Journal of Parallel and Distributed Computing 171 (2023) 28–39

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Online computation offloading with double reinforcement learning algorithm in mobile edge computing

Linbo Liao [a], Yongxuan Lai [a,b,*], Fan Yang [c], Wenhua Zeng [a]

[a] School of Informatics / Shenzhen Research Institute, Xiamen University, Xiamen/Shenzhen, China
[b] School of Mathematics and Information Engineering, Longyan University, Longyan, China
[c] Department of Automation, School of Aerospace Engineering, Xiamen University, Xiamen, China

ARTICLE INFO

Article history:
Received 31 August 2021
Received in revised form 27 June 2022
Accepted 7 September 2022
Available online 14 September 2022

Keywords:
Mobile edge computing
Power control
Computation offloading
Deep deterministic policy gradient
Double Deep Q-Networks

ABSTRACT

Smart mobile devices have recently emerged as a promising computing platform for computation tasks. However, the task performance is restricted by the computing power and battery capacity of mobile devices. Mobile edge computing, an extension of cloud computing, solves this problem well by providing computational support to mobile devices. In this paper, we discuss a mobile edge computing system with a server and multiple mobile devices that need to perform computation tasks with priorities. The limited resources of the mobile edge computing server and mobile device make it challenging to develop an offloading strategy to minimize both delay and energy consumption in the long term. To this end, an online algorithm is proposed, namely, the *double reinforcement learning computation offloading* (DRLCO) algorithm, which jointly decides the offloading decision, the CPU frequency, and transmit power for computation offloading. Concretely, we first formulate the power scheduling problem for mobile users to minimize energy consumption. Inspired by reinforcement learning, we solve the problem by presenting a power scheduling algorithm based on the deep deterministic policy gradient (DDPG). Then, we model the task offloading problem to minimize the delay of tasks and propose a double Deep Q-networks (DQN) based algorithm. In the decision-making process, we fully consider the influence of task queue information, channel state information, and task information. Moreover, we propose an adaptive prioritized experience replay algorithm to improve the model training efficiency. We conduct extensive simulations to verify the effectiveness of the scheme, and the simulation results show that compared with the conventional schemes, our method reduces the delay by 48% and the energy consumption by 53%.

1. Introduction

With the development of wireless communication and the Internet of Things (IoT), smart mobile devices (MDs) have become a new mobile computing platform on which applications such as video surveillance, face recognition, and natural language processing are widely deployed [36,14]. These pose strict requirements on the computational power of MDs, especially for computation-intensive applications. The contradiction between resource-constrained MDs and computation-intensive applications becomes the bottleneck when providing satisfactory quality of experience (QoE) [19].

As an effective method to solve the above-mentioned problem, recently various computation offloading schemes have been proposed, which migrate the computation tasks to other devices or platforms for execution [5]. The cloud computing systems, for example, would transfer all or part of the computation tasks to the cloud server to alleviate the heavy burden on the MDs. The main drawback of this cloud-based offloading approach is that it usually causes an unacceptable transmission delay as the cloud is usually far away from the clients. In contrast, mobile edge computing (MEC) deploys servers or micro-servers close to the MDs to reduce the transmission delay [24]. Therefore, MEC has become a promising computing paradigm for the various mobile applications [21].

There has been some research on the problem of computation offloading in MEC systems, most of which aim at enhancing the users' QoE. Viewed as a convex optimization problem, the computation offloading ratio, the processor clock rate, the bandwidth allocation, as well as the transmit power are taken into account for the optimization which minimizes the weighted sum of the

* Corresponding author at: School of Informatics / Shenzhen Research Institute, Xiamen University, Xiamen/Shenzhen, China.
E-mail addresses: lbliao@stu.xmu.edu.cn (L. Liao), laiyx@xmu.edu.cn (Y. Lai), yang@xmu.edu.cn (F. Yang), whzeng@xmu.edu.cn (W. Zeng).

https://doi.org/10.1016/j.jpdc.2022.09.006
0743-7315/

execution delay and the energy consumption for the computation offloading problem in [11]. To improve the delay performance, the general dependency among tasks is analyzed and a genetic algorithm-based solution was adopted in [2]. Recently, the computation offloading in UAV-assisted multi-user MEC scenarios is also discussed by [38].

In the previous works, however, all the tasks are assumed to be equally important, and they neglect the fact that the computation tasks are of different importance to users. For example, in MDs, security tasks (such as road detection, and vehicle detection) have the highest priority, followed by real-time tasks (such as games, AR / VR), and the lowest priority for non-real-time tasks (such as user behavior analysis task). Moreover, the queue waiting and execution delay on the MEC servers should also be considered when solving the optimal offloading problem. A high-performance computation offloading scheme that tails for the priority-based tasks and resource-constraint MEC servers is highly required.

The challenges of computation offloading in MEC systems lie in three folds: 1) The mobile network with edge servers is stochastic and dynamic. The task execution is affected by multiple factors, such as the channel state information (CSI), and the task queue state information. So the cost and performance of computation tasks change with the states and the execution modes of MDs and edge servers; 2) The energy consumption of MDs is constrained by the battery capacity. Hence the transmit power and CPU frequency of the MDs should be reasonably scheduled to save energy when offloading the computation tasks; 3) As the states evolve in a continuous space, one offloading decision will influence the latter one. Thus, the delay-energy tradeoff optimization problem is a long-term mixed-integer linear programming problem (LT-MILP), which has been proven to be NP-hard [20].

In this paper, we propose an efficient computation offloading scheme, called double reinforcement learning computation offloading (DRLCO), in resource-constraint MEC systems. Our scheme includes two aspects of optimization objectives: 1) To minimize the weighted sum of the execution delay and the energy consumption of computation tasks in the long-term; 2) To improve the users' QoE by reducing the delay of important tasks. Both the MDs and MEC servers are assumed resource-limited, and the MEC server can provide computational support for multi-MDs to handle tasks with priorities. Due to the stochastic and dynamic nature of the MEC system, we reformulate the computation offloading problem as a Markov Decision Process (MDP) problem and solve it by utilizing reinforcement learning techniques. The main contributions of this work are summarized as follows:

- We consider a scenario where both the MEC server and MDs are resource-constraint. And we develop a computation offloading model, including the offloading controller, the task execution queue on the MEC server and MD, and the task transmission queue between the MEC server and MD. Besides, different types of tasks have different execution priorities in the MEC system.
- In this paper, the optimization goal of computation offloading is defined as reducing task execution delay and mobile device energy consumption, and then a *double reinforcement learning computation offloading* algorithm is proposed. Specifically, we decompose the computation offloading process into power scheduling process and task offloading process. In the process of power scheduling, we propose a Deep Deterministic Policy Gradient (DDPG) based approach to reduce the energy consumption of MDs by scheduling the transmit power and CPU frequency of the MDS. In the process of task offloading, we propose a double Deep Q Network (DQN) based approach to reduce the execution delay of tasks by making offloading decisions for computing tasks. And an adaptive prioritized experience replay algorithm is proposed to improve the model training efficiency. Besides, the task queues are arranged and sorted according to their priorities to reduce the waiting delay and improve the users' QoE.
- We conduct extensive experiments to evaluate the performance of the proposed algorithms. The simulation results verify that our approach outperforms other state-of-the-art schemes. It reduces the delay by 48% and the energy consumption by 53% compared with other schemes.

The rest of this paper is organized as follows. Section 2 discusses the related works. Section 3 presents the system model. Section 4 formulates the cost minimization problem and reformulates it as an MDP problem. Section 5 presents the detailed Double DQN based approach and DDPG based approach, which obtains the optimal tasks offloading policy and schedules the transmit power and CPU frequency of the MDs. Section 6 evaluates the performance of the proposed method based on extensive simulations. Finally, section 7 concludes the paper and gives some future directions.

2. Related work

Existing research on computation offloading in MEC systems can be roughly classified into three categories according to the optimization objectives, i.e., the delay-optimal computation offloading, the energy-optimal computation offloading, and the energy-delay tradeoff computation offloading.

For delay-sensitive applications, improving the delay performance is the main objective of computation offloading. Liu et al. [23] proposed a one-dimensional search algorithm to minimize the total delay. Meng et al. [27] modeled the computation offloading problem as an infinite level average cost MDP and derived a closed-form multi-level water-filling computation offloading policy to minimize the average delay in long term. Qian et al. [29] aimed to reduce the transmission delay by improving the spectrum utilization with the non-orthogonal multiple access (NOMA) technology. Besides, Apostolopoulos et al. [4] proposed an offloading strategy based on non-cooperative game theory to achieve delay-optimal in the multi-MDs scenario. Chen et al. [9] applied reinforcement learning technology to the process of computing unloading and proposed an Advanced DQN algorithm. The algorithm improves the original DQN by adding a priority buffer mechanism and an expert buffer mechanism.

For the energy-optimal computation offloading problem, Kamoun et al. [18] proposed an offloading strategy based on the constrained MDP to minimize the energy consumption under a delay constraint. Wang et al. [35] derived a convex optimization-based method by considering time-varying channels to reduce the energy consumption of transmissions. MDs have strict requirements for energy consumption, due to the limitation of battery capacity. Hence the energy minimization problem in the scenario of multi-MDs is also investigated [39,37]. Zhou et al. [39] proposed a distributed solution based on the consensus alternating direction method of multipliers (ADMM), in which the energy minimization problem is decomposed into a bunch of subproblems distributed on MDs and solved in parallel. Yao et al. [37] proposed an optimal total energy consumption algorithm (OTCA) based on bipartite matching to reduce the system energy consumption, and an optimal energy consumption assignment algorithm (OECAA) to reduce the energy consumption of MDs. Huang et al. [16] proposed an edge computing offload model for wireless charging, in which the MDs use the energy obtained by wireless charging to perform computation tasks and computation offloading. And the author proposes a Deep Reinforcement learning-based Online Offloading

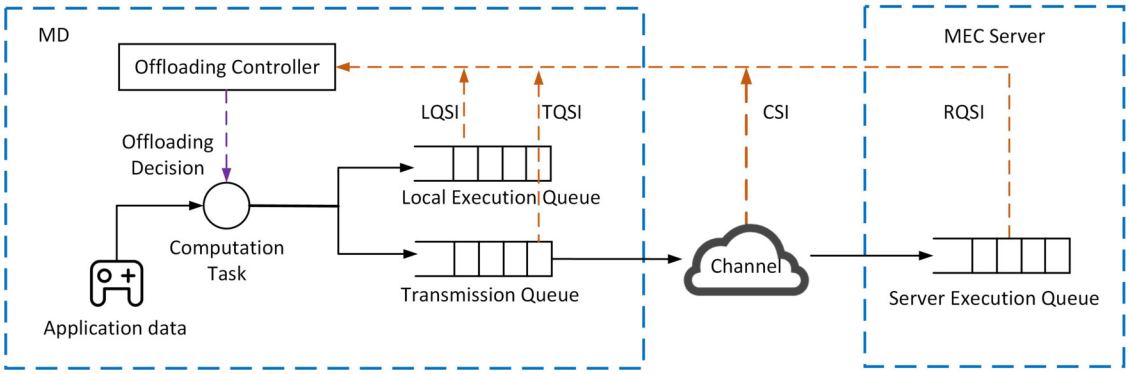


Fig. 1. The architecture of computation offloading model.

(DROO) framework, which uses the experience replay mechanism to learn binary offloading decisions.

Furthermore, some works [25,15,13,10,12] take the energy consumption and delay into consideration at the same time and stride a tradeoff between them. Mao et al. [25] adopted a Lyapunov optimization-based offloading algorithm to minimize the total cost which consists of delay and energy consumption. To minimize both the delay and energy consumption, Dinh et al. [13] proposed a linear relaxation-based approach and a semidefinite relaxation (SDR)-based approach to jointly optimize the offloading decision and the CPU frequency of MD. Chen et al. [10] solved the joint optimization problem in the multi-MDs scenarios by non-cooperative game theory. In the same case, Ding et al. [12] proposed a decentralized computation offloading algorithm by jointly deciding the offloading mode, CPU frequency, and transmit power. Alfakih et al. [3] proposed a SARSA-based computation offload algorithm to minimize energy consumption and computing delay. Song et al. [33] defined the computation offloading process as a multi-objective Markov decision process, and proposed a multi-objective reinforcement learning (MORL) algorithm based on DQN to reduce energy consumption and computation delay.

Most of the above-mentioned delay-energy optimal schemes do not consider the relationships between the types of tasks and users' QoE. Different from existing research, in this paper we focus on the computation offloading of tasks with different priorities in the MEC systems. To minimize the delay and energy consumption, an algorithm based on reinforcement learning is proposed for the computation offloading.

3. System model

In this section, we first introduce the system model studied in this paper, and then elaborate on the computation model and the tasks queue model.

3.1. MEC system model

In [23,27,1], different MEC systems are proposed for different edge scenarios. In the paper, we consider computation offloading problems in a confined edge network/space (e.g. an enterprise, a campus, a home), and conduct a MEC system with one MEC server that can provide computational support for multi-MDs. As shown in Fig. 1, the MD can execute its computation tasks in two different approaches: 1) executes locally at the MD, and 2) offloads the computation to the MEC server via wireless transmissions.

In our computation offloading model, the MD contains three parts: an offloading controller, a transmission queue, and a local execution queue. The offloading controller is in charge of making the offloading decision for the computation task; the execution and transmission queues set the orders of tasks when executing or transmitting orders. And the MEC server executes the tasks received from the MDs. Time is slotted with duration τ, and the time slot index set is $T = \{0, 1, ..., T\}$. For ease of reference, we list the key notations of our system model in Table 1.

Table 1
Main notations in this paper.

Notation	Description
T	the time slot set
$m(b, s, c)$	computation task with priority b, data size s, and complexity c
I_t^i	offloading indicator of task i at time t
f_l^t	CPU frequency of the MD at time t
$D_l^{t,i}$	local execution delay of task i at time t
$E_l^{t,i}$	execution energy consumption of task i at time t
r^t	transmission rate at time t
g^t	the channel gain between the MD and the MEC server at time t
p_{tx}^t	the transmit power of the MD at time t
D_{tx}^i	transmission delay of task i
E_{tx}^i	transmission energy consumption of task i
f_s	CPU frequency of the MEC server
D_s^i	execution delay at MEC server of task i
q_l^t, q_{tx}^t, q_s^t	state information of local queue, transmit queue, and server queue
D_w^i	waiting delay of task i
$cost_t^i$	execution cost of task i at time t
S_p, A_p, r_p	state space, action space, and reward function of the MDP model for power scheduling problem
S_o, A_o, r_o	state space, action space, and reward function of the MDP model for task offloading problem
$\mu_\psi(s_p)$	the policy function of actor model of power scheduling algorithm
$Q_\theta(s_p, a_p)$	the Q function of critic model of power scheduling algorithm
$Q_\xi(s_o, a_o)$	the Q function of critic model of task offloading algorithm
Ws_i	the weight of sample i
P_i	the sampling probability of sample i

3.2. Computation model

Without any loss of generality, the computation task is denoted as a tuple $m(b, s, c)$, where b is the priority of the task, s (in bits) is the data size of the task, and c is the number of CPU cycles required to process one bit of task. The offloading indicator of the task is denoted by $I \in \{0, 1\}$, where $I = 0$ indicates the local execution approach and $I = 1$ stands for the offloading approach.

3.2.1. Local execution model

With the dynamic voltage frequency scaling (DVFS) techniques [30], MDs can dynamically adjust their CPU frequency based on

demands, which affects the delay and energy consumption of the task [25]. Specifically, the CPU frequency of the MD is denoted by f_l^t, and the local execution delay of task i: $m^i(b^i, s^i, c^i)$ is denoted by:

$$D_l^{t,i} = \frac{w^i}{f_l^t}, \tag{1}$$

where $w^i = s^i c^i$ presents the number of CPU cycles required to execute the task, and f_l^t is constrained by f_{max}, i.e., $f_l^t \leq f_{max}$, $t \in T$. The energy consumption of task executing at the MD is denoted by:

$$E_l^{t,i} = k(f_l^t)^2 D_l^{t,i}, \tag{2}$$

where k is the effective switched capacitance that depends on the chip architecture [8].

3.2.2. Task offloading model

The task offloading process is divided into two phases: the transmission phase and the execution phase.

In the transmission phase, the task is transmitted from the MD to the MEC server. According to the Shannon formula, the transmission rate at time slot t is:

$$r^t = Blog_2(1 + \frac{p_{tx}^t g^t}{N_0}), \tag{3}$$

where B is the bandwidth, N_0 is the noise power at the receiver, and p_{tx}^t is the antenna transmit power of the MD at time slot t which is constrained by p_{max}, i.e., $p_{tx}^t \leq p_{max}, t \in T$. Besides, the channel gain g^t between the MD and the MEC server at the time slot t can be expressed as:

$$g^t = G_0(\frac{d_0}{d^t})^{\gamma}, \tag{4}$$

where d^t is the distance between the MD and the MEC server, which is changing as time goes by, d_0 is the antenna far-field test distance of the base station to which the MEC server is connected, G_0 is the channel gain constant, and γ is the path-loss exponent.

Similar to other works [23,25,27], the transmission delay of sending the results back to the MD from the MEC server is assumed to be negligible. Therefore, the transmission delay of task i is related to the data size and uplink rate of each time slot. For simplicity, we assume the transmission rate is fixed during the transmission of the task. Thus, the transmission delay can be expressed as:

$$D_{tx}^i = \frac{s^i}{r^t}. \tag{5}$$

The energy consumption is given by

$$E_{tx}^i = p_{tx}^t D_{tx}^i. \tag{6}$$

In the execution phase, the task is executed at the MEC server. Similar to other works, the CPU frequency of the MEC server f_s is assumed to be fixed. Hence the execution delay is:

$$D_s^i = \frac{w^i}{f_s}. \tag{7}$$

3.3. Tasks queue model

For resource-constrained MEC systems, the queue status information (QSI) is a significant factor affecting the performance of offloading decisions. Let q_l^t, q_{tx}^t, q_s^t represent the LQSI and TQSI at the MD, and RQSI at the MEC server, respectively. Assume that the number of tasks generated in each time slot is random and obeys the i.i.d. $\chi(t)$ with $\mathbb{E}[\chi] = \bar{\chi}$, where $\bar{\chi}$ is the average generating rate of tasks. Thus, the number of computation tasks generated in each time slot can be expressed as $\chi(t)\tau$, τ is the duration of a time slot.

We also use different attributes to represent the state of different queues, i.e., the state of the execution queue is represented by the workload of the tasks and that of the transmission queue is represented by the data size. Hence, the dynamics of the LQSI are given by:

$$q_l^{t+1} = q_l^t + \sum_{i=1}^{\chi(t)\tau} (1 - I) w_i - f_l^t \tau. \tag{8}$$

And the dynamics of the TQSI is given by:

$$q_{tx}^{t+1} = q_{tx}^t + \sum_{i=1}^{\chi(t)\tau} I s_i - r^t \tau. \tag{9}$$

The dynamics of the SQSI is given by:

$$q_s^{t+1} = q_s^t + \sum_{i=1}^{n} w_i - f_s \tau, \tag{10}$$

where n is the number of tasks received by the MEC server in time slot t.

The waiting delay of tasks in the queue is not negligible for the MEC systems. The waiting delay of task i can be calculated by summing the execution or transmission delay of previous tasks in the queue. In the local execution queue, the task waiting delay can be expressed as:

$$D_{w,l}^i = \sum_{j=0}^{i-1} D_l^j, \tag{11}$$

where D_l^j is the execution delay of task j at MDs. Similar to the local execution queue, the waiting delay of tasks in the transmission queue $D_{w,tx}^i$ and the server execution queue $D_{w,s}^i$ is obtained by summing the delays of previous tasks.

Therefore, the total execution delay of task i can be expressed as:

$$D_{tot}^i = \begin{cases} D_{w,l}^i + D_l^{t,i}, & I = 0 \\ D_{w,tx}^i + D_{w,s}^i + D_{tx}^{t,i} + D_s^i. & I = 1 \end{cases} \tag{12}$$

Besides, the execution energy consumption of task i is defined as $E^{t,i} = E_l^{t,i}$ when $I = 0$, and $E^{t,i} = E_{tx}^{t,i}$ when $I = 1$.

4. Problem formulation

In this section, we first introduce the execution cost of a task and formulate the execution cost minimization (ECM) problem. Then, we define the power scheduling and task offloading problems based on the MDP model [7].

4.1. Execution cost minimization problem

Execution delay and energy consumption are two key factors for users' QoE, which are adapted for optimizing the computation offloading policy in the MEC systems. Considering the priorities of tasks, the execution cost of task i can be expressed as:

$$cost_t^i = \lambda b D_{tot}^i + (1 - \lambda) E^{t,i} \tag{13}$$

where $\lambda \in [0, 1]$ is a parameter used to stride a tradeoff between the two objectives.

Our goal is to minimize the execution cost of tasks by jointly calculating the CPU frequency, transmission power, and task execution mode. Given a time span of T, the ECM problem in the long-term can be formulated as:

$$\begin{aligned} \min \quad & \lim_{T\to+\infty} \frac{1}{T}\sum_{t=0}^{T}\frac{1}{\chi(t)\tau}\sum_{i=0}^{\chi(t)\tau} \boldsymbol{cost_t^i} \\ \text{s.t.} \quad & 0 \le p_{tx}^t \le p_{max}, \quad t \in T, \\ & 0 \le f_{mobile}^t \le f_{max}, \quad t \in T, \end{aligned} \tag{14}$$

where $cost_t^i$, consists of task execution delay and energy consumption of the MD, is the task execution cost defined at Eq. (13).

The ECM problem is a multi-objective optimization problem with coupled constraints, and the goal of the ECM problem is to minimize the execution cost by scheduling CPU frequency, transmission power, and task offloading mode. We decouple the ECM problem by dividing the computation offloading process into two processes: 1) schedule the CPU frequency and transmission power at the beginning of each time slot to adjust the length of task queues; 2) In each time slot, the offloading decision is made for each computation task to reduce the execution cost of the task. Therefore, the computation offloading problem can be decomposed into power scheduling problem and task offloading problem. These problems are further reformulated as MDP problems that have advantages over sequential decision-making problems.

A typical MDP model is defined as a tuple $\langle S, A, r, T\rangle$, which consists of a set of states S, a set of actions A, a reward function r, and a transition function T. In the following, we define the corresponding components for power scheduling and task offloading problems, respectively.

4.2. Power scheduling problem

At the beginning of each time slot of the offloading process, the MD needs to set the CPU frequency and transmission power to reduce the length of the task queues and the energy consumption.

4.2.1. State space

In this problem, we consider each state of state space contains two parts: the queue state information (QSI) and the channel state information (CSI). Specifically, we denote the state at the beginning of time slot t as a vector, i.e., $s_p^t = (q^t, g^t) \in S_p$, in which $q^t = (q_l^t, q_{tx}^t, q_s^t)$.

4.2.2. Action space

The MD needs to choose the CPU frequency and transmit power according to the current state s_p^t. And the action at time slot t can be denoted as $a_p^t = (f_l^t, p_{tx}^t) \in A_p$, and the feasible action space is $A = [0, f_{max}] \times [0, p_{max}]$.

4.2.3. Reward function

After taking action a_p^t, the MD can obtain the reward from the environment. In the power scheduling process, to minimize both queues length and energy consumption, the reward r_p^t is defined as:

$$r_p^t = -(w_1 q^{t+1} + w_2 E^t), \tag{15}$$

where E^t is the total delay of the MD in time t, w_1, w_2 denote the weights of the two objectives in the combined reward function.

4.2.4. Transition

According to the equations (8), (9), and (10), the state transition probability is related to the mode of task execution and environment in time slot t.

According to the MDP model given above, the power scheduling problem can be reformulated as a problem of finding the optimal policy π_p that maximizes the cumulative reward in the long-term. Specifically, the definition of the reformulated problem is given as:

$$\max_{\pi_p} \sum_{t=0}^{T} \zeta_p^{t-1} r_p^t, \tag{16}$$

where $\zeta_p \in [0, 1]$ is a discount factor that indicates the impact of long-term rewards on the current decision making.

4.3. Task offloading problem

For each computation task generated in any time slot, the mobile device needs to select an appropriate execution mode for them to reduce the task execution cost.

4.3.1. State space

In the task offloading problem, the state refers to the environment information when generating a task. Each state in the state space contains four kinds of information, including the CSI, the CPU frequency of the MD, the QSI, and the task information. Specifically, we denote the state as $s_o^i = [q^i, f_l^i, g^i, m_i(b_i, s_i, c_i)]$.

4.3.2. Action space

The action represents the mode of task execution, and can be expressed as $a_o^i \in \{0, 1\}$, where $a_o^i = 0$ indicates local execution and $a_o^i = 1$ stands for computation offloading.

4.3.3. Reward function

To minimize the weighted sum of the execution delays and energy consumption, the reward function can be defined as $r_o^i = -cost^i$, where $cost^i$ is defined at eq. (13).

4.3.4. Transition

Different from the power scheduling problem, the environment state changes after each task execution. Concretely, the CSI and tasks vary randomly according to Gaussian distribution and QSI varies according to equations (8), (9), and (10).

Hence, the task offloading problem can be reformulated as:

$$\max_{\pi_o, \pi_p} \sum_{t=0}^{T}\sum_{i=0}^{\chi(t)\tau} \zeta_o^{i-1} r_o^i, \tag{17}$$

where π_o is the task offloading policy, and $\zeta_o \in [0, 1]$ is the discount factor.

5. Reinforcement learning for computation offloading

This section presents a reinforcement learning-based approach for the computation offloading in the MEC system. We first briefly introduce and analyze existing reinforcement learning methods, then we describe the DDPG algorithm which solves the power scheduling problem [22], and the Double DQN algorithm that solves the task offloading problem in detail [34].

5.1. Basic idea of reinforcement learning

Reinforcement learning (RL) is an important branch of machine learning that can develop an online policy for smart agents to maximize expected cumulative rewards by interacting with the environment. The basic idea of RL is introduced in [17], and there are

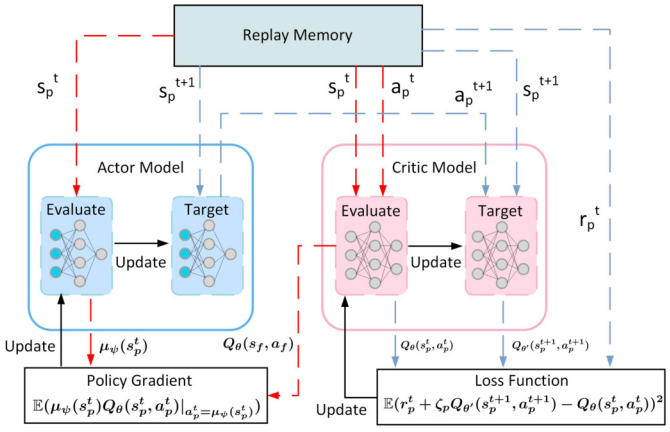


Fig. 2. Architecture of the DDPG consists of actor model, critic model, and replay memory [22].

three major types of RL algorithms: 1) critic-model (value-based approach); 2) actor-model (policy-based approach); 3) actor-critic learning approach. The critic model-based RL algorithm makes decisions by calculating and comparing the value of each action. The actor model-based RL algorithm learns a stochastic policy to choose the action, which is suitable for continuous action space. And the actor-critical approach utilizes value functions to conduct agents to make decisions in a continuous action space by combining the above two methods.

To reduce the length of task queues and energy consumption, we develop a power scheduling algorithm based on the Deep Deterministic Policy Gradient to obtain CPU frequency and transmission power at the beginning of each time slot. Then, we present a Double DQN-based algorithm to make offloading decisions for the computation tasks generated in each time slot to reduce the task execution cost. We will introduce our RL algorithm from model architecture, decision making, and network training.

5.2. The DDPG based power scheduling algorithm

5.2.1. The architecture of DDPG

DDPG is a reinforcement learning algorithm of an actor-critic model, which holds good performance in continuous action spaces. The architecture of DDPG is shown in Fig. 2, which consists of three modules: the actor model, the critic model, and the replay memory. In the actor model and the critic model, there are two neural networks, called evaluate network and target network, with the same structure (5 fully connected layers) but different parameters. We denote $\mu_\psi(s_p), \mu_{\psi'}(s_p)$ as the policy function of two networks in actor model, and $Q_\theta(s_p, a_p), Q_{\theta'}(s_p, a_p)$ as the Q-function for critic model.

5.2.2. Decision making

The actor-evaluate network is responsible for interacting with the environment and outputting action $a_p^t = \mu_\psi(s_p^t)$ according to the states. To avoid local optimum, Gaussian noise is added to the output of the actor-evaluate network. After taking the action a_p^t, the environment changes to the next state s_p^{t+1} and gives rewards r_p^t according to equation (15). Besides, the interactive experience $(s_p^t, a_p^t, r_p^t, s_p^{t+1})$ is stored in the replay memory for training the model.

5.2.3. Network training

In each training episode, a batch of experiences is randomly sampled, which can reduce the correlation of experiences and accelerate the network convergence [34]. We take one experience $(s_p^t, a_p^t, r_p^t, s_p^{t+1})$ as an example to introduce the training procedure as follows.

To improve the performance of the policy function, the actor-evaluate network is updated with the support of the critic-evaluate network. Concretely, s_p^t and a_p^t are input into critic-evaluate network to obtain the estimated action-value of a_p^t (denote as $Q_\theta(s_p^t, a_p^t)$). According to Silver et al. [32], the actor-evaluate network can be updated by minimizing the loss function:

$$L\psi = \mathbb{E}(\mu_\psi(s_p^t) Q_\theta(s_p^t, a_p^t)|_{a_p^t=\mu_\psi(s_p^t)}). \tag{18}$$

Then the critic-evaluate network is trained to obtain accurate estimated action-value. First, s_p^{t+1} is input into the actor-target network and critic-target network to obtain Q-value of state s_p^{t+1} (denoted as $Q_{\theta'}(s_p^{t+1}, a_p^{t+1})$), where $a_p^{t+1} = \mu_{\psi'}(s_p^{t+1})$. Then, according to the Bellman equation [6], the critic-evaluate network can be updated by minimizing the loss function:

$$L_\theta = \mathbb{E}(r_p^t + \zeta_p Q_{\theta'}(s_p^{t+1}, a_p^{t+1}) - Q_\theta(s_p^t, a_p^t))^2. \tag{19}$$

Moreover, the parameters of the target network are fixed while training evaluate network, which is helpful to the convergence of the evaluate network. We update the parameters of the target network by copying the parameters of the evaluate network. The detailed description of the network training of the DDPG-based power scheduling algorithm is given in Algorithm 1.

Algorithm 1 Network training of the DDPG based power scheduling algorithm.

Input:
The LQSI q_l, the TQSI q_{tx}, the SQSI q_s, and channel gain g.
Output:
The policy π_p.
Initialize the networks of actor and critic model.
for e ← 0 to $MAX\_EPISODE$ **do**
  Sample a batch of transitions from the replay memory according to Adaptive Prioritized Experience Replay Algorithm;
  for t ← 0 to $BATCH\_SIZE$ **do**
    Obtain experience $(s_p^t, a_p^t, r_p^t, s_p^{t+1})$;
    Given s_p^t, a_p^t, the critic-evaluate network outputs $Q_\theta(s_p^t, a_p^t)$;
    Update the actor-evaluate network according to Eq. (18);
    Given s_p^{t+1}, the actor-target network outputs a_p^{t+1};
    Given s_p^{t+1}, a_p^{t+1}, the critic-target network outputs $Q_{\theta'}(s_p^{t+1}, a_p^{t+1})$;
    Update the critic-evaluate network according to Eq. (19);
    if e >$UPDATE\_EPISODE$ **then**
      Copy the parameters of evaluate network to target network;
    end if
  end for
end for

5.3. The double DQN based offload decision algorithm

5.3.1. The architecture of double DQN

Double DQN is a critic model-based RL algorithm, which is suitable for discrete action prediction. The architecture of Double DQN is shown in Fig. 3, which consists of three modules, including evaluate network, target network, and replay memory. The evaluate network and the target network are neural networks with the same structure (5 fully connected layers) but different parameters. We denote $Q_\xi(s_o, a_o), Q_{\xi'}(s_o, a_o)$ as Q-function for the two networks.

5.3.2. Decision making

In Double DQN, the evaluate network is in charge of decision making. When the computation task is generated, the state s_o^i is input into the evaluation network to obtain the offloading decision a_o^i by

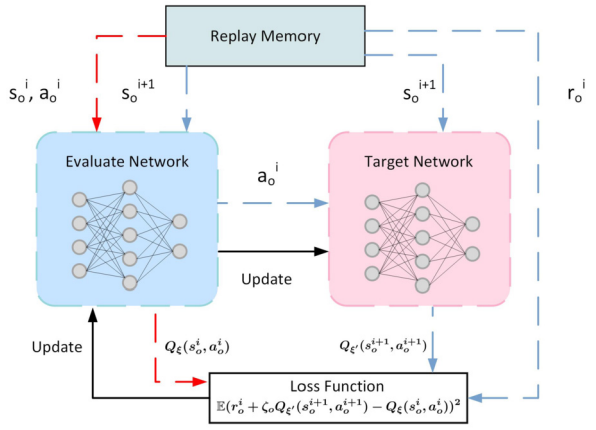


Fig. 3. The architecture of Double DQN consists of evaluate network, target network, and replay memory [34].

$$a_o^i = \arg\max_a(Q_\xi(s_o^i, a)). \tag{20}$$

After taking action a_o^i, the system changes to the next state s_o^{i+1} and gives rewards r_o^i. Also, the replay memory model is constructed to store a series of historical experiences, i.e. $(s_o^i, a_o^i, r_o^i, s_o^{i+1})$.

5.3.3. Network training

Here we take one experience $(s_o^i, a_o^i, r_o^i, s_o^{i+1})$ as an example to describe the training procedure. First, s_o^i and s_o^{i+1} are input into the evaluate network to obtain the estimated action-value of a_o^i (denote as $Q_\theta(s_o^i, a_o^i)$) and the action a_o^{i+1} with maximum Q-value under s_o^{i+1} state. The a_o^{i+1} can be obtained by:

$$a_o^{i+1} = \arg\max_a(Q_\xi(s_o^{i+1}, a)). \tag{21}$$

Then, we leverage the target network to obtain the Q-value of state s_o^{i+1} : $Q_{\xi'}(s_o^{i+1}, a_o^{i+1})$. According to the Bellman equation [6], the parameters of evaluate network can be updated by minimizing the loss function:

$$L_\xi = \mathbb{E}(r_o^i + \zeta_o Q_{\xi'}(s_o^{i+1}, a_o^{i+1}) - Q_\xi(s_o^i, a_o^i))^2. \tag{22}$$

Similar to the DDPG algorithm, the parameters of the target network are fixed when training the evaluate network and updated by copying the parameters of the evaluate network. The detailed description of the network training of the Double DQN-based task offloading algorithm is given in Algorithm 2.

5.4. The adaptive prioritized experience replay algorithm

In the DDPG and double DQN algorithms, agents sample uniformly from the replay buffer which can not only meet the assumption of the independent distribution of samples but also accelerate convergence. However, uniform sampling ignores the differences in importance between samples. To improve the model training efficiency, we propose an adaptive prioritized experience replay algorithm. Concretely, we first assign a weight to each experience and then calculate its sampling probability based on the weights.

We define the weight function Ws to represent the importance of samples. Specifically, the weight function is mainly composed of two parts: the reward weight of samples Wr and the sampling frequency Fs.

The reward weight of samples Wr can be denoted as follows:

$$Wr_i = |TD_i| \times R_i + \alpha, \tag{23}$$

Algorithm 2 Network training of the Double DQN based offload decision algorithm.

Input:
The LQSI q_l, the TQSI q_{tx}, the SQSI q_s, the CPU frequency f_l, the transmit rate r, and task information $m(b, s, c)$.
Output:
Task offloading policy π_o.
Initialize the evaluate networks and critic networks.
for e ← 0 to $MAX\_EPISODE$ **do**
  Sample a batch of transitions from the replay memory according to Adaptive Prioritized Experience Replay Algorithm;
  for t ← 0 to $BATCH\_SIZE$ **do**
    Given s_o^{i+1}, the evaluate network outputs $a_o^{i+1} = \arg\max_a(Q_\xi(s_o^{i+1}, a))$;
    Given s_o^i, a_o^i, the evaluate network outputs $Q_\xi(s_o^i, a_o^i)$;
    Given s_o^{i+1}, a_o^{i+1}, the target network outputs $Q_{\xi'}(s_o^{i+1}, a_o^{i+1})$;
    Update the evaluate network according to Eq. (22);
    if e >$UPDATE\_EPISODE$ **then**
      Copy the parameters of evaluate network to target network;
    end if
  end for
end for

where TD_i denotes temporal difference error of sample i, which is represented by formula $r_p^t + \zeta_p Q_{\theta'}(s_p^{t+1}, a_p^{t+1}) - Q_\theta(s_p^t, a_p^t)$ in power scheduling algorithm and by formula $r_p^t + \zeta_p Q_{\theta'}(s_p^{t+1}, a_p^{t+1}) - Q_\theta(s_p^t, a_p^t)$ in offload decision algorithm. We set α to a small positive number to avoid sampling failure when TD error equals 0. To facilitate calculation, the rewards r_i are normalized to [-1, 1], and the weight of reward R can be expressed as:

$$R_i = \log(|r_i| + 1) + 1. \tag{24}$$

To prevent the weight of some samples from being too high, we add the sampling frequency to the weight function, which is denoted as:

$$Fs_i = e^{-(num_i)^x}, \tag{25}$$

where num_i represents the sampling times of samples i and x is a constant. Then, the weight function Ws can be denoted as:

$$Ws_i = Wr_i + \kappa Fs_i, P \tag{26}$$

where κ is a hyper-parameter used to stride a tradeoff between the two functions. Using the above weight function, we can calculate the sampling probability of the sample by:

$$P_i = \frac{(Ws_i)^\beta}{\sum_i (Ws_i)^\beta} \tag{27}$$

where $\beta \in \{0, 1\}$ is a random sampling factor. Concretely, the replay buffer adopts uniform sampling when $\beta = 0$ and priority sampling when $\beta = 1$. In addition, to reduce the sampling complexity, a more effective update and sampling implementation, sum tree, is adopted in this algorithm [31]. The detailed description of the adaptive prioritized experience replay is given in Algorithm 3.

Algorithm 3 Adaptive prioritized experience replay algorithm.

Input:
Batch size of replay memory K.
Output:
K training samples.
Initialize the replay memory according to equations (26).
for i ← 0 to K **do**
  Sample a transition from the replay memory according to equations (27) based on sum tree;
  Calculate TD error of transition;
  Update sample weight equations (26);
end for

Algorithm 4 The Double RL computation offloading policy.

Initialize parameters for DDPG and Double DQN.
while $t \in T$ **do**
  Observe state s_f^t from environment.
  Chose CPU frequency and transmit power a_f^t according to DDPG.
  while $i \in \chi(t)\tau$ **do**
    Observe state s_o^i from environment.
    Make offloading decision a_o^i according to Double DQN.
    Sort task queue, execute task, and transmit task.
    Obtain reward r_o^i and transfer state into s_o^{i+1}.
    Store $(s_o^i, a_o^i, r_o^i, s_o^{i+1})$ for training Double DQN.
  end while
  Obtain reward r_f^t and transfer state into s_f^{t+1}.
  Store $(s_f^t, a_f^t, r_f^t, s_f^{t+1})$ for training DDPG.
  if t > $BATCH\_SIZE$ **then**
    Train DDPG model.
    Train Double DQN model.
  end if
end while

5.5. The queue sorting algorithm

Considering the difference in the importance of tasks, we develop an algorithm to enhance the users' QoE by sorting the execution order of tasks in the queues. The sorting algorithm utilizes four attributes of tasks, including the workload, data size, priority, and the waiting delay.

$M = (m_{ij})_{n\times 4}$, where m_{ij} is the j-th attribute of the i-th task, denotes the attributes of the tasks. By normalizing each column, the matrix M is transformed into $U = (u_{ij})_{n\times 4}$. Then, the weight matrix of the tasks can be expressed as:

$$W = UK^{\mathrm{T}}, \tag{28}$$

where K is the weight coefficient vector of task attributes, and it has different values for different queues. Specifically, we prioritize the data size in the transmission queue and the workload in the execution queue.

Those algorithms above are combined to achieve the optimal delay-energy computation offloading for the considered MEC system. The whole procedure of DRLCO is presented in Algorithm 4. In DRLCO, for any time slot t in the computation offloading process, the MD obtains the state information s_p^t at the beginning of the time slot. Then s_p^t is input into Algorithm 1 to obtain the CPU frequency and server transmission power. For any task i generated in time slot t, the MD obtains the environment information s_o^i and sends the task to the execution queue or transmission queue according to Algorithm 2. Then, the task queues are sorted by the queue sorting algorithm when a task enters the queue. Besides, the interaction experience between mobile and environment is stored for training DDPG and Double DQN models.

6. Performance analysis

In this section, we first introduce the simulation setup and the baselines. Then the performance of our computation offloading policy is evaluated through extensive simulations. The schemes are implemented in Python 3.6 and experiments are run on a computer with Intel Core i5 CPU, 2.9 GHz, NVIDIA GeForce GTX 1660 GPU, 16 G RAM under Windows 10.

6.1. Simulation setup

The default simulation settings are set as follows unless otherwise stated. The CPU frequency f_s of the MEC server is set at 50 GHz (2.5 GHz per core, 20 cores), and the average generating rate of tasks $\bar{\chi}$ is 5 tasks per second [27]. According to [28], the data size s of each task is set between 500 Kbits and 2 Mbits, and complexity c is between 500 and 2000 cycles per byte. Besides, the CPU frequency of MD is constrained by $f^{max} = 2$ GHz and the maximum antenna transmit power $P_{tx}^{max} = 2$ W. And the effective switched capacitance is set as $k = 10^{-28}$ according to [25]. Furthermore, we set the wireless channel model according to [26]. Concretely, the antenna far field test distance $d_0 = 1$ m, the channel gain constant $G_0 = 10^{-3}$, path loss exponent $\gamma \in (1.6, 3.5)$, the wireless bandwidth $B = 10$ MHz, and noise power $N_0 = 10^{-13}$ W. Besides, Table 2 describes the parameter settings for the training of DRLCO.

Table 2
Training parameters for DRLCO.

Parameter	Value
Number of MDs	35
Number of MEC server	1
Task generate rate of MDs	5 per time slot
Time slot duration	1 s
Length of time slot	10
Learning rate for actor of DDPG	0.005
Learning rate for critic of DDPG	0.001
Learning rate for Double DQN	0.01
Batch size for DDPG	64
Batch size for Double DQN	32
Discount factor ζ_p, ζ_o	both 0.9

6.2. Compared schemes

To evaluate the performance of the proposed computation offloading policy, six baselines are introduced as follows:

- *Local Execution*: The mobile device executes all computation tasks locally with the maximum CPU frequency.
- *Server Execution*: The mobile device transmits all computation tasks to the MEC server with the maximum transmit power.
- *Greedy Offloading*: According to the execution delay and transmission delay, the mobile device selects the execution mode with the least cost for each computing task.
- *LODCO Algorithm* [25]: Both the computation power and the computation offloading power are obtained by Lyapunov optimization. And the offloading decision is obtained according to the greedy algorithm.
- *TSO Algorithm* [23]: Offloading decision is made according to offloading probability which is obtained by the one-dimensional linear search.
- *MORL Algorithm* [33]: Offloading decision is made by multi-objective reinforcement learning to reduce the execution delay and energy consumption.

6.3. Performance evaluation

6.3.1. Overall performance

Next, we evaluate the performance of six offloading methods in terms of delay, energy consumption, and execution cost.

Table 3 shows the overall performance of different approaches. Compared to other methods, *Local Execution* has the highest execution delay (11.396 s) and the highest execution cost (11.701), which indicates that the MD cannot fully meet the computational requirements for performing computation tasks. In contrast, *Server Execution* offloads all tasks to the MEC server, which greatly reduces the delay (1.923 s) but incurs high energy consumption (0.995×10^{-3} J). This is because the transmission task consumes a large amount of energy, resulting in high execution costs (2.918). Besides, *Greedy Execution* has significant reduction in delay (about 23% reduction) and energy consumption (about 31% reduction)

Table 3
Performance of the computation offloading schemes.

Schemes	Delay (S)	Energy (10^{-3} J)	Total cost	Decision delay (10^{-3} S)	Training delay (10^{-3} S)
Local Execution	11.396	**0.305**	11.701	N/A	N/A
Server Execution	1.923	0.995	2.918	N/A	N/A
Greedy Offloading	1.668	0.680	2.348	0.516	N/A
LODCO	1.639	0.643	2.282	1.262	N/A
TSO	1.425	0.724	2.149	0.938	N/A
MORL	1.341	0.527	1.868	0.739	3.347
DRLCO	**1.002**	0.467	**1.469**	0.997	6.153

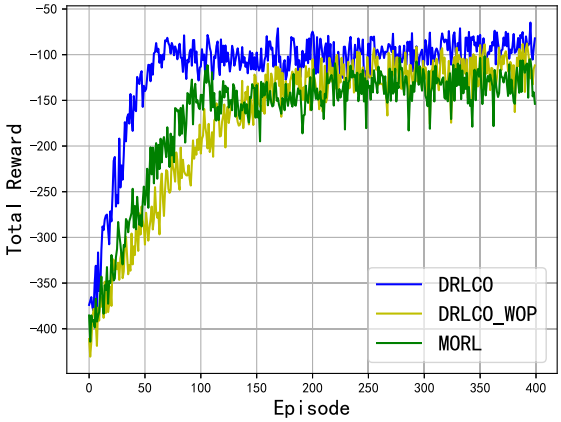


Fig. 4. Learning curve of three reinforcement learning methods.

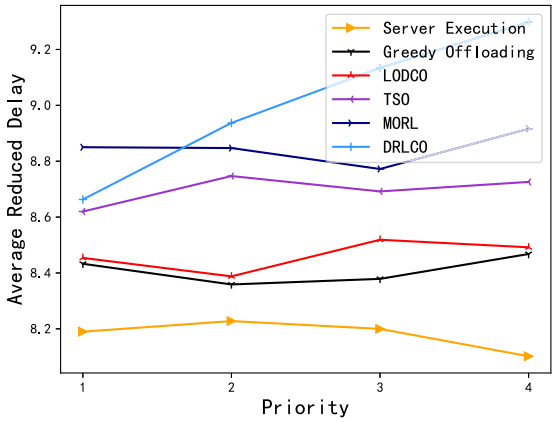


Fig. 5. Execution delay vs. different task priorities.

compared with *MEC Server Execution*. *LODCO* reduces the time delay to 1.639 s and the energy consumption to 0.643×10^{-3} J. *TSO* performs well on the delay (13% decrease) while failing to control the energy consumption (12% increase) of the MD compared with *LODCO*. This is because *TSO* focuses on reducing task delay and ignores energy consumption. In addition, the *MORL* algorithm can effectively reduce delay (1.341 s) and energy consumption (0.527×10^{-3} J). However, MORL only focuses on the impact of task offloading mode on execution cost, while ignoring the impact of CPU frequency and transmission power of the MD. *DRLCO* has the lowest execution cost (1.469), which is about 87% and 50% reduction compared to *Local Execution* and *Server Execution*. And *DRLCO* achieves the lowest execution delay (1.002 s) compared with other baselines. It is because our task offloading algorithm can make dynamic decisions based on environmental information. Our method also obtains a good performance in energy consumption (0.417×10^{-3} J) because our power scheduling algorithm can dynamically adjust the CPU frequency and transmit power of the MD.

To discuss the overhead of three reinforcement learning models, we record the decision delay of different algorithms. The delay of our algorithm to make a decision is 0.938×10^{-3} s, which is far below the task execution delay of 1.002 s. Besides, we compared the training delay of DRLCO and MORL. The DRLCO training an episode takes 6.153×10^{-3} s and the MORL training takes 3.427×10^{-3} s. This is because the MORL model has fewer parameters for decision-making.

Fig. 4 shows the learning curves of DRLCO, MORL and DRLCO without Adaptive Prioritized Experience Replay (DRLCO WOP). As shown in the figure, the DRLCO WOP has the slowest convergence speed (converge at episode 180), which shows that Algorithm 3 can accelerate the convergence of the model. In addition, the MORL starts to converge at episode 100, and the final reward is about -130.

Fig. 5 depicts the average reduced delay of the tasks with different priorities over six approaches compared to *Local Execution*. In this experiment, the execution delay of tasks with different priorities is different, which is due to that the data size and complexity are random when generating tasks.

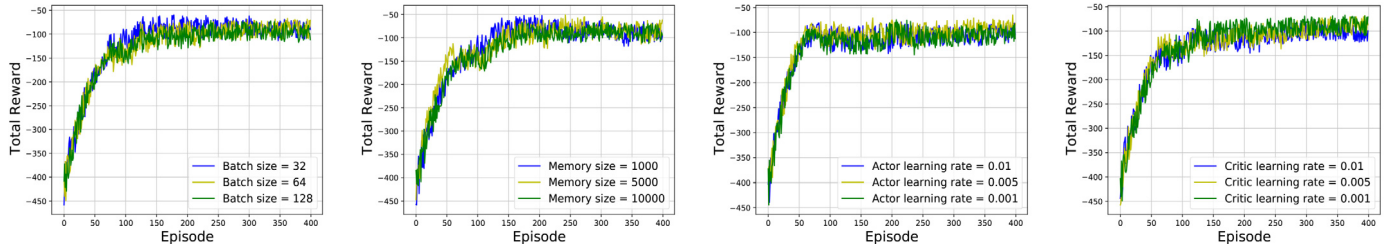
Therefore, we compared the delay reduction of different priorities of tasks of each algorithm relative to *Local Execution*. We can find that the six baselines did little to reduce execution delays for high-priority tasks. In contrast, in our algorithm, the higher the task priority, the higher the delay reduction. Tasks with low to high priority are each reduced by 8.663 s, 8.937 s, 9.134 s, and 9.298 s. This is because we developed a queue sorting algorithm and considered the priority of the task when making the offloading decisions to reduce the delay of high-priority tasks.

6.3.2. Parameter tuning

We firstly study the effects of different parameters on the total reward for task execution. Concretely, we discuss different batch sizes, memory sizes, and learning rates for both power scheduling and task offloading algorithms. In Fig. 6(a), we plot the total reward of task execution when the batch size of DDPG varies from 32 to 128. It can be seen from the figure that the three curves converge at about 150 episodes, and the total rewards stabilized around -90 at the final. We find that the curve converges the fastest (about 110 episodes) when batch size = 32 but the lower total reward at final. It is because small batch size can store little experience, and the reward drops when a new state is sampled. Also, there is a fact that a larger batch size consumes more time for training. Thus, we set batch size = 64 for the power scheduling algorithm.

Then we present the simulation results in Fig. 6(b) with the size of replay memory ranging from 1000 to 10000. We find that the larger the memory size, the slower the convergence rate of our algorithm, but the greater the total reward at the final since a large memory size can reduce the correlation of sampled data. However, the storage space occupied by replay memory is positively related to its size. Therefore, we set the memory size to 5000 for our algorithm.

Fig. 6(c) and Fig. 6(d) plot the effects of different learning rates (range from 0.001 to 0.01) on the total reward for the actor and

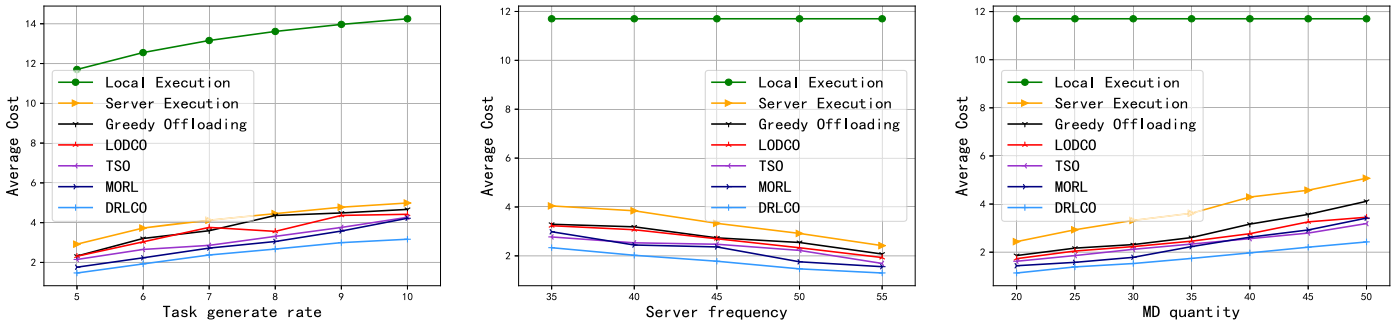


(a) Batch size (b) Memory size (c) Learning rate of actor model (d) Learning rate of critic model

Fig. 6. Impacts of different parameters of power scheduling algorithm on the total reward, including (a) batch size, (b) memory size, (c) learning rate of actor model, and (d) learning rate of critic model.

(a) Batch size (b) Memory size (c) Learning rate

Fig. 7. Impacts of different parameters of task offloading algorithm on the total reward, including (a) batch size, (b) memory size, (c) learning rate.



(a) Average generating rate of tasks (b) Frequencies of the MEC server (c) Numbers of the MDs

Fig. 8. Performance of task execution with different (a) average generating rate of tasks, (b) frequencies of the MEC server, and (c) number of the MDs.

critic model. It can be seen from the two figures that the higher the learning rate, the faster the curve converges. In Fig. 6(c), we find that the model received the highest total reward (about -80) when the learning rate of actor was 0.005. In Fig. 6(d), the curve (learning rate = 0.001) converges the slowest (converge at 150 episodes), but obtains the highest total reward (about -75) at final. To maximize the total reward, the learning rates of the actor and critic models are set to 0.005 and 0.001.

We also tuned the parameters of the task offloading algorithm. Fig. 7(a) illustrates the effects of the batch size on the total reward. We find that the three curves fall into the local optimum between 50 and 150 episodes, which may be due to the agent's insufficient exploration of the environment. But the two curves (batch size = 32 and 64) perform well after 150 episodes. Thus, we set the batch size to 32 considering the training time.

Fig. 7(b) depicts the convergence performance when the size of replay memory varies from 1000 to 10000. After 250 episodes, the MD received the highest total reward (about -75) when memory size = 5000, while the MD received the lowest total reward (about -90) when memory size = 1000. Thus, we set the memory size to 1000 to reduce the storage space.

Finally, we present the simulation results in Fig. 7(c) with the learning rate ranging from 0.005 to 0.05. We can find that the curve (learning rate = 0.005) converges the slowest, while the curve (learning rate = 0.05) fluctuates the most, which is related to the nature of deep learning. Then, we set the learning rate to 0.01.

Besides, we find that the parameter tuning of the task offloading algorithm has a greater effect on the convergence rate than that of the power scheduling algorithm when comparing Fig. 6 and Fig. 7. This is mainly because the task offloading algorithm has a direct impact on the execution mode of the task.

6.3.3. *Impact factors*

We also vary several critical factors, i.e., the average generating rate of tasks $\bar{\chi}$, the CPU frequency of the MEC server f_s, and the number of the MDs to study their impact on the *DRLCO*.

Fig. 8(a) illustrates the impact of the average generating rate of tasks $\bar{\chi}$ of the MD, which varies from 5 to 10 per second. We find that the execution cost of the tasks increases gradually as $\bar{\chi}$ increases. Compared with *Server Execution*, *DRLCO* reduces the execution cost by half. This performance can be attributed to two features of our algorithm: 1) maintain the load balance between the MEC server and the MD; 2) maintain a balance between power consumption and delay.

Fig. 8(b) depicts the impact of the CPU frequency of the MEC server f_s on the execution cost. We can find that the higher the CPU frequency of the server, the lower the execution cost of the task. When f_s = 50 GHz (2.5 GHz per core, 20 cores), the execution costs of our algorithm are reduced by 15%, 32%, 35%, 41%, 48% and 87% than other six baselines.

Fig. 8(c) depicts the impact of the number of MDs on the execution cost. We can find that the higher the number of MDs, the higher the execution delay of the task due to the number of tasks increased. When there are 20 MDs, the execution cost of our algorithm is reduced by 21%, 30%, 34%, 39%, 55%, and 90% than the other six baselines.

7. Conclusion

In this article, we model a computation offloading framework for the resource-constrained MEC system. Then, we take task execution delay and energy consumption as optimization objectives which are key factors to measure the QoE of mobile users. To minimize both the execution delay and energy consumption, we propose a double reinforcement learning computing offloading algorithm that can jointly schedule CPU frequency, transmission power, and task offloading mode. As compared with other benchmark schemes, simulation results have demonstrated that the proposed algorithm can effectively reduce the execution delay of the tasks and the energy consumption of the MD.

For future work, we are going to investigate the computation offloading problem in the MEC scenario with multiple MEC servers and multi-MDs. We will consider the problem of resource scheduling and the cooperation between the MDs in this scenario.

CRediT authorship contribution statement

This paper studies the problem of computation offloading in resource constrained MEC systems, and proposes a double reinforcement learning computation offloading algorithm. Experiments show that our method reduces the delay by 48% and the energy consumption by 53%. In this work, the main contribution of Liao is experiments and write paper. The experiment was conducted under the guidance of Lai. In addition, Yang and Zeng provided technical guidance and financial support.

We declare that the material presented in this manuscript has not been previously published, except in abstract form, nor is it simultaneously under consideration by any other journal.

Declaration of competing interest

We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of, the manuscript entitled “Online Computation Offloading with Double Reinforcement Learning Algorithm in Mobile Edge Computing”.

Acknowledgments

This work is partially supported by the Natural Science Foundation of Guandong (2021A1515011578), the Funding of Longyan Institute of Industry and Education Integration, Xiamen University (20210302), Natural Science Foundation of China (61872154), the Natural Science Foundation of Fujian (2018J01097), and Shenzhen Basic Research Program (JCYJ20190809161603551).

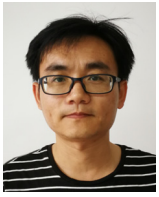
References

[1] A.A. Al-Habob, A. Ibrahim, O.A. Dobre, A.G. Armada, Collision-free sequential task offloading for mobile edge computing, IEEE Commun. Lett. 24 (2019) 71–75.
[2] A.A. Al-Habob, O.A. Dobre, A.G. Armada, S. Muhaidat, Task scheduling for mobile edge computing using genetic algorithm and conflict graphs, IEEE Trans. Veh. Technol. 69 (2020) 8805–8819.
[3] T. Alfakih, M.M. Hassan, A. Gumaei, C. Savaglio, G. Fortino, Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa, IEEE Access 8 (2020) 54074–54084.
[4] P.A. Apostolopoulos, E.E. Tsiropoulou, S. Papavassiliou, Cognitive data offloading in mobile edge computing for Internet of things, IEEE Access 8 (2020) 55736–55749.
[5] S. Barbarossa, S. Sardellitti, P. Di Lorenzo, Communicating while computing: distributed mobile cloud computing over 5g heterogeneous networks, IEEE Signal Process. Mag. 31 (2014) 45–55.
[6] R. Bellman, Dynamic programming, Science 153 (1966) 34–37.
[7] D.S. Bernstein, R. Givan, N. Immerman, S. Zilberstein, The complexity of decentralized control of Markov decision processes, Math. Oper. Res. 27 (2002) 819–840.
[8] T.D. Burd, R.W. Brodersen, Processor design for portable systems, J. VLSI Signal Process. Syst. Signal Image Video Technol. 13 (1996) 203–221.
[9] N. Chen, S. Zhang, Z. Qian, J. Wu, S. Lu, When learning joins edge: real-time proportional computation offloading via deep reinforcement learning, in: 2019 IEEE 25th International Conference on Parallel and Distributed Systems, ICPADS, IEEE, 2019, pp. 414–421.
[10] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Trans. Netw. 24 (2015) 2795–2808.
[11] X. Chen, Y. Cai, Q. Shi, M. Zhao, B. Champagne, L. Hanzo, Efficient resource allocation for relay-assisted computation offloading in mobile-edge computing, IEEE Int. Things J. 7 (2019) 2452–2468.
[12] Y. Ding, C. Liu, X. Zhou, Z. Liu, Z. Tang, A code-oriented partitioning computation offloading strategy for multiple users and multiple mobile edge computing servers, IEEE Trans. Ind. Inform. 16 (2019) 4800–4810.
[13] T.Q. Dinh, J. Tang, Q.D. La, T.Q. Quek, Offloading in mobile edge computing: task allocation and computational frequency scaling, IEEE Trans. Commun. 65 (2017) 3571–3584.
[14] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): a vision, architectural elements, and future directions, Future Gener. Comput. Syst. 29 (2013) 1645–1660.
[15] S. Hu, G. Li, Dynamic request scheduling optimization in mobile edge computing for iot applications, IEEE Int. Things J. 7 (2019) 1426–1437.
[16] L. Huang, S. Bi, Y.J.A. Zhang, Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks, IEEE Trans. Mob. Comput. 19 (2019) 2581–2593.
[17] L.P. Kaelbling, M.L. Littman, A.W. Moore, Reinforcement learning: a survey, J. Artif. Intell. Res. 4 (1996) 237–285.
[18] M. Kamoun, W. Labidi, M. Sarkiss, Joint resource allocation and offloading strategies in cloud enabled cellular networks, in: 2015 IEEE International Conference on Communications, ICC, IEEE, 2015, pp. 5529–5534.
[19] Othman M. Khan, S.A. Madani, S.U. Khan, et al., A survey of mobile cloud computing application models, IEEE Commun. Surv. Tutor. 16 (2013) 393–413.
[20] J. Kleinberg, E. Tardos, Algorithm Design, Pearson Education India, 2006.
[21] K. Kumar, J. Liu, Y.H. Lu, B. Bhargava, A survey of computation offloading for mobile systems, Mob. Netw. Appl. 18 (2013) 129–140.
[22] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, in: 4th International Conference on Learning Representations, ICLR, 2016, pp. 1–14.
[23] J. Liu, Y. Mao, J. Zhang, K.B. Letaief, Delay-optimal computation task scheduling for mobile-edge computing systems, in: 2016 IEEE International Symposium on Information Theory, ISIT, IEEE, 2016, pp. 1451–1455.
[24] P. Mach, Z. Becvar, Mobile edge computing: a survey on architecture and computation offloading, IEEE Commun. Surv. Tutor. 19 (2017) 1628–1656.
[25] Y. Mao, J. Zhang, K.B. Letaief, Dynamic computation offloading for mobile-edge computing with energy harvesting devices, IEEE J. Sel. Areas Commun. 34 (2016) 3590–3605.
[26] T.L. Marzetta, Noncooperative cellular wireless with unlimited numbers of base station antennas, IEEE Trans. Wirel. Commun. 9 (2010) 3590–3600.
[27] X. Meng, W. Wang, Y. Wang, V.K. Lau, Z. Zhang, Closed-form delay-optimal computation offloading in mobile edge computing systems, IEEE Trans. Wirel. Commun. 18 (2019) 4653–4667.
[28] A.P. Miettinen, J.K. Nurminen, Energy efficiency of mobile clients in cloud computing, in: 2nd USENIX Workshop on Hot Topics in Cloud Computing, in: HotCloud, vol. 10, 2010, pp. 1–7.
[29] L. Qian, Y. Wu, J. Ouyang, Z. Shi, B. Lin, W. Jia, Latency optimization for cellular assisted mobile edge computing via non-orthogonal multiple access, IEEE Trans. Veh. Technol. 69 (2020) 5494–5507.
[30] B. Rountree, D.H. Ahn, B.R. De Supinski, D.K. Lowenthal, M. Schulz, Beyond dvfs: a first look at performance under a hardware-enforced power bound, in: 2012

IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IEEE, 2012, pp. 947–953.
[31] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, in: 4th International Conference on Learning Representations, ICLR, 2016, pp. 1–21.
[32] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic policy gradient algorithms, in: International Conference on Machine Learning, PMLR, 2014, pp. 387–395.
[33] F. Song, H. Xing, X. Wang, S. Luo, P. Dai, K. Li, Offloading dependent tasks in multi-access edge computing: a multi-objective reinforcement learning approach, Future Gener. Comput. Syst. 128 (2022) 333–348.
[34] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2016, pp. 2094–2100.
[35] F. Wang, J. Xu, S. Cui, Optimal energy allocation and task offloading policy for wireless powered mobile edge computing systems, IEEE Trans. Wirel. Commun. 19 (2020) 2443–2459.
[36] S. Wang, S. Dey, Adaptive mobile cloud computing to enable rich mobile multimedia applications, IEEE Trans. Multimed. 15 (2013) 870–883.
[37] M. Yao, L. Chen, T. Liu, J. Wu, Energy efficient cooperative edge computing with multi-source multi-relay devices, in: 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems, HPCC/SmartCity/DSS, IEEE, 2019, pp. 865–870.
[38] T. Zhang, Y. Xu, J. Loo, D. Yang, L. Xiao, Joint computation and communication design for uav-assisted mobile edge computing in iot, IEEE Trans. Ind. Inform. 16 (2019) 5505–5516.
[39] Z. Zhou, J. Feng, Z. Chang, X. Shen, Energy-efficient edge computing service provisioning for vehicular networks: a consensus admm approach, IEEE Trans. Veh. Technol. 68 (2019) 5087–5099.

Linbo Liao received the B.S degree in Department of Internet of Things Engineering from Nanchang Hangkong University in 2019, the master degree from School of Informatics, Xiamen University in 2022. His research interests are edge and distributed computing and machine learning.

Yongxuan Lai received the bachelor degree at Management of Information System from Renmin University of China in 2004, the Ph.D. degree in Computer Science from Renmin University of China in 2009. He is currently a professor in Software Engineering Department, School of Informatics, Xiamen University, China. He is also the dean of School of Mathematics and Information Engineering, Longyan University, China. He was an visiting scholar during Sep. 2017 - Sep. 2018 at University of Queensland, Australia. His research interests include network data management, intelligent transportation systems, and big data management and analysis.



Fan Yang received his Ph.D. degree in Control Theory and Control Engineering from Xiamen University in 2009. He is currently an associate professor in the Department of Automation at Xiamen University. His research interests include feature selection, ensemble learning, and intelligent transportation systems.

Wenhua Zeng received his Ph.D. degree in Industrial Automation from Zhejiang University in 1989. He is currently a professor in the Department of Software Engineering at Xiamen University. His research interests include embedded system, embedded software, internet of things, and cloud computing.