Hindawi
Journal of Electrical and Computer Engineering
Volume 2023, Article ID 6134837, 16 pages
https://doi.org/10.1155/2023/6134837

Research Article

A Dual-Agent Approach for Coordinated Task Offloading and Resource Allocation in MEC

Jiadong Dong, Kai Pan, Chunxiang Zheng, Lin Chen, Shunfeng Wu, and Xiaolin Zhang

School of Electronic Engineering and Intelligent Manufacturing, Anqing Normal University, Anqing 246133, China

Correspondence should be addressed to Chunxiang Zheng; zhcx@aqnu.edu.cn

Received 27 August 2023; Revised 10 November 2023; Accepted 8 December 2023; Published 21 December 2023

Academic Editor: B. Rajanarayan Prusty

Multiaccess edge computing (MEC) is a novel distributed computing paradigm. In this paper, we investigate the challenges of task offloading scheduling, communication bandwidth, and edge server computing resource allocation for multiple user equipments (UEs) in MEC. Our primary objective is to minimize system latency and local energy consumption. We explore the binary offloading and partial offloading methods and introduce the dual agent-TD3 (DA-TD3) algorithm based on the deep reinforcement learning (DRL) TD3 algorithm. The proposed algorithm coordinates task offloading scheduling and resource allocation for two intelligent agents. Specifically, agent 1 overcomes the action space explosion problem caused by the increasing number of UEs, by utilizing both binary and partial offloading. Agent 2 dynamically allocates communication bandwidth and computing resources to adapt to different task scenarios and network environments. Our simulation experiments demonstrate that the binary and partial offloading schemes of the DA-TD3 algorithm significantly reduce system latency and local energy consumption compared with deep deterministic policy gradient (DDPG) and other offloading schemes. Furthermore, the partial offloading optimization scheme performs the best.

1. Introduction

Nowadays, with the rapid advancements in artificial intelligence (AI) and 5G technology, particularly in computer vision (CV) and natural language processing (NLP), an increasing number of diverse application scenarios have emerged. These scenarios include but are not limited to intelligent driving, smart homes, medical diagnosis, and intelligent manufacturing [1]. This has resulted in an increased demand for computational resources to process task requests from terminal users, in order to ensure better quality of experience (QoE) and quality of service (QoS). However, the computational capabilities and battery capacity of UEs are limited, and MEC provides an effective way to solve this problem [2, 3]. By offloading tasks wholly or partially to edge servers for execution, it reduces the computation delay and local energy consumption on UEs [4]. Designing and developing a rational offloading strategy to minimize system latency and local energy consumption is a crucial challenge.

DRL has shown promising results in achieving intelligent task offloading and resource allocation by learning from various data sources, such as terminal user behavior, network topology, and available computing resources. Its potential for broad application in computing offloading has been widely recognized [5, 6]. For the problem of task offloading among multiple UEs in the same time slot in MEC, it is necessary to first determine the task offloading decision for each UE. Existing research often utilizes centralized and distributed deep Q-network (DQN) algorithms to produce binary offloading decisions for multiple UEs. The centralized DQN algorithm generates offloading decisions for all UEs using a single agent, with a multiuser offloading decision vector serving as its action space [7]. This approach is simple and efficient in decision-making, but as the number of users increases, the exponential growth of the discrete action space makes it difficult for the algorithm to converge. In contrast, in the distributed DQN algorithm, each UE has an agent, and the action space of each agent is the set of nodes that can be offloaded to [8]. Although it can avoid the

problem of large action space caused by the increase of UEs, it also makes network training more difficult. To overcome the limitations of the above algorithms, this paper proposes the DA-TD3 algorithm that supports continuous action space, in which agent 1 can directly generate the task offloading rate for all users, significantly reducing the complexity of the problem.

When determining which tasks to offload to edge nodes, it is crucial to consider the allocation of communication bandwidth and edge node computing resources. In MEC, resource allocation needs to meet the task requirements while also taking into account the limitations of computing capability and network bandwidth of the edge nodes [9–11]. To address these challenges, we have designed agent 2 in the DA-TD3 algorithm to dynamically allocate communication bandwidth and edge server computing resources for multiple UEs. This approach allows for the adjustment of resource allocation ratios based on actual conditions, thereby avoiding resource waste and idle time, improving resource utilization, and better adapting to different task scenarios and network environments, thereby maximizing system performance.

The task offloading and resource allocation is a nondeterministic polynomial (NP) hard problem [12]. This paper proposes the DA-TD3 algorithm, which, under resource constraints, guides the coordination between agent 1 and agent 2 through setting the same reward function, to achieve the goal of minimizing system latency and local energy consumption. The main contribution of this paper can be summarized as follows:

(1) A binary offloading strategy for DA-TD3 was designed, where agent 1 generates continuous actions for multiple users, mapping continuous values to binary offloading decisions of 0 and 1. This approach effectively circumvents the difficulties associated with training distributed DQN algorithms and the problem of large action spaces in centralized DQN algorithms that hinder algorithm convergence.

(2) The partial offloading scheme of DA-TD3 was designed, in which agent 1 generates the offloading ratio for multiple users and agent 2 generates the allocation ratio of communication bandwidth and computing resources, both collaboratively optimizing the objective function to achieve satisfactory performance.

(3) The "binary and partial schemes of DA-TD3" outperform the "random offloading scheme," "all-offloading scheme," and "all-local computing scheme" in reducing system delay and local energy consumption.

2. Related Work

Both binary offloading and partial offloading are common task offloading methods [13]. Binary offloading refers to the approach of completely offloading a task to a local or edge node for processing. Although this implementation is simple and efficient, it can result in a significant network transmission overhead when multiple UEs simultaneously offload tasks to edge nodes. Partial offloading includes two implementation methods. One is to split the task into two subtasks according to the offloading rate and process them in parallel on both local and edge nodes. The other method is to divide the entire task into multiple subtasks based on the directed acyclic graph (DAG) [14] and offload them according to the binary offloading method. It can simultaneously utilize the computing resources of both local and edge nodes, greatly improving resource utilization and reducing transmission overhead compared to binary offloading, thereby improving processing efficiency.

2.1. Binary Offloading. For the scenario of the binary offloading approach [15], the authors of [16] proposed a DTORA algorithm to address the task offloading and computing resource allocation issues in MEC. The algorithm utilizes the entropy weight method as a weighting method for computing objective weights and completes the process of task offloading and resource allocation, with the goal of reducing the average response time of tasks and total system energy consumption. The authors of [17] have investigated an MEC system in which each mobile terminal can offload multiple tasks to an edge server. They have developed a joint task offloading decision-making and bandwidth allocation optimization method based on the DQN algorithm. The proposed method aims to minimize the energy, computation, and delay costs associated with the offloading process. The authors of [18] proposed a collaborative optimization scheme for the computation offloading optimization problem in vehicular networks involving edge computing and cloud computing. The optimization problem was decomposed into two subproblems. The computation offloading decision problem was handled using game theory, while resource allocation was achieved using the Lagrange multiplier method. Experimental results demonstrated that the proposed scheme improved system utilization and reduced task processing delay. The authors of [19] investigated a multi-UAV assisted MEC system and proposed the MATD3 algorithm to address the limited computing and energy capacity of UAVs. This algorithm jointly optimizes trajectory design, computation task allocation, and communication resource management to minimize execution delay and energy consumption. Numerical simulation results demonstrate that the proposed method can adapt to the mobility of users, changes in communication and computing resources, and the dynamic nature of computation tasks. The authors of [20] investigated the joint optimization problem of computation offloading and resource allocation in a dynamic multi-user MEC system, taking into account the uncertain resource demands and latency constraints of heterogeneous computing tasks. To avoid the curse of dimensionality, they proposed a method based on double deep Q-networks (DDQNs) to minimize the energy consumption of the entire MEC system. Simulation results demonstrate that the proposed method outperforms other baseline methods in various scenarios. The authors of [21] investigated the problem of task offloading in MEC networks

with dynamic weighted tasks. They proposed a dynamic task offloading algorithm for MEC networks based on deep supervised learning (DSLO) to address the weak adaptability of traditional deep learning-based offloading decision algorithms to new environments. Batch normalization was introduced to accelerate the model's convergence process and improve its robustness. Simulation results show that the proposed algorithm can quickly adapt to new MEC scenarios with only a small number of training samples.

2.2. Partial Offloading. For the scenario of the partial offloading approach [22], the authors of [23] investigated a UAV-assisted mobile edge computing (MEC) system, which utilized the DDPG algorithm to jointly optimize user scheduling, task offloading rate, UAV flight angle, and flight speed, in order to minimize processing latency. The results showed a significant improvement in processing latency achieved by the proposed system. The authors of [24] proposed a framework for a multistatic and vehicle-assisted MEC server, which utilizes a partially offloading method based on DRL to learn the optimal offloading decision according to randomly arrived loads, changing channel states, and dynamic distances between users and edge servers. The results showed that the DRL-based algorithm can autonomously learn the optimal computing offloading strategy without prior knowledge and effectively reduce system costs and delays. The authors of [25] proposed a DRL-based algorithm to address the problem of maximizing the sum computation rate (SCR) in wireless-powered edge computing networks under partial offloading strategies. Specifically, the algorithm learns near-optimal wireless power transfer (WPT) durations from previous offloading experiences and optimizes the time allocation for offloading among nodes as well as the energy allocation for each node given a fixed WPT duration. The authors of [26] proposed partial computation offloading strategies based on the Q-learning algorithm and the DDPG algorithm to address the problem of partial computation offloading in multiuser MEC systems in industrial scenarios. Compared to the binary computation offloading strategy, both proposed strategies effectively reduced system latency, with the DDPG algorithm performing the best. The authors of [27] proposed a DRL framework based on the actor-critic structure for the task of offloading decision-making and resource allocation in DAGs. The framework utilizes the policy network actor to generate binary offloading decisions for each task based on wireless channel gains and the computing frequency status of edge servers as inputs, with the aim of minimizing energy consumption and time cost. The authors of [28] proposed an ACED algorithm based on DRL for DAG-based multitask offloading strategies in MEC systems. By jointly considering the application structure and wireless interference of user transmission, ACED aims to reduce the average ETC of all users. Experimental results show that the ACED algorithm outperforms existing works in terms of reducing the average ETC of users. The authors of [29] proposed a hierarchical computing offloading strategy based on multiagent reinforcement learning (MARL) to address the computation offloading problem in multiuser multiserver energy-harvesting MEC systems. This strategy optimizes task offloading location selection and task offloading rates to minimize system delay and energy consumption. Simulation results demonstrate that the proposed strategy outperforms other baseline algorithms in terms of average task delay, energy consumption, and dropout rates.

3. System Model

In this paper, we consider an MEC network, which consists of a single edge server, small-cell base stations (SBS), and N UEs, as shown in Figure 1. The edge server and UEs are key components of the MEC network and can serve as computing nodes to collaborate in task offloading and resource allocation, improving the overall computational performance and efficiency of the network and meeting the demand of UEs for low latency and low energy consumption.

We discretize the continuous communication period T into equally spaced time slots $t \in (0, 1, 2..., T)$, where ∇t denotes the length of each time slot and each UE $i \in (1, 2..., N)$ has a computationally intensive task arriving in each time slot t. We define a task as a triple $A_i(t) = \{d_i(t), c_i(t), s_i(t)\}$, where $d_i(t)$ and $c_i(t)$, respectively, denote the data size and the required CPU cycles of the task i at time slot t and $s_i(t)$ denotes the offloading rate of the task $A_i(t)$ to the edge node. In the same time slot, the task $A_i(t)$ can be divided into two subtasks based on the offloading rate $s_i(t)$ and processed in parallel at the local or edge node, ignoring any dependencies between the subtasks.

To fully utilize communication resources, the uplink communication between UEs and MEC servers employs orthogonal frequency division multiple access (OFDMA) technology. The 10 MHz bandwidth channel is divided into N subchannels using OFDMA technology, and mutual interference between subchannels can be ignored. To simulate a more realistic offloading process and consider the mobility of UEs, we assume that the SBS coordinates are defined as (0.0) in a two-dimensional Cartesian coordinate system and that UE i is uniformly distributed around the SBS with coordinates $(x_i(t), y_i(t))$. Thus, the straight-line distance between UE i and SBS is

$$l_i(t) = \sqrt{x_i(t)^2 + y_i(t)^2}. \tag{1}$$

The communication transmission rate between UE i and the edge server in time slot t is

$$r_i(t) = a_i(t) B \log_2\left(1 + \frac{p_i g l_i(t)^{-\tau}}{\sigma^2}\right), \tag{2}$$

where $\alpha_i(t)$ represents the proportion of communication bandwidth allocated to UE i in time slot t, B is the total bandwidth between UE and SBS, p_i represents the transmission power of UE i, g is the channel gain at a reference distance of 1 meter, τ represents the path loss factor, and σ^2 represents the noise power.

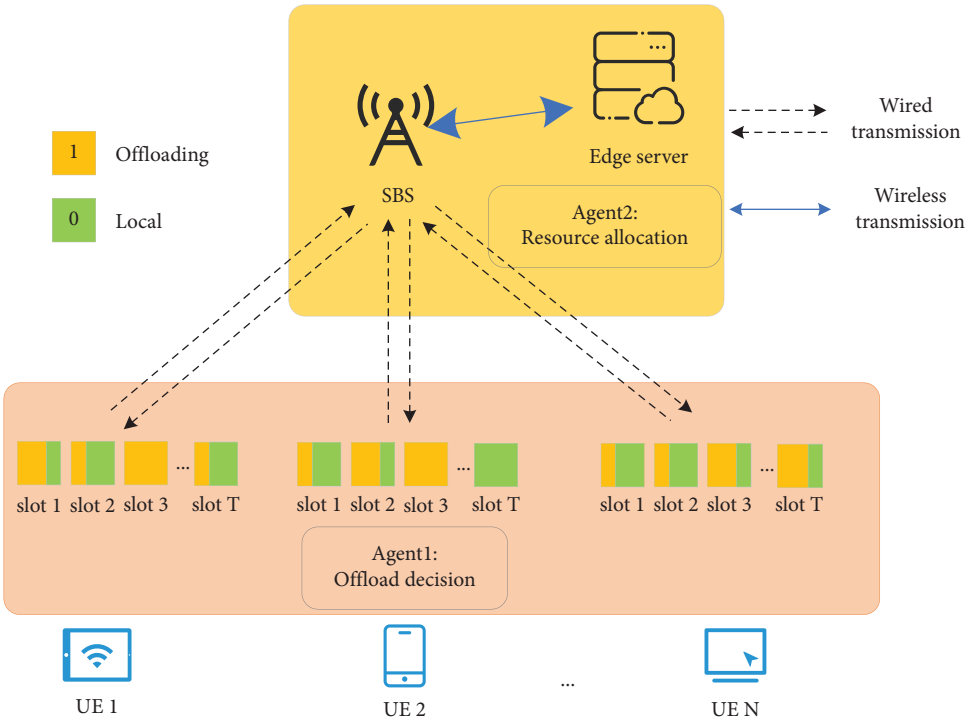


Figure 1: System network model.

3.1. Local Computing Model. The local computing model is a common computing model in MEC. In this model, when the task $A_i(t)$ is offloaded to a local device for processing either entirely or partially, local computing latency can be calculated based on the required number of CPU cycles for completing the task and the processing capability of the local device. Its local computing latency is given by

$$T_{\text{local}}(i,t) = \frac{(1 - s_i(t)) * c_i(t)}{f_i}. \tag{3}$$

In addition, the energy consumption of local computing can be calculated based on the power consumption of the UE and the task execution time, as shown in the following formula:

$$E_{\text{local}}(i,t) = \delta (f_i)^2 (1 - s_i(t)) c_i(t). \tag{4}$$

In this equation, f_i represents the CPU frequency of UE i, which varies among different devices, and δ is the energy consumption coefficient. When offloading rate $s_i(t) = 0$, it indicates that the task $A_i(t)$ is completely processed locally, and when $s_i(t) = 1$, it indicates that the task $A_i(t)$ is completely offloaded to the edge node.

3.2. Computing Offloading Model. At time slot t, the MEC server at the SBS can provide computing services to multiple UEs simultaneously. When the task $A_i(t)$ is scheduled to be partially or completely offloaded to the MEC server through the offloading rate $s_i(t)$, UE i first uploads the task data to the MEC server through the assigned communication channel N, and then, the MEC server computes and returns the result to UE. The time delay of the computation offloading process mainly includes the task uploading delay $T_{\text{up}}(i,t)$, the MEC server computing delay $T_{\text{calc}}(i,t)$, and the result transmission delay (which is ignored in this paper), expressed as follows:

$$\begin{aligned} T_{\text{up}}(i,t) &= \frac{s_i(t) d_i(t)}{r_i(t)}, \\ T_{\text{calc}}(i,t) &= \frac{s_i(t) c_i(t)}{\beta_i(t) F}. \end{aligned} \tag{5}$$

Here, F represents the CPU frequency of the edge server and $\beta_i(t)$ represents the proportion of computing resources allocated to UE i at time slot t. The total delay of the offloading process is given by

$$T_{\text{offload}}(i,t) = T_{\text{up}}(i,t) + T_{\text{calc}}(i,t). \tag{6}$$

Similarly, the energy consumption of the offloading process includes the transmission energy consumption $E_{\text{tran}}(i,t)$ of the UE and the idle energy consumption $E_{\text{wait}}(i,t)$ as

$$\begin{aligned} E_{\text{tran}}(i,t) &= T_{\text{up}}(i,t) p_i, \\ E_{\text{wait}}(i,t) &= T_{\text{calc}}(i,t) p_w, \end{aligned} \tag{7}$$

where p_w is the idle power consumption of the UE during the transmission process. The total local energy consumption of the offloading process is given by

$$E_{\text{offload}}(i,t) = E_{\text{tran}}(i,t) + E_{\text{wait}}(i,t). \tag{8}$$

3.3. Problem Formulation. We aim to jointly optimize the offloading rate $s_i(t)$ for each UE, as well as the optimal allocation of communication bandwidth proportion $\alpha_i(t)$ and MEC server computing resource proportion $\beta_i(t)$ for the partitioned offloading subtasks, with the goal of minimizing the computation maximum delay and corresponding energy consumption for all UEs. Based on previous modeling, the calculation methods for the maximum delay $T_{\text{all}}(i,t)$ and total energy consumption $E_{\text{all}}(i,t)$ for all UEs within a time slot are as follows:

$$\begin{aligned} T_{\text{all}}(i,t) &= \max\{T_{\text{local}}(i,t), T_{\text{offload}}(i,t)\}, \\ E_{\text{all}}(i,t) &= \sum_{i=1}^{N} E_{\text{local}}(i,t) + E_{\text{offload}}(i,t). \end{aligned} \tag{9}$$

The total delay and energy consumption of the system over T time slots are represented as

$$\text{Cost} = \sum_{t=0}^{T}\sum_{i=1}^{N} \rho_0 T_{\text{all}}(i,t) + \rho_1 E_{\text{all}}(i,t). \tag{10}$$

Therefore, the joint optimization problem of task offloading scheduling and resource allocation can be modeled as the following equation:

$$\begin{aligned} &O\text{: } \min(Cost) \\ &\text{s.t.} \\ &C_1\text{: } 0 \le s_i(t) \le 1, \quad \forall i,t, \\ &C_2\text{: } \sum_{i=1}^{N} \alpha_i(t) = 1,\ 0 \le \alpha_i(t) \le 1, \quad \forall i,t, \\ &C_3\text{: } \sum_{i=1}^{N} \beta_i(t) = 1,\ 0 \le \beta_i(t) \le 1, \quad \forall i,t, \\ &C_4\text{: } 0 \le \rho_0 \le 1,\ 0 \le \rho_1 \le 1. \end{aligned} \tag{11}$$

The constraint C_1 represents different offloading strategies based on the different offloading rates. Specifically, in binary offloading, the offloading rate $s_i(t)$ of each UE i consists of discrete values of 0 or 1, while partial offloading consists of continuous values between 0 and 1. Constraints C_2 and C_3, respectively, ensure that the allocated communication bandwidth resource ratio $\alpha_i(t)$ and the MEC computing resource ratio $\beta_i(t)$ of all UEs in the same time slot add up to 1. In the constraint C_4, ρ_0 and ρ_1 represent the weight coefficients of time and energy consumption. These constraints ensure the feasibility and correctness of the optimization problem.

The problem at hand has been proven to be an NP-hard problem. Previous studies on offloading decision-making mostly employed binary offloading methods. In this paper, we introduce offloading rates $s_i(t)$ to extend the offloading decision-making from the discrete domain to the continuous domain. In addition, to handle more realistic scenarios, where prior information about the task request pattern within a time slot is unknown, and the complexity of the problem grows exponentially with an increase in the number of UEs, traditional model-based approaches may not be suitable to adapt to the dynamic nature and make intelligent decisions. To address these challenges, we propose a dual-agent collaborative computation offloading strategy called DA-TD3 based on reinforcement learning, aiming to minimize the maximum computation delay of all UEs tasks and the corresponding energy consumption.

4. Reinforcement Learning Optimization Algorithm

In this section, we first introduce the relevant theory and background knowledge of the TD3 algorithm based on DRL. Second, building on the TD3 algorithm, we propose the DA-TD3 computation offloading algorithm for dual agents, construct MDP, and define the states, actions, and rewards for the dual agents. Finally, we describe the design details of the algorithm.

4.1. TD3 Algorithm Framework. DRL is a technology that combines deep learning and reinforcement learning to solve complex tasks with high-dimensional state and action spaces. TD3 is a DRL algorithm based on the actor-critic framework and an improvement over the DDPG algorithm [30, 31]. The algorithm consists of six neural networks: one actor policy network, two critic value networks, and their corresponding target actor and target critic networks, as shown in Figure 2.

4.1.1. The Experience Generation Process. The actor network is a key network in DRL, which takes the current environment state as input $s(t)$ and calculates the output action $\mu_{\phi_1}(s(t))$. In the TD3 algorithm, in order to encourage the agent to explore the action space in a more diverse way, a random noise is added to the original action, as shown in the following equation:

$$a(t) = \mu_{\phi}(s(t)) + \varepsilon, \varepsilon \sim N(0,\sigma), \tag{12}$$

where ϕ represents the parameters of the actor network. The agent interacts with the environment by taking noisy actions $a(t)$ and receiving rewards $r(t)$ and the next state $s'(t)$, forming a transition sample $(s(t), a(t), r(t), s'(t))$ that is stored in the experience replay buffer as a dataset for training the online network.

4.1.2. Training and Updating the Network. When the number of state transition samples stored in the experience replay buffer exceeds the set capacity, a batch of state transition samples $(\text{bs}, \text{ba}, \text{br}, \text{bs}')$ is randomly selected from the buffer for training the online network. Unlike the DDPG algorithm, the TD3 algorithm introduces a regularization strategy by adding truncated normal distribution noise to the output action $\mu_{\theta'}(\text{bs}')$ of the target actor network, as shown in the following equation:

$$\text{ba}' = \mu_{\phi'}(\text{bs}') + \varepsilon',\ \varepsilon' \sim \text{clip}\left(N\left(0,\sigma^2\right), -c, c\right). \tag{13}$$

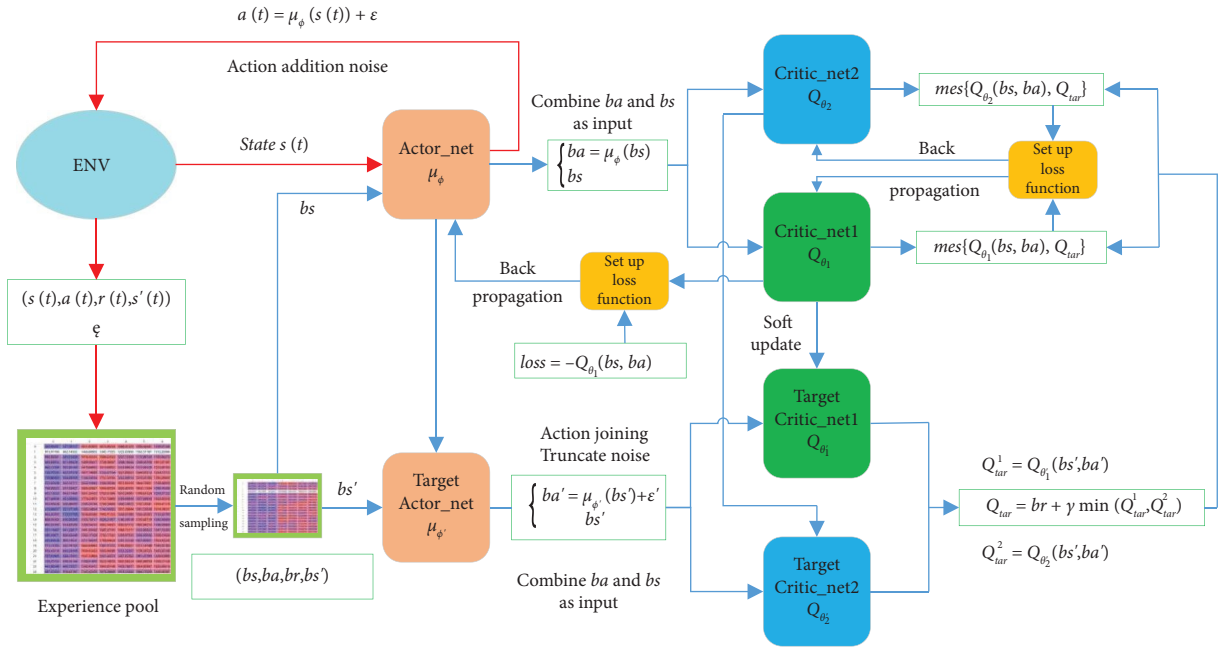


Figure 2: TD3 network architecture and algorithm flow.

In the equation, ϕ' represents the parameters of the target actor network. Adding noise can make the two target critic networks output smoother Q-values and reduce overfitting. In addition, to alleviate the overestimation of Q-values, the minimum Q-value among the outputs of the two target critic networks is selected, and the TD target is calculated based on the Bellman expectation equation of the state-action value function, which is written as

$$Q_{\text{tar}} = \text{br} + \gamma \min\left(Q_{\theta_1'}(\text{bs}', \text{ba}'), Q_{\theta_2'}(\text{bs}', \text{ba}')\right), \tag{14}$$

where br represents the batch reward, γ is the discount factor, and θ_1', θ_2' are the parameters of the two target critic networks. To estimate Q-values closer to the target Q-values, two MSE loss functions are established by computing the TD error, and then, the gradients are calculated by derivation to update the parameters of the critic networks. The update equations are as follows:

$$L(\theta_i) = \frac{1}{b} \sum \left(Q_{\theta_i}(\text{bs}, \text{ba}) - Q_{\text{tar}}\right)^2; \quad i = 1, 2, \tag{15}$$

$$\theta_i \longleftarrow \theta_i - l_c \nabla_{\theta_i} L(\theta_i); \quad i = 1, 2, \tag{16}$$

where $Q_{\theta_{1,2}}(\text{bs}, \text{ba})$ refers to the estimated Q-values output by the two critic networks with parameters θ_1 and θ_2, respectively, while l_c represents the learning rate of the value network.

In RL, the goal of an agent is to maximize cumulative rewards, and Q-values can measure the long-term cumulative rewards of taking a certain action in a current state. During training, the actor network is expected to output actions that maximize the Q-values, in order to maximize cumulative rewards. Therefore, the loss function and the parameter update equation for the actor network are as follows:

$$J(\phi) = -\frac{1}{b} \sum Q_{\theta_1}\left(\text{bs}, \mu_\phi(\text{bs})\right), \tag{17}$$

$$\phi \longleftarrow \phi - l_a \nabla_\phi J(\phi). \tag{18}$$

Finally, the parameters ϕ of the actor network and the parameters θ_1 and θ_2 of the two critic networks are updated to their corresponding target network parameters using a soft update approach, as shown in the following equation:

$$\theta_i' \longleftarrow \tau\theta_i + (1 - \tau)\theta_i', \quad i = 1, 2, \tag{19}$$

$$\phi' \longleftarrow \tau\phi + (1 - \tau)\phi', \tag{20}$$

where τ represents the soft update coefficient and ϕ' represents the parameters of target actor network.

4.2. DA-TD3 Offloading Algorithm. In MEC, the binary offloading decisions for multiple users are typically made using the DQN algorithm to output Q-values for all actions and then indirectly selecting the action with the highest Q-value as the output. However, with the increasing number of UEs, the range of action space also increases, which leads to difficulties in algorithm convergence and training. To address this challenge, this paper proposes a DA-TD3 algorithm for collaborative offloading scheduling and resource allocation based on the TD3 algorithm with a continuous action space for two agents. Specifically, agent 1 generates offloading decisions for multiple users directly, while agent 2 allocates communication and computing resources based on the offloading decisions, and both agents collaboratively optimize the objective function. The advantage of this algorithm is that it greatly alleviates the problem of large action dimensionality, thereby improving the training efficiency and convergence speed of the algorithm, as shown in Figure 3.

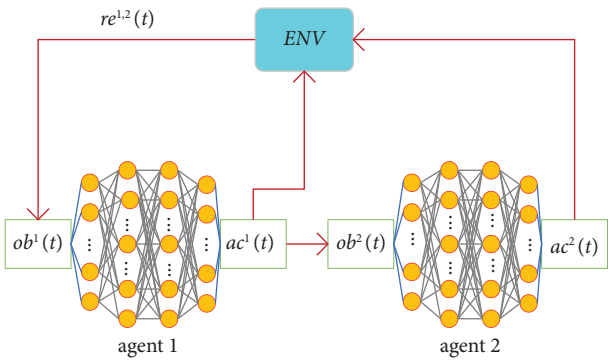


Figure 3: Cooperative optimization mechanism for dual agents.

4.2.1. State Space. It is critical to define the state that matches the characteristics of the problem. In this paper, the amount of data and the distance between the UEs and the SBS determine the transmission time, while the computing resources required determine the execution time and energy consumption of the task on the computing node. Therefore, using these factors as state inputs can help the agent better understand the relationship between different tasks and offloading scheduling and make more effective offloading decisions. Thus, the state input of agent 1 is defined as follows:

$$\mathbf{ob^1(t)} = \begin{bmatrix} d_1(t) & c_1(t) & x_1(t) & y_1(t) \\ . & . & . & . \\ d_N(t) & c_N(t) & y_N(t) & y_N(t) \end{bmatrix}. \tag{21}$$

Agent 2 focuses on the status information of tasks offloaded to edge nodes and allocates resources accordingly. When a task is scheduled to be executed locally, this part of the status information is not relevant. Specifically, when the offloading rate $s_i(t) = 0$, the state vector of UE i is all 0. The state input of agent 2 is as follows:

$$\mathbf{ob^2(t)} = \begin{bmatrix} s_1(t)d_1(t) & s_1(t)c_1(t) & x_1(t) & y_1(t) \\ . & . & . & . \\ s_N(t)d_N(t) & s_N(t)c_N(t) & x_N(t) & y_N(t) \end{bmatrix}. \tag{22}$$

4.2.2. Action Space. Agent 1 is responsible for the offloading decisions of all UEs tasks. A sigmoid activation function is added to the output layer of its policy network to limit the N-dimensional action vector between 0 and 1. When using binary offloading, the sigmoid function output is binary classified. The output value greater than or equal to 0.5 is mapped to an offloading rate of 1, and the output value less than 0.5 is mapped to an offloading rate of 0. When using partial offloading, the output action directly represents the offloading rate. Thus, at time slot t, the action vector of agent 1 is

$$\mathbf{ac^1(t)} = \left[s_1(t), s_2(t), \ldots, s_{(N-1)}(t), s_N(t)\right]. \tag{23}$$

Agent 2 adopts a softmax activation function for the policy network output, ensuring that the ratio of communication and computation resource allocation to all UEs in the same time slot is equal to 1. The action vector is as follows:

$$\mathbf{ac^2(t)} = \left[\alpha_1(t), \alpha_1(t), \ldots, \alpha_{N-1}(t), \alpha_N(t), \beta_1(t), \beta_1(t), \ldots, \beta_{N-1}(t), \beta_N(t)\right]. \tag{24}$$

The dimension of the action vector is $2N$, with the first N values representing the communication bandwidth resource allocation ratio and the last N values representing the computation resource allocation ratio.

4.2.3. Reward. The reward design has a significant impact on the decisions of the agents and the optimal solution to the task. Using the negative of the optimization objective as the reward function can easily be converted into the minimization of a cost function. Since agents 1 and 2 need to cooperatively optimize the offloading policy to minimize the cost function, the same reward function is set for agents 1 and 2 at time slot t, as follows:

$$\mathrm{re}^{1,2}(t) = -\left(\sum_{i=1}^{N} \rho_0 T_{\mathrm{all}}(i,t) + \rho_1 E_{\mathrm{all}}(i,t)\right). \tag{25}$$

The detailed process of DA-TD3 is shown in Algorithm 1.

5. Simulation Experiment

In this section, we first introduce the experimental simulation environment and parameter settings. Then, we design multiple different offloading schemes. Finally, we compare and analyze the various offloading schemes in different experimental environments. Through experimental validation, we demonstrate the effectiveness of the binary and partial offloading schemes based on the DA-TD3 algorithm.

5.1. Experimental Settings. We utilized the Python 3.8 development environment and the PyTorch deep learning framework as the simulation software platform for our experiments, with the hardware platform consisting of an Intel Xeon Gold 6148 processor and a 3090 graphics card. The experiment parameters and hyperparameter settings for the DA-TD3 algorithm were established based on experimental requirements, as illustrated in Tables 1 and 2.

Input: UE task information $\text{ob}^1(t)$.
Output: Offloading decision vector $\text{ac}^1(t)$ and resource allocation vector $\text{ac}^2(t)$.
Initialize the network parameters for agents 1 and 2.
Set the capacity of experience buffer and specify the batch size for training.
for *episode* = 1, 2, . . ., *Max_Episode* **do**
  Resetting the environment to obtain the initial state $\text{ob}^1(t)$.
  for $t = 1, 2, \ldots, T$ **do**
    Input state $\text{ob}^1(t)$ to the actor network μ_ϕ^1 of agent 1 to obtain the action $\text{ac}^1(t)$.
    Calculate the state input $\text{ob}^2(t)$ for the actor μ_ϕ^2 of agent 2 based on the $\text{ac}^1(t)$.
    Input state $\text{ob}^2(t)$ to the actor μ_ϕ^2 of agent 2 to obtain the action $\text{ac}^2(t)$.
    Calculate the reward $\text{re}^{1,2}(t)$ by jointly considering the $\text{ac}^1(t)$ and $\text{ac}^2(t)$.
    Store the $(\text{ob}^1(t), \text{ac}^1(t), \text{re}^{1,2}(t), \text{ob}^1(t+1))$ in the replay buffer of agent 1.
    Store the $(\text{ob}^2(t), \text{ac}^2(t), \text{re}^{1,2}(t), \text{ob}^2(t+1))$ in the replay buffer of agent 2.
    if batch size < the current capacity of buffer **Then**
      for agent $i = 1, 2$ **do**
        Sample a batch of experiences randomly.
        Calculate the loss of critic net $Q^i_{\theta_{1,2}}$ according to equation (15).
        Update parameters θ_1, θ_2 of the critic net according to equation (16).
        Calculate the loss of actor net μ^i_ϕ according to the equation (17).
        Update parameter ϕ of the critic according to equation (18).
        Update parameters of the target nets according to (19) and (20).

Algorithm 1: DA-TD3.

Table 1: Experiment parameter settings.

Notation	Definition	Value
F	CPU frequency of the MEC server	8 Ghz
f_i	CPU frequency of UE	Unif (2, 2.5) Ghz
B	Bandwidth of the channel	10 Mhz
d_i	Task data size	Unif (4, 8) Mbits
c_i	CPU cycles for task calculation	Unif (0.8, 1.6) Gigacycles
l_i	The distance between UEs and SBS	Unif (100, 300) m
p_i	The transmission power of UE	Unif (400, 500) mw
p_w	The standby power of UE	Unif (80, 100) mw
τ	The path loss factor	2
σ^2	The noise power	−100 dBm
δ	Energy consumption coefficient	10^{-28}
T	Total time slots	30
ρ_0	Weight coefficient of time	0.7
ρ_1	Weight coefficient of energy consumption	0.3

Table 2: Hyperparameter settings for DA-TD3.

Definition	Hyperparameter
Max_episode	1000
The learning rate of actor l_a	0.00001
The learning rate of critic l_c	0.0001
Memory size	10000
Batch size	256
Discount factor γ	0.99
Soft update coefficient τ	0.05
Action noise ε	$N(0, 0.1)$

In the DA-TD3 algorithm, two agents are employed to solve the task offloading decision and resource allocation, respectively. Each agent consists of 2 actor networks and 4 critic networks. The structural parameters of these actor and critic networks are shown in Table 3.

Figure 4 illustrates the changes in the reward function throughout training episodes with varying learning rates in the DA-TD3 algorithm. Typically, the critic network's learning rate is set higher than that of the actor network to enhance sensitivity to value function approximation. We conducted four sets of experiments to assess the impact of learning rates on convergence performance. The results indicate that the optimal convergence effect occurs when the actor learning rate is 0.00001 and the critic learning rate is 0.0001. Learning rates that are excessively large or small can compromise stability. Extremely high rates induce dynamic instability during training, while excessively low rates prolong convergence time.

Figure 5 shows the reward function changes over training episodes under different memory sizes and batch sizes of the DA-TD3 algorithm. We set up 4 groups of experiments with positively correlated values for the two

Table 3: Network structure for DA-TD3.

Networks	Number	Network structure
Actor of agent 1	2	fc1 (state_dim, 256), relu fc2 (256, 128), relu fc3 (128, action_dim), sigmoid
Critic of agent 1	4	fc1 (state_action_dim, 256), relu fc2 (256, 128), relu fc3 (128, 1)
Actor of agent 2	2	fc1 (state_dim, 256), relu fc2 (256, 128), relu fc3 (128, action_dim), softmax
Critic of agent 2	4	fc1 (state_action_dim, 256), relu fc2 (256, 128), relu fc3 (128, 1)

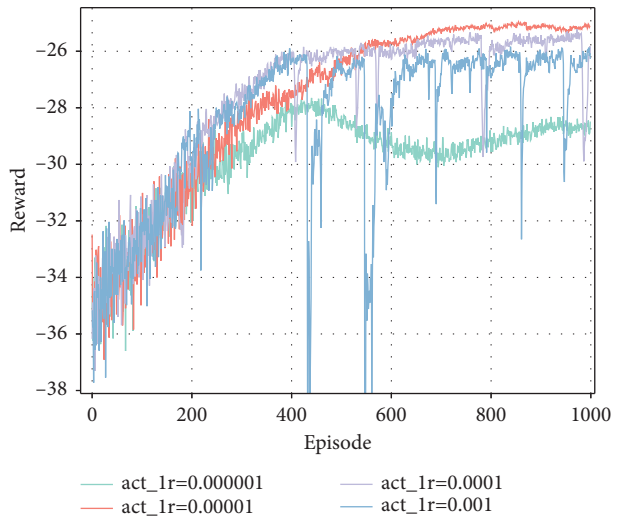


Figure 4: Reward under different learning rate.

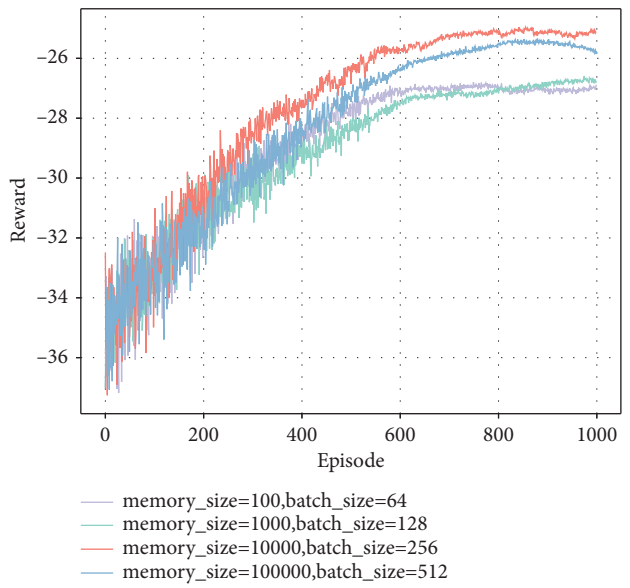


Figure 5: Reward under different memory_size and batch_size.

hyperparameters. This is because the memory buffer needs sufficient samples for batch training. Ultimately, we determined the combination of memory size = 10,000 and batch size = 256, which demonstrated the best convergence performance in experiments. In practical tuning, we noticed that excessively small memory and batch sizes can cause the algorithm to get stuck in local optima, while excessively large values may lead to unstable training.

To assess the efficacy of deep neural network models in managing high-dimensional state spaces, we varied the number of fully connected layers in the algorithm, specifically setting them to 3, 4, and 5 layers. The subsequent comparison included their influence on training time, convergence performance, and other factors, as illustrated in Figure 6. The findings reveal that an increase in layers correlates with accelerated convergence, yet the ultimate reward improvement upon convergence remains marginal, as outlined in Table 4. Acknowledging the elevated computational cost associated with additional network layers, we opted for the 3-layer network architecture after a comprehensive consideration of various factors.

A heterogeneous objective function was constructed to achieve flexible multiobjective optimization, which introduces the adjustable time weight ρ_0 and energy weight ρ_1. By tuning these two weights, the relative importance of latency and energy consumption in the objective function can be dynamically balanced to optimize different performance metrics. Here, ρ_0 is set to 0.7 and ρ_1 is set to 0.3, with the larger ρ_0 reflecting the priority of reducing latency and ρ_1 considering energy consumption when balancing the two metrics. This setting can be adjusted based on specific scenarios and requirements to meet different optimization goals.

5.2. Comparison Scheme. To verify the superiority of both the "DA-TD3 binary offloading scheme" and "DA-TD3 partial offloading scheme," we designed 6 comparison schemes as follows:

(i) All_Local: all tasks are offloaded to local computing.

(ii) All_Edge: All tasks are offloaded to the edge nodes, and the bandwidth and computing resources are evenly distributed.

(iii) Random_Offload: tasks are randomly offloaded and scheduled with evenly distributed bandwidth and computing resources.

(iv) DDPG_All: The network architecture of the DA-TD3 algorithm is replaced with the DDPG algorithm, and the binary offloading scheme with dual agent outputs for offloading decisions and resource allocation ratios is adopted.

(v) DDPG_Part: The network architecture of the DA-TD3 algorithm is replaced with the DDPG algorithm, and the partial offloading scheme with dual agent outputs for offloading decisions and resource allocation ratios is adopted.

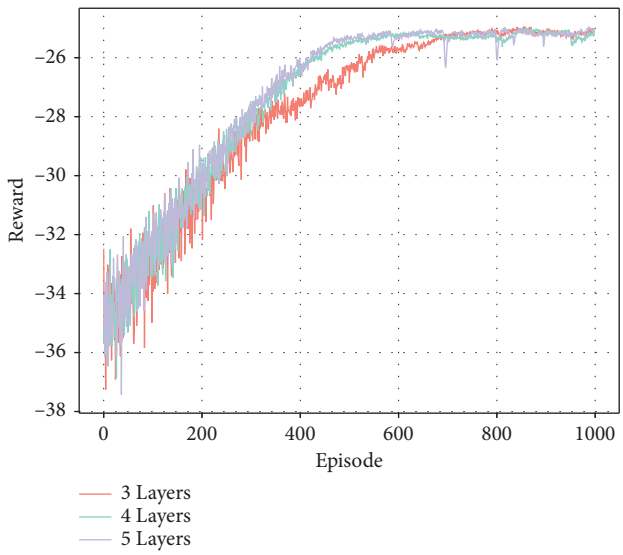


Figure 6: Reward under different net structure.

Table 4: Performance of DA-TD3 with different number of layers.

Number of layers	Training duration (s)	Average reward	Maximum reward
3	362	−24.96	−27.71
4	392	−24.99	−27.36
5	435	−24.94	−27.27

(vi) DQN_All: The DQN algorithm is used to generate binary offloading decisions for multiple users, with evenly distributed bandwidth and computing resources. Since the DQN algorithm can only generate discrete actions, this paper did not consider the partial offloading scheme of DQN.

(vii) HDMAPPO: Following the framework design idea of the HDMAPPO algorithm in [29], this paper proposes a dual-agent framework HDMAPPO algorithm based on PPO (proximal policy optimization) for joint optimization of offloading decision and resource allocation.

5.3. Experimental Results and Analysis. Figure 7 compares the changes in rewards and training episodes for five offloading schemes. With the same batch of random seeds, 30 time slots and five UEs are involved. As shown in the figure, the offloading scheme based on the actor-critic framework gradually smooths its curve as the training progresses. This is because the policy network's output action is added with a normal distribution noise that decreases over time, enabling more exploration in the initial phase and more exploitation in the later phase. The "DQN binary offloading scheme" shows poor convergence and high noise, indicating that the DQN algorithm is not suitable for high-dimensional action spaces. The HDMAPPO algorithm exhibits large convergence oscillations during training. This is because HDMAPPO is based on the PPO algorithm, which is prone to high variance itself, resulting in unstable training. In addition, the convergence curves of "DA-TD3 algorithm

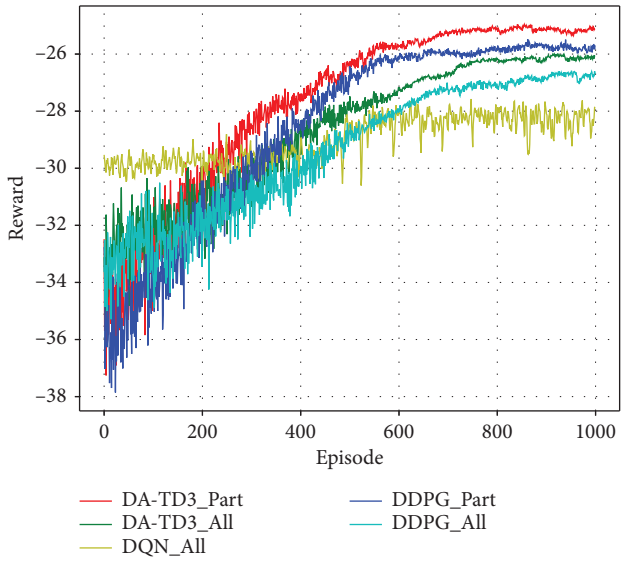


Figure 7: Convergence performance comparison of different schemes.

binary and partial offloading schemes" are better than those of "DDPG binary and partial offloading schemes," since TD3's action search is superior to DDPG. TD3 employs double Q learning to reduce the bias of action search and improve the algorithm's efficiency. Furthermore, partial offloading outperforms binary offloading, as it provides more flexibility in scheduling tasks and makes full use of limited communication bandwidth and computing

resources, enabling efficient parallel computing of offloading tasks at local and edge nodes.

Figure 8 shows the system cost changes within one episode under different offloading schemes. Compared with other schemes, the partial and binary offloading schemes of DA-TD3 achieved lower system costs at each time slot. Specifically, the average system costs of the 30 time slots under the DA-TD3 partial and binary offloading schemes are 34.43% and 31.48% lower than those of the local-only scheme, 23.35% and 19.9% lower than those of the full offloading scheme, and 18.01% and 14.32% lower than those of the random offloading scheme, respectively. In addition, the performance of partial offloading outperforms binary offloading at each time slot, with 4.3% lower average system cost. By dynamically adjusting offloading decisions and optimizing resource allocation according to the time-varying environment and task characteristics, DA-TD3 achieves adaptive minimization of system latency and energy consumption. Its online learning capability enables efficient adaptation to dynamic MEC environment changes, thereby significantly reducing the cost at each control time slot.

Figures 9 and 10, respectively, present the performance of the "DA-TD3 binary offloading scheme" and the "DA-TD3 partial offloading scheme" in terms of time delay under the conditions of the time weight coefficient $\rho_0 = 1$ and the energy weight coefficient $\rho_1 = 0$. In the binary offloading scheme, three UEs are offloaded to the edge nodes for computation, resulting in a maximum time delay of 0.79 seconds. In contrast, in the partial offloading scheme, local and edge nodes are parallelly computed based on the offloading rate $s_i(t)$, resulting in a reduced maximum time delay to 0.52 seconds. In the binary offloading scheme, when the offloading rate $s_i(t) = 1$, local computing resources will be idle, causing resource waste. Therefore, the partial offloading scheme is more adaptable to the network environment and task scenarios, with higher resource utilization, and can significantly shorten the overall computation time delay.

Figure 11 compares the successful offloading quantities at each time slot between the DA-TD3 scheme, where all tasks are offloaded, and the all-edge scheme with equally distributed resources. This is done to validate the load balancing effectiveness of DA-TD3 under the settings of latency weight = 1 and energy weight = 0. The results show that the amount of successful offloading of DA-TD3 is higher than that of all-edge at each time slot. Overall, the offloading success rate of DA-TD3 (60.67%) is also superior to that of all-edge (54.67%). Here, the max delay threshold for successful offloading is set as 1.2s. This demonstrates that DA-TD3 can optimize offloading decisions and resource scheduling according to latency requirements, utilize computing resources more fully, and thus achieve load balancing to some extent.

Figure 12 illustrates the changes in system cost under different schemes and varying numbers of UEs. The time weight coefficient ρ_0 is set to 0.7, and the energy weight coefficient ρ_1 is set to 0.3, with UE numbers of 5, 10, 15, 20, and 25. It can be observed from the figure that the system cost of all schemes increases as the number of UEs increases, with the "DA-TD3 partial offloading scheme" achieving the lowest system cost. Although the "all-offloading scheme" can significantly reduce system energy consumption, in limited resource conditions, an increasing number of UEs result in less communication bandwidth and edge server computing resources allocated to each user in the same time slot, leading to an increase in transmission and computing latency. On the other hand, the "all-local computing scheme" significantly increases system energy consumption. The "DA-TD3 binary and partial offloading schemes" can simultaneously consider time delay and local energy consumption according to the reward function, fully utilizing resources to achieve the goal of reducing system cost. Among them, the offloading decision of the "DA-TD3 partial offloading scheme" is more flexible, leading to lower system cost and demonstrating performance advantages.

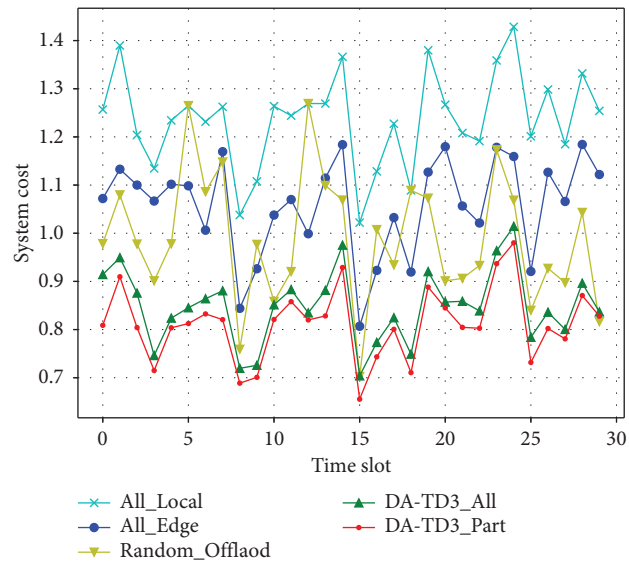
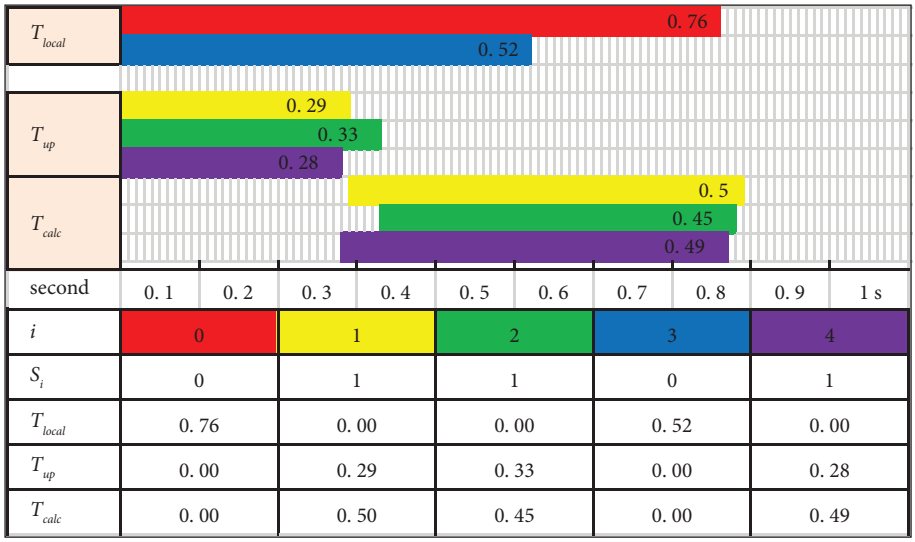


Figure 8: System cost for a single time slot.

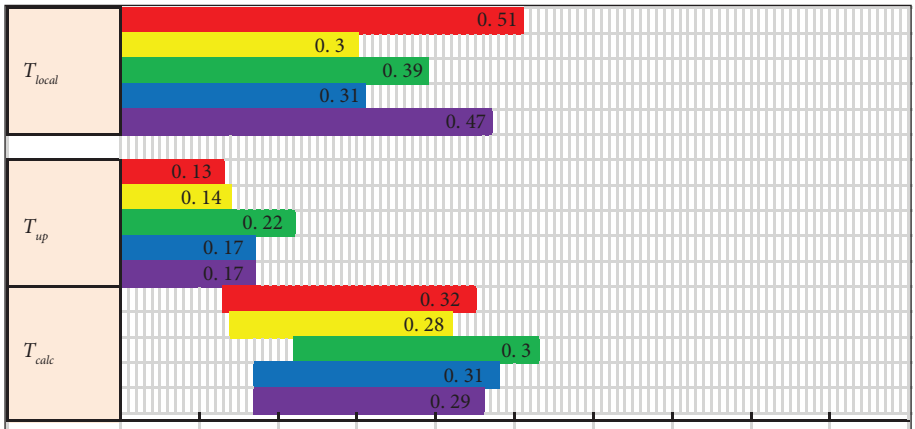
Figure 13 compares the changes in system latency and the number of UEs, with a time weight coefficient $\rho_0 = 1$ and an energy consumption weight coefficient $\rho_1 = 0$. As the number of UEs increases, the system latency of different offloading schemes also increases. From the figure, it can be seen that the performance of both the binary offloading and partial offloading schemes gradually decreases when the number of UEs exceeds 20. This is due to limited communication bandwidth and computing resources, resulting in limited optimization effect of the algorithm. Agent 1 tends to schedule tasks locally to reduce system latency and approach the "all-local computation scheme."

Figure 14 presents the variations of system energy consumption with respect to the number of UEs, where the weight coefficient for energy consumption ρ_1 is set to 1 and the weight coefficient for time ρ_0 is set to 0. As the energy consumption is calculated by accumulating the energy consumption of all UEs in the same time slot, an increase in the number of UEs will inevitably lead to an overall increase



i	0	1	2	3	4
S_i	0	1	1	0	1
T_{local}	0. 76	0. 00	0. 00	0. 52	0. 00
T_{up}	0. 00	0. 29	0. 33	0. 00	0. 28
T_{calc}	0. 00	0. 50	0. 45	0. 00	0. 49

Figure 9: Binary offloading scheme latency of DA-TD3.



i	0	1	2	3	4
S_i	0. 33	0. 45	0. 38	0. 40	0. 36
T_{local}	0. 51	0. 30	0. 39	0. 31	0. 47
T_{up}	0. 13	0. 14	0. 22	0. 17	0. 17
T_{calc}	0. 32	0. 28	0. 30	0. 31	0. 29

Figure 10: Partial offloading scheme latency of DA-TD3.

in the system energy consumption. The "DA-TD3 global and partial offloading schemes" outperform other schemes in the figure, because the offloading decision of agent 1 is solely guided by the system energy consumption as a reward. Agent 1 tends to offload tasks to the edge node as a whole, resulting in similar optimization performance between the "DA-TD3 global offloading scheme" and the "DA-TD3 partial offloading scheme."

To investigate the relationship between CPU frequency of the MEC server and system cost, Figure 15 compares the system cost of different offloading schemes under different CPU frequencies of the MEC server. The weight coefficients for energy consumption ρ_1 and time ρ_0 are set to 0.3 and 0.7, respectively. Simulation results show that, except for the "all-local computing scheme," the system cost of other schemes decreases with the increase of the server computation frequency. Under the condition of fixed server computation frequency, the "DA-TD3 joint optimization scheme for binary and partial offloading" outperforms the other three schemes. As the CPU frequency of the MEC server increases, agent 1 tends to offload tasks to the edge node. Therefore, from the figure, we can observe that the performance gap between the joint optimization schemes for binary and partial offloading gradually decreases when the server computation frequency is greater than 10 GHz.

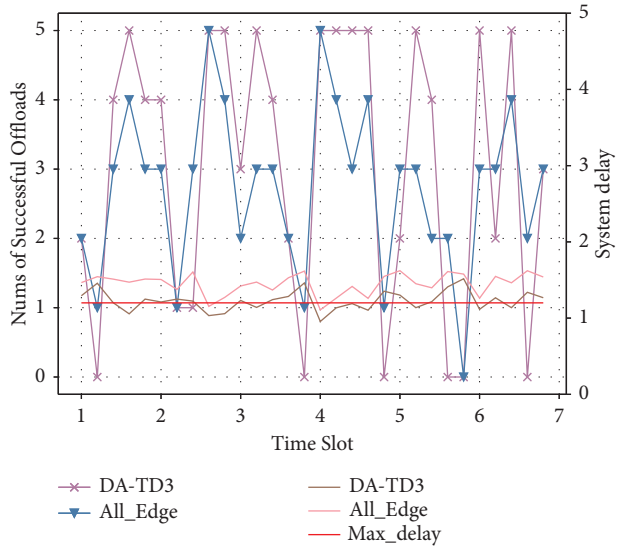


Figure 11: Number of offloading successes per time slot.

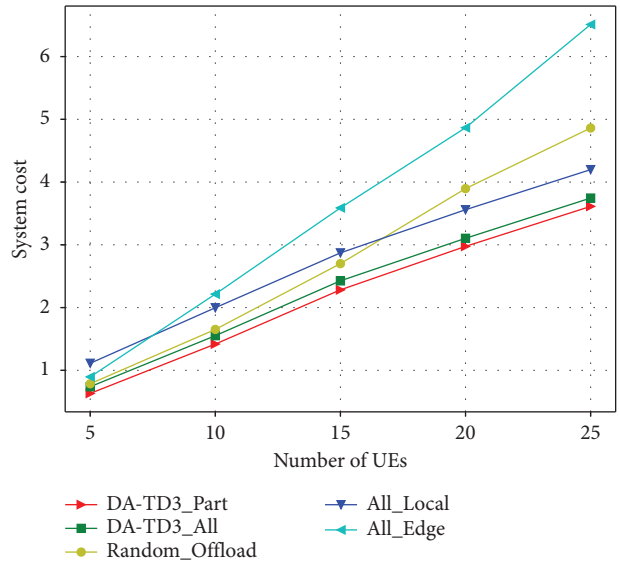


Figure 12: Total system cost for different numbers of UEs.

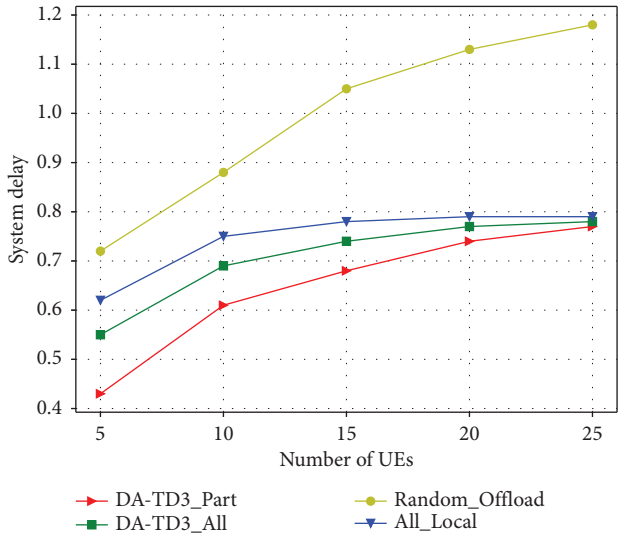


Figure 13: System latency for different numbers of UEs.

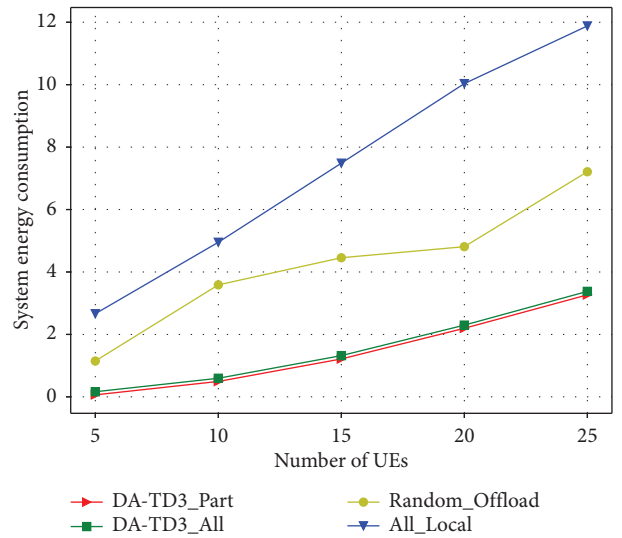


Figure 14: System energy consumption for different numbers of UEs.

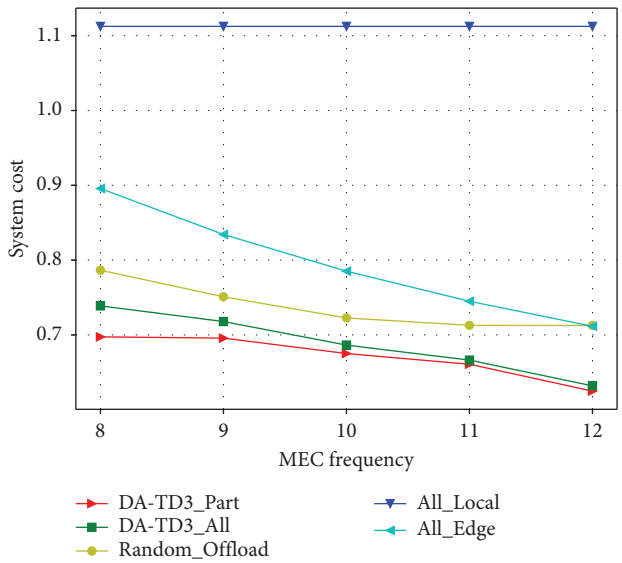


Figure 15: Total system cost for different server computing frequencies.

6. Conclusions

This paper studied an MEC system with multiple terminal users concurrently executing multiple tasks and compared two task offloading strategies: binary offloading and partial offloading. To realize the collaborative optimization of offloading decision and resource allocation, a dual-agent DA-TD3 reinforcement learning algorithm was designed. DA-TD3 adopts a continuous action space representation, which is easier for learning and convergence compared to the discrete action space in DQN. The experimental results show that both the binary and partial offloading schemes based on DA-TD3 can significantly reduce system latency and local computation energy consumption, with partial offloading being more effective. Looking forward, future research could consider expanding to large-scale MEC systems with multiple servers and users, exploring collaboration and competition mechanisms between different user tasks and servers, and designing efficient joint offloading decision and resource allocation optimization algorithms for such complex environments. For the algorithm framework, deep neural networks such as CNN and RNN could be attempted to extract sophisticated features from the high-dimensional state space, replacing the simply fully connected network with limited representation capability, so as to handle large-scale system state spaces and make better decisions.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was funded by the National Natural Science Foundation of China (Grant no. 62205005) and 2021 China Universities Industry-University-Research Innovation Fund—New Generation Information Technology Innovation Project (Grant no. 2021ITA01022).

References

[1] C. Zhang and Y. Lu, "Study on artificial intelligence: the state of the art and future prospects," *Journal of Industrial Information Integration*, vol. 23, no. 1, Article ID 100224, 2021.

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

[3] P. Mach and Z. Becvar, "Mobile edge computing: a survey on architecture and computation offloading," *IEEE Communications Surveys and Tutorials*, vol. 19, pp. 1628–1656, 2017.

[4] E. Mustafa, J. Shuja, S. K. uz Zaman et al., "Joint wireless power transfer and task offloading in mobile edge computing: a survey," *Cluster Computing*, vol. 25, no. 1, pp. 2429–2448, 2022.

[5] Y. Qian, J. Wu, R. Wang, F. Zhu, and W. Zhang, "Survey on reinforcement learning applications in communication networks," *Journal of Communications and Information Networks*, vol. 4, no. 2, pp. 30–39, 2019.

[6] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: a deep reinforcement learning approach," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1529–1541, 2021.

[7] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *Proceedings of the 2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, Barcelona, Spain, April 2018.

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

[9] L. Huang, X. Feng, A. Feng, Y. Huang, and L. P. Qian, "Distributed deep learning-based offloading for mobile edge computing networks," *Mobile Networks and Applications*, vol. 23, no. 1, pp. 1–8, 2018.

[10] H. Xu, W. Huang, Y. Zhou, D. Yang, M. Li, and Z. Han, "Edge computing resource allocation for unmanned aerial vehicle assisted mobile network with blockchain applications," *IEEE Transactions on Wireless Communications*, vol. 20, no. 5, pp. 3107–3121, 2021.

[11] Y. Wang, J. Liu, Y. Yin, Y. Tong, and J. Liu, "Space information network resource scheduling for cloud computing: a deep reinforcement learning approach," *Wireless Communications and Mobile Computing*, vol. 2022, Article ID 1927937, 16 pages, 2022.

[12] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2015.

[13] R. Chen and X. Wang, "Maximization of value of service for mobile collaborative computing through situation-aware task offloading," *IEEE Transactions on Mobile Computing*, vol. 22, no. 2, pp. 1049–1065, 2023.

[14] Y. Y. Cui, D. G. Zhang, T. Zhang, J. Zhang, and M. Piao, "A novel offloading scheduling method for mobile application in mobile edge computing," *Wireless Networks*, vol. 28, no. 6, pp. 2345–2363, 2022.

[15] K. Sadatdiynov, L. Cui, L. Zhang, J. Z. Huang, S. Salloum, and M. S. Mahmud, "A review of optimization methods for computation offloading in edge computing networks," *Digital Communications and Networks*, vol. 8, no. 1, pp. 1–13, 2022.

[16] Z. Tong, X. Deng, F. Ye, S. Basodi, X. Xiao, and Y. Pan, "Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment," *Information Sciences*, vol. 537, pp. 116–131, 2020.

[17] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digital Communications and Networks*, vol. 5, no. 1, pp. 10–17, 2019.

[18] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7944–7956, 2019.

[19] N. Zhao, Z. Ye, Y. Pei, Y. C. Liang, and D. Niyato, "Multi-agent deep reinforcement learning for task offloading in

UAV-assisted mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 21, no. 9, pp. 6949–6960, 2022.

[20] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 1517–1530, 2021.

[21] S. Yang, G. Lee, and L. Huang, "Deep learning-based dynamic computation task offloading for mobile edge computing networks," *Sensors*, vol. 22, no. 11, p. 4088, 2022.

[22] T. P. Truong, T. V. Nguyen, W. Noh, and S. Cho, "Partial computation offloading in NOMA-assisted mobile-edge computing systems using deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 8, no. 17, pp. 13196–13208, 2021.

[23] Y. Wang, W. Fang, Y. Ding, and N. Xiong, "Computation offloading optimization for UAV-assisted mobile edge computing: a deep deterministic policy gradient approach," *Wireless Networks*, vol. 27, no. 4, pp. 2991–3006, 2021.

[24] J. Wang, H. Ke, X. Liu, and H. Wang, "Optimization for computational offloading in multi-access edge computing: a deep reinforcement learning scheme," *Computer Networks*, vol. 204, Article ID 108690, 2022.

[25] S. Zhang, H. Gu, K. Chi, L. Huang, K. Yu, and S. Mumtaz, "DRL-based partial offloading for maximizing sum computation rate of wireless powered mobile edge computing network," *IEEE Transactions on Wireless Communications*, vol. 21, no. 12, pp. 10934–10948, 2022.

[26] X. Deng, J. Yin, P. Guan, N. N. Xiong, L. Zhang, and S. Mumtaz, "Intelligent delay-aware partial computing task offloading for multiuser industrial internet of things through edge computing," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 2954–2966, 2023.

[27] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task Graph in mobile edge computing: a deep reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 19, no. 8, pp. 5404–5419, 2020.

[28] J. Chen, Y. Yang, C. Wang, H. Zhang, C. Qiu, and X. Wang, "Multitask offloading strategy optimization based on directed acyclic graphs for edge computing," *IEEE Internet of Things Journal*, vol. 9, no. 12, pp. 9367–9378, 2021.

[29] Y. Sun and Q. He, "Computational offloading for MEC networks with energy harvesting: a hierarchical multi-agent reinforcement learning approach," *Electronics*, vol. 12, no. 6, p. 1304, 2023.

[30] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *Proceedings of Machine Learning Research*, vol. 80, pp. 1587–1596, 2018.

[31] T. P. Lillicrap, J. J. Hunt, A. Pritzel et al., "Continuous control with deep reinforcement learning," *Computer Science*, vol. 8, no. 6, p. A187, 2015.