. Netw. (MSN)*, 2020, pp. 785–790.

[6] 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 27th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, 2021, pp. 225–232.

[7] R. Chai, M. Li, T. Yang, and Q. Chen, "Dynamic priority-based computation scheduling and offloading for interdependent tasks: Leveraging parallel transmission and execution," *IEEE Trans. Veh. Technol.*, vol. 70, no. 10, pp. 10970–10985, Oct. 2021.

[8] J. Fang and A. Ma, "IoT application modules placement and dynamic task processing in edge-cloud computing," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12771–12781, Aug. 2021.

[9] H. Li, H. Xu, C. Zhou, X. Lü, and Z. Han, "Joint optimization strategy of computation offloading and resource allocation in multi-access edge computing environment," *IEEE Trans. Veh. Technol.*, vol. 69, no. 9, pp. 10214–10226, Sep. 2020.

[10] 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.

[11] W. Zhan, C. Luo, G. Min, C. Wang, Q. Zhu, and H. Duan, "Mobility-aware multi-user offloading optimization for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 3, pp. 3341–3356, Mar. 2020.

[12] P. Wei et al., "Reinforcement learning-empowered mobile edge computing for 6G edge intelligence," *IEEE Access*, vol. 10, pp. 65156–65192, 2022.

[13] C. Li et al., "Dynamic offloading for multiuser multi-CAP MEC networks: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 70, no. 3, pp. 2922–2927, Mar. 2021.

[14] J. Heydari, V. Ganapathy, and M. Shah, "Dynamic task offloading in multi-agent mobile edge computing networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.

[15] J. Yan, S. Bi, and Y. J. Angela 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.

[16] J. Zou, T. Hao, C. Yu, and H. Jin, "A3C-DO: A regional resource scheduling framework based on deep reinforcement learning in edge scenario," *IEEE Trans. Comput.*, vol. 70, no. 2, pp. 228–239, Feb. 2021.

[17] 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, Sep. 2021, doi: 10.1109/TCCN.2021.3066619.

[18] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[19] C. Li, G. Zhou, Y. Guo, J. Hou, H. Zhang, and S. Li, "Resource allocation strategy for virtualized wireless sensor networks based on actor-critic," in *Proc. IEEE 21st Int. Conf. Ubiquitous Comput. Commun. (IUCC/CIT/DSCI/SmartCNS)*, 2022, pp. 132–139.

[20] S. Leng and A. Yener, "An actor-critic reinforcement learning approach to minimum age of information scheduling in energy harvesting networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2021, pp. 8128–8132.

[21] S. Leng and A. Yener, "Age of information minimization for wireless ad hoc networks: A deep reinforcement learning approach," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.

[22] S. Araf, A. S. Saha, S. H. Kazi, N. H. Tran, and M. G. Rabiul Alam, "UAV assisted cooperative caching on network edge using multi-agent actor-critic reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 72, no. 2, pp. 2322–2337, Feb. 2023.

[23] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 18, no. 2, pp. 319–333, Feb. 2019.

[24] W. Zhang and Y. Wen, "Energy-efficient task execution for application as a general topology in mobile cloud computing," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 708–719, Jul.-Sep. 2018.

[25] 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)*, 2016, pp. 1451–1455.

[26] 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 Distrib. Syst.*, vol. 32, no. 11, pp. 2777–2792, Nov. 2021.

[27] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017, doi: 10.1109/TWC.2016.2633522.

[28] J. Zhou, D. Tian, Z. Sheng, X. Duan, and X. Shen, "Distributed task offloading optimization with queueing dynamics in multiagent mobile-edge computing networks," *IEEE Internet Things J.*, vol. 8, no. 15, pp. 12311–12328, Aug. 2021.

[29] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 668–682, Mar. 2019.

[30] G. Qu, H. Wu, R. Li, and P. Jiao, "DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 3448–3459, Sep. 2021.

[31] H. Tang, H. Wu, Y. Zhao, and R. Li, "Joint computation offloading and resource allocation under task-overflowed situations in mobile-edge computing," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 1539–1553, Jun. 2022.

[32] 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.

[33] L. Huang, S. Bi, and Y.-J. Angela 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.

Juan Fang (Member, IEEE) received the M.S. degree from the Jilin University of Technology, Changchun, China, in 1997, and the Ph.D. degree from the College of Computer Science, Beijing University of Technology, Beijing, China, in 2005. In 1997, she joined the College of Computer Science, Beijing University of Technology, where she has been a Professor since 2015. Her research interests include high performance computing, edge computing, and big data technology.

Huijie Chen (Member, IEEE) received the B.Eng. degree from the School of Computer Science, Henan University of Economics and Law, Zhengzhou, China, in 2010, the M.S.Eng. degree from the School of Computer Science, Taiyuan University of Science and Technology, Taiyuan, China, in 2013, and the Ph.D. degree in computer science from the Beijing Institute of Technology, Beijing, China, in 2020. He currently works with the School of Computer Science, Beijing University of Technology, Beijing. His research interests include smart sensing, crowd-sensing, and mobile-edge computing.

Dezheng Qu (Student Member, IEEE) received the B.S. degree from Shandong University, Weihai, China, in 2020. He is currently pursuing the M.Sc. degree with the Beijing University of Technology, Beijing, China. His main research direction is mobile-edge computing and task offloading. He is a Student Member of CCF.

Yaqi Liu (Student Member, IEEE) received the B.S. degree from the Beijing University of Technology, Beijing, China, in 2020, where she is currently pursuing the M.Sc. degree. Her main research direction is mobile-edge computing and task offloading. She is a Student Member of CCF.