Improved DQN-Based Computation Offloading Algorithm in MEC Environment

Zhao, Zheyu; Cheng, Hao; Xu, Xiaohua

DOI
10.1109/ICPADS56603.2022.00012

Publication date
2023

Document Version
Final published version

Published in
Proceedings - 2022 IEEE 28th International Conference on Parallel and Distributed Systems, ICPADS 2022

2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS) | 978-1-6654-7315-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICPADS56603.2022.00012

Improved DQN-Based Computation Offloading Algorithm in MEC Environment

Zheyu Zhao*, Hao Cheng†, Xiaohua Xu*
*School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
†Department of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, Netherland

***Abstract*—Massive terminal users have brought explosive need of data residing at edge of overall network. Multiple Mobile Edge Computing (MEC) servers are built in/near base station to meet this need. However, optimal distribution of these servers to multiple users in real time is still a problem. Reinforcement Learning (RL) as a framework to solve interaction problem is a promising solution. In order to apply RL based algorithm into a multi-agent environment, we propose an iterative scheme: select individual users with priorities to interact with the environment iteratively one at a time. Furthermore, we tried to optimize the overall system performance based on this scheme. Hence, we construct three objective system performance indicators: average processing cost, delay and energy consumption, improve the existing Deep Q-learning Network (DQN) by using the cost as reward function, changing the fixed exploitation rate into dynamic one that associated with reward and episode time. In order to explore the performance potential of the proposed algorithm, we have simulated the proposed algorithm, DQN algorithm and greedy algorithm under different users and data sizes. The results show that the proposed algorithm had reduced at least 12% of system average processing cost comparing to the greedy algorithm. It also outperform the greedy algorithm and DQN algorithm in delay and energy consumption significantly.**

***Index Terms*—Mobile Edge Computing, Computation Offloading, Reinforcement Learning, Deep Q-Learning Network**

I. Introduction

In recent years, with the development of mobile communication technology, a large amount of physical environment data are generated from the devices such as behavior records, audio, video devices etc. at the edge of the network [1]. MEC is an emerging paradigm that pushes computing resources from the network to network edge, which means massive computing and storage resources are placed close to mobile devices or sensors that is the edge of overall network [2]. In this way, mobile device users can exchange their data with nearest MEC servers rather than cloud server far away. Therefore, MEC has the advantage of low delay, low energy consumption, and high bandwidth [3].

In this paper, with the goal of minimizing average processing cost of the system, we try to solve the problem of computation offloading in the MEC environment. Computation offloading means that the user will offload the computing task to the MEC server for processing, and the MEC server will calculate the task and return the calculation result to the user. When the tasks arrive, we need to decide whether the tasks should be processed locally or offloaded to MEC servers. If offloaded, which MEC server should be offloaded to. Traditional optimization algorithms use fixed mathematical models, and it is difficult to adapt to this rapid changing edge computing network environment. Hence, researchers want to find solutions in adaptive algorithm.

Reinforcement learning algorithm as a algorithm framework designed specifically to solve interaction problem with changing environment would be a better choice [4]. The DQN algorithm is a representative reinforcement learning algorithm for solving discrete problems, which was proposed by DeepMind in 2013 to solve the Atari game problem, and achieved great success [5]. As for mobile edge computing scenarios, some researchers apply it to optimize the resource distribution [6]. In this paper, an Improved-DQN algorithm is proposed to solve the problem of computation offloading in the distributed networks.

Currently, the computation offloading problem in the MEC scenario faces many challenges. System resources such as channel capacity and computing capacity of the MEC servers are always changing. Secondly, as users and MEC serves both increase, the offloading decision set becomes very huge, which brings the problem of dimension explosion to the RL algorithm. To address these challenges, this paper proposes a method that applies an Improved-DQN algorithm to build up adaptive task distribution choice between individual user to multiple MEC servers at one time, then iterate this task distribution choice over all potential users with priorities. Furthermore, we want to reduce the overall system processing cost based on existing DQN algorithm and design the reward function considering system average processing delay and energy comprehensively, propose a dynamic exploitation based scheme, which is doing less exploitation in early stage and more in later stage according to episodes. The main contributions of this paper are as follows:

1) **Improved DQN algorithm:** This paper introduces the idea of dynamic exploitation factor ε into DQN algorithm, associate the ε factor with the number of iterations and the current reward. As Equation 8 shows, the exploitation rate will change according to environment feedback and its iteration state. According to our observation(refer to Figure 5), the value of ε is large at the beginning of algorithm convergence, it slowly decrease to a stable lower value as time goes by. This means the proposed algorithm would like to explore more in the early stage but exploit more in the later

978-1-6654-7315-6/23/$31.00 ©2023 IEEE
DOI 10.1109/ICPADS56603.2022.00012

stage.

2) **Iterative task distribution by RL algorithm:** In the simulation stage, we establish a multi-user multi-MEC server system scenario. In order to solve the problem of computing offloading for multiple users, we first sort users according to their priorities, and process the computing offloading decisions of each user in turn by way of round robin. In this way, the proposed algorithm is still an individual agent that can interact with the multi-MEC servers as an environment. This process will be taken iteratively for all users with its priority.
3) **System performance indicators and corresponding simulation:** In order to evaluate our proposed algorithm's performance on this multi-user multi-MEC servers scenario, we construct three system performance indicators as averaging processing cost, average processing delay and average processing energy. The initial parameter of system are set randomly to approximate to the real situation. The proposed Improved DQN algorithm are compared with greedy algorithm and DQN algorithm under different users and data size. The final simulation results demonstrate the the proposed algorithm's superiority.

The rest of the paper is organized as follows: Section II summarizes the related work of previous researchers. Section III presents the system model and performance indicators of multi-user and multi-MEC server. Section IV introduces the flow of DQN algorithm and the innovation of this paper. Section V illustrates the details of our proposed Improved-DQN algorithm. Section VI simulates proposed algorithm for the system performance indicator and compare with baseline algorithm. Section VII summarizes our work and draw the conclusion.

II. Related Work

In this section, we study previous researches related to computation offloading and analyze the limitations of these work.

Traditional optimal algorithms for computation offloading problems: Zhao et al. proposed a joint optimization scheme of collaborative computation offloading and resource allocation, which decomposed the optimization problem into two sub-problems of computation offloading decision and resource allocation [7]. Dong et al. formulated the computation offloading problem as a graph cut problem and proposed a solution based on spectral clustering computation [8]. In [9], the authors proposed an edge-cloud collaborative computation offloading model, and used the inertia weight particle swarm algorithm to solve the problem, which was made up for the premature convergence of the standard particle swarm algorithm. In paper [10], the authors proposed a novel hybrid metaheuristic algorithm based on genetic simulated annealing and particle swarm optimization, which achieved lower energy consumption in a shorter convergence time.

Reinforcement Learning algorithms for computation offloading problems: However, the studies above only focus on the performance of static systems, and it is difficult for these traditional mathematical optimization models to accurately model complex networks that change in real time. To solve this problem, some researchers introduced reinforcement learning algorithms into the computation offloading problem. In [11], the authors used the Q-Learning algorithm to solve the computation offloading problem. In [12], the authors proposed a joint optimization method based on deep Q-learning for device-level and edge-level task offloading. Zhou et al. further proposed a method based on Double Deep Q-Learning Network (DDQN), which can effectively approximate the value function of Q-learning, when the number of users is 5, compared with the local first and unloading first algorithm, the average energy consumption is reduced by 35% and 53% respectively [13].

Current limitations of the above researches: However, the researchers above rarely consider the scenario of multiple MEC servers, and the number of users is too small. In the scenario where multiple servers were considered, the author set the parameters of the servers to be constant [11], and only used the distance as the indicator to select the MEC servers, which was unpractical for the real scenario. In this paper, we consider the problem of computation offloading in the multi-user and multi-MEC Servers scenario. In the deployment of the MEC server, we set the computing power and location of the MEC as different parameters. The simulation results prove the feasibility of the algorithm.

III. Multi-user and Multi-MEC server system

In this section, we first analyze the multi-user multi-MEC server network, then build model for computation offloading problem for this network. Finally, the system performance indicator average processing cost as our optimal object along with average processing delay and average processing energy consumption are given out at the end.

A. System model of terminal users and multiple MEC servers

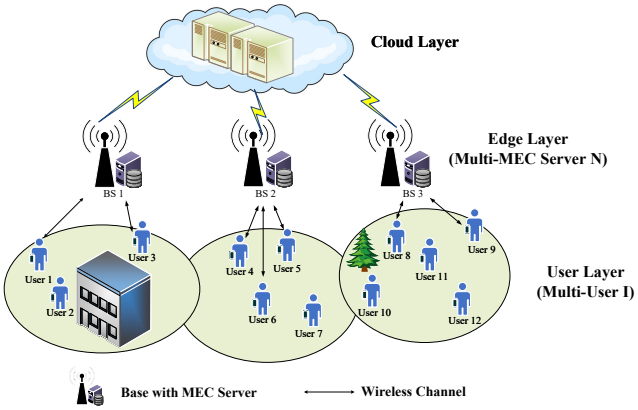


Fig. 1. System model

Figure 1 shows a typical multi-user multi MEC server network and its position inside overall communication system. In this scenario, the user set is defined as $\mathcal{I} = \{1, 2, \cdots, I\}$,

TP_i represents the priority of user i, $i \in \mathcal{I}$. The MEC server set is defined as $\mathcal{N} = \{1, 2, \cdots, N\}$. I and N stands for the maximum number of users and MEC servers respectively. The MEC server n has the computing capability is F_{ser}^n (cycles/s). The terminal users generate tasks that need to be processed in real time. The task set is denoted as $\mathcal{K} = \{1, 2, \cdots, K\}$, and for any $k \in \mathcal{K}$, let k be denoted by the triplet (d_k, θ_k, p_k). Where d_k represents the data size of task k (bits), θ_k represents the computational complexity of task k (cycles/bit), p_k represents the popularity of the task. Normally, when the user's local device is under heavy load, the burden tasks are sent to cloud server. In this way, the time response is related slow for some delay sensitive tasks. One of the solution to speed up respond is to build MEC servers in base station so that the terminal users can put their tasks direct to local MEC servers rather than to the cloud server. In this scene, users can offload the tasks to the MEC servers, and the MEC servers calculate the tasks and return the calculation results to the users. This mode reduce the overall network bandwidth usage and provide quicker respond to local users, but it brings the new problem that: how to distribute multiple users' tasks to multiple MEC servers so that the local network can have lower processing delay, processing energy consumption and cost.

In order to focus on the importance, we assume that 1) once the task is offloaded, it will be processed on selected edge server; 2) Task download delay will be ignored, wich is too small comparing with upload time delay [14]. With the purpose to measure the system performance objectively, we construct the performance indicators: average processing cost, delay and energy. We further choose the cost as a linear combination of delay and energy as our object function and try to minimize it.

B. Processing delay and energy consumption of the task

In this part, we analyze the processing delay and energy consumption of the task. Let $a_{i,t}^k$ denote the offloading decision of user i for task k at time t.

$$a_{i,t}^k = \begin{cases} 0 & \text{Execute locally} \\ n & \text{Execute in } MECn \end{cases} \tag{1}$$

When the task is executed locally, the task processing delay is the calculation delay of the task, which is defined as: $T_{loc,i,t}^k = \frac{d_k \times \theta_k}{F_{loc}^i}$. F_{loc}^i (cycles/s) is the computing capability of user i. Meanwhile, the energy consumption is generated when the user executes the task, which is defined as: $E_{loc,i,t}^k = P_{i,exe} \times T_{loc,i,t}^k$, where $P_{i,exe}$ (J/s) is the computing power of the user i.

When task k is generated by user i at time t and it is decided to offload to MEC server n for processing, user i needs to offload the task k to the MEC server n first, and then the MEC server calculates the task k and returns the calculation result to the user. The task offloading processing delay is the sum of offloading delay, the execution delay of the MEC server, and the download delay of the task, which is defined as:

$$T_{n,i,t}^k = T_{n,i,t}^{k,tran} + T_{n,i,t}^{k,exe} + T_{n,i,t}^{k,back} \tag{2}$$

$T_{n,i,t}^{k,tran}$ is the offloading delay of task k transmitted from user i to MEC server n at time t, which is expressed as $T_{n,i,t}^{k,tran} = \frac{d_k}{R_{n,i,j}}$. Where $R_{n,i,j}$ is the transmission rate of channel j, defined as:

$$R_{n,i,j} = B_j \log_2 \left(1 + \frac{h_j^2 P_{i,tran} d_{i,n}^{-\theta}}{N_0}\right) \tag{3}$$

In the Equation 3, B_j denotes the bandwidth of channel j, h_j denotes the fading factor of channel j. $P_{i,tran}$ is the transmit power of user i and $d_{i,n}$ is the transmission distance from user i to MEC server n. θ is a constant. N_0 is a stochastic norm distribution variable which stands for Gaussian white noise.

$T_{n,i,t}^{k,exe}$ represents the calculation delay of MEC server n processing task k at time t. There may be more than one task offloaded to MEC server n at time t. When multiple tasks are offloaded to MEC server n at the same time, the computing resources of MEC server n are allocated according to the data size and computing density of each task. $T_{n,i,t}^{k,exe}$ is defined as:

$$T_{n,i,t}^{k,\text{ exe}} = \frac{d_k \times \theta_k}{\frac{d_k \times \theta_k}{\sum_{k=1}^{K} d_k \times \theta_k \times o_{n,t}^k} \times F_{MEC}^n} \tag{4}$$

$o_{n,t}^k \in \{0, 1\}$ is the indicator variable. When task k is waiting for execution on MEC server n at time t, $o_{n,t}^k$ is 1, otherwise it is 0.

$T_{n,i,t}^{k,back}$ represents the transmission delay of the calculation result of task k returned from MEC server n to user i at time t. Compared with the data amount of the uploaded data, the data amount of the calculation result is small. In addition, the transmission rate of the backhaul is high, so $T_{n,i,t}^{k,back}$ is often ignored.

To sum up, when $a_{i,t}^k = n$, the processing delay of the task in MEC server n is expressed as: $T_{n,i,t}^k = T_{n,i,t}^{k,tran} + T_{n,i,t}^{k,exe}$, at this stage, we only consider the energy consumption of the user side, and the energy consumption of the terminal users is the energy consumed by the user when offloading tasks, so $E_{n,i,t}^k$ is defined as: $E_{n,i,t}^k = P_{i,tran} \times T_{n,i,t}^{k,tran}$.

C. Processing cost of the task

In this section, we aim to minimize the user's task processing delay and energy consumption simultaneously, so the cost of user i processing task k at time t is defined as follows:

$$C_{i,t}^k = \begin{cases} \alpha T_{loc,i,t}^k + \beta E_{loc,i,t}^k & a_{i,t}^k = 0 \\ \alpha T_{n,i,t}^k + \beta E_{n,i,t}^k & a_{i,t}^k = n \end{cases} \tag{5}$$

α and β ($\alpha \in \{0, 1\}$, $\beta \in \{0, 1\}$) are weight factors, which are used to describe the proportion of delay and energy consumption in the cost function.

The total cost of the system is defined as: $\sum_{t=1}^{T} \sum_{i=1}^{I} C_{i,t}^k \left(a_{i,t}^k\right)$, our goal is to take the optimal offloading decision to minimize the expectation of the

average processing cost of the system, which is formulated as:

$$\min E\left[\frac{1}{T}\frac{1}{I}\sum_{t=1}^{T}\sum_{i=1}^{I}C_{i,t}^{k}\left(a_{i,t}^{k}\right)\right]$$
$$\text{s.t.}\begin{cases} a_{i,t}^{k}\in\{0,1,2,\cdots,N\} \\ t\in\{0,1,2,\cdots,T\} \\ i\in\{0,1,2,\cdots,I\} \\ T_{loc,i,t}^{k}\mathbb{I}\left\{a_{i,t}^{k}=0\right\}+T_{n,i,t}^{k}\left(1-\mathbb{I}\left\{a_{i,t}^{k}=0\right\}\right)\leq\tau \end{cases} \tag{6}$$

$\mathbb{I}$ is an indicator vector, which returns 1 when $a_{i,t}^{k}=0$ is true, and 0 otherwise. We set the upper limit of the execution time of the task to τ, and the task is considered invalid if it exceeds this threshold.

IV. Introduction to DQN Algorithm and the Challenges in MEC Environment

In this section, we first discuss the challenges of using the DQN algorithm to solve the computation offloading problem in the MEC scenario. Then we discuss how to use the the idea of dynamic ε and iterative task distribution to deal with the users' offloading decision in the multi-user and multi-MEC server scenario.

A. Challenges of applying DQN in MEC environment

In the multi-user multi-MEC server scenario, with the increase of both the number of users and the number of MEC servers, the space for user offloading decisions increases exponentially, which brings difficulties to the convergence of the DQN algorithm. In order to speed up the convergence, we introduce the idea of dynamic exploitation into the DQN algorithm, so that ε changes adaptively in different stages of training. At the same time, we implement the offloading decision of users in the scenario one by one to avoid the problem of dimensional explosion of action sets.

B. Proposed dynamic exploitation factor

In the traditional ε-greedy strategy, the exploitation factor ε is a constant which means the algorithm adopts a greedy strategy with a fixed probability ε to select the optimal decision all along. On the contrary, the DQN algorithm explores with a probability of $1-\varepsilon$, that is, to select actions randomly. This means the probability of selecting each action is equal, which is also possible to select the optimal action. Let the space of the action set be $N\_actions$, we have:

$$\pi(a\mid s)=\begin{cases}\frac{1-\epsilon}{N\_actions}+\epsilon & \text{if } a^{*}=\underset{a\in A}{\operatorname{argmax}}Q(s,a)\\ \frac{1-\epsilon}{N\_actions} & \text{otherwise}\end{cases} \tag{7}$$

However, the fixed ε parameter cannot be adaptively changed according to the training phase of the DQN algorithm. On the basis of the classical DQN algorithm, this paper dynamically changes the ϵ parameter corresponding to the exploitation rate, and define:

$$\varepsilon=clip\left(1-\frac{e^{(V_1-R)}}{t},0.8,0.95\right) \tag{8}$$

Where the clip function limits the size of ε between 0.8 and 0.95. When $1-\frac{e^{(V_1-R)}}{t}$ is less than 0.8, the return value of the clip function is 0.8, and when $1-\frac{e^{(V_1-R)}}{t}$ is greater than 0.95, the return value of the clip function is 0.95. When the number of iterations t increases, the neural network parameters gradually stabilize, the algorithm gradually converges, and the value of $1-\frac{e^{(V_1-R)}}{t}$ increases. At the same time, $e^{(V_1-R)}$ is used to measure the pros and cons of the current selection. When V_1-R is large, the algorithm is still unstable. At this time, the value of $1-\frac{e^{(V_1-R)}}{t}$ value is small.

C. Proposed iterative task distribution scheme

In this part, we discuss how to apply the DQN algorithm to the offloading decision problem in the multi-user multi-MEC server scenario. Suppose that each user generates a computationally intensive task k at the beginning of each time slot of the system. Then we need to decide the offloading decision of these I tasks at time t. Since users in the scene compete with each other for the computing resources of the MEC server, if each user is regarded as an agent, there will be multiple competing agents in the scene. Traditional RL algorithm is difficult to apply to multi-agent environment because if each agent is trained separately, all of them are becoming intelligent that will make the environment unstable. That is:

$$P\left(s'|s,a,\pi_1,\ldots,\pi_N\right)\neq P\left(s'|s,a,\pi_1',\ldots,\pi_N'\right)\left(\forall\pi_i\neq\pi_i'\right) \tag{9}$$

where $\pi_1,\ldots,\pi_N$ is an ordered set of n player strategies. When the strategy of any one of the players is different, the state transition probability of the system will be different. In order to solve this problem, some researchers make offloading decisions for all users simultaneously [13]. However, the action space under this strategy faces the problem of dimensional explosion. For example, when the number of MEC servers is 3, the user's offloading decision is to process locally or offload the task to a certain MEC server for processing, so the size of the action space is 4. For the joint decision problem of I users, the action space size is 4^I. When the number of users is large, the problem becomes unsolvable because the action space is too large.

In this paper, we solve each agent's offloading decision serially. We first sort users according to their priorities as agent $1,2,\ldots,N$, and make offloading decisions for these sorted agent sequence in turn. As shown in Figure 2, at the beginning $(t=0)$, agent 1 that has the highest priority over other agents gets initial environment state s_0, transmit its own action a_1 and receive corresponding feedback reward r_1 by interaction with environment sequentially. The environment updates its state from s_0 to s_1 and interact similarly with agent 2 that has secondary priority at time $t=1$. These interaction continues to the lowest priority agent N at time $t=N-1$ so that all the offloading decision in this iteration are finished.

At the same time, instead of creating a new Q network and target network for each agent, we store the offloading decisions of all agents together. The specific method is that we

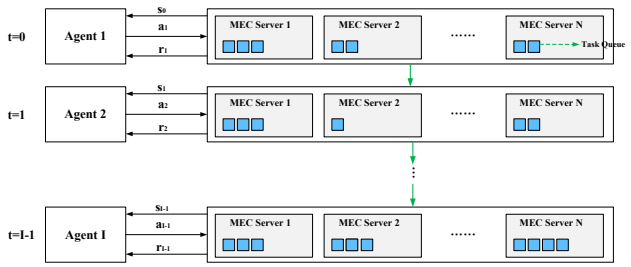


Fig. 2. Offloading decisions for multiple users

store the user's ID into the state set, and index the offloading decision of different users in the current state through different user IDs. As shown in Figure 3, we define $R^{'}_{i,a}$ to represent the corresponding reward when user i takes action a. At the same time, we use the superscript of R to distinguish the corresponding rewards in different states.

State \ Action	Action 1	Action 2	Action 3	Action 4
...	...	...	...	...
<0,0,0,0,id=I>	$R^{'}_{I,1}$	$R^{'}_{I,2}$	$R^{'}_{I,3}$	$R^{'}_{I,4}$
<0,2,0,1, id =I>	$R^{''}_{I,1}$	$R^{''}_{I,2}$	$R^{''}_{I,3}$	$R^{''}_{I,4}$
...	...	...	...	...

Fig. 3. Content of Q network

V. Computation Offloading Decision Based on Improved-DQN Algorithm

In this section, an Improved-DQN based computation offloading algorithm to minimize the average processing cost of the system is proposed. Since the objective function 6 is a mixed integer nonlinear programming problem that is one kind of NP-hard (Non-deterministic Polynomial) problem, which is difficult to be solved by traditional optimization methods [13]. We use the Improved-DQN algorithm to solve the offloading decision of the tasks. In the following, we first model the problem as a markov decision process. Then we give the flow of the Improved-DQN algorithm in detail.

A. Markov decision process

The optimization problem is modeled as a Markov Decision Process (MDP), and the problem is formalized with $< S, A, R >$:

(1) State:

$$S = \{\text{tas}\,k_{MEC1,t}, \text{task}_{MEC2,t}, \cdots, \text{tas}\,k_{MECN,t}, id\}$$

$\text{task}_{MECn,t}(n \in \mathcal{N})$ represents the number of tasks to be processed by MEC server n at time t. $\text{task}_{MECn,t}$ varies with user offloading decision. id represents the number of the user.

(2) Action:

$$A = \left\{a^k_{i,t} \mid a^k_{i,t} \in \{0, 1, 2, \cdots, N\}\right\}$$

$a^k_{i,t}(i \in \mathcal{J})$ represents the offloading decision of user i for task k at the current moment t. When $a^k_{i,t}$ is 0, it is considered that task k is processed locally, otherwise, task k will be offload to MEC server n.

(3) Reward:

$$R = \min\left\{V_1, \frac{V_2}{C^k_{i,t}\left(a^k_{i,t}\right)}\right\}$$

Since $C^k_{i,t}\left(a^k_{i,t}\right)$ is a negative indicator, we define the reward as the inverse of the cost function. At the same time, we delimit the reward, let the reward is the smaller one of V_1 and $\frac{V_2}{C^k_{i,t}\left(a^k_{i,t}\right)}$.

In order to ensure that the algorithm satisfies the constraints in the objective function 6, the task processing delay is calculated first before the cost. If the processing delay of the task exceeds the threshold τ, the reward function is negative. So that, the situation where the processing delay exceeds the threshold can be avoided.

B. Flow of the Improved-DQN algorithm

Algorithm 1 describes the flow of Improved-DQN algorithm.

Algorithm 1 Improved-DQN algorithm

```
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode = 1, 2, ..., E do
    Initialize sequence s = {0, ..., 0} and get its feature vector φ_1 = φ(s_1)
    Initialize ep_reward = 0
    for t = 1, 2, ..., T do
        for user = 1, 2, ..., I do
            Calculate ε = max(1 − e^(V1−R)/t, 0.9)
            With probability ε select a_t = max_a Q*(φ(s_t), a; θ)
            Otherwise select a random action a_t
            Execute action a_t in model and observe reward r_t and next state s_(t+1)
            Get the feature vector of s_(t+1), that is φ_{t+1} = φ(s_{t+1})
            Store transition (φ_t, a_t, r_t, φ_{t+1}) in D
            Sample random minibatch of transitions (φ_j, a_j, r_j, φ_{j+1}) from D
            Set y_j according to φ_{j+1}
            Perform a gradient descent step on (y_j − Q(φ_j, a_j; θ))^2 with respect to the network parameters θ
        end for
    end for
end for
```

Specifically, the algorithm first initializes a memory pool that can store N pieces of data and an action-value function Q for random network parameters. Then the algorithm enters the first loop of each episode. At the beginning of the new

episode, the parameters of the algorithm are reset, and the current state is the initial state. Here, the state is the number of tasks currently requesting the MEC server and the number of the user with the highest priority, so the initial state is an n-dimensional 0 vector, and the feature vector of the state is obtained through preprocessing. After that, the current round enters the second loop, which is T-step iteration. In each iteration, we iterate over I users in the scene, which is the third loop.

The third loop is the training subject. First, the dynamically changing parameter ϵ is calculated according to the Equation 8. If a high probability event occurs, the optimal action is selected according to the greedy strategy, that is, $a_t = \max_a Q^*(\phi(s_t), a; \theta)$; otherwise the algorithm explores and executes random selected action a_t. Here, executing the action means that the task k is processed locally or offloaded to a certain MEC server for processing. At this time, the system returns the time and energy consumption of executing the task, and calculates the reward r_t. Meanwhile, the state vector changes based on the user's offloading decision. s_{t+1} is processed as feature vector, and the transformed sample $(\phi_t, a_t, r_t, \phi_{t+1})$ is stored in memory pool D. Next, we randomly sample a small batch of transformation samples $(\phi_t, a_t, r_t, \phi_{t+1})$ from the memory pool of the target network, and judge whether ϕ_{t+1} is a terminal state, if so, $y_j = r_j$; otherwise $y_j = r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta)$. DQN updates the parameters of the neural network through back-propagation of gradients, namely:

$$\theta_{t+1} = \theta_t + \alpha \left[y_j - Q(\phi_j, a_j; \theta) \right] \nabla Q(\phi_j, a_j; \theta) \qquad (10)$$

y_j is obtained from the target network, and $Q(\phi_j, a_j; \theta)$ is obtained from the Q-value network. This operation can effectively reduce the correlation between sample data and speed up the convergence process of the algorithm.

VI. Simulation Results

In this section, we present our simulation results. We deployed 3 MEC servers in the MEC scenario, the computing power and spatial location of these MEC servers are different. At the same time, several stationary users are discretely distributed. In this section, we set the baseline algorithm as the greedy algorithm and DQN algorithm which are the basis of our work. The greedy algorithm avoids exploration and continuously selects the optimal decision estimated by the algorithm at the current moment. The simulation parameters are summarized in Table I.

TABLE I
System Parameters

Parameters	Value
Channel Bandwidth B_i	5 MHz
PathLoss Parameters θ	4
Transmit Power P_i^{tr}	100 mW
Tolerate delay of the task τ	0.35 ms
Location of MEC 1	[1316, 93]
Location of MEC 2	[1985, 374]
Location of MEC 3	[464, 398]
Computation Frequency of WBAN F_i^{loc}	1 GHz
Computing Complexity of task X_i	100 cycles/bit
Computation Frequency of MEC F^{ser}	[9 GHz, 10GHz, 8GHz]
Target network replace frequency	100
Maximum number of episodes to train	1000

A. Hyper-parameter setting according to convergence rate of the Improved-DQN algorithm

In this part, two hyper parameters (Learning rate α and discount factor γ) setting according to the convergence rate of the Improved-DQN algorithm are discussed.

We set α to be 0.01, 0.001, 0.002 to analyze the convergence of the algorithm. As shown in Figure 4 (a), when the α is 0.01, 0.001, 0.002, the algorithm has achieved good convergence effect. Among them, when the α is 0.002, the algorithm converges in nearly 300 steps, and the convergence speed is the fastest. Therefore we choose α = 0.002.

After setting learning rate α to 0.002, we turn to discount factor which is delay coefficient that determines the importance of future rewards in learning. Here we set γ to be 0.9, 0.95, 0.99. As can be seen from Figure 4 (b), when γ is 0.9 and 0.95, the convergence effect of the algorithm is better. The algorithm converges at about 400 steps. Here, we choose 0.9 as the value of γ.

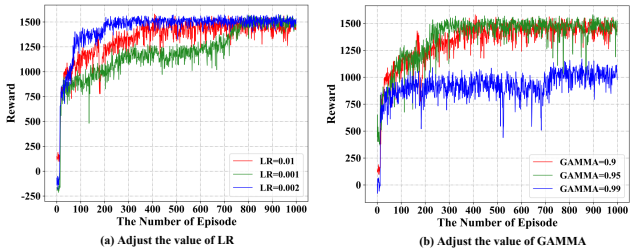


Fig. 4. Adjust the value of LR and GAMMA

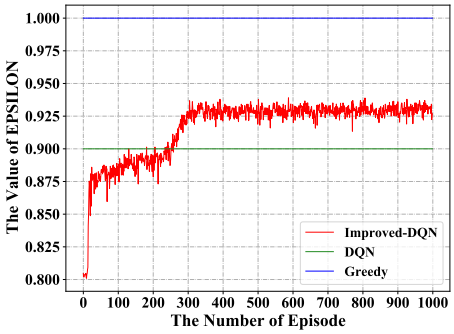
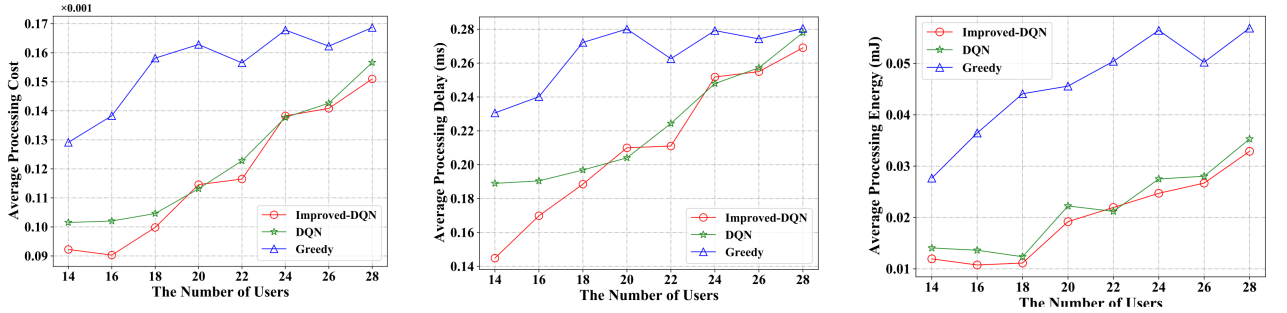


Fig. 5. Change of ε during the training process

After that, we fix the hyperparameter α to 0.002, γ to 0.90, and analyze the value change of ε during the algorithm training process. As shown in Figure 5, the Greedy algorithm adopts a greedy strategy and continuously uses the current optimal decision. Therefore, during the algorithm training process, ε is always 1. In the DQN algorithm, ε is a fixed value, which



(a) Average processing cost under different user numbers (b) Average processing delay under different user numbers (c) Average processing energy under different user numbers

Fig. 6. Average processing cost under different user numbers

remains unchanged during the training process. The Improved-DQN algorithm introduces the idea of dynamic exploitation. In the early stage of algorithm training, ε is small, and the algorithm is mainly based on exploration. As the number of episode increases, the algorithm gradually converges that made the value of epsilon increase to a stable value. At this time, the algorithm is mainly based on exploitation. As can be seen from the Figure 5, when the number of training episode is greater than 300, the Improved-DQN algorithm is gradually stabilized, and ε remains around 0.925.

B. Performance comparison under different users

In this part, the comparison between Improved DQN algorithm and other two algorithm (DQN algorithm and greedy algorithm) under different users are summarized.

As shown in Figure 6 (a), as the number of users increases, the average processing cost of all three algorithm increase. The reason is the increased tasks by increased users race limited resource. To be specific, the tasks generated sequentially in a short time period are competing for the computing resources of the MEC server simultaneously, so the average processing cost of tasks increases. The improved-DQN performs better than the other two, this advantage is more obvious when number of users lie in between (16, 20). The distance between the Improved-DQN and other two algorithm shrink when number of users increase more. Generally speaking, the average processing cost is reduced by using Improve-DQN algorithm around 12% comparing with the Greedy algorithm.

As shown in Figure 6 (b), as the number of the users increases, the average processing delay of the Improved-DQN algorithm, DQN algorithm and the greedy algorithm all tend to increase. As can be seen from the figure, compared to the Improved-DQN algorithm and the DQN algorithm, the greedy algorithm has some jitter. The reason for this phenomenon may be that the greedy algorithm randomly assigns weights to the parameters of the neural network during initialization, and then continuously selects the optimal decision. The contingency in the initialization stage has caused certain fluctuations to the algorithm. Compared with the baseline algorithm, the Improved-DQN algorithm has more advantages.

From Figure 6 (c), we can see that as the number of users in the scene increases, the average processing energy consumption of all the algorithms increases. Meanwhile, when the number of users lie in between (14,18), the average processing energy consumption of DQN algorithm and Improved-DQN algorithm decrease. This may be related to the geographical distribution of users. In addition, when the number of users in the scenario is small, the resource competition phenomenon is not serious, so there is no significant increase in the average processing energy consumption index. Compared with the greedy algorithm, the Improved-DQN algorithm and the DQN algorithm have obvious advantages.

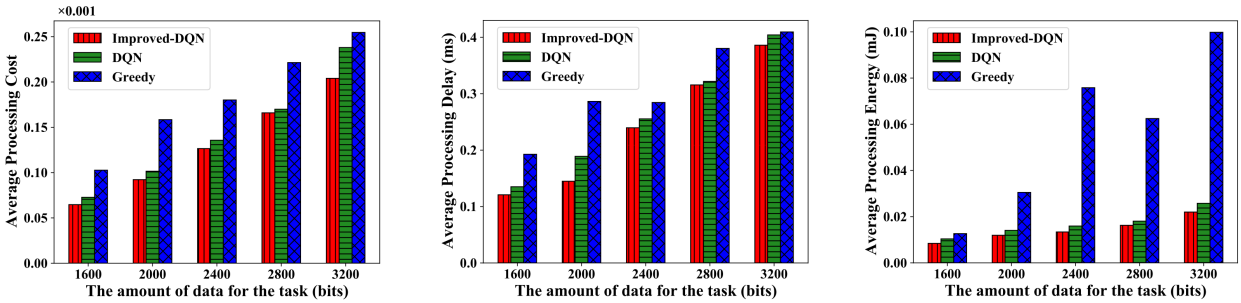
C. Performance comparison under different data size

In this part, we want to see the data size's effect on the Improved DQN algorithm, DQN algorithm and the greedy algorithm respectively.

As shown in Figure 7 (a), as the size of the amount of data for the task increases, the average processing cost of the task increases. This is due to that the increased processing cost required to process a single task. We can see that the data size of the task has doubled from 1600 bits to 3200 bits. However, the average processing cost of the task has increased by about 3 times. The main reason is that when the data volume of a task increases, which leads to an increase in the processing time of the task on the MEC server, so the computing resources that can be allocated by the MEC server per unit time decrease. As can be seen from the Figure 7 (a), the Improved-DQN algorithm has obvious advantages compared to DQN algorithm and the greedy algorithm.

Next, we use the average processing delay as an indicator to compare and analyze the Improved-DQN algorithm, DQN algorithm and the greedy algorithm. In Figure 7 (b), as the amount of data for the task increases, the average processing delay of the task increases. Figure 7 (b) shows that the Improved-DQN algorithm performs better than the baseline algorithms.

Finally, we compare the two algorithms under the average processing energy consumption. With the increase in the amount of data for the task, the average processing energy consumption of the Improved-DQN algorithm and DQN algo-



(a) Average processing cost under different data size of the task (b) Average processing delay under different data size of the task (c) Average processing energy under different data size of the task

Fig. 7. Average processing cost under different data size of the task

rithm increases slightly, while the average processing energy consumption of the greedy algorithm fluctuates. The main reasons for this phenomenon are that the greedy algorithm randomly assigns network weights during initialization, resulting in a certain degree of randomness. Under the criteria of the average processing energy consumption, the Improved-DQN algorithm still performs well.

VII. Conclusion

In this paper, we propose an Improved-DQN algorithm based method to solve computation offloading decision problem in multi-user multi-MEC server scenario. In order to apply RL framework into this scenario with multi-agent, we treat each user as individual agent that can interact with all MEC servers as environment at one time. After the interaction fished, the algorithm finds another agent with lower priority iteratively. Furthermore, we try to minimize the average processing cost by novel dynamic exploitation rate scheme that can adapt to reward and convergent time. The final simulation results carried among the proposed algorithm, greedy algorithm and DQN algorithm under different users and data size show that the proposed algorithm has great advantages in terms of average processing cost, average processing delay and average processing energy consumption. However, there is some optimal space for our work. In the future, we will reorder the priorities of agents in the waiting list and using more adapative RL based model from both new invention and the existing to improve our work.

ACKNOWLEDGEMENT

This research is partially supported by National Key Research and Development Program of China 2021ZD0110403, National Natural Science Foundation of China (NSFC) with Grant No. 62172383 and No.62231015. Anhui Provincial Key R&D Program with No. S202103a05020098. Xiaohua Xu is the contact author.

References

[1] Guisong Yang, Ling Hou, Xingyu He, Daojing He, Sammy Chan, and Mohsen Guizani. Offloading time optimization via markov decision process in mobile-edge computing. *IEEE Internet of Things Journal*, 8(4):2483–2493, 2021.

[2] Dali Zhu, Ting Li, Haitao Liu, Jiyan Sun, Liru Geng, and Yinlong Liu. Privacy-aware online task offloading for mobile-edge computing. *Wirel. Commun. Mob. Comput.*, 2021:6622947:1–6622947:16, 2021.

[3] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2018.

[4] Georgios Fragkos, Nicholas Kemp, Eirini Eleni Tsiropoulou, and Symeon Papavassiliou. Artificial intelligence empowered uavs data offloading in mobile edge computing. In *ICC 2020*, pages 1–7, 2020.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[6] Ming Tang and Vincent W.S. Wong. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing*, pages 1–1, 2020.

[7] Junhui Zhao, Qiuping Li, Yi Gong, and Ke Zhang. Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks. *IEEE Transactions on Vehicular Technology*, 68(8):7944–7956, 2019.

[8] Luobing Dong, Meghana N. Satpute, Junyuan Shan, Baoqi Liu, Yang Yu, and Tihua Yan. Computation offloading for mobile-edge computing with multi-user. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 841–850, 2019.

[9] Jinze Wu, Zhiying Cao, Yingjun Zhang, and Xiuguo Zhang. Edge-cloud collaborative computation offloading model based on improved partical swarm optimization in mec. In *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 959–962, 2019.

[10] Jing Bi, Haitao Yuan, Shuaifei Duanmu, MengChu Zhou, and Abdullah Abusorrah. Energy-optimized partial computation offloading in mobile-edge computing with genetic simulated-annealing-based particle swarm optimization. *IEEE Internet of Things Journal*, 8(5):3774–3785, 2021.

[11] Dongyu Wang, Xinqiao Tian, Haoran Cui, and Zhaolin Liu. Reinforcement learning-based joint task offloading and migration schemes optimization in mobility-aware mec network. *China Communications*, 17(8):31–44, 2020.

[12] Peizhi Yan and Salimur Choudhury. Deep q-learning enabled joint optimization of mobile edge computing multi-level task offloading. *Computer Communications*, 180:271–283, 2021.

[13] Huan Zhou, Kai Jiang, Xuxun Liu, Xiuhua Li, and Victor C. M. Leung. Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing. *IEEE Internet of Things Journal*, 9(2):1517–1530, 2022.

[14] Xingqiu He and Sheng Wang. Peer offloading in mobile-edge computing with worst case response time guarantees. *IEEE Internet of Things Journal*, 8(4):2722–2735, 2021.