Dynamic Offloading Loading Optimization in Time-Varying Mobile Edge Networks with Deep Reinforcement Learning Approach

Liang Yu\textsuperscript{a}, Rui Wang\textsuperscript{b,}\textsuperscript{*}, Minyan Shi\textsuperscript{b}

\textsuperscript{a}Institute of Vibration, Shock and Noise, State Key Laboratory of Mechanical System and Vibration, Shanghai Jiao Tong University, Shanghai, 200240, China
\textsuperscript{b}College of Electronics and Information Engineering, Tongji University, Shanghai, 201804, China

Abstract

Mobile edge computing (MEC) has been regarded as a promising wireless access architecture to alleviate the burden of performing computation for large content size tasks at resource limited mobile terminals. By allowing the mobile terminals to offload the computation tasks to MEC servers, the task processing delay can be significantly decreased. As the resources at the MEC and user are limited, performing reasonable resources allocation optimization can improve the performance, especially for a multi-user offloading system. In this study, with an aim to minimize the task computation delay, we jointly optimize the local content splitting ratio, the transmission/computation power allocation, and the MEC server selection under a dynamic environment with stochastic task arrivals and time-varying wireless channels. To address the challenging dynamic joint optimization problem, we formulate it as a reinforcement learning (RL) problem. We design the computational offloading policies to minimize the long-term average delay cost. Two deep RL strategies, that is, deep Q-learning network (DQN) and deep deterministic policy gradient (DDPG), are adopted to efficiently learn the computational offloading policies adaptively. The proposed DQN strategy takes the MEC selection as a unique action while using convex

\textsuperscript{*}Corresponding author
Email address: ruiwang@tongji.edu.cn (Rui Wang)
optimization approach to obtain the local content splitting ratio and the transmission/computation power allocation. Simultaneously, the actions of DDPG strategy are selected as all dynamic variables including the local content splitting ratio, the transmission/computation power allocation, and the MEC server selection. Our numerical results demonstrate that both proposed strategies perform better than the traditional non-learning scheme, and the DDPG strategy is superior to the DQN strategy as it can online learn all variables although it requires large complexity.

**Keywords:** mobile edge computing, multi-user offloading, time-varying channels, reinforcement learning

1. Introduction

With the development of mobile communication technology, more and more applications have put forward higher requirements on computation resources and time delay, such as intelligent transportation and virtual reality. These applications generate a large number of computational tasks on the mobile devices. Due to the limited computational resources at the mobile devices, we need to offload the computational intensive contents to the servers with strong computational capability for processing. The traditional cloud computing mode is to offload the computing tasks to remote servers deployed in the core networks. However, the cloud servers are usually far away from the devices, which causes waste of resources during transmission and introduces message lag. The emergence of mobile edge computing (MEC) provides a solution to these problems. Compared with cloud computing, MEC deploys servers which called mobile edge servers closer to the users. The task processing can be completed at near MEC servers with a greatly shorten time delay.

To improve the performance, computing offloading, as one of the core techniques of the MEC, has received great attentions recently. For simple and indivisible or highly integrated tasks, binary offloading strategies are generally adopted, and tasks can only be computed locally or all offloaded to the
servers [5]. The authors in [6] formulated the binary computation offloading decision problem as a convex problem, which minimizes the transmission energy consumption under the time delay constraint. The computation offloading model studied in [7] assumed that the application has to complete the computing task with a given probability within a specified time interval, and the optimization goal is the sum of local and offloading energy consumption. This work concluded that in some cases, offloading computing tasks to the MEC servers can be more efficient.

In practice, offloading decisions can be more flexible. The computation tasks can be divided into two parts which are performed in parallel, one part is processed locally, and the other part is offloaded to the MEC servers for processing [8]. Under this computation offloading model, the authors in [9] proposed a task-call graph model to illustrate the dependency between users and MEC servers, and investigated the joint offloading-scheduling decision, latencies and the energy saved by formulating it as a linear programming problem.

More recently, reinforcement learning (RL), which is a model-free machine learning algorithm that can perform self-iterative training based on the data it generates, has been employed as a new solution to the problem of mobile edge computation offloading [10, 11, 12, 13]. For time-sensitive applications, task processing delay is an important optimization parameter. In [10], the authors studied the problem of computation offloading in an internet of things (IoT) network. A Q-learning based RL approach was proposed for an IoT device to select an proper device and to determine the proportion of the computation task to offload. The authors in [11] investigated the joint communication, caching and computing for the vehicular networks by considering the vehicle’s mobility. A deep Q-learning based RL with multi-timescale framework was developed to solve the joint online optimization problem. In [12], the authors studied the offloading for the energy harvesting (EH) MEC network. An after-state RL algorithm was proposed to address the large time complexity problem and polynomial value function approximation was introduced to accelerate the learning process. In [13], the authors also studied the MEC network with EH device.
The authors proposed a hybrid based actor-critic learning for optimizing the offloading ratio, local computation capacity, and server selection.

As seen from [10, 11, 12, 13], efficient computation offloading decisions made by RL based approach can help applications improve performance. In this work, we studied the offloading decision in a dynamic MEC environment. Different from current studies, we consider a more practical scenario where the current wireless channel state information (CSI) cannot be observed when making offloading decision. The offloading policy including local content splitting ratio, the transmission/computation power allocation, and the MEC server selection should be made in accordance to the predicted CSI and task arrival rates under the user and server energy constraints in a multi-user multi-MEC server network, with an aim to minimize the long-term average delay cost. We first establish a low-complexity deep Q-learning network (DQN) based offloading framework where the action includes only discrete MEC server selection, while local content splitting ratio, the transmission/computation power allocation are optimized by convex optimization method. Then we develop a deep deterministic policy gradient (DDPG) based framework which includes both discrete MEC server selection variable, and continuous local content splitting ratio, the transmission/computation power allocation variable as actions. Our numerical results demonstrates that both proposed strategies perform better than the traditional non-learning scheme, and the DDPG strategy is superior to the DQN strategy as it can online learn all variables although it requires large complexity.

2. System Model

2.1. Network Model

Consider an offloading-enabled wireless network that includes $M$ MEC servers and $K$ users as shown in Fig. 1. Let $\mathcal{M} = \{1, \cdots, M\}$ and $\mathcal{K} = \{1, \cdots, K\}$ be the index sets of the MEC servers and the users, respectively. Suppose that
the MEC server has stronger computing capability than the user, so part of the tasks may be offloaded to the MEC server. The system time is divided into consecutive time frames with equal time period $\tau_0$ and the time indexed by $t \in T = \{0, 1, \cdots\}$. Denote the channel state information between the $m$-th MEC server and the $k$-th user as $h_{m,k}$ and the size of the task at user $k$ as $C_k$. The channel state information of the MEC network $\{h_{m,k}(t)\}$ and the task arrival $C_k(t)$ at each user change for each time interval $t \in T$. To save the energy consumption at both the user and MEC server, and shorten the task processing delay, a central agent node needs to determine task ratio of local executed content size and the offloading content size, the power splitting ratio between the local task processing and data transmission. Moreover, if an MEC server is selected to help process the tasks coming from more than one user, the power splitting at this MEC server between multiple users should be also determined. In what follows, we will introduce the communication model and the computational model in detail.

2.2. Communication Model

In the considered MEC network, we assume that communications are operated in an orthogonal frequency division multiple access (OFDMA) framework.
and a dedicate subchannel with bandwidth $B$ is allocated for each user for the partial task offloading. Supposing that user $k$ communicates with MEC server $m$, the received signal at MEC $m$ receiver can be represented as

$$y_{m,k} = h_{m,k}\sqrt{p_k^o(t)}s_k + n_{m,k}$$

where $s_k$ denotes the symbols transmitted from user $k$ and $p_k^o(t)$ is the utilized power at user $k$. $n_{m,k}$ denotes the received additive Gaussian noise with power $N_0$. Here we assume that the wireless channel gains $h_{m,k}(t)$ follows the finite-state Markov chain (FSMC), and thus the communication rate between MEC server $m$ and user $k$ is give by

$$r_{m,k}^o(t) = B \log_2 \left( 1 + \frac{p_k^o(t)|h_{m,k}(t)|^2}{N_0} \right).$$

2.3. Computing Model

In the considered MEC system, it is supposed that the task $C_k(t)$ received at user $k$ at time $t$ need to be processed during time interval $t$. Denote the task splitting ratio as $\alpha_k \in [0, 1]$ which indicates that at time interval $t$, $\alpha_k C_k(t)$ bits are executed at the user device and the remaining $(1 - \alpha_k)C_k(t)$ bits are offloaded to and processed by the MEC server.

1) Local computing: In local computation, the CPU of the user’s device is the primary engine, which adopts the dynamic frequency and voltage scaling (DVFS) technique, and the performance of the CPU is controlled by the CPU-cycle frequency $\kappa_u$. Let $p_k^l(t)$ denote the local processing power at user $k$, then the user’s computing speed (cycles per second) $f_k^l(t)$ at $t$-th slot is given by

$$f_k^l(t) = \frac{3}{\kappa_u} \frac{p_k^l(t)}{\kappa_u}$$

Let $D_k$ denote the number of CPU cycles required for user $k$ to accomplish one
task bit. Then the local computation rate for user $k$ at time slot $t$ is given by

$$r_{l,k}^t = \frac{f_{l,k}^t}{D_k} = \frac{3/\sqrt{\kappa_u}}{D_k}$$

(2)

2) Mobile Edge Computation Offloading: The task model for mobile edge computation offloading is data-partition model, where the task-input bits are bit-wise and can be arbitrarily divided into different groups. At the beginning of the time slot, the user chooses which MEC server to connect according to the channel state. Assume that the processed power which is allocated to the user $k$ by the MEC server $m$ is $p_{m,k}$, then the computation rate $r_{c,m,k}$ at MEC server $m$ for user $k$ is:

$$r_{c,m,k} = \frac{3/\sqrt{p_{m,k}}}{\kappa_m}$$

(3)

where $D_m$ is the number of CPU cycles required for MEC server to accomplish one task bit, and $\kappa_m$ denotes the the CPU-cycle frequency at the MEC server. It is noted that a MEC server simultaneously processes the tasks from multiple users, and we assume that multiple application can be executed in parallel with a negligible processing latency. Moreover, the feedback time from the MEC to the user is ignored due to the small sized computational output.

3. DQN Based Offloading Design

In this section, we develop a DQN based offloading framework for minimizing the long-term processing delay cost. As the development of the traditional Q-learning algorithm, DQN is particularly suitable for high-dimensional state spaces and posses fast convergence behavior. In the considered DQN offloading design framework, the whole MEC system constructs the DQN environment and there exists a central agent node which observes the states, performs actions and receives the fed-back reward. The central can be the cloud server or a MEC server.

In the following, we introduce the DQN based offloading framework and the corresponding state space, action space and reward will be defined. In the
overall DQN paradigm, it is assumed that the instantaneous CSI is estimated at MEC servers using the training sequences, and then delivered to the agent. Here, due to the channel estimation operations and feedback delay, the CSI observed at the agent is the delayed version. Moreover, for each MEC server, only local CSI of users which connect this MEC server is acquired.

3.1. System state and action spaces

System State Space: In the considered DQN paradigm, the state space observed by the agent includes the CSI of the overall network and the received task size $C_k(t)$ at time $t$. As agent needs consumes extra communication overhead to connect the CSI from all MEC servers, at time $t$ it only observes delayed version of CSI at time $t-1$, i.e., $\{h_{m,k}(t-1)\}$. Denote

$$h(t) = \{h_{1,1}(t), h_{1,2}(t), \ldots, h_{M,K}(t)\},$$
$$C(t) = \{C_1(t), C_2(t), \ldots, C_K(t)\}. \quad (4)$$

The state space observed at time $t$ can be represented as

$$S(t) = \{h(t-1), C(t)\}.$$ 

System Action Space: With the observed state space $S(t)$, the agent will take certain actions to interact with the environment. As DQN can only take care of the discrete actions, the actions defined in the proposed DQN paradigm constitute only the MEC server selection. Denote the MEC server selection action as $a(t)$, which can be represented as

$$a(t) = \{x_{m,k}(t) | x_{m,k}(t) \in \{0, 1\}\}.$$ 

Here, $x_{m,k}(t) = 0$ means that the user $k$ does not select the MEC server $m$ at $t$-th time slot, while $x_{m,k}(t) = 1$ indicates that the user $k$ selects the MEC server $m$ at $t$-th time slot.
3.2. Reward Function

In the considered DQN paradigm, the reward is defined as the maximum time delay required to complete all the tasks received at all users. After taking the actions, a dedicated MEC server can basically calculate the time delays required for the users choosing this MEC server to offload, as all MEC can observe the local CSI. With loss of generality, we assume that user $k$ with $k \in \mathcal{O}_k$ offloads the tasks to MEC $m$, where set $\mathcal{O}_k$ defines the indexes of the users selecting MEC server $m$ to offload tasks. To minimize the required time delays, the MEC server needs to formulate an optimize problem to find optimal $\alpha_k(t)$, $p_k^l(t)$, $p_k^o(t)$, and $p_{m,k}^c(t)$. It is worth noting that as the MEC server knows the instantaneous CSI at time $t$, the solution can be obtained based on $h(t)$, which is different from the MEC server selection taken based on $h(t-1)$. For the users which do not offload tasks to the MEC servers, the required time delays for local task processing can be known by these users. Then, the agent collects all the time delay consumptions from the users and the MEC servers to obtain the final reward.

Next we detail how to compute the time delay for user $k$ assuming that it selects MEC server $m$ to offload. Denote the total time consumption for completing the task processing at user $k$ by $t_k$, which equals to $t_k = \max\{t^l_k, t^o_{m,k} + t^c_{m,k}\}$, where $t^l_k$, $t^o_{m,k}$, and $t^c_{m,k}$ denote the times required for user local task processing, task offloading transmission from user $k$ to MEC server $m$, and task processing at MEC server, respectively.

With the computation rate $r^l_k$ defined in eq. (2), time $t^l_k$ can be represented as

$$t^l_k = \frac{\alpha_k C_k(t)}{r^l_k(t)}.$$

As the size the offlaoded task is $(1-\alpha_k)C_k$, with the communication rate defined in eq. (1), time $t^o_{m,k}$ can be calculated as

$$t^o_{m,k} = \frac{(1-\alpha_k)C_k(t)}{r^o_{m,k}(t)}.$$


With the computation rate $r_{m,k}$ allocated by MEC server $m$ to user $k$ in eq. (2), time $t_{m,k}^c$ can be computed as

$$t_{m,k}^c = \frac{(1 - \alpha_k)C_k(t)}{r_{m,k}(t)}.$$  

To maximize the reward, we need to minimize the required time delay for each user under the total energy constraint at users and MEC servers. To illustrate the way to find optimal $\alpha_k(t)$, $p_k^l(t)$, $p_k^o(t)$, and $p_{m,k}^c(t)$ for different types of MEC server selection, we next present two typical offloading scenarios, that is, a MEC server serves one user and a MEC server serves two users. It is noted that the proposed way of solving $\alpha_k(t)$, $p_k^l(t)$, $p_k^o(t)$, and $p_{m,k}^c(t)$ can be extended to the case where a MEC server serves arbitrary number of users.

1) Scenario 1: one MEC server serves one user

The energy consumption at user $k$, denoted by $E_k$, includes two parts, i.e., one part for local partial task processing and another for partial task transmission. Therefore, $E_k$ can be written as

$$E_k = p_k^l(t)t_k^l + p_k^o(t)t_k^o.$$  

The energy consumption at the MEC server $m$ for processing the partial task offloaded from user $k$ is denoted by $E_{m,k}$, and can be represented as

$$E_{m,k} = p_{m,k}^c(t)t_{m,k}^c.$$  

The optimization problem formulated to find optimal $x_k(t) = \{\alpha_k(t), p_k^l(t), p_k^o(t), p_{m,k}^c(t)\}$ is given by

$$\max_{x_k(t)} \quad t_k = \max\{t_k^l, t_{m,k}^o + t_{m,k}^c\} \quad \text{(5a)}$$

s.t. $$E_k \leq E_{\max,k}, \forall k \in K \quad \text{(5b)}$$

$$E_m \leq E_{\max,m}, \forall m \in M \quad \text{(5c)}$$
where \( E_{\text{max},k} \) and \( E_{\text{max},m} \) denote the maximum available energy at user \( k \) and MEC server \( m \), respectively. Problem (5) can be rewritten as

\[
\min_{x_k(t)} \quad \max \left\{ \frac{\alpha_k(t) C_k(t)}{\frac{3/4}{D_k}}, \frac{\alpha_k(t) C_k(t)}{\frac{3/4}{D_m}} \right\}
\]

subject to

\[
p_k(t) t_k^l + p_m^c(t) t_{m,k}^c \leq E_{\text{max},k}
\]

\[
p_m^c(t) t_{m,k}^c \leq E_{\text{max},m}
\]

\[
\alpha_k \in [0, 1]
\]

To solve problem (6), we first find that at optimal solution, constraint (6c) must be active, which can minimize the objective value (6a). We thus have

\[
p_c^c(t) t_{m,k}^c = \left( \frac{E_{\text{max},m}}{\frac{3/4}{D_m}} \right)^{3/2}
\]

which produces

\[
p_c^c(t) = \left( \frac{E_{\text{max},m}}{(1 - \alpha_k(t)) C_k(t) \kappa_m^{1/3} D_m} \right)^{3/2}
\]

Substituting (7) to problem (6), we have

\[
\min_{x_k(t)} \quad \max \left\{ \frac{\alpha_k(t) C_k(t)}{\frac{3/4}{D_k}}, \frac{\alpha_k(t) C_k(t)}{\frac{3/4}{D_m}} \right\}
\]

subject to

\[
p_k(t) t_k^l + p_m^c(t) t_{m,k}^c \leq E_{\text{max},k}
\]

\[
p_m^c(t) t_{m,k}^c \leq E_{\text{max},m}
\]

\[
\alpha_k \in [0, 1]
\]

It is noted that problem (8) is a non-convex optimization problem. To find
efficient solution, we propose an alternating algorithm to separately solve \( p'_k(t) \), \( p''_k(t) \) and \( \alpha_k(t) \) in different subproblems.

In the first subproblem, we solve \( p'_k(t) \) for given \( \alpha_k(t) \), and \( p''_k(t) \). To minimize the objective function, the optimal solution of \( p'_k(t) \) should activate constraint (8b), that is, \( p'_k(t)t_k^l = E_{\text{max},k} - p''_k(t)t_{m,k}^o \), which implies

\[
p'_k(t) = \left( \frac{E_{\text{max},k} - p''_k(t)t_{m,k}^o}{\alpha_k(t)C_k(t)D_l\kappa_l^{1/3}} \right)^{3/2}.
\] (9)

In the second subproblem, we solve \( p''_k(t) \) with given \( p'_k(t) \) and \( \alpha_k(t) \). The corresponding optimization problem is given by

\[
\begin{align*}
\min_{p''_k(t)} & \quad \frac{(1 - \alpha_k(t))C_k(t)}{B \log_2(1 + \frac{p''_k(t)|h_{m,k}|^2}{N_0})} \\ \text{s.t.} & \quad \frac{(1 - \alpha_k(t))C_k(t)p''_k(t)}{B \log_2(1 + \frac{p''_k(t)|h_{m,k}|^2}{N_0})} \leq E_{\text{max},k} - p'_k(t)t_k^l
\end{align*}
\] (10a)

Problem 9 is a convex optimization problem and can be efficiently solved by for example interior point algorithm etc.

In the third problem, \( \alpha_k(t) \) is solved with given \( p'_k(t) \) and \( p''_k(t) \). The corresponding optimization problem is given by

\[
\begin{align*}
\min_{\alpha_k(t)} & \quad \max \left\{ \frac{\alpha_k(t)C_k(t)}{B \log_2(1 + \frac{p'_k(t)|h_{m,k}|^2}{N_0})} \left( \frac{(1 - \alpha_k(t))^{3/2}C_k(t)^{3/2}}{\left( \frac{p'_k(t)}{p''_k(t)} \right)^{1/6}D_m^{1/2}E_{\text{max},m}^{1/2}} \right) \\
& \quad + \frac{(1 - \alpha_k(t))C_k(t)}{B \log_2(1 + \frac{p''_k(t)|h_{m,k}|^2}{N_0})}\right\} \\
\text{s.t.} & \quad \alpha_k(t) \in [0, 1]
\end{align*}
\] (11b)

For the min-max problem 11, by denoting \( f_1(\alpha_k(t)) = \frac{\alpha_k(t)C_k(t)}{\sqrt{p''_k(t)}} \), \( f_2(\alpha_k(t)) = \frac{(1 - \alpha_k(t))^{3/2}C_k(t)^{3/2}}{\kappa_m^{1/2}D_m^{1/2}E_{\text{max},m}^{1/2}} + \frac{(1 - \alpha_k(t))C_k(t)}{B \log_2(1 + \frac{p''_k(t)|h_{m,k}|^2}{N_0})} \) and \( f(\alpha_k(t)) = f_1(\alpha_k(t)) + f_2(\alpha_k(t)) \), it is known that the optimal \( \alpha_k(t) \), denoted by \( \alpha_k^*(t) \), occurs in the following three cases, that is, \( \alpha_k^*(t) = 0 \), \( \alpha_k^*(t) = 1 \) or \( f_1(\alpha_k^*(t)) = f_2(\alpha_k^*(t)) \). Note that
here the solution of the third case can be obtained by solving a cubic equation. The final solution is given as

$$\alpha^*_k(t) = \begin{cases} 
\alpha^1_k(t) & \text{if } f(\alpha^1_k(t)) \leq \{f(\alpha^2_k(t)), f(\alpha^3_k(t))\} \\
\alpha^2_k(t) & \text{if } f(\alpha^2_k(t)) \leq \{f(\alpha^1_k(t)), f(\alpha^3_k(t))\} \\
\alpha^3_k(t) & \text{if } f(\alpha^3_k(t)) \leq \{f(\alpha^1_k(t)), f(\alpha^2_k(t))\} 
\end{cases} \quad (12)$$

By alternating three subproblems with the solutions given in (9), (10) and (12) until convergence, we obtain the final solution.

2) Scenario 2: one MEC server serves two users

Assume that MEC server $m$ serves two users, e.g., user $k$ and user $k'$, then the optimization problem can be formulated as follows

$$\min \max \{t^o_{m,k}, t^o_{m,k'} + t^o_{m,k}, t^o_{m,k'} + t^e_{m,k'}\}$$

s.t. $p^l_k(t) + p^e_k(t) t^o_{m,k} \leq E_{\max,k}$

$p^l_{k'}(t) t^o_{m,k'} + p^e_{k'}(t) t^o_{m,k'} \leq E_{\max,k'}$

$p^e_{m,k}(t) t^e_{m,k} + p^e_{m,k'}(t) t^e_{m,k'} \leq E_{\max,m}$

The previously proposed iterative algorithm can still be applied here to solve $\alpha_i(t)$, $p_i(t)$, $p^c_i(t)$ and $p^c_{m,i}(t)$ with $i = \{k, k'\}$. Here the only difference lies in solving $p^c_{m,k}(t)$ and $p^c_{m,k'}(t)$. The corresponding optimization problem can be formulated as

$$\min \max \left\{ \frac{t^o_{m,k}}{D_m} + \frac{(1 - \alpha_k(t)) C_k(t)}{\sqrt[3]{\frac{p^c_{m,k}(t)}{D_m}}}, \frac{t^o_{m,k'}}{D_m} + \frac{(1 - \alpha_{k'}(t)) C_{k'}(t)}{\sqrt[3]{\frac{p^c_{m,k'}(t)}{D_m}}} \right\}$$

s.t. $\frac{(1 - \alpha_k(t)) C_k(t)}{\sqrt[3]{\frac{p^c_{m,k}(t)}{D_m}}} + \frac{(1 - \alpha_{k'}(t)) C_{k'}(t)}{\sqrt[3]{\frac{p^c_{m,k'}(t)}{D_m}}} \leq E_{\max,m}$

It is worth noting that the optimal solution must activate the constraints and
make the two terms within the objective function equal to each other. Therefore, the optimal $p_{m,k}(t)$ and $p'_{m,k'}(t)$ can be obtained by solving the following equations

\[
t^0_{m,k} + \frac{(1 - \alpha_k(t))C_k(t)}{\sqrt{\frac{P_{m,k}(t)}{\kappa_mD_m}}} = t^0_{m,k'} + \frac{(1 - \alpha_{k'}(t))C_{k'}(t)}{\sqrt{\frac{P'_{m,k'}(t)}{\kappa_mD_m}}}
\]

\[
\frac{(1 - \alpha_k(t))C_k(t)}{\sqrt{\frac{P_{m,k}(t)}{\kappa_mD_m}}} + \frac{(1 - \alpha_{k'}(t))C_{k'}(t)}{\sqrt{\frac{P'_{m,k'}(t)}{\kappa_mD_m}}} = E_{\text{max},m}
\]

Hence, under an action $a(t)$, the system reward can be obtained as

\[
r_t = -\max \left\{ t_k | k \in \mathcal{K}, a(t) \right\}.
\]

The structure of the DQN-based offloading algorithm is illustrated in Fig. 2 and the pseudocode is presented in Algorithm 1.

4. DDPG Based Offloading Design

Note that only the discrete actions can be handled by the DQN based offloading design, where the reward acquisition mainly depends on solving the
Algorithm 1 The DQN-based Offloading Algorithm

1: Initialize the experience replay buffer $B$;
2: Initialize action-value function $Q$ with random weights $\theta$;
3: Initialize target action-value function $Q'$ with random weights $\theta^- = \theta$;
4: for each episode $n = 1, 2, \cdots, N$ do
   5: Reset simulation parameters for the environment;
   6: Randomly generate an initial state $s_1$;
   7: for each time slot $t = 1, 2, \cdots, T$ do
      8: Generate an action $a_t = \mu(s_t|\theta^\mu) + \nabla \mu$ to determine which MEC server to connect to;
      9: Execute action $a_t$ and solving corresponding optimization to obtain reward $r_t$;
     10: Receive reward $r_t$ and observe the next state $s_{t+1}$;
     11: Store the tuple $(s_t, a_t, r_t, s_{t+1})$ into $B$;
     12: Sample a random mini-batch of $N$ transitions $(s_t, a_t, r_t, s_{t+1})$ from $B$;
     13: Perform gradient descent and update Q-network;
     14: Every $C$ steps reset $Q' = Q$;
   7: end for
5: end for

formulated optimization problems at MEC servers, which may increase extra computing burden at the MEC servers. In this section, we rely on the DDPG to design offloading policy considering that DDPG can deal with both the discrete and continuous value actions. Different from DQN, DDPG uses the Actor-Critic network to improve the accuracy of the model. In this section, we directly regard $a(t)$, $\alpha_k(t)$, $p_c(t)$, $p_k(t)$, and $p_{c,m,k}(t)$ as the output action instead of disassembling the problem into two parts.

**System State Space:** In the DDPG offloading paradigm, the system state space action is the same with the DQN based offloading paradigm, which is given by

$$S(t) = \{h(t-1), C(t)\}$$

with $h(t-1)$ and $C(t)$ defined in (4). As in the DQN offloading paradigm, the agent can only observe the delayed version of CSI due to channel estimation operations and feedback delay.

**System Action Space:** In DDPG offloading paradigm, we utilize the value of $p_{c,m,k}(t)$ to indicate the MEC server selection, that is, $p_{c,m,k}(t) = 0$ represents
that there is no partial task at user $k$ offloaded to the MEC server $m$. In other words, the MEC server $m$ is not chosen by user $k$. If $p_{m,k}^c(t)$ is not equal to 0, it means that the user $k$ decides to offload partial tasks to the MEC server $m$. Since the user can only connect to one MEC server at one time slot, only one $p_{m,k}^c(t)$ in any time slot is not 0, and the remaining ones are 0. The action space of DDPG offloading paradigm can be expressed as

$$a(t) = \{\alpha_k(t), p_k^l(t), p_k^o(t), p_{m,k}^c(t)\}, \forall k, m.$$ 

It is noted that here the continuous actions $p_k^l(t), p_k^o(t), p_{m,k}^c(t)$ can be obtained based on state $S(t)$ with delayed CSI $h(t-1)$.

**System Reward Function**: In the DDPG offloading algorithm, $\alpha_k(t)$, $p_k^l(t), p_k^o(t)$, and $p_{m,k}^c(t)$ can be obtained from a continuous action space. With the decisions, the agent tells each user $k$ the selected MEC server and delivers $p_k^l(t), p_k^o(t)$ to it to perform the offloading. Moreover, the agent needs to send $p_{m,k}^c(t)$ to each server for the computing resources allocation. After that, the reward is obtained as in (15) by collecting $t_k$ observed at the MEC servers or users.

Compared to the DQN based offloading paradigm, DDPG based offloading
Algorithm 2 The DDPG-based Offloading Algorithm

1: Randomly initialize the actor network $\mu_{\theta^\mu}$ and the critic network $Q_{\theta^Q}$ with weights $\theta^\mu$ and $\theta^Q$;
2: Initialize target network $\mu$ and $Q$ with weights $\theta^\mu' \leftarrow \theta^\mu$, $\theta^Q' \leftarrow \theta^Q$;
3: Initialize the experience replay buffer $B$;
4: for each episode $n = 1, 2, \cdots, N$ do
5: Reset simulation parameters for the environment;
6: Randomly generate an initial state $s_1$;
7: for each time slot $t = 1, 2, \cdots, T$ do
8: Select an action $a_t = \mu(s_t | \theta^\mu) + \nabla \mu$ to determine the power for transmission and computation;
9: Execute action $a_t$ and receive reward $r_t$ and observe the next state $s_{t+1}$;
10: Store the tuple $(s_t, a_t, r_t, s_{t+1})$ into $B$;
11: Sample a random mini-batch of $N$ transitions $(s_t, a_t, r_t, s_{t+1})$ from $B$;
12: Update the critic network by minimizing the loss $L$:
   $$L = \frac{1}{N} \sum_{t=1}^{N} \left( r_t + \max_{a \in A} Q(s_{t+1}, a | \theta^Q') - Q(s_t, a_t | \theta^Q) \right)^2 ;$$
13: Update the actor network by using the sampled policy gradient:
   $$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_{t=1}^{N} \nabla_a Q(s_t, a | \theta^Q) \bigg|_{a=a_t} \nabla \theta^\mu \mu(s_t | \theta^\mu);$$
14: Update the target networks by:
   $$\theta^\mu' \leftarrow \tau \theta^\mu + (1 - \tau) \theta^\mu';$$
   $$\theta^Q' \leftarrow \tau \theta^Q + (1 - \tau) \theta^Q';$$
15: end for
16: end for

paradigm does not need the MEC servers to solve the optimization problems, which can release the computation burden at the MEC servers. However, as DDPG algorithm is generally more complex than the DQN algorithm, the computation complexity unavoidably increases at the agent.

The structure of the DDPG-based offloading algorithm is illustrated in Fig. 3.

We provide the pseudocode in Algorithm 2.

5. Numerical Results

In this section, we present the numerical simulation results to illustrate the performance of the proposed two offloading paradigms. Assume that the time interval of the system is 1 ms, and the bandwidth of the system is 1 MHz. Additionally, the required CPU cycles per bit is 300 cycles/bit at the users and 120 cycles/bit at MEC servers. In the training process, the learning rate
of the DQN-based offloading algorithm is 0.01. In the DDPG-based offloading algorithm, the learning rate of the actor network is 0.001, and the learning rate of the critic network is 0.001.

In Fig. 4, we plot the training process of the DQN-based algorithm and the DDPG-based algorithm. At the beginning, the delay of the system is in an unstable state with large fluctuations, indicating that the agent is constantly exploring the environment randomly. After a period of learning, the delay decreases slowly, and the fluctuation range gradually gets smaller. After about 1200 iterations, the DDPG-based algorithm converges to a stable value and after about 1500 iterations, the DQN-based algorithm converges. At this time, the average reward of each episode no longer changes, and the training process is completed. We observe that the DDPG-based algorithm converges faster and can obtain a lower latency than the DQN-based algorithm. This indicates that the performance of the DDPG-based algorithm is better than that of the DQN-based algorithm for our offloading problem.

In Fig. 5, we illustrate the offloading delay as the function of the amount of tasks at users. Three benchmarks, namely “random scheme”, “Local com-

Figure 4: Dynamics of delay per iteration in the training process.
puting” and “MEC computing”, are chosen to compare the performance with the proposed two offloading paradigms. Here “random scheme” means that the computing resources are allocated in a random manner; “Local computing” and “MEC computing” means that the tasks are processed only at users and only at MEC servers, respectively. The curves in Fig. 5 show that as the amount of tasks grows, the required time delay increases correspondingly. As there is not much local computing capacity at users, the computation delay of “Local computing” is the largest. “MEC computing” performs better than “random scheme” when the task arrival rate is greater than 2.7 Mbps, which indicates that when the task arrival rate increases, task offloading to MEC servers can obtain a lower time delay. When the task arrival rate is greater than 4 Mbps, the offloading time delay of “MEC computing” is close to the DQN-based computation offloading algorithm, indicating that most tasks are offloaded to the MEC servers with large task sizes. In particular, both proposed DQN and DDPG offloading paradigms achieve better performance than other benchmarks, proving the effectiveness of the proposed methods. On the other hand, the DDPG-based computation offloading paradigm achieves lower computation delay than the DQN-based computation offloading paradigm, which further verifies the superiority of the DDPG algorithm in dealing with high-dimensional continuous action-state space problems.
Fig. 6 shows the impact of computing capabilities of users and MEC servers on the processing delay. We fix the local computing capability as a constant value and increase the computing capacity of the MEC server continuously, so the computation delay of “Local computing” is not affected by the ratio of computing capacity between user and MEC server. It can be again seen from the figure that under different computing capabilities, our proposed DQN and DDPG offloading paradigms can achieve better performance than other three benchmarks, and the performance of DDPG-based offloading paradigm is slightly better than DQN-based offloading paradigm. When the ratio of MEC server computing capacity to user computing capacity locates between 2 and 3, as the ratio increases, the processing speed of MEC server is faster than the user, and the user chooses to offload more tasks to the MEC server. Therefore, the computation delay is significantly reduced, and the computation delay of “MEC computing” is smaller than “random scheme”. When the ratio exceeds 3, it is observed that as the ratio increases, the processing speed of the MEC server is significantly higher than the users. At this time, the user prioritizes the task offloading, and the task processing delay is still decreasing but the downward trend slows down. The computation delay of “MEC computing” is lower than “random scheme” and closes to the DQN-based offloading paradigm which indicates that most or all tasks are offloaded to the MEC servers. The decrease in delay is mainly due to the increase in computing capacity of the MEC servers.

Fig. 7 illustrates the computation delay as the functions of the user energy. The curves show that “MEC computing” is not affected by the varying of user energy. However, “Local computing” highly depends on the user energy constraint, and the computation delay decreases significantly as the user energy increases. The increase in user energy indicates a fast local processing speed and high available transmission at the users, which can reduce the computation delay to a certain extent. When the user’s energy reaches a certain level, the computation delay gradually decreases which shows that within a certain range, the user’s energy constraint has a greater impact on the computation.
delay. When the user’s energy exceeds a certain range, it has a weaker impact on the computation delay. Again, the DQN-based and the DDPG-based offloading paradigms achieve better performance than other offloading methods under different user energy constraints, which indicates the effectiveness of the proposed computational offloading algorithms. What’s more, the performance of the DDPG-based offloading paradigm is slightly better than the DQN-based offloading paradigm.
6. Conclusion

In this paper, we studied the resource allocation scheme of multi-user mobile edge computing offloading based on deep reinforcement learning. Two different deep reinforcement learning algorithms, namely, DQN and DDPG, were investigated to solve the formulated offloading optimization problem. The effectiveness and convergence of the proposed algorithms were verified through simulation results. By comparing with different offloading schemes, we show that the proposed deep reinforcement learning based learning methods can effectively reduce task processing delay under different system parameters.

7. Acknowledgements

This work was supported by the National Science Foundation China under Grant 61771345 and Shanghai Science and Technology Innovation Action Plan Project No. 21220713100.

8. References

References

[1] M. Satyanarayanan, “Fundamental challenges in mobile computing,” in Proc. ACM Symp. Principles of Distributed Computing, pp. 1-7, 1996.

[2] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, “A survey of computation offloading for mobile systems,” Mobile Networks Applicat., vol. 18, no. 1, pp. 129-140, Feb. 2013.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the Internet of Things”. Proc. MCC Workshop Mobile Cloud Comput., Helsinki, Finland, 2012, pp. 13-16

[4] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, “A survey of computation offloading for mobile systems,” Mobile Networks Applicat., vol. 18, no. 1, pp. 129-140, Feb. 2013.
[5] Y. Mao, J. Zhang and K. B. Letaief, “Dynamic computation offloading for mobile-edge computing with energy harvesting devices,” IEEE Journal on Selected Areas in Communications, vol. 34, no. 12, pp. 3590-3605, Dec. 2016.

[6] S. Barbarossa, S. Sardellitti, and P. D. Lorenzo, “Communicating while computing: Distributed mobile cloud computing over 5G heterogeneous networks,” IEEE Signal Process. Mag., vol. 31, no. 6, pp. 45-55, Nov. 2014.

[7] W. Zhang et al., “Energy-optimal mobile cloud computing under stochastic wireless channel,” IEEE Trans. Wireless Commun., vol. 12, no. 9, pp. 4569-4581, Sep. 2013.

[8] Y. Zhang, H. Liu, L. Jiao, and X. Fu, “To offload or not to offload: An efficient code partition algorithm for mobile cloud computing,” Proc. 1st Int. Conf. Cloud Netw., Paris, France, 2012, pp. 80-86.

[9] S. E. Mahmoodi, R. N. Uma and K. P. Subbalakshmi, “Optimal joint scheduling and cloud offloading for mobile applications,” IEEE Transactions on Cloud Computing, vol. 7, no. 2, pp. 301-313, 1 April-June 2019.

[10] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu and W. Zhuang, “Learning-Based Computation Offloading for IoT Devices With Energy Harvesting,” IEEE Transactions on Vehicular Technology, vol. 68, no. 2, pp. 1930-1941, Feb. 2019.

[11] L. T. Tan and R. Q. Hu, “Mobility-Aware Edge Caching and Computing in Vehicle Networks: A Deep Reinforcement Learning,” IEEE Transactions on Vehicular Technology, vol. 67, no. 11, pp. 10190-10203, Nov. 2018.

[12] Z. Wei, B. Zhao, J. Su and X. Lu, “Dynamic Edge Computation Offloading for Internet of Things With Energy Harvesting: A Learning Method,” IEEE Internet of Things Journal, vol. 6, no. 3, pp. 4436-4447, June 2019.

[13] J. Zhang, J. Du, Y. Shen and J. Wang, “Dynamic Computation Offloading With Energy Harvesting Devices: A Hybrid-Decision-Based Deep Reinforce-
ment Learning Approach,” *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9303-9317, Oct. 2020.