Service Oriented Computing and Applications
https://doi.org/10.1007/s11761-024-00419-6

ORIGINAL RESEARCH PAPER

Deep reinforcement learning based task offloading and resource allocation strategy across multiple edge servers

Bing Shi[1,2] · **Yuting Pan**[1] · **Lianzhen Huang**[1]

Received: 28 November 2023 / Revised: 2 May 2024 / Accepted: 6 July 2024

Abstract
In the mobile edge computing environment, multiple edge servers are often deployed in task-dense areas, however, the service coverage of these edge servers may overlap with each other. In such scenarios, users within the overlapping areas need to determine which server is chosen to offload the task. However, unreasonable decision of task offloading may result in imbalanced loads, thereby affecting the number of served users and the latency and energy consumption of user task offloading. Furthermore, the complexity of task offloading and resource allocation is further heightened by the dynamic arrival of user tasks. Therefore, it is crucial to design an effective task offloading and resource allocation strategy in an environment with multiple edge servers. In this paper, we propose a task offloading and resource allocation strategy aimed at meeting task latency requirements while maximizing the number of served users and minimizing the average energy consumption of all completed tasks. To timely obtain information about user tasks and the status of edge servers, we adopt a central controller to manage multiple edge servers. Then, we model the problem as a parameterized action Markov decision process and utilize the parameterized deep Q-network algorithm, a deep reinforcement learning algorithm, to solve it. Additionally, we conducted experiments to evaluate the performance of our proposed strategy against five benchmark strategies. The results demonstrate the superiority of our strategy in terms of the number of served users and the average energy consumption per task while meeting task latency constraints.

Keywords Multiple edge servers · Task offloading · Resource allocation · Deep reinforcement learning

1 Introduction

With the rapid development of Internet of Things (IoT) technology, mobile devices such as smartphones and tablets have become an indispensable part of our daily lives. These devices are increasingly utilized for various computationally intensive applications, such as real-time video processing, online gaming, and facial recognition [1–3]. However, due to the limited computing capabilities of mobile devices, these tasks often require offloading to cloud servers for processing. Nevertheless, the increased number of tasks uploaded to cloud servers creates significant server load pressure, while the long transmission distance introduces high latency and energy consumption.

To address these issues, mobile edge computing (MEC) has emerged as a solution [4] that deploys edge servers with computing and storage resources closer to users at the network edge, enabling task offloading with lower latency and energy consumption. However, as the number of mobile users increases, the limited computing resources of edge servers can impact task execution latency and energy consumption. Therefore, it is common to deploy multiple edge servers in mobile device-dense areas, where each server can provide computing services to users within their service coverage areas. For example, in the Internet of Vehicles, smart vehicles are equipped with sensors, automatic driving assistance systems, and navigation systems, and these vehicles have very high requirements for computational needs and response speed in tasks such as obstacle detection, path planning, image, and data processing. Multiple edge servers usually need to be deployed along the roadway to provide the necessary computing services to meet the computing requirements

✉ Bing Shi
bingshi@whut.edu.cn

[1] School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan 430000, China

[2] Shenzhen Research Institute of Wuhan University of Technology, Shenzhen 518000, China

Published online: 17 July 2024

for these tasks and ensure traffic safety. In the industrial Internet of Things, production equipment also needs to handle many computational tasks such as real-time monitoring, data analysis, and fault prediction. To ensure that these tasks can be processed instantly, multiple edge servers need to be deployed within the industrial production area to provide computing services. However, in an environment with multiple edge servers, service coverage areas of multiple servers may be overlapped with each other. For tasks located in these overlapped areas, multiple choices for task offloading arise. If a random selection is made to offload a task to an edge server, it may be offloaded to an edge server with a lower computing capacity or heavier workload. This may lead to task offloading latency and increased energy consumption. Furthermore, it may happen that some edge servers become overloaded while others are underutilized, leading to wasted computing resources and a decreased number of users served by the edge servers. Therefore, overlapping service coverage of multiple edge servers complicates the problem of task offloading and resource allocation. Moreover, with the dynamic arrival of user demands, it is necessary to make rational task offloading and resource allocation decisions based on the specific requirements of each task and the real-time status information of each edge server. This further increases the complexity of task offloading and resource allocation in an environment with multiple edge servers. Therefore, in the edge computing system where multiple edge servers coexist and their service coverage overlaps, we need to design an effective task offloading and resource allocation strategy for intensive and latency-sensitive tasks.

Currently, there exist many works analyzing task offloading strategy and resource allocation strategy independently in an environment with multiple edge servers. In this paper, we consider both task offloading decisions and edge server resource allocation as a whole. Therefore, the study in this paper involves a mixed-action space, where task offloading decisions are discrete actions and resource allocation is a continuous action. Furthermore, when the service areas of edge servers overlap, unreasonable task offloading and resource allocation decisions may cause an overload of some edge servers while underutilizing the resources of other servers. Such an imbalance in resource allocation can increase task completion latency and energy consumption and decrease the overall efficiency of the entire edge server system. Moreover, users' tasks are dynamically generated in real time. Although a few works have started to consider dynamic environments, they may ignore the mutual influence among edge servers. Therefore, task offloading and resource allocation strategy needs to be studied, taking into account the dynamic nature of user tasks and the impact between edge servers.

To tackle the above challenges, we propose an effective task offloading and resource allocation strategy that maximizes the number of served users and minimizes the average energy consumption of completed tasks, while meeting task latency constraints. This paper advances the state of the art in the following ways. We use a central controller[1] to coordinate task offloading and resource allocation among multiple edge servers. Since the current user task offloading decision is affected by the last offloading and will also affect the next task offloading decision, this is a sequential decision problem. Furthermore, this problem involves a discrete-continuous hybrid action space, i.e. task offloading is a discrete action while resource allocation is a continuous action. Therefore, we transform the task offloading and resource allocation problem into a parameterized action Markov decision process (PAMDP), and use a parameterized deep Q-network algorithm (*P-DQN*) to address it in an environment with multiple edge servers. Furthermore, we experimentally evaluate this strategy against five typical benchmark strategies. Experimental results show that our strategy can outperform five typical benchmark strategies in terms of the number of served users and the average energy consumption of all completed tasks under task latency constraints. This means that our task offloading and resource allocation strategy is effective for dynamically arrived tasks by considering the remaining resources of each edge server and the computing capability of the user's mobile device.

The rest of this paper is organized as follows. In Sect. 2, we introduce the related work. In Sect. 3, we introduce the system model of this paper. In Sect. 4, we describe how to use *P-DQN* algorithm to design the task offloading and resource allocation strategy in an environment with multiple edge servers. We evaluated this strategy in different experimental settings, the experimental analysis of which is shown in Sect. 5. Finally, we conclude in Sect. 6.

2 Related work

Currently, task offloading and resource allocation in MEC environments are primarily investigated in single and multiple-edge server scenarios. The main approaches include (1) heuristic algorithm-based methods and (2) machine learning algorithm-based methods. In the following, we introduce the related work from these two different approaches and summarize them in Table 1.

[1] This kind of central controller is adopted in some existing works, such as using software-defined networking to control MEC systems or using cloud radio access network technology to put all MEC servers under the control of one central unit. For example, Wang et al. [5] use software-defined networking to control distributed edge servers. Both Tran et al. [6] and Fang et al. [7] use a central unit to manage multiple edge servers.

Table 1 Summary of related work

Research scenario	References	Objective	Proposed solution	Weakness
In a single-edge server environment	[8]	Minimized latency	Applying heuristic algorithms to partially offload tasks to single-edge servers for processing	Ignoring task offload energy consumption
	[9]	Minimized energy consumption	Modelling as a stochastic optimisation problem to be solved	Computationally complex, not applicable to latency-sensitive tasks
	[10]	Increased rate of task completion	Applying heuristic algorithms for partial offloading	Ignoring the dynamics of user tasks
	[11]	Minimized energy consumption	Designing task offloading strategy based on deep reinforcement learning algorithm	Not applicable to task latency-sensitive tasks
	[12]	Minimized the sum of task latency and energy consumption	Designing task offloading and resource allocation strategies based on two reinforcement learning algorithms for dynamic multiple user task scenarios	Computationally complex
	[13]	Increased rate of task completion	Designing an online task offloading algorithm based on deep reinforcement learning and a resource allocation algorithm based on Lyapunov optimisation	Ignoring the dynamics of user tasks, the algorithm is more complex
In an environment with multiple edge servers	[14]	Minimized energy consumption	Modelling as a mixed integer nonlinear programming problem and solving it using a two-stage heuristic algorithm	Ignoring the dynamics of user tasks
	[15]	Minimized the sum of task latency and energy consumption	Proposing a queuing algorithm to calculate the priority of user offloading tasks for task offloading	Computationally complex, ignoring the dynamics of user tasks
	[16]	Increased number of tasks served	Designed an online distributed algorithm based on Lyapunov optimisation and game theory	Computationally complex, not applicable to task latency-sensitive tasks
	[5]	Minimized latency	Proposing a resource allocation strategy based on a deep reinforcement learning algorithm to allocate computing resources among edge servers adaptively	Ignoring the dynamics of user tasks
	[17]	Minimized energy consumption	Task offloading using non-orthogonal multiple access techniques and resource allocation based on deep reinforcement learning	Computationally complex

Table 1 continued

Research scenario	References	Objective	Proposed solution	Weakness
	[18]	Increased number of tasks served	Determining optimal task offloading strategy using game theory and dynamic resource allocation for edge servers based on reinforcement learning	Computationally complex, ignoring the dynamics of user tasks

2.1 Heuristic algorithm-based methods

First, we introduce related work on heuristic algorithm-based computational offloading methods in different edge server environments. In a single-edge server environment, Lyu et al. [8] proposed a heuristic offloading decision algorithm in a near-end cloud system with multiple users and a single edge server, where mobile users' tasks with higher utility were offloaded to the edge server while other users' tasks executed locally, resulting in less latency for the edge server to complete the tasks. Chen et al. [9] studied the energy efficient task offloading problem and formulated the problem as a stochastic optimization problem to minimize the energy consumption of task offloading while ensuring the average length of queues. Ning et al. [10] proposed an iterative heuristic resource allocation strategy to offload a portion of users' computational tasks to edge servers while processing the remaining tasks locally, aiming to improve the task completion rate of edge servers by reducing task latency. In an environment with multiple edge servers, Li et al. [14] investigated the task offloading and resource allocation problem in terminal devices with varying performance and edge servers with limited computing resources, modeling the problem as a mixed integer nonlinear programming problem and employing a two-stage heuristic algorithm that minimized energy consumption. Deng et al. [15] modeled the queuing state of mobile users' tasks offloaded to edge servers as a Markov decision process and then proposed a queuing algorithm to calculate the priority of user offloading tasks, optimizing total user latency and device energy consumption. Xia et al. [16] proposed an online distributed algorithm based on perturbed Lyapunov optimization and game theory, aiming to increase the number of tasks served by the system by optimizing computational resources and battery energy while minimizing task completion latency and energy consumption.

2.2 Machine learning algorithm-based methods

We introduce the related work on machine learning algorithm-based computational offloading methods in different edge server environments. In a single-edge server environment, Yan et al. [11] considered a single-edge server system for recording users' generic tasks and proposed a task offloading strategy based on the deep reinforcement learning algorithm to achieve the optimization goal of minimizing task completion energy consumption. Li et al. [12] provided an in-depth study of task offloading and resource allocation for dynamically changing multi-user mobile edge computing environments and proposed a solution based on the Q-learning algorithm and deep reinforcement learning algorithm, aiming to minimize the sum of latency and energy consumption. Bi et al. [13] introduced a deep reinforcement learning-based online task offloading algorithm and a Lyapunov optimization-based resource allocation algorithm to reduce task execution latency and improve task offloading efficiency. In an environment with multiple edge servers, Wang et al. [5] investigated the problem of allocating computing resources to edge servers and proposed a resource allocation strategy based on deep reinforcement learning algorithms that could adaptively allocate computing resources among edge servers, thus reducing the service time for tasks. Qian et al. [17] proposed a non-orthogonal multiple access technique for task offloading and jointly designed a distributed algorithm under a static channel and a deep reinforcement learning online algorithm under a dynamic channel for resource allocation to realize the reduction of the energy consumption of the whole system while satisfying the task latency. Jiang et al. [18] proposed the use of game theory to determine the optimal task offloading strategy and reinforcement learning to achieve dynamic resource allocation of edge servers to achieve the maximization of the number of users to be served while satisfying their quality of service.

However, these works mainly focus on static optimization problems, while the real-world mobile edge computing environment is complex with dynamic computing demands. Moreover, some works only consider scenarios with a single edge server, and when examining multiple edge servers, they fail to consider the mutual interactions between them, especially in cases where the edge server service coverage areas are partially overlapped with each other. In this paper, we address task offloading and resource allocation problem in an environment with multiple edge servers by considering the above factors, with the aim to maximize the number of served users and minimize the average energy consumption

Table 2 Key symbols

Notation	Description
T	The number of time slots
N	The number of users
M	The number of edge servers
u_i	User i
e_j	Edge server j
ω_j^t	The remaining computing resources of edge server e_j at time slot t
l_{ij}	Distance from user u_i to edge server e_j
H_i	The task of user u_i
$fl_{\max}$	The maximum computing capability of mobile devices
p_{trans}	Transmission power
h_{ij}	Channel gain power
B	The bandwidth of edge server
k_l	The energy consumption factor of users' mobile devices
k_e	The energy consumption factor of edge server

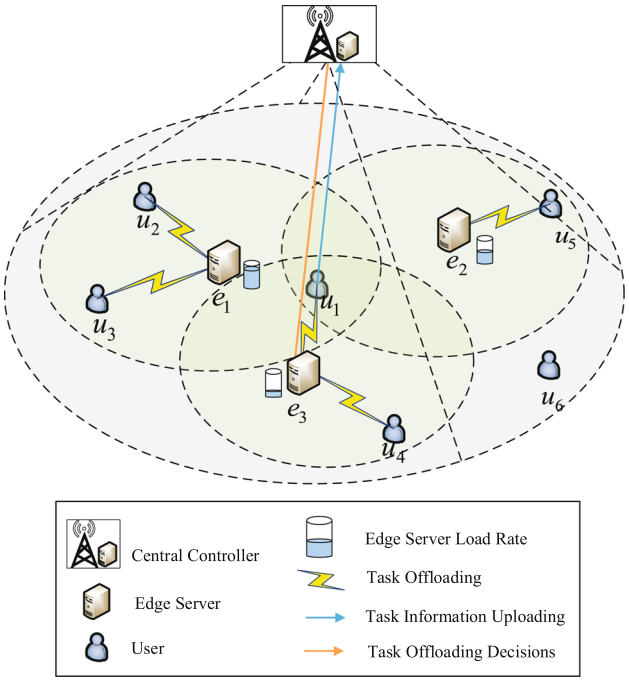


Fig. 1 Task offloading across multiple edge servers

for all completed tasks while ensuring compliance with task latency constraints.

3 System model

In this section, we first describe the task offloading and resource allocation scenario and the related settings of this paper. Then, we detail the process where tasks are executed on their own mobile devices or offloaded to edge servers for processing. Finally, we give the optimization objectives.

We assume that N users arrive dynamically in a finite number of time slots $t \in \{1, 2, \ldots, T\}$. Each user's mobile device generates a computationally intensive task, which is sensitive to latency with different computing resource requirements. To meet the task's latency constraints, the edge server needs to decide how to offload tasks and allocate resources. The mathematical symbols used in this paper are shown in Table 2.

In an environment with multiple edge servers, we consider that there is no direct physical link for communication between edge servers. Therefore, they independently handle computing tasks within their service coverage respectively. This means that only users located within the service coverage of an edge server can offload tasks to that server for processing. Considering each server's service coverage is limited, several edge servers' service coverage may overlap. Therefore users in areas of overlapped service coverage have a variety of options for task offloading. Furthermore, we assume that the task cannot be further divided, i.e., it can only be processed on its mobile device or the edge server.

For example, in Fig. 1, there are 6 users denoted as u_1, u_2, u_3, u_4, u_5 and u_6, and 3 edge servers denoted as e_1, e_2 and e_3. We can find that u_1 is in the overlapped service coverage of three edge servers, and its task can be offloaded to one of these three edge servers. As u_2 and u_3 are only in the service coverage of e_1, these two tasks can only be offloaded to the edge server or executed on their own mobile devices. If edge server e_1 is nearly fully loaded and the task u_1 is offloaded to it, this edge server may become overloaded. At this point, the central controller must make reasonable decisions for dynamically offloading tasks to appropriate edge servers and allocating resources. At the same time, we discover that u_6 is not covered by the servers' service coverage. Therefore, the task of u_6 can only be processed on its own mobile device. In the sections afterward, we will introduce the specific process for tasks performed on their own mobile devices or edge servers.

3.1 Tasks executed on their own mobile devices

This section describes how tasks are executed on their own mobile devices. We define the user's computing task at time slot t as a three-tuple $H_i = (d_i, c_i, \tau_i), i \in N$, where d_i is the data size of the task, c_i is the total computing resources required to complete the task, and τ_i is the maximum completion time that the user can tolerate. Since each mobile device has a limited number of computing resources, tasks can be

executed locally when the computing resource requirements of the task are less than the maximum computing resources of the user's mobile device. The time at which task H_i is executed on its own mobile device is:

$$t_i^{local} = \frac{c_i}{f_{i0}} \tag{1}$$

where f_{i0} is computing resources allocated to task H_i by its own mobile device.

When $t_i^{local} \leq \tau_i$, the computing resources allocated by the user's mobile device to task H_i can meet the specific latency requirements, and then task H_i can be completed on its own mobile device. Similar to work in [19], since the energy consumed to process task H_i on its own mobile device is related to CPU frequency, the energy consumed to complete task H_i is expressed as:

$$E_i^l = k_l (f_{i0})^2 c_i \tag{2}$$

where k_l is the user's mobile device's energy consumption coefficient, which is based on the chip structure of the device [20], and $k_l (f_{i0})^2$ is the energy consumption per unit of computing resources.

3.2 Tasks executed on edge servers

In a fixed area, we assume that there are M edge servers and N users, and since each user arrives dynamically, we divide the time into a finite number of time slots $t \in \{1, 2, \ldots, T\}$. We define M edge servers as a set $E = \{e_1, e_2, \ldots, e_M\}$ and N users as a set $U = \{u_1, u_2, \ldots, u_N\}$. Each edge server has a three-tuple denoted $e_j = (\beta_j, F_j, p_j^e)$, where β_j represents the service coverage radius of the edge server, F_j represents the computing capability of the edge server and p_j^e is the location of the edge server, which is denoted as $p_j^e = (x_j^e, y_j^e)$ using two-dimensional Euclidean coordinates. Similarly we use $p_i^u = (x_i^u, y_i^u)$ to indicate the user's location.

Then the distance between user u_i and edge server e_j is expressed as:

$$l_{ij} = \sqrt{\left(x_i^u - x_j^e\right)^2 + \left(y_i^u - y_j^e\right)^2} \tag{3}$$

When $l_{ij} \leq \beta_j$, task H_i can only be transferred to edge server e_j for processing.

Then, according to the formula of Shannon's theorem, the data transfer rate between user's mobile device u_i and edge server e_j can be calculated, which can be expressed as:

$$r_{ij} = B \log_2 \left(1 + \frac{p_{trans} h_{ij}}{\sigma^2}\right) \tag{4}$$

where h_{ij} is channel power gain and σ^2 is the noise power. At this time, the transmission time of task H_i is:

$$t_{ij}^{trans} = \frac{d_i}{r_{ij}} \tag{5}$$

During this process, task transfer consumes some energy, which is:

$$e_{ij}^{trans} = p_{trans} t_{ij}^{trans} \tag{6}$$

After task H_i transmission is completed, edge server e_j allocates a certain amount of computing resources to meet the task's latency constraints. Since each edge server has limited computing resources and can only serve a limited number of tasks, the computing resources allocated by edge server e_j should be less than or equal to the remaining computing resources, which is $f_{ij} \leq \omega_j^t$. After the resource allocation is complete, task H_i is executed. The time required for task H_i execution is:

$$t_{ij}^{exe} = \frac{c_i}{f_{ij}} \tag{7}$$

Similarly, the energy consumed by edge server e_j to perform this task is similar to the energy consumed by its own mobile device to perform task H_i, which is:

$$e_{ij}^{exe} = k_e {f_{ij}}^2 c_i \tag{8}$$

where k_e is the edge server's energy consumption coefficient, which is based on the chip structure of the device [20].

After the server has processed task H_i, it needs to send the result to the mobile device. Since the calculation result of task H_i is usually small, we ignore this part of the latency when considering the downlink transmission latency [21]. As a result, the total latency required when task H_i is offloaded to edge server e_j for processing is denoted as:

$$T_{ij}^{off} = t_{ij}^{trans} + t_{ij}^{exe} \tag{9}$$

where t_{ij}^{trans} represents the transmission latency, and t_{ij}^{exe} represents the execution latency.

Similarly, the energy consumed by the edge server to accomplish the task includes the task transmission energy and the edge server processing energy, which is:

$$E_{ij}^{off} = e_{ij}^{trans} + e_{ij}^{exe} \tag{10}$$

When a task is completed, the computing resources occupied edge server e_j are released. Therefore, at the end of time slot t, the remaining computing resources on edge server e_j

is denoted as:

$$\omega_j^{t+1} = \omega_j^t + b_j^t \tag{11}$$

where ω_j^t represents the remaining computing resources of edge server e_j and b_j^t represents the computing resources returned to edge server e_j by task H_i.

3.3 Problem formulation

Our problem involves multiple optimization objectives, i.e., designing reasonable offloading decisions and resource allocation schemes for dynamically arriving user tasks in an environment with multiple edge servers to maximize the number of users served and minimize the average energy consumption for all task completions while meeting the latency requirements of the offloading tasks. Therefore, the optimization problem in this paper is expressed as follows.

$$P0: \quad \max \sum_{i \in N} \left(z_{i0} + \sum_{j \in M} \mathrm{z}_{ij} \right) \xi_i \tag{12}$$

$$\min \frac{\sum_{i \in N} \sum_{j \in M} \left(z_{i0} E_i^l + \mathrm{z}_{ij} \mathrm{E}_{ij}^{off} \right)}{\sum_{i \in N} (z_{i0} + \sum_{j \in M} \mathrm{z}_{ij}) \xi_i} \tag{13}$$

$$s.t. \quad C1: z_{i0} t_i^{local} + \sum_{j \in M} \mathrm{z}_{ij} t_{ij}^{off} \leq \tau_i \tag{14}$$

$$C2: z_{i0} + \sum_{j \in M} \mathrm{z}_{ij} \leq 1, \ \forall i, z_{i0}, \mathrm{z}_{ij} \in \{0, 1\} \tag{15}$$

$$C3: l_{ij} = \sqrt{(x_i^u - x_j^e)^2 + (y_i^u - y_j^e)^2} \leq \beta_j \tag{16}$$

$$C4: \begin{cases} f_{ij} \leq \omega_j^t, & \forall z_{ij} = 1 \\ f_{i0} \leq f l_{\max}, & \forall z_{i0} = 1 \end{cases} \tag{17}$$

where Eq. 12 means the maximization of the number of task-based users served in total, in which z_{i0} means processed locally, and z_{ij} means processed on edge servers. The constrained objective $C2$ means that these two variables can only take values 0 or 1. The variable ξ_i denotes whether the task processing meets the task latency requirement, if so, $\xi_i = 1$, otherwise $\xi_i = 0$. Equation 13 means the minimization of the average energy consumption of completed tasks, where E_i^l denotes processed locally consumes the energy consumption and E_{ij}^{off} denotes the total energy consumption set by processing on the edge server. The constraint $C1$ denotes the latency constraint of task H_i. The constraint $C2$ denotes that task H_i can be processed on edge server e_j or executed on its own mobile device, but it cannot be split to be executed in both locations at the same time. The constraint $C3$ states that task H_i can only be transmitted to the server if the user is within the edge server's service coverage. The first inequality of constraint $C4$ states that the allocation of computing resources cannot exceed the edge server's current remaining computing resources, while the second inequality states that the allocation of computing resources is not greater than its own mobile device's maximum computing capability.

4 The approach

In this section, we first explain how to represent the task offloading and resource allocation problem as a parameterized action Markov decision process (PAMDP) and then introduce how to apply a deep reinforcement learning *P-DQN* to solve it.

4.1 Parameterized action Markov decision process

As mentioned above, the central controller across multiple edge servers needs to make a decision for each task at each time slot in sequence. The decision affects the computing resources available in edge server e_j, and affects future decisions. Therefore, task offloading and resource allocation problem for multiple edge servers is a sequential-decision problem. In addition, the problem involves a discrete-continuous hybrid action space, i.e. task offloading decision is a discrete action, while the resource allocation decision is a continuous action, and thus we model it as a parameterized action Markov decision process (PAMDP) [22].

PAMDP is defined as a tuple $<S, A, r, P, \gamma>$, where S denotes a set of states, A denotes a parameterised action space with discrete-continuous hybrid actions, r denotes the immediate reward function of the agent, P denotes a Markovian state transition probability function, γ is a discount factor. The parameterised action space consists of a set of discrete actions that is $A_d = [K] = \{k_1, k_2, \ldots, k_K\}$, each discrete action k corresponds to a m_k dimensional continuous action parameter $x_k \in \mathcal{X}_k \subseteq \mathbb{R}^{m_k}$, and thus the parameterized action space can be written as:

$$A = \bigcup_{k \in [K]} \{A_k = (k, x_k) \mid x_k \in \mathcal{X}_k\} \tag{18}$$

Specifically, at first, at state $s_t \in S$, the central controller collects global state information of the MEC environment, including resource request information of tasks and state information of all edge servers. Afterwards, the central controller executes action $A_k \in A$ and obtains the reward r_t from the environment feedback. Then, the next moment state of the MEC environment is transformed to $s_{t+1} \in S$. Thus at each time slot t, running through the sequence $\{s_1, A_1, s_2, A_2, \ldots, s_t, A_t, \ldots\}$, PAMDP is formed. The detailed setup of this process is represented as follows:

– *State* we define it as: $s_t = \{H_i(t), p_i^u(t), \omega(t), b(t-1)\} \in S$, where $H_i(t) = (d_i, c_i, \tau_i)$ represents task H_i information at time slot t, which includes the data size of task H_i, the total computing resources required to complete task H_i, and the maximum completion time that user can accept to complete task. In addition, the remaining computing resources of M edge servers in the current time slot are expressed as: $\omega(t) = (\omega_1^t, \omega_2^t, \ldots, \omega_M^t)$, and the computing resources for processing users' tasks in the previous time slot are expressed as: $b(t-1) = (b_1^{t-1}, b_2^{t-1}, \ldots, b_M^{t-1})$.
– *Action* the action A is a parameterised action space with discrete-continuous hybrid actions, which is defined as: $k \in A_d = \{z_{i0}, z_{i1}, \ldots, z_{iM}\}$, where $z_{i0} = 1$ means that task H_i is executed on its own mobile device, and $z_{ij} = 1, j \in \{1, 2, \ldots, M\}$ means that task H_i is executed on edge server e_i. Similarly, we use $x_k = f_{i0}$ and $x_k = f_{ij}, j \in \{1, 2, \ldots, M\}$ to define whether task H_i gets computing resources on its own mobile device or on edge server e_j.
– *Reward* we define it as:

$$r(s_t, A_k) = \begin{cases} -z_{i0}E_i^l(t) - z_{ij}E_{ij}^{off}(t) + \lambda, \\ \qquad\qquad \tau_i \geq z_{i0}t_i^{local} + z_{ij}t_{ij}^{off} \\ -\mu, \qquad \tau_i < z_{i0}t_i^{local} - z_{ij}t_{ij}^{off} \end{cases} \tag{19}$$

where λ and μ being positive hyperparameters. When the completion time of a task satisfies its latency constraints, indicated by $\tau_i \geq z_{i0}t_i^{local} + z_{ij}t_{ij}^{off}$, a positive reward of $-z_{i0}E_i^l(t) - z_{ij}E_{ij}^{off}(t) + \lambda$ is awarded. This reflects the energy consumed on the mobile device $E_i^l(t)$ and the edge server $E_{ij}^{off}(t)$, along with a reward λ for meeting the latency constraints. If the task's completion time fails to meet its latency constraints, a penalty of $-\mu$ is awarded, representing a negative reward. This reward function is designed to incentivize the agent to perform task offloading and resource allocation to satisfy the latency demand and minimize energy consumption. This paper aims to maximize the number of served users and minimize the average energy consumption of completed tasks while meeting task latency constraints. Therefore, we pay special attention to the cumulative reward during the time period T, which is represented as $R = \sum_{t=0}^{t=T} r(s_t, A_k)$. This cumulative reward can indirectly reflect the number of completed tasks during the time period. Specifically, the more tasks are successfully processed, the more the agent obtains positive immediate rewards. Furthermore, the lower the energy consumption generated by each agent's action, the higher the positive immediate reward it obtains. Therefore, the higher the number of successfully processed tasks by the agent and the lower the energy consumption generated per offloaded task, the higher the cumulative reward obtained. Therefore, the reward function is closely related to our optimization objectives in this paper.

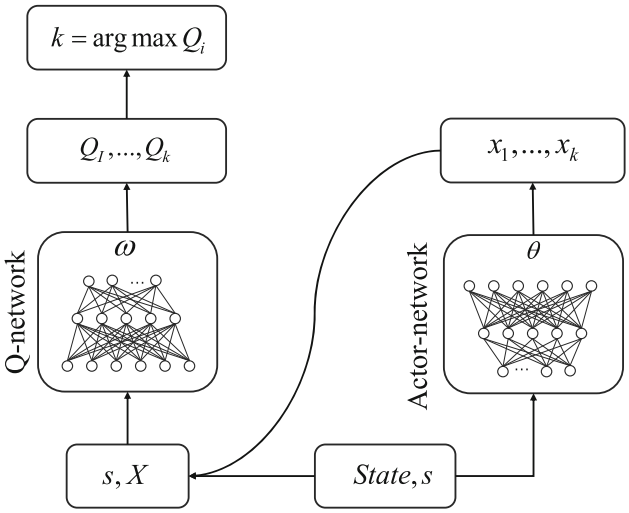


Fig. 2 Algorithmic structure of *P-DQN* [25]

– *The state transition probability function* we define it as: $P_{ss'}^a = \mathbb{P}\left[s_{t+1} = s' \mid s_t = s, k, x_k\right]$, where $s_t, s_{t+1} \in S$ and $A_k = \{k, x_k\} \in A$.

4.2 Task offloading and resource allocation strategy based on P-DQN

Currently, deep reinforcement learning algorithms are widely used to address sequential decision-making problems [23, 24]. Additionally, in an environment with multiple edge servers, the task offloading and resource allocation problems involve a mixed action space, where task offloading actions are discrete, and resource allocation actions are continuous. To tackle this challenge, we adopt the Parameterized Deep Q-Network (*P-DQN*) [25] algorithm. Specifically, *P-DQN* combines the advantages of traditional Deep Q-Network (*DQN*) and Deep Deterministic Policy Gradient (*DDPG*) algorithms. It utilizes an Actor network to determine the parameters of the continuous action space and inputs these continuous action parameters and the environment state into a Critic network to obtain the optimal discrete actions and corresponding continuous action parameters, where the structure is shown in Fig. 2. Therefore, the *P-DQN* algorithm is chosen for solving problems with a mixed action space, to avoid exhaustive searches over multiple continuous action parameters when seeking the optimal discrete actions.

In the algorithm, it takes global state information s from the MEC environment into the Actor network with parameter θ as inputs, and then outputs the continuous parameter of all discrete actions $x_1, x_2, \ldots, x_K$. Next, state s and all continuous parameter vectors are fed into the Q network with parameter ω. Finally, the Q network outputs the Q values

of all discrete actions and executes the discrete action corresponding to the largest Q value. In the following, we will describe the controller-managed task offloading and resource allocation process, in connection with the PAMDP and the *P-DQN* algorithm.

For action $a \in A$, we use k_t to denote the offloading action at the current time slot and use x_{k_t} to denote the corresponding continuous action parameter. The action value function can be expressed as $(s, a) = Q(s, k, x_k)$. *P-DQN* algorithm uses a deep neural network $Q(s, k, x_k; \omega)$ to estimate the action value function $Q(s, k, x_k)$, where ω represents the weight of the Q network. Furthermore, *P-DQN* estimates $x_k^Q(s)$ using the determined Actor network $x_k(\cdot; \theta) : S \rightarrow \mathcal{X}_k$, where θ represents the weight of this network. That is when ω is fixed, the goal of *P-DQN* is to find θ, such that:

$$Q(s, k, x_k(s; \theta); \omega) \approx Q(s, k, x_k; \omega) \tag{20}$$

where the value of ω is obtained by the least mean squares of gradient descent. The weights of the Q network and the Actor network are denoted by ω_t and θ_t, respectively. Then the objective value y_t can be expressed by the formula as:

$$y_t = r_t + \max_{k \in [K]} \gamma Q(s_{t+1}, k, x_k(s_{t+1}, \theta_t); \omega_t) \tag{21}$$

where s_{t+1} is the next state after the executed action $a = (k, x_k)$, $Q(s_{t+1}, k, x_k(s_{t+1}, \theta_t); \omega_t)$ can be obtained through the target network, and γ is discount factor.

Then, using the stochastic gradient descent method, we update the neural network weights and the loss function of the Q network is expressed as:

$$l_t^Q(\omega) = \frac{1}{2}[Q(s_t, k_t, x_k; \omega) - y_t]^2. \tag{22}$$

Similarly, the loss function of the Actor network is expressed as:

$$l_t^\Theta(\theta) = -\sum_{k=1}^{K} Q(s_t, k, x_k(s_t; \theta); \omega_t). \tag{23}$$

Specifically, we use step length α_1 and α_2 to update the weight of the Q network and the Actor network.

The overview of *P-DQN* algorithm is shown in Algorithm 1. Specifically, the algorithm first takes the maximum computing capability of all edge servers, the dynamically arrived task information H_i, and the user's location coordinate p_i^u as inputs. Line 1 indicates the initialization of the maximum number of training times, random noise function and parameters, the size of the experience pool, the number of samples for gradient descent, the Q network and the Actor network. Line 3 indicates that the state of the MEC environment is obtained at that time. Line 5 indicates inputting the obtained state into the Actor network and then outputting the continuous action parameters. Line 6 indicates that the computing resources and states are input into the Q network to obtain the Q value corresponding to each offloading action, and select the corresponding discrete action to execute according to the ϵ-greedy strategy. Line 7 indicates that users execute the selected action a_t and get reward and enter into the next state. Lines 8–10 indicate that the resource allocation result is recorded in a list and stored in the experience pool $\mathcal{D}$, and the computing resources are released at the end of time slot t. Line 11 indicates the state enters the next state s_{t+1}. Lines 12 indicates that Φ samples are taken from the experience replay pool $\mathcal{D}$ to calculate the objective value function y_t. Line 13–14 indicate that the weights of the Q network is updated as well as the Actor network using the random gradient method.

Algorithm 1 Task offloading and resource allocation strategy algorithm based on **P-DQN**

Input: Maximum computing capability F of the edge server, task information H_i and user's location coordinate p_i^u
Output: Task offloading and resource allocation strategy π with parameters ω and θ
1: Initialize weights ω and θ of the Q network and the Actor network, maximum number of iterations Γ, random noise function $\mathcal{N}$, initialize experience replay pool $\mathcal{D}$, the number of samples for gradient descent Φ, the update parameters α_1 and α_2 for the Q network and the Actor network
2: **for** $i = 1 \rightarrow \Gamma$ **do**
3: The central controller gets the state values of the MEC environment s_t
4: **for** $t = 1 \rightarrow T$ **do**
5: Calculate continuous action parameters: $x_k \leftarrow x_k(s_t, \theta_t) + \mathcal{N}$
6: Select the discrete action $a_t = (k_t, x_k)$ according to strategy ϵ-greedy:
$$a_t = \begin{cases} \text{random discrete action}, rnd < \epsilon \\ (k, x_k), k = \arg max_{k \in [K]} Q_\omega, rnd \geq \epsilon \end{cases}$$
7: Execute action a_t and observe rewarded r_t and next state s_{t+1}
8: Store the resource allocation result of the edge server in the record list
9: Store (s_t, a_t, r_t, s_{t+1}) in replay pool $\mathcal{D}$
10: Update the record list of edge servers and release the returned computing resources
11: $s_t \leftarrow s_{t+1}$
12: Take ϕ samples (s_j, a_j, r_j, s_{j+1}) from the replay pool $\mathcal{D}$ and calculate target value y_t
13: Calculate the gradients of $\nabla_\omega l_t^Q(\omega)$ and $\nabla_\theta l_t^\Theta(\theta)$ according to Eq.22 and Eq.23
14: Update the weights of the Q network: $\omega_{t+1} \leftarrow \omega_t - \alpha_1 \nabla_\omega l_t^Q(\omega_t)$, and the Actor network: $\theta_{t+1} \leftarrow \theta_t - \alpha_2 \nabla_\theta l_t^\Theta(\theta_t)$
15: **end for**
16: **end for**

5 Experiments

In this section, we experimentally evaluate the task offloading and resource allocation strategy proposed in the appeal based on the *P-DQN* algorithm. We first introduce the dataset and parameter settings, and then introduce the benchmark algorithms and evaluation metrics. We finally discuss the experimental results.

5.1 Dataset and parameter settings

Considering that real datasets usually do not consider scenarios with multiple edge servers overlapped with each other, and those datasets also do not include intensive task data that are dynamic and latency-sensitive. Therefore, we conduct experiments using synthetic datasets. Specifically, we set the experiment's area as 200 m × 200 m. For users, we adopt the Random Waypoint (RWP) [26] model to produce the movement trajectory of users, because this model is frequently used in mobile network simulation. Similar to work in [12, 27], we assume that the data size of the user's task, the total computing resources required to complete the task, and the maximum completing task time are independent and identically drawn from a uniform distribution within $d_i \sim U(8000, 10{,}000)$ kbits, $c_i \sim U(7000, 10{,}000)$ MHz and $\tau \sim U(3, 5)$ s, respectively. At the same time, the computing capability $fl_{\max}$ of the user's mobile device is 2 GHz.

For edge servers, in order to ensure full usage of edge server computing resources and avoid incurring excessive deployment costs, we set up $M = 3$ edge servers e_1, e_2 and e_3, and the coordinates of each edge server are $p_1^e = (50, 50)$, $p_2^e = (150, 50)$ and $p_3^e = (100, 100)$ respectively. The edge server service coverage radius β, bandwidth B and computing capability F are set to 100 m, 15 MHz and 4 GHz respectively.

For an environment with multiple edge servers, we divide the time into $T = 100$ time slots. We assume that the transmission power p_{trans} from the user's mobile device to the edge server is 0.5 W, the channel gain power h is 1.02×10^{-13}, the mobile device-edge server noise power is 10^{-13} W, the energy consumption factor of user's mobile device k_l and the energy consumption factor of edge server k_e are 10^{-27} and 10^{-29} respectively. The experimental parameters of this paper are shown in Table 3.

Then, we describe the parameter settings related to reinforcing learning. In *P-DQN* algorithm, both the Actor network and Q network adopt a two-layer neural network model, and the number of neurons in each layer is 256 and 128, respectively. Meanwhile, to enhance the nonlinear processing capability of the networks, both the Actor network and the Q network adopt the ReLU (Rectified Linear Unit) activation function. During the training process, the discount factor γ is set to 0.95, the software update factor is set to 0.001 for both Actor and Q networks, and the learning rates are set to = 0.001 and 0.0001 for both Actor and Q networks, respectively. We use the Adam optimizer to update the networks' weights to optimize the training process. At the same time, we set the size of the experience pool to 40,000, and the number of samples used for gradient descent ϕ is set to 256. The initialization of the network is also an important step. We initialize the parameters of the Actor network and the Q network to 0.5. The maximum number of iterations for the whole experiment was is to 5000. After training, we observe that the algorithm converges at about 2300 rounds. The code is released at https://github.com/Anya999999/RL_edge-computing.

The experiments in this paper are conducted in the environment with the following specifications: CPU model AMD Ryzen 7 5800 H with 8 cores, GPU model NVIDIA GeForce RTX 3060 with 6 GB memory, CUDA version 11.6, operating system Ubuntu 18.04.4 LTS, Python version 3.6.13 and PyTorch version 1.10.2. In addition, we repeat the experiments 10 times and compute the average values as the experimental results.

5.2 Benchmark algorithms and evaluation metrics

To evaluate the effectiveness of the proposed task offloading and resource allocation strategies in an environment with multiple edge servers, we consider five strategies as the benchmark, which are as follows:

- *Random* this strategy randomly determines whether user tasks are computed locally or executed on an edge server.
- *Greedy* this strategy offloads user tasks to the edge server with the highest available computing resources. If there are no free edge servers, the task is executed locally.
- *Nearest offloading (NO)* this strategy offloads user tasks to the nearest edge server for processing. If the nearest edge server does not have sufficient resources, the task is executed locally.
- *HTR* this strategy uses a heuristic algorithm to make the task offloading decision and then the edge server assigns computing resources to the task, which is similar to the work in [28].
- *PA-DDPG* this strategy is an improved reinforcement learning strategy based on the Deep Deterministic Policy Gradient (*DDPG*) [29] algorithm.

In this paper, we intend to maximize the number of served users and minimize the average energy consumption for all completed tasks, while meeting task latency constraints. Therefore, we consider the following three metrics for evaluation, which are the total number of served users by edge

Table 3 Experimental parameters

Parameter	Description
$T = 100$	Total time slot
$M = 3$	Number of edge servers
$d_i \sim U(8000, 10{,}000)$ kbits	Data size of the task
$c_i \sim U(7000, 10{,}000)$ MHz	Total computing resources required to complete the task
$\tau \sim U(3, 5)$ s	Maximum completion time acceptable to the user
$fl_{\max} = 2$ GHz	The maximum computing capability for users' mobile devices
$\beta = 100$ m	Service coverage radius of edge server
$B = 15$ MHz	Channel bandwidth of edge servers
$F = 4$ GHz	The maximum computing capability of edge servers
$p_{trans} = 0.5$ W	Transmission power
$h = 1.02 \times 10^{-13}$	Channel gain power
$\sigma^2 = 10^{-13}$ W	Noise power
$k_l = 10^{-27}$	Energy consumption factor of users' mobile devices
$k_e = 10^{-29}$	Energy consumption factor of edge servers

servers, the average energy consumption to accomplish all tasks, and the average service time to accomplish all tasks.

5.3 Experimental analysis

To evaluate the effectiveness of our strategy, we considered the impact of three parameters on the evaluation metrics: the number of users N, the computing capability F of the edge server, and the bandwidth B of the edge server. Then, we conducted three separate sets of experiments to analyze.

5.3.1 The impact of the number of users' tasks on different metric

Firstly, we show the performance of different task offloading and resource allocation strategies with different numbers of users. As an example, we set $F = 4$ GHz, $B = 15$ MHz, the number of selected users to 50, 100, 150, and 200, respectively, and the experimental results are shown in Fig. 3a–c.

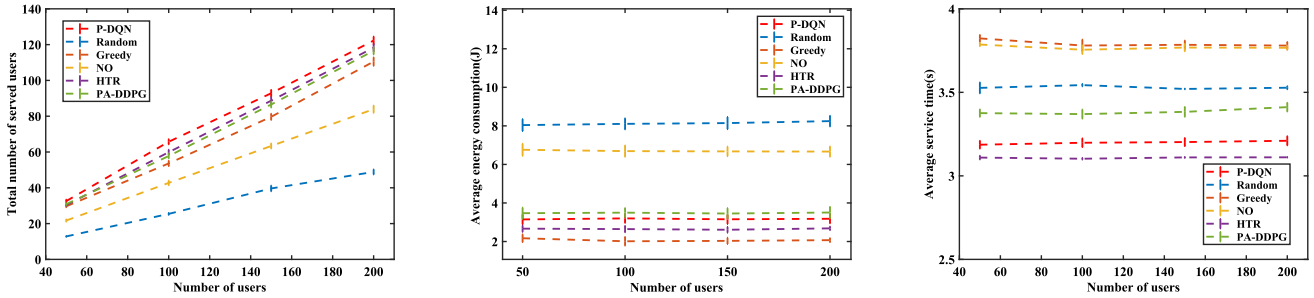
In Fig. 3a, we find that the *P-DQN* strategy consistently outperforms the other five baseline strategies in terms of the number of served users across different user numbers. *HTR*, *PA-DDPG* and *Greedy* strategies exhibit similar performance in terms of the total number of served users, but they serve fewer users compared to *P-DQN* strategy. Although *PA-DDPG* strategy simultaneously outputs a combination of discrete and continuous actions for further optimization, it may overlook the relationship between task offloading actions and resource allocation actions in the current research problem. *HTR* and *Greedy* strategies prioritize utilizing the computing resources of edge servers to process users' tasks. However, since each task is highly sensitive to latency, these two strategies are limited in the number of users they can serve simultaneously within a given time.

In Fig. 3b, we find that the average energy consumption remains relatively constant as the number of users increases. This is because users arrive dynamically, and once their tasks are completed, the corresponding computing resources are released, minimizing the overall energy consumption. However, among the strategies, *Greedy* and *HTR* strategies exhibit lower average energy consumption. This is because it consistently provides service to users in a manner that minimizes energy consumption. Although *P-DQN* strategy has a slightly higher average energy consumption compared to *Greedy* and *HTR* strategies, it serves a greater number of users. This trade-off between average energy consumption and the number of served users shows the effectiveness of *P-DQN* strategy in maximizing the number of served user while still maintaining average energy consumption at a reasonable level.

In Fig. 3c, we find that the average service time remains relatively constant as the number of users increases. Among the strategies, *HTR* strategy exhibits the shortest average service time. This is because *HTR* strategy prioritizes serving tasks in a manner that minimizes both overall latency and energy consumption. *P-DQN* strategy has a slightly higher average service time compared to *HTR* strategy. However, in subsequent experiments, it is observed that the performance gap between *P-DQN* and *HTR* decreases as the computing capability and bandwidth of the edge server increase. Additionally, the number of users served by *P-DQN* strategy remains higher than that of *HTR* strategy.

5.3.2 The impact of edge server computing capability on different metrics

In the second setting, we set the number of users $N = 100$, the bandwidth $B = 15$ MHz, and the computing capability



(a) Total number of served users (b) Average energy consumption (c) Average service time

Fig. 3 The impact of the number of users on the experimental metrics

F of the edge server to 4 GHz, 5 GHz, 6 GHz and 7 GHz respectively. The experimental results are shown in Fig. 4a–c.

As shown in Fig. 4a, we find that as the computing capability of the edge server increases, the total number of served users using various strategies also increases. *P-DQN* strategy consistently serves the highest number of users compared to other strategies. This is because *P-DQN* strategy can make efficient task offloading and resource allocation decisions based on task latency constraints and the remaining resources of the edge servers.

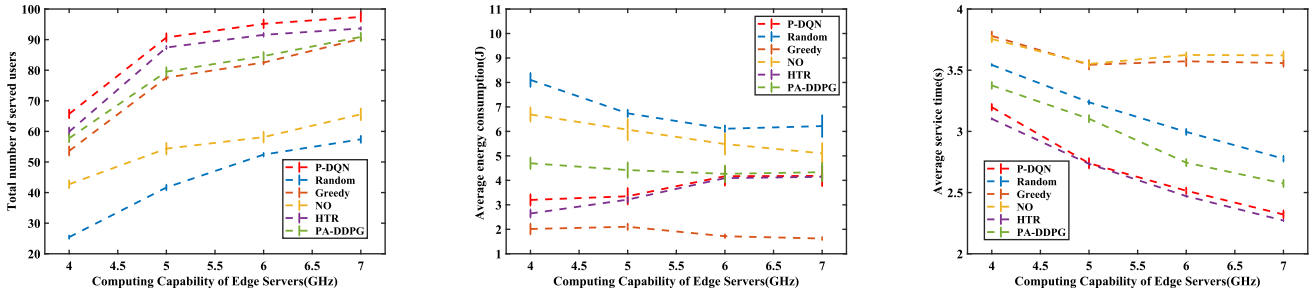
In Fig. 4b, as the edge server's processing power rises, we find that *HTR* and *P-DQN* strategies reveal that the average energy consumption increases slowly with the increased computing capability. The reason is that in order to shorten the task completion time and increase the number of served users, more tasks are offloaded to edge servers for processing in order to free up more computing resources for forthcoming tasks. Although *Greedy* strategy has the lowest average energy consumption, it serves far fewer users compared to *P-DQN* and *HTR* strategies.

In Fig. 4c, we find that as the computing capability of the edge server increases, the task offloading and resource allocation strategies of *HTR*, *P-DQN*, *PA-DDPG*, and *Random* result in a reduction in average service time. Among these strategies, both *HTR* and *P-DQN* show approximately equal and the shortest average service time. This is because these strategies allocate more computing resources for the tasks, thereby shortening the task completion time, at the cost of increased average energy consumption of the edge servers. Both *NO* and *Greedy* strategies have approximately equal and the longest average service time. This is because *NO* strategy offloads tasks to the nearest edge server and allocates computing resources that are equivalent to *Greedy* strategy. However, when *NO* strategy executes a task on its own mobile device, it utilizes all the computing resources of the mobile device to complete the task, resulting in a similar average service time as the *Greedy* strategy but higher average energy consumption.

5.3.3 The impact of edge server bandwidth on different metrics

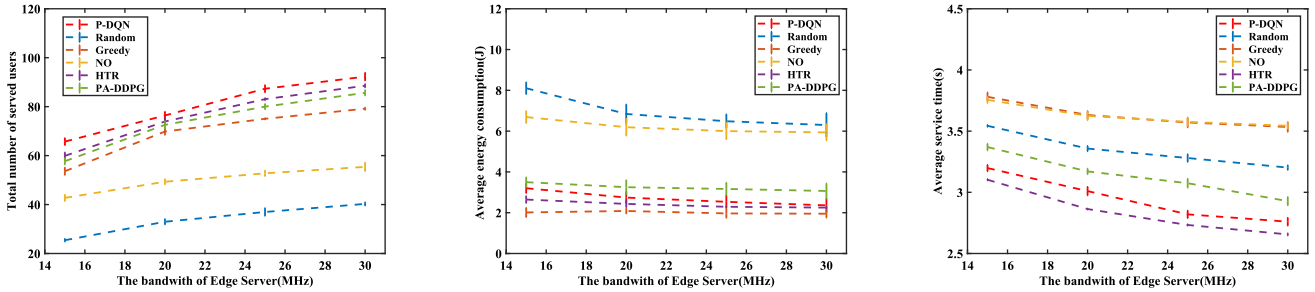
In the third setting, we analyze the experimental results by changing the bandwidth of the edge server. We set the number of users $N = 100$, the computing capability $F = 4$ GHZ, and the bandwidth B of the edge server to 15 MHz, 20 MHz, 25 MHz, and 30 MHz, respectively. The experimental results are shown in Fig. 5a–c. In Fig. 5a, we find that *P-DQN* strategy still serves the highest total number of users compared to other strategies. As the edge server's bandwidth increases, so do the number of served users by different strategies, while average energy consumption and average service time decrease (as shown in Fig. 5b, c). This is because as the bandwidth increases, the task transmission time also decreases, reducing the task completion time. Therefore, the edge server can serve more tasks in a limited amount of time.

In general, *Random* strategy is to randomly offload tasks to the user's device or to an edge server for processing. Therefore, this strategy is unable to meet the time latency constraints of the arrived tasks and performs poorly. *Greedy* strategy focuses solely on lowering the energy consumption of edge servers. This strategy cannot perform well in terms of the number of served users and the average service time, even though it has the lowest energy consumption. *NO* strategy can offload tasks to the nearest edge server for processing. Even though its transmission time decreases as the bandwidth of the edge server increases, it always has to wait for the nearest edge server, i.e. the waiting time is not improved, the total service time is large, the energy consumption of the server is high, and the number of served users is not great. *HTR* strategy reduces latency and energy consumption, but the total number of served users is not the largest. *PA-DDPG* strategy converts discrete actions such as task offloading into continuous actions and optimizes them simultaneously, without considering the relationship between task unloading and resource allocation actions in the hybrid action space, which restricts its ability to make effective task unloading and resource allocation decisions. As a result, it exhibits poor



(a) Total number of served users (b) Average energy consumption (c) Average service time

Fig. 4 The impact of edge server computing capability on the experimental metrics



(a) Total number of served users (b) Average energy consumption (c) Average service time

Fig. 5 The impact of edge server bandwidth on the experimental metrics

performance in terms of the number of served users, average energy consumption, and average service time. *P-DQN* strategy considers both latency and energy usage, and thus it outperforms other strategies in terms of the total number of served users. This strategy also achieves good results in terms of average energy consumption and average service time.

6 Conclusion

In this paper, we examine the problem of task offloading and resource allocation in the MEC system with multiple edge servers. After modeling the problem as a parameterized action Markov decision process (PAMDP), we present a task offloading and resource allocation strategy that is based on deep reinforcement learning *P-DQN*. In addition, we conduct extensive experiments to evaluate the proposed strategy against five typical benchmark strategies in terms of the number of served users, the average energy consumed and the time required to complete all tasks. The experimental results show that our strategy can take into account the remaining resources of each edge server as well as the computing capability of mobile devices to make effective task offloading and resource allocation decisions for the dynamically arrived tasks. As a result, within the task latency constraint, the algorithm is able to maximize the number of served users while simultaneously minimizing the average energy consumption of all tasks that have been completed. Our work can provide valuable insight for practical task offloading and resource allocation strategy design.

As the number of edge servers increases, the algorithm we use may be challenged with a dramatic increase in the state action space, which will limit the scalability of our solutions. Therefore, in the future, we will consider extending the method to the complex setting with more edge servers. Moreover, considering the collaboration among multiple edge servers, we will investigate the load-balancing problem among multiple edge servers.

Acknowledgements This paper was funded by the Humanity and Social Science Youth Research Foundation of Ministry of Education of China (Grant No. 19YJC790111) and the Philosophy and Social Science Post-Foundation of Ministry of Education of China (Grant No. 18JHQ060).

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Ananthanarayanan G, Bahl P, Bodík P et al (2017) Real-time video analytics: the killer app for edge computing. Computer 50(10):58–67. https://doi.org/10.1109/MC.2017.3641638
2. Elbamby MS, Perfecto C, Bennis M et al (2018) Toward low-latency and ultra-reliable virtual reality. IEEE Netw 32(2):78–84. https://doi.org/10.1109/MNET.2018.1700268
3. Premsankar G, Di Francesco M, Taleb T (2018) Edge computing for the internet of things: a case study. IEEE Internet Things J 5(2):1275–1284. https://doi.org/10.1109/JIOT.2018.2805263
4. Abbas N, Zhang Y, Taherkordi A et al (2017) Mobile edge computing: a survey. IEEE Internet Things J 5(1):450–465. https://doi.org/10.1109/JIOT.2017.2750180
5. Wang J, Zhao L, Liu J et al (2019) Smart resource allocation for mobile edge computing: a deep reinforcement learning approach. IEEE Trans Emerg Top Comput 9(3):1529–1541. https://doi.org/10.1109/TETC.2019.2902661
6. Tran TX, Pompili D (2018) Joint task offloading and resource allocation for multi-server mobile-edge computing networks. IEEE Trans Veh Technol 68(1):856–868. https://doi.org/10.1109/TVT.2018.2881191
7. Fang F, Wang K, Ding Z et al (2021) Energy-efficient resource allocation for NOMA-MEC networks with imperfect CSI. IEEE Trans Commun 69(5):3436–3449. https://doi.org/10.1109/TCOMM.2021.3058964
8. Lyu X, Tian H, Sengul C et al (2016) Multiuser joint task offloading and resource optimization in proximate clouds. IEEE Trans Veh Technol 66(4):3435–3447. https://doi.org/10.1109/TVT.2016.2593486
9. Chen Y, Zhang N, Zhang Y et al (2019) Energy efficient dynamic offloading in mobile edge computing for internet of things. IEEE Trans Cloud Comput 9(3):1050–1060. https://doi.org/10.1109/TCC.2019.2898657
10. Ning Z, Dong P, Kong X et al (2018) A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things. IEEE Internet Things J 6(3):4804–4814. https://doi.org/10.1109/JIOT.2018.2868616
11. Yan J, Bi S, Zhang YJA (2020) Offloading and resource allocation with general task graph in mobile edge computing: a deep reinforcement learning approach. IEEE Trans Wirel Commun 19(8):5404–5419. https://doi.org/10.1109/TWC.2020.2993071
12. Li J, Gao H, Lv T et al (2018) Deep reinforcement learning based computation offloading and resource allocation for MEC. In: 2018 IEEE wireless communications and networking conference (WCNC). IEEE, pp 1–6. https://doi.org/10.1109/wcnc.2018.8377343
13. Bi S, Huang L, Wang H et al (2021) Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks. IEEE Trans Wirel Commun 20(11):7519–7537. https://doi.org/10.1109/TWC.2021.3085319
14. Li H, Xu H, Zhou C et al (2020) Joint optimization strategy of computation offloading and resource allocation in multi-access edge computing environment. IEEE Trans Veh Technol 69(9):10214–10226. https://doi.org/10.1109/TVT.2020.3003898
15. Deng Y, Chen Z, Chen X et al (2021) Throughput maximization for multiedge multiuser edge computing systems. IEEE Internet Things J 9(1):68–79. https://doi.org/10.1109/JIOT.2021.3084509
16. Xia S, Yao Z, Li Y et al (2021) Online distributed offloading and computing resource management with energy harvesting for heterogeneous MEC-enabled IoT. IEEE Trans Wirel Commun 20(10):6743–6757. https://doi.org/10.1109/TWC.2021.3076201
17. Qian L, Wu Y, Jiang F et al (2020) Noma assisted multi-task multi-access mobile edge computing via deep reinforcement learning for industrial internet of things. IEEE Trans Ind Inform 17(8):5688–5698. https://doi.org/10.1109/TII.2020.3001355
18. Jiang Q, Xu X, He Q, et al (2021) Game theory-based task offloading and resource allocation for vehicular networks in edge-cloud computing. In: 2021 IEEE International Conference on Web Services (ICWS). IEEE, pp 341–346. https://doi.org/10.1109/ICWS53863.2021.00052
19. Lyu X, Tian H, Ni W et al (2018) Energy-efficient admission of delay-sensitive tasks for mobile edge computing. IEEE Trans Commun 66(6):2603–2616. https://doi.org/10.1109/TCOMM.2018.2799937
20. Zhao H, Deng S, Zhang C, et al (2019) A mobility-aware cross-edge computation offloading framework for partitionable applications. In: 2019 IEEE international conference on web services (ICWS). IEEE, pp 193–200. https://doi.org/10.1109/ICWS.2019.00041
21. Du W, Lei Q, He Q, et al (2019) Multiple energy harvesting devices enabled joint computation offloading and dynamic resource allocation for mobile-edge computing systems. In: 2019 IEEE international conference on web services (ICWS). IEEE, pp 154–158. https://doi.org/10.1109/ICWS.2019.00035
22. Masson W, Ranchod P, Konidaris G (2016) Reinforcement learning with parameterized actions. In: Thirtieth AAAI conference on artificial intelligence
23. Gao H, Huang W, Duan Y (2021) The cloud-edge-based dynamic reconfiguration to service workflow for mobile ecommerce environments: a QoS prediction perspective. ACM Trans Internet Technol TOIT 21(1):1–23. https://doi.org/10.1145/3391198
24. Ma X, Xu H, Gao H et al (2022) Real-time virtual machine scheduling in industry IoT network: a reinforcement learning method. IEEE Trans Ind Inform 19(2):2129–2139. https://doi.org/10.1109/TII.2022.3211622
25. Xiong J, Wang Q, Yang Z, et al (2018) Parametrized deep q-networks learning: reinforcement learning with discrete-continuous hybrid action space. arXiv preprint arXiv:1810.06394. https://doi.org/10.48550/arXiv.1810.06394
26. Johnson DB, Maltz DA (1996) Dynamic source routing in ad hoc wireless networks. In: Mobile computing, pp 153–181. https://doi.org/10.1007/b102605
27. Dai Y, Zhang K, Maharjan S et al (2020) Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond. IEEE Trans Veh Technol 69(10):12175–12186. https://doi.org/10.1109/TVT.2020.3013990
28. Wang Z, Du H, Ye Q (2021) HTR: a joint approach for task offloading and resource allocation in mobile edge computing. In: ICC 2021-IEEE international conference on communications. IEEE, pp 1–6. https://doi.org/10.48550/arXiv.1511.04143
29. Hausknecht M, Stone P (2015) Deep reinforcement learning in parameterized action space. arXiv preprint arXiv:1511.04143. https://doi.org/10.1109/JIOT.2018.2868616