EURASIP Journal on Advances in Signal Processing

RESEARCH **Open Access**

A deep reinforcement approach for computation offloading in MEC dynamic networks

Yibiao Fan[1*] and Xiaowei Cai[1]

*Correspondence: 82006017@lyun.edu.cn

[1] School of Physics and Mechanical and Electrical Engineering, Longyan University, Longyan 361000, Fujian, China

Abstract

In this study, we investigate the challenges associated with dynamic time slot server selection in mobile edge computing (MEC) systems. This study considers the fluctuating nature of user access at edge servers and the various factors that influence server workload, including offloading policies, offloading ratios, users' transmission power, and the servers' reserved capacity. To streamline the process of selecting edge servers with an eye on long-term optimization, we cast the problem as a Markov Decision Process (MDP) and propose a Deep Reinforcement Learning (DRL)-based algorithm as a solution. Our approach involves learning the selection strategy by analyzing the performance of server selections in previous iterations. Simulation outcomes show that our DRL-based algorithm surpasses benchmarks, delivering minimal average latency.

Keywords: Edge servers, Dynamic users, Computation offloading, Dynamic tasks, Reinforcement learning

1 Introduction

Amidst the development of fifth generation (5G) networks and the rising popularity of mobile devices, applications such as artificial intelligence, big data processing, the burgeoning use of smartphones, augmented reality (AR) and natural language processing are generating an unprecedented volume of data streams [1–3]. These applications typically require substantial computational resources. Yet, mobile devices often have limited computing capabilities due to hardware constraints. Despite the advanced performance of contemporary devices, they fall short in addressing the needs of computation-heavy applications and the demand for low-latency, high-reliability communication.

To tackle the issue of inadequate computing resources on mobile devices, several solutions have emerged, with Mobile Cloud Computing (MCC) [4–7] being among the first. Cloud servers boast vast computational resources, which can effectively compensate for the limited capacity of mobile devices to handle intensive computational tasks. However, the use of MCC comes with its own set of challenges. For instance, the network topology of MCC means that cloud servers are typically situated at a considerable distance from mobile devices, necessitating a reliance on the network for cloud connectivity. This can result in reduced efficiency and a degraded user experience due to factors such as mobile network bandwidth and latency. Furthermore, MCC poses certain security risks; data and applications on mobile devices often contain sensitive information, creating a potential for data breaches during transmission to the cloud.

To mitigate these issues, Mobile Edge Computing (MEC) has gained traction [8–10]. The fundamental concept of MEC is to decentralize computational resources from the core network to the network's edge, by outfitting edge devices like base stations (BSs) with high-speed computational servers. This shift provides users with computationally intensive applications access to computational support in closer proximity. It is projected that, in the future, up to 75% of data generated by enterprises are likely to undergo processing at the edge of the network [11].

In MEC networks, application providers capitalize on the mobility of mobile devices to gather users' preferences and location data by tracking the devices' trajectories. This enables the providers to dynamically select the most suitable MEC server for each user, thereby decreasing both task processing latency and energy consumption. For instance, literature [12] introduces services that accommodate the random movement and task arrivals of multiple mobile terrestrial users by integrating unmanned aerial vehicles (UAVs) into the MEC framework. In literature [13], the system's profitability is optimized by strategically managing the pricing of MEC computation services, the amount of data offloaded, and the selection of MEC servers while acknowledging the dynamic and unpredictable nature of user behavior. The study in [14] demonstrates how a UAV-assisted MEC system can enhance overall stability and reduce both energy usage and computational delay by managing the UAVs' flight paths and fine-tuning the offloading ratio.

However, it is insufficient to only focus on user mobility; one must also account for the dynamic changes in the number of users accessing edge services. Most existing network architectures inaccurately assume a constant number of user accesses within their mathematical models [15–19]. In multi-user MEC systems, the complexity increases due to intricate resource competition and potential interactions among mobile terminals, on top of the natural limitations imposed by finite computational and communication resources. The arrival of tasks is unpredictable, occurring at various times and involving variable sizes depending on the application. This unpredictability, combined with the variety in task sizes, makes accurate forecasting a challenge. Furthermore, the dynamic nature of users' tasks, influenced by emergency procedures, mobility, and the uncertain operational state of the mobile terminal-including random tasks and transmission state-poses substantial challenges for effectively offloading applications to fully benefit from MEC. Emergency procedures are primarily influenced by the type of device, the user's sense of urgency or importance, and the critical nature of the computational task in emergency situations (e.g., myocardial infarction, heart attack, urgent applications, etc.). Users in such critical states should be given utmost priority, with their tasks being processed immediately. Hence, it is a significant challenge to incorporate the impact of user mobility, the stochastic nature of task arrivals, and the urgency of tasks into the modeling process.

Within edge computing networks, the offloading of tasks is categorized into two approaches: partial offloading and binary offloading, which depend on whether a computational task is divisible. In the binary offloading model, an indivisible task must be processed entirely either locally or on the MEC server [20–25]. With partial offloading, it is possible to offload portions of a task to the edge server by determining an optimal offloading ratio [26, 27]. Given the varying requirements of tasks, this study adopts the partial offloading approach. Given the variability of wireless channels, offloading all computational tasks may not always be beneficial. Conversely, opportunistic offloading that adapts to the fluctuating channel conditions can yield substantial performance improvements. The resulting challenge is that variations in backend application demands can alter the computational capacity available on edge servers. Therefore, developing a logical and efficient computational offloading strategy remains a formidable challenge.

Unlike cloud computing systems, edge servers typically possess constrained resources. Consequently, selecting the right edge server is a crucial component of the computation offloading process [28]. The research presented in literature [29] focuses on UAV-assisted mobile edge computing with the objective of minimizing system latency by simultaneously optimizing UAV flight paths, time slot allocation, and compute resource distribution. Xing et al. introduced a computational offloading strategy in [30], aimed at reducing the user's offloading latency through the combined optimization of offloading duration and task processing time. However, given that the coverage of edge servers is finite and users may move frequently, mobile users may transition across various edge server coverage zones. Inappropriate server selection can lead to increased latency and energy consumption, thereby degrading the user experience. The population of users accessing edge servers is constantly fluctuating. As a user offloads tasks, it escalates the workload of the corresponding edge server, impacting the computational costs for all other users connected to that server. This interdependence among users' choices complicates server selection further. The ongoing nature of user services, coupled with the dynamics introduced by user mobility, adds to the complexity of server selection. Thus, it becomes essential to estimate the long-term optimality of computation and communication expenses over a sequence of time slots within a dynamic setting.

Additionally, most existing research on time-variant challenges in MEC systems relies on conditions such as channel statistical data to precisely monitor and update network-wide channel information, which incurs significant signaling overheads, like the time-slotting strategy discussed in literature [31–35]. However, in edge environments, minimizing delay is imperative. The primary difficulty in multi-user edge systems is the allocation of limited communication and computational resources. Each endpoint generates tasks unpredictably, which, if not managed promptly, may cause network bottlenecks and queuing at the edge servers, ultimately diminishing system performance. To ensure timely task dispatch and execution among devices, our goal is to determine how to measure task state updates within the available time slots for mobile users with random movement and data arrival patterns to deliver computational services effectively.

In edge computing frameworks, system state transitions are primarily induced by elements such as the randomness of user engagement with the system, server workload fluctuations, and unpredictable task generation. These variables are not known beforehand, making it arduous to identify the optimal policy using conventional methods

[36–39]. To navigate these challenges, we employ deep reinforcement learning (DRL) techniques. DRL is capable of proposing the most appropriate action by processing vast amounts of high-dimensional raw data as input to the deep neural network, leveraging the deep neural network's robust approximation capabilities. It is not necessary to foresee state transitions in advance, as DRL excels in managing control in stochastic and dynamic environments. Instead, it directly assesses mobile user dynamics based on observed outcomes, in accordance with the current system state, to facilitate server selection.

Based on the observations outlined previously, this study concentrates on the collaborative optimization of offloading decisions and resource allocation for task execution in MEC with the objective of minimizing the latency across the entire MEC system. The key contributions of this study are summarized as follows:

- A mixed integer nonlinear programming model is presented to optimize task offloading and resource allocation decisions. We propose a time slot optimization scheme that accounts for a time-varying MEC system, characterized by dynamic and real-time changes. Mobile users initiate tasks with a certain probability that follows a uniform distribution. These tasks are unsynchronized, vary in size, and are generated by a constantly changing number of users. This study takes into account the stochastic nature of application requests from mobile users, as well as the unpredictable states of MTs, which include operational states and mobility patterns.
- We facilitate dynamic optimization of joint resource allocation and task offloading decisions. Unlike most existing studies that are static and do not update resource allocation synchronously with the offloading decision, this study considers the offloading strategy for computational tasks, varying user priorities, and the resource demand of users with uncertain transmission power. We define the corresponding problem as a mixed integer nonlinear optimization challenge to simultaneously optimize the offloading decisions of mobile users and their access to the network, aiming to minimize the long-term latency of the whole MEC system. We model the user's server selection decision as a Markov decision process, considering short intra-time slot resource optimization as well as long-time slot resource optimization. To address this, we propose a Deep Deterministic Policy Gradient (DDPG)-based algorithm, which is designed to adapt to dynamically changing user conditions.
- We conduct experimental simulations to assess the performance of our proposed algorithm and benchmark its effectiveness against existing algorithms.

In the following sections, we will delve into the system model, the proposed DRL-based server selection algorithm, and the results of our experimental simulations in detail.

2 Problem statement and formulation

2.1 System model

We consider a MEC system consisting of edge servers and mobile users, containing N users and K MEC servers as shown in Fig. 1. The set of users is denoted as $\mathcal{N} = \{1, 2, \ldots, n\}$. To provide computing services to the users, the MEC servers are deployed at the access points (APs). Furthermore, the time model is discrete. The

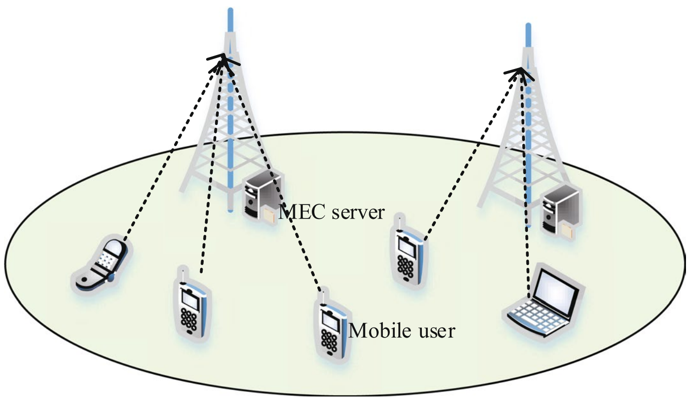

Fig. 1 System model

length of each time slot is t, where $t \in \{0, 1, 2, \ldots, \tau\}$. In each time slot, each user will generate only one computationally intensive task. In the system model, random arrival of tasks and real-time dynamic processing are used. The allocation of system spectrum and computational resources is uniformly scheduled by the MEC server.

Without loss of generality, we characterize the computational tasks arriving on the user i $(i \in \mathcal{N})$ in time slot t. are characterized $N_{\max}$ is the maximum number of users that the edge system can accommodate, with the number of access servers varying in each time slot, and denotes the task of the ith user at the moment of t, obeying a uniform distribution. Parameter elements $U_i(t) = \{S_i\text{t}, D_i(t), P_{i,\max}(t), \theta_{i,\max}(t), \lambda_i(t)\}$ represent The characteristics of the user i, where $S_i(t)$ represents the data size of the computational task, and $D_i(t)$ reflects the resources required to accomplish the task, i.e., the total number of CPU cycles required. $P_{i,\max}(t)$ is the maximum transmit power of the user. $\theta_{i,\max}(t)$ denotes the maximum tolerable delay of the task. λ_i is the priority of the user i, which is computed by the type of the device, and the degree of urgency/importance of the user. With a larger λ_i denoting that the matter is more urgent (which can be categorized or prioritized thresholds, with a greater than the threshold of 1, and a smaller value being sorted by the weighted order).

The edge server k feature is denoted as parameter element $\left\{C_k(t), C_k^r(t)\right\}$, where $C_k(t)$ represents the server processing capability, which is constant as the basic parameter of the edge server. But at moment t, the computational capability that the edge server can provide to the user is variable and is denoted by $C_k^r(t)$.

In the traditional time slot system scheme, the tasks generated by users in a certain time slot have to wait for all the users' tasks to be processed before the resources are released together to process the tasks in the next time slot; and thus, the new tasks generated during this waiting period are in a waiting state. This greatly reduces the user experience and demand. To reduce the latency, a novel scheme is proposed for the time slot system. The new scheme is shown in Fig. 2. At the moment t1, user 1 and user 2 generate a new task Task1 and task2, respectively. On that basis, the proposed system is able to dynamically adjust the program of mobile user access server in real-time. When ending the task of the previous time slot, if a new task arrives (generated by user 3), the server immediately releases the resources (of the previous time slot

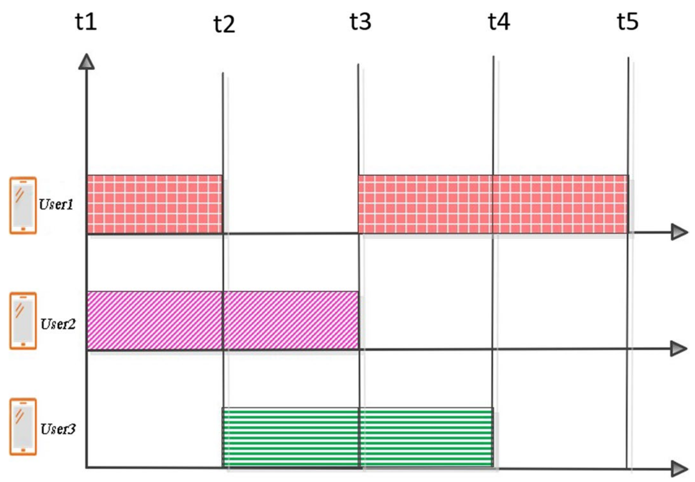

Fig. 2 Dynamic time slot allocation scheme

to process user 1) to process the newly arrived task, thus satisfying the low latency requirements of the edge system users.

2.2 Communications model

In the edge server system model, since edge servers are densely deployed, the coverage areas of various edge servers often overlap with each other, and a mobile user can be covered by multiple edge servers exhibit the capability of covering a mobile user at the same time. When user i device handles its computing tasks locally, the processing time is determined by computing capability of the user, which differs from users.

In practical environments, the task characteristics and computational capabilities of an edge server may be time-varying due to the changing environment. To compute task A_i(t), user i offloads a portion $\rho_{i,k}$ of the task A_i(t) to the edge server k over a wireless link, where $0 \leq \rho_{i,k} \leq 1$. All edge server k return the computation results to the user over dedicated feedback links. An offload vector of tasks for the user i is expressed as $\rho_i = [\rho_{i,0}, \rho_{i,1}, \ldots, \rho_{i,k}]$, where $\rho_{i,0}$ is local computation and $\rho_{i,k}$ is the proportion of tasks offloaded by the user i on the edge server k.

The user's transmit power affects the transmission data rate, on that basis, the user's transmit power should be optimized. $p_i = [p_{i,0}, p_{i,1}, \ldots, p_{i,k}]$ denotes the vector of the user's transmit power.

In this study, we study the problem of minimizing the delay in the communication and computation process to measure the system cost within each time slot. The uplink transmission rate from user i to the edge server k on the wireless link can be expressed as

$$R_{i,k}(t) = B\log_2(1 + \frac{p_i(t)h_i^k(t)}{I_{\mathrm{i}}(t) + \sigma(t)^2}), \quad \forall t, i, \tag{1}$$

where B is the bandwidth of the edge server channel, σ^2 is the noise power, $I_{\mathrm{i}}(t)$ is the interference caused to the user i by other users in the channel, $h_i^k(t)$ is the channel gain between the mobile user i and the edge server k at time slot t, and $p_i(t)$ is the uplink power of the user.

2.3 Calculation model

When tasks are offloaded from the user to the MEC, the complete task execution latency covers the communication latency between the user and the MEC as well as the computation latency at the MEC servers. Since each MEC server always handles other computational tasks simultaneously, the background workload may overload the MEC servers. Drawing upon the help of multiple MEC servers, users can select the associated MEC servers to minimize the computation delay. Thus, MEC server selection serves as a new dimension that reduces task execution latency and user's energy consumption. The computational tasks can be executed locally by the user or by computational offloading in a certain ratio $\rho_{i,k}$ on the MEC servers, and the latency is given as follows, respectively.

2.3.1 Locally computed delay

When user device i processes its computational tasks locally, the processing time is determined by its own computing capability, which is various for various users, the computational capability of the user is $f_i^{\mathrm{loc}}(t)$, and the computational capability of the edge server is expressed as by $f_k^{mec}(t)$. In each time slot the tasks are randomly generated, following the mean distribution $A_j(t) \sim U()$.Subsequently the user local computation time can satisfy:

$$T_i^{loc} = \frac{D_i(t)}{f_i^{loc}(t)}, \tag{2}$$

where $D_i(t)$ denotes the resources required to accomplish computational task $S_i(t)$.

2.3.2 Calculate offloading delay

When the user i offloads the computation task to the MEC server, the delay mainly consists of the uplink transmission time, MEC server task execution time, and the time for the output result to be transmitted from the MEC back to the user (which is negligible), then the uplink transmission time is

$$T_i^{up}(t) = \frac{S_i(t)\rho_{i,k}}{R_{i,k}(t)}, \tag{3}$$

The total delay for the MEC to process the offloading task for user i is

$$T_i^{m}(t) = T_i^{mec}(t) + T_i^{up}(t) = \left(\frac{D_i(t)}{\xi_{i,k}(t) f_k^{mec}(t)} + \frac{S_i(t)}{R_{i,k}(t)}\right)\rho_{i,k}. \tag{4}$$

where $\xi_{i,k}(t)$ represents the computing power allocated by the edge server k for the user i thereof.

For the entire edge system, the total delay for all users can be expressed as

$$\begin{aligned} T_i^{mec,loc}(t) &= \sum_{i=1}^{N_{\max}} \max\left\{T_i^{loc}, T_i^{m}(t)\right\} \\ &= \sum_{i=1}^{N} \max\left\{\frac{D_i(t)\left(1-\rho_{i,k}\right)}{f_i^{loc}(t)}, \frac{D_i(t)\rho_{i,k}}{\xi_{i,k}(t) f_k^{mec}(t)} + \frac{S_i(t)\rho_{i,k}}{R_{i,k}(t)}\right\}. \end{aligned} \tag{5}$$

2.4 Problem formulation

To simultaneously safeguard the task processing latency and computational cost, for the edge server collaborative computing system, the objective function is

$$\sum_{t=1}^{T}\sum_{i=1}^{N_{\max}} \lambda_i f\left(T_i^{mec,loc}(t), \theta_{i,\max}(t)\right), \tag{6}$$

where λ_i denotes the priority of the user accessing the server, and the larger the value of λ, the higher the priority of the user. $f\left(T_i^{mec,loc}(t), \theta_{i,\max}(t)\right)$ is the reward function. Our proposed optimization problem can be completely expressed as

$$\begin{array}{ll} \min \sum\limits_{t=1}^{T}\sum\limits_{i=1}^{N_{\max}} \lambda_i f\left(T_i^{mec,loc}(t), \theta_{i,\max}(t)\right) & \\ \text{s.t. } \mathrm{C1}: & a_{i,k} \in \{0,1\}, \forall i \in N, k \in K \\ \quad\;\; \mathrm{C2}: & 0 \leq \rho_{i,k} \leq 1, \forall i \in N \\ \quad\;\; \mathrm{C3}: & 0 < p_i \leq P_{\max} \\ \quad\;\; \mathrm{C4}: & \xi_{i,k}(t) \in (0,1] \end{array} \tag{7}$$

where C1 denotes the user's task offloading server selection, assuming that the user's task can only select one server. C2 denotes the offloading vector of user *i*. C3 ensures the constraint of uplink power, and C4 determines the computational resource allocation strategy. The optimization problem proposed in this study is a hybrid nonlinear programming challenge that is nonconvex and NP-hard. To address this problem, we need to determine the offloading decision vector for each time slot, which encompasses the choice of server for offloading, the offloading ratio, and the user's transmission power, all aimed at minimizing the total delay cost of the system while adhering to a specified delay constraint.

It is important to note that the offloading decision variables $a_{i,k}$, $\rho_{i,k}$ and p_i variables are dynamic. The system must gather information to ensure that offloading strategies and resource allocation decisions are informed by an overarching awareness of the network state. Furthermore, we explore a more realistic scenario where the pattern of task requests over time is not known in advance. Given the dynamic nature of the problem at hand, conventional optimization methods fail to deliver swift decisions in a constantly changing state, and the complexity of the algorithms scales up exponentially with the system model's expansion. Hence, we propose a DRL-based method to tackle the problem presented in this study.

3 Approach design

In this section, we conceptualize the challenge of minimizing service delay as a Markov decision process (MDP). Initially, we define the state, action, and reward functions within the MDP framework. Subsequently, we employ the DDPG algorithm to resolve the problem.

3.1 Markov decision process model

For each discrete time slot *t*, the agent ascertains the presence of a new user and the generation of a new task. Upon the creation of a new task, the agent collects environmental information such as the allocatable computing capacity of the MEC node, the data being transmitted by users currently accessing the service, and the power and state of the environment. The agent then selects an action following the relevant strategy, interacts with the environment to acquire an updated state, and receives a reward signal generated by the environment. The agent iteratively refines its strategy in response to the reward, accumulating rewards after each action until the strategy stabilizes. Given that the agent must consider both immediate and future rewards, the principal learning objective is to maximize cumulative rewards through the continuous refinement of its strategy. In our model, one of the users is designated as the intelligent agent; while, all other components of the edge computing system constitute the environment. Below, we provide a detailed account of the state space, action space, and reward function.

3.1.1 State space

The state in MDP is a space reflecting the environment, encompassing user state and edge computing server state. The state space is represented by $Z(t) = \{U(t), C(t), I(t)\}$, where the user state is $U(t) = \left\{U_1(t), U_2(t), \cdots, U_n(t)_{N_{\max}}\right\}$, $N_{\max}$ represent the maximum number of users that the edge system can accommodate.

User state: The state characteristics of the *i*st user can be expressed as $U_i(t) = \{S_i(t), D_i(t), P_{i,\max}, \theta_i(t), \lambda_i(t)\}$, where $0 < i \leq N_{\max}$, $S_i(t)$ represent the data size of the computational task, $D_i(t)$ indicate the resources required to complete the task, $P_{i,\max}$ is the maximum transmit power of the user. $\theta_i(t)$ is the delay requirement, the maximum tolerable time of the task. λ_i is the priority system of the user i which is determined by the type of the device, the degree of urgency/importance of the user, and the larger λ_i the greater the urgency, the greater the urgency of the matter. For instance, when user i has no access or access but no new task is generated, then there is $S_i(t) = 0$, $D_i(t) = 0$, $P_{i,\max} = 0$, $\theta_{i,\max}(t) = 0$, $\lambda_i(t) = 0$.

The state characteristics of an edge computing server can be represented as $C(t) = \left\{C_1^r(t), C_2^r(t), \ldots, C_K^r(t)\right\}$. C_k denotes the computing capability of the edge server k, $C_k^r(t)$ is the computing capability that the edge server k can provide to the user at time slot t. $C_k^{used}(t)$ is the computing capability of the edge server k that has been assigned other tasks at time slot t, and $C_k^r(t) = C_k - C_k^{used}(t)$.

Interference with other users when the environment user sends data is $I(t)$.

3.1.2 Action space

An agent aims to choose the offloading tactics for various users throughout each time slot. The offloading strategy $A(t) = \{X(t), \rho(t), P(t), \xi(t)\}$ can be divided into four parts:

1. $X(t)$ indicates that the user task offload selection. Here, we assume that user *i*'s task can only select one server, and

$$X(t) = \begin{pmatrix} x_{1,1}(t) & x_{1,2}(t) & \cdots & x_{1,k}(t) \\ x_{2,1}(t) & x_{2,2}(t) & \cdots & x_{2,k}(t) \\ \vdots & \vdots & \cdots & \vdots \\ x_{N_{mzx},1}(t) & x_{N_{mzx},2}(t) & \cdots & x_{N_{mzx},K}(t) \end{pmatrix}, \tag{8}$$

where$\sum_{j=1}^{K} x_{i,j}(t) = \begin{cases} 1, & user\ i\ has\ a\ task\ and\ the\ task\ is\ offloaded\ to\ edge\ server\ j \\ 0, & else \end{cases}$

2. $\rho(t)$ indicates the percentage of user tasks offloaded. $\rho(t) = \left(\rho_1(t), \rho_2(t), \cdots, \rho_{N_{\max}}(t)\right)$, $\rho_i(t) \in [0, 1]$ indicates the proportion of user *i*'s data and computation tasks uploaded to the edge computing server. When $\rho_i(t) = 0$, indicates that user *i*'s tasks are completed locally, and $\rho_i(t) = 1$ indicates that user *i*'s tasks are completed locally.
3. $P(t)$ indicates the transmit power of the user. $P(t) = \left(P_1(t), P_2(t), \cdots, P_{N_{\max}}(t)\right)$, $P_i(t) <= P_{i,\max}$ denotes the task transmit power for user *i*, and $P_{i,\max}$ denotes the maximum transmit power.
4. The computational capability allocated by the edge server to user tasks can be represented by matrix $\xi(t)$, i.e.,

$$\xi(t) = \begin{pmatrix} \xi_{0,1}(t) & \xi_{0,2}(t) & \cdots & \xi_{0,K}(t) \\ \xi_{1,1}(t) & \xi_{1,2}(t) & \cdots & \xi_{1,K}(t) \\ \vdots & \vdots & \vdots & \vdots \\ \xi_{N_{\max},1}(t) & \xi_{N_{\max},2}(t) & \cdots & \xi_{N_{\max},K}(t) \end{pmatrix}. \tag{9}$$

$\xi(t)$ must fulfill the following conditions: a) $C_j^r(t) = \xi_{0,j}(t) + \sum_{i=1}^{i=N_{\max}} x_{i,j}(t)\xi_{i,j}(t)$, where $\xi_{0,j}(t)$ is very critical. It indicates the computational capability reserved by the edge server for future tasks which can be compared by simulation with or without reservation. For example, a set of data $\xi_{0,j}(t) = 0$ and another set of normal training. b) $f_i^{Mec}(t) = \sum_{j=1}^{j=K} x_{i,j}(t)\xi_{i,j}(t)$, $f_i^{Mec}(t)$ denotes the computational capability obtained by user *i*.

3.1.3 Reward space

The reward function is pivotal as it delineates the overarching objective of the agent's learning journey. With each action completed, the agent garners a reward from the environment. This reward reflects the benefit of executing said action within the current state and, through sustained interaction, ultimately steers the agent toward refining its strategy to maximize cumulative gain. In light of the optimization challenge proposed, our aim is to minimize latency across the entire MEC system. Reinforcement learning endeavors to realize this by maximizing the sum of discounted rewards over time. As with any learning algorithm, during the training phase, once an action is taken, the corresponding reward is conveyed to the agent at time slot *t*. Based on the received reward, the agent updates its policy (π) toward the optimal policy-that is, the policy that consistently yields high rewards for actions taken across various environmental states. The reward issued to the agent is denoted by $r : Z \times A \to R$.

In this study, we design the following reward function

$$r_i(t) = \begin{cases} 0, & when\ \log_2\left(\frac{\theta_i(t)}{T_i(t)}\right) > 0 \\ \log_2\left(\frac{\theta_i(t)}{T_i(t)}\right), & else \end{cases} \tag{10}$$

When the reward function is $r_i(t)$, the system optimizes the objective function to minimize the service delay and increase the proportion of tasks that satisfy the delay qualification.

3.2 Deep reinforcement learning model design

and-error interactions with the environment, where state transitions and rewards are initially unknown. DRL-based server selection relies on gradient-based strategy learning. Within the context of this study, we need to ascertain whether long-term planning can be effectively executed in dynamic environments and how to manage high-dimensional state spaces efficiently. Subsequently, we will outline the resolution to these challenges. For neural network training, we have utilized the DDPG algorithm. This deterministic policy framework does not produce the likelihood of an action; instead, it outputs the specific numerical value of each dimension that corresponds to the action, thereby obviating the need for action sampling. Given that the training data is time-dependent, it can sometimes lead to slow convergence or even a lack of convergence in neural network training. To counteract this, we implement experience replay, a technique that disrupts temporal correlations to expedite convergence. In reinforcement learning, samples are sequentially correlated, presenting challenges, as neural networks function optimally with samples that are independent and identically distributed. Experience replay addresses the correlation issue inherent in sequential decision-making and enhances sample efficiency. Once the experience pool reaches a predetermined size, the oldest data is typically removed to ensure that the pool remains current. Algorithm 1 presents the proposed computational offloading algorithm for dynamic MEC networks based on deep reinforcement learning. The DDPG network structure is illustrated in Fig. 3.

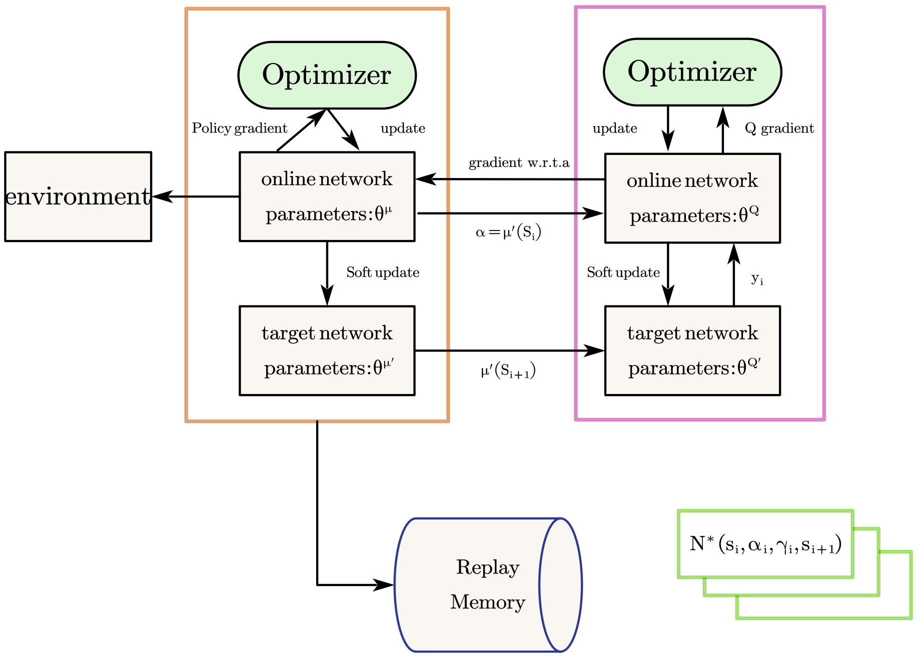

Fig. 3 DDPG network

Algorithm 1 Deep reinforcement learning-based computational offloading algorithm for MEC dynamic networks

```
1: Set hyperparameters: discount factor $\gamma$, learning rate of Actor network $\alpha_{\mathrm{a}}$, learning rate of Critic network $\alpha_c$.
2: Initialize Actor network parameters $\theta$ and Critic current network $w$; Actor target network $\theta' = \theta$, Critic target network $w' = w$, Empty experience replay collection $D$.
3: Repeat
4:    Randomly initialize the environment and observe the initial state of the environment $Z(1) = \{U(1), C(1), I(1)\}$.
   for $i = 1, 2, ..., T$, do
5:    Get action $z(t)$ in Actor's current network based on state $A(t) = \mu(z(t)) + n(t)$.
6:    Perform action $A(t) = \{X(t), \rho(t), P(t), \xi(t)\}$ on the environment.
7:    Get status $Z(t+1) = \{U(t+1), C(t+1), I(t+1)\}$ and reward $r(t+1)$.
8:    Put the quaternion store $\{z(t), A(t), Z(t+1), r(t+1)\}$ into the experience replay set $D$.
9:    Take $m$ random samples of data from the experience replay set $D$, i.e., $\{z(t), A(t), Z(t+1), r(t+1)\}, t = 1, 2, \cdots, m$.
10:   Calculate the mean squared loss function $J(w) = \frac{1}{m}\sum_{t=1}^{m}(y(t) - Q_w(z(t), A(t)))^2$ to update all parameters of Critic's current network by backpropagation of the gradient of the neural network $w$.
11: Calculation $J(\theta) = -\frac{1}{m}\sum_{t=1}^{m} Q_w(z(t), A(t))$ to update all parameters of Actor's current network by backpropagation of the gradient of the neural network $\theta$.
12: Update the target network: $w' \leftarrow \tau w + (1-\tau)w'$, $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$.
13: End for
```

The DDPG algorithm comprises four neural networks: the Actor network $\mu_\theta(t)$, the Critic network $\mu_Q(t)$, the Target Actor network $\mu_{\theta^1}(t)$, and the Target Critic network $\mu_{Q^1}(t)$. The workflow of the DDPG algorithm operates as follows:

1. Initialization: The Actor and Critic networks are initialized along with their respective target networks.
2. Sampling: The Actor network generates actions for a given environmental state, which are then executed in the environment to observe rewards and subsequent states.
3. Storage: The experiences, consisting of states, actions, and rewards, are stored in a replay buffer for future learning.
4. Training the Critic Network: A mini-batch of experiences is randomly sampled from the replay buffer. The Critic network evaluates these experiences, the Temporal Difference (TD) error is computed, and the network's parameters are updated through backpropagation to minimize this error.
5. Training the Actor Network: The gradient of the error calculated by the Critic network is used to update the parameters of the Actor network via backpropagation.
6. Updating the Target Networks: The parameters of the target networks are gradually adjusted toward the parameters of their respective current networks, using a soft update approach.

7. Loop: Steps 2–6 are repeated, continuously refining the network parameters until the algorithm converges.

DDPG, being a deterministic policy-based approach, requires sampling fewer data points, making the algorithm efficient. However, it may struggle with generalizing to unseen actions. To compensate for the action exploration ability sacrificed by the intelligent body, a random noise N is added to the selected action A at the strategy network to enhance the generalization. Ultimately, the expression for an action A that interacts with the environment is

$$A(t) = \mu(z(t)) + n(t),$$

where $n(t)$ is Gaussian white noise.

Next is the loss function for DDPG. For the Critic current network, the loss function is the mean square error, i.e.,

$$J(w) = \frac{1}{T}\sum_{i=1}^{T}\left(y(t) - Q_w(z(t), A(t))\right)^2. \tag{11}$$

In terms of the Actor current network, the loss function is

$$J(\theta) = -\frac{1}{T}\sum_{i=1}^{T} Q_w(z(t), A(t)). \tag{12}$$

Building upon the DQN algorithm, the DDPG algorithm introduces three significant enhancements:

First, DDPG improves the stability of learning by adopting the dual neural network architecture from DQN. This architecture involves two sets of neural networks-the primary networks for evaluation and the target networks for occasional updates of the parameters. DDPG distinguishes itself by employing a soft update method for the target networks, providing a more stable learning process.

Second, to address the issue of correlated and nonuniformly distributed samples, DDPG utilizes the experience replay mechanism, a concept borrowed from DQN. This mechanism preserves the data generated during the agent's interaction with the environment in a structured memory known as the experience replay buffer. During the learning phase, the algorithm samples a batch of experiences at random from the buffer to train the model. This method ensures a diversified learning experience, which is essential for the robust development of the policy.

The third enhancement addresses the exploration-exploitation dilemma, a fundamental challenge in reinforcement learning where the agent must balance the act of exploring new possibilities with leveraging existing knowledge. DDPG introduces exploration noise to this end. By adding stochastic noise, which often follows a Gaussian or uniform distribution, to the selected actions, the algorithm equips the agent with better exploration capabilities. This noise enables the agent to investigate uncharted areas of the state and action space more effectively, facilitating the discovery of optimal strategies.

Table 1 Simulations parameters

The notation	Physical meaning	Value
N	Mobile user	20
K	Edge server	4
f_k^{mec}	Server capacity (CPU cycles)	[10, 100] GHz
f_m^{loc}	User computing capacity (CPU cycles)	[0.5, 2] GHz
D	Amount of user-generated task data	[1, 5] Mbits
B	System Bandwidth	20 MHz
$P_{\max}$	Maximum uplink transmit power	0.2 W
σ^2	Background noise power	−100 dBm
$h_i^k(t)$	Channel gain	$127 + 30log(L)$ [41, 42]
t	Time slot	3 ms
$\theta_i(t)$	The maximum tolerable delay	3 s
C	Capacity of empirical replay buffer	500
U	Capacity of the small batch of samples	32
ε	Learning rate	0.1
ϕ	Reward	0.9 ms

4 Simulations and discussions

4.1 Simulation setup

In this section, extensive simulations are conducted to evaluate the performance of the proposed DDPG algorithm, and this algorithm is compared with benchmark algorithms.

A small cell with a radius of 0.3×0.3 km in a 5G mobile environment is considered, where there is K AP with MEC servers, and N mobile users with computation tasks exhibit random dispersion in the coverage area of the AP. We consider various users with various computational capabilities and the computing power exhibits a uniform distribution between 0.5 and 2 GHz. The MEC system is capable of leveraging the DSA technique for the allocation of the channel resources according to the demand of the terminals. Other simulation parameters are listed in Table 1.

4.2 Performance comparison

Figure 4 illustrates the convergence of the proposed DDPG-based learning method when the system accommodates 20 user terminals. Initially, the cumulative reward experiences minimal fluctuation. This is attributed to the user's lack of environmental knowledge at the outset, resulting in nearly random action selection. As the user aggregates sufficient samples over time, these samples are used to train the network. Overall, the DDPG-based method demonstrates robust performance, stabilizing after approximately 50 training sessions. It is evident that with an increasing number of training sessions, the system's cumulative reward swiftly escalates, enabling the effective learning of computational offloading strategies through ongoing interactions.

For a comparative analysis of performance, we introduce four benchmark algorithms: (a) A brute-force search to ascertain an approximate optimal solution (denoted as "Exhaustion"). (b) A strategy that prioritizes the offloading of tasks to MEC servers, distributing all communication and computation resources equally among users (denoted as "Offloading"). (c) A user-centric approach that favors local

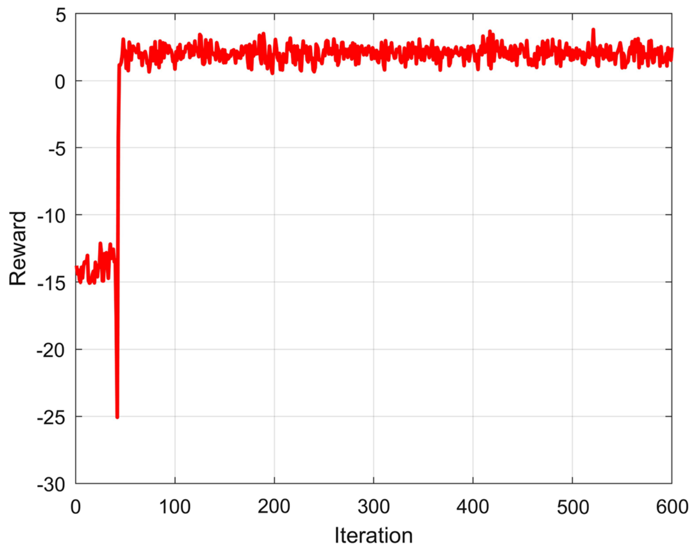

Fig. 4 Convergence of the proposed DDQN-based method

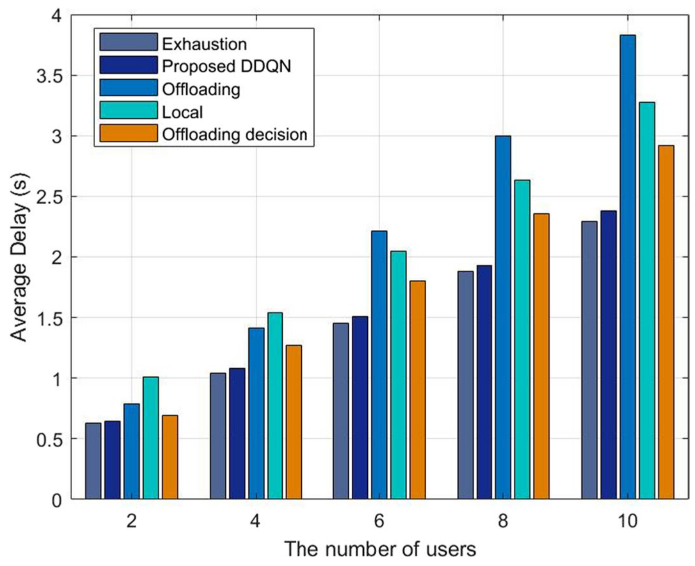

Fig. 5 Average delay versus the number of users

task execution with maximum tolerated latency (denoted as “Local”). (d) An optimization of offloading decisions that does not factor in the optimization of resource allocation (denoted as “Offloading Decision”).

Figure 5 presents a comparison of the proposed algorithm’s performance against these benchmarks in terms of average latency with more users. The latency for all algorithms escalates with the addition of more users. The exhaustive method serves as a benchmark for peak performance. The proposed DDPG-based method delivers performance closely aligned with this exhaustive approach. Notably, with eight users, the proposed algorithm significantly diminishes average latency by 20%, 33% and 55%

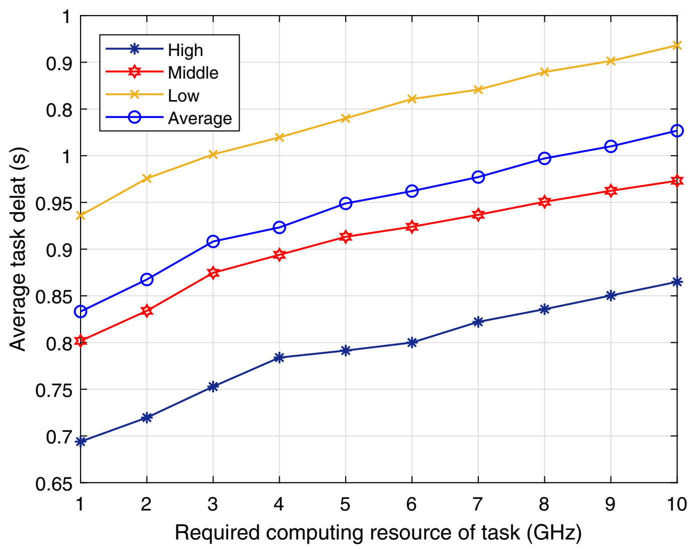

Fig. 6 Average delay versus task requirement

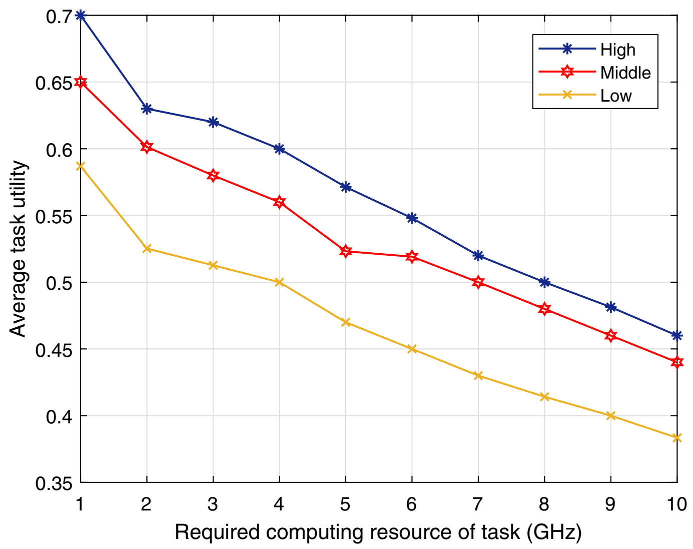

Fig. 7 Average utility versus task requirement

compared to the other three methods, respectively. Furthermore, the typical latency associated with the DDPG algorithm is also lower than those of the benchmark algorithms, suggesting the effectiveness of our proposed strategy.

The various tasks are categorized into three priorities based on the value of priority system λ_i. $0.75 < \lambda_i \leq 1$ for high priority, $0 < \lambda_i \leq 0.75$ for medium priority and $0.1 < \lambda_i \leq 0.4$ for low priority. The number of users is set to $N = 20$ and the input data are fixed to an average value of 200 kB.

Figure 6 depicts the latency of three priority tasks under varying computational task loads. As the computational load intensifies, the latency for all priority levels increases, with the high priority tasks experiencing the least latency and the low priority tasks the most. The average system latency exceeds that of high-priority tasks, indicating that reducing latency for high-priority tasks incurs increased latency for lower-priority ones. Figure 7 presents the average task utility for the three levels of prioritized tasks under various computational burdens. Our proposed approach not only ensures reduced system latency but also stratifies task priority effectively, allowing urgent tasks to be completed more swiftly by users with pressing needs.

5 Conclusions

In this study, we address the server selection problem within dynamic time slot schemes in MEC. To tackle the NP-hard challenges stemming from dynamic factors, we model the ongoing server selection issue as a MDP and introduce an algorithm based on DRL. Our DRL-based server selection algorithm accounts for user states, inter-user interference, and the processing capabilities of edge servers. We incorporate historical data and the dynamic nature of these elements through neural network encoding. Our simulation results indicate that the DDPG algorithm, developed as part of this study, consistently outperforms established benchmarks by delivering the lowest average latency.

Abbreviations

MEC	Mobile edge computing
MDP	Markov Decision Process
DRL	Deep Reinforcement Learning
5G	Fifth generation
BS	Base station
AR	Augmented reality
UAVs	Unmanned aerial vehicles
MINLP	Mixed integer nonlinear programming
DDPG	Deep Deterministic Policy Gradient algorithm
AP	Access point

Acknowledgements
This research was funded by the Fujian Provincial Natural Science Fund under Grant (2023J01967)

Additional files
This is as a reference to check the layout of the article as the authors intended.

Author contributions
The major writer of this study is Y.F., who put forward the main idea, carried out the simulations, and analyzed it. X.C. assisted the review of this study. All authors read and issued the approval of the final manuscript.

Funding
This research was funded by the Fujian Provincial Natural Science Fund under Grant (2023J01967).

Availability of data and materials
Not applicable.

Declarations

Competing interests
The authors declare that they have no competing interest.

Received: 7 December 2023 Accepted: 24 March 2024
Published online: 08 April 2024

References
1. J. Liu, L. Zhong, J. Wickramasuriya, V. Vasudevan, uWave: Accelerometer-based personalized gesture recognition and its applications. Pervasive Mob. Comput. **5**(6), 657–675 (2009)

2. A. Al-Shuwaili, O. Simeone, Energy-efficient resource allocation for mobile edge computing-based augmented reality applications. IEEE Wirel. Commun. Lett. **6**(3), 398–401 (2017)
3. W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
4. V. Farhadi et al., Service placement and request scheduling for data-intensive applications in edge clouds. In: *Presented at the IEEE INFOCOM 2019—IEEE Conference on Computer Communications* (2019)
5. L. Zhao, W. Sun, Y. Shi, J. Liu, Optimal placement of cloudlets for access delay minimization in SDN-based internet of things networks. IEEE Internet Things J. **5**(2), 1334–1344 (2018)
6. B. Shen, X. Xu, F. Dar, L. Qi, X. Zhang, W. Dou, Dynamic task offloading with minority game for internet of vehicles in cloud-edge computing. In: *Presented at the 2020 IEEE International Conference on Web Services (ICWS)* (2020)
7. H. Baraki, A. Jahl, S. Jakob, C. Schwarzbach, M. Fax, K. Geihs, Optimizing applications for mobile cloud computing through MOCCAA. J. Grid Comput. **17**(4), 651–676 (2019)
8. Y.C. Hu, M. Patel, D. Sabella, N. Sprecher, V. Young, Mobile edge computing—A key technology towards 5G. ETSI White Paper **11**(11), 1–16 (2015)
9. Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: the communication perspective. IEEE Commun. Surv. Tutor. **19**(4), 2322–2358 (2017)
10. Y. Zhang et al., Edge intelligence for plug-in electrical vehicle charging service. IEEE Netw. **35**(3), 81–87 (2021)
11. Varadharajan V, Mantri S, Shah B, et al. Emerging edge computing applications. In: *IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)* (2022). p. 1–4
12. Z. Yang, S. Bi, Y.-J.A. Zhang, Dynamic trajectory and offloading control of UAV-enabled MEC under user mobility. In: *Presented at the 2021 IEEE International Conference on Communications Workshops (ICC Workshops)* (2021).
13. S. Li, X. Hu, Y. Du, Deep reinforcement learning and game theory for computation offloading in dynamic edge computing markets. IEEE Access **9**, 121456–121466 (2021)
14. L. Zhang et al., Task offloading and trajectory control for UAV-assisted mobile edge computing using deep reinforcement learning. IEEE Access **9**, 53708–53719 (2021)
15. P.Q. Huang, Y. Wang, K. Wang, Z.Z. Liu, A bilevel optimization approach for joint offloading decision and resource allocation in cooperative mobile edge computing. IEEE Trans Cybern **50**(10), 4228–4241 (2020)
16. T. Nisha, D.T. Nguyen, V.K. Bhargava, A bilevel programming framework for joint edge resource management and pricing. IEEE Internet Things J. **9**(18), 17280–17291 (2022)
17. Y. Liu, J. Yan, X. Zhao, Deep reinforcement learning based latency minimization for mobile edge computing with virtualization in maritime UAV communication network. IEEE Trans. Veh. Technol. **71**(4), 4225–4236 (2022)
18. X. Xu et al., Game theory for distributed IoV task offloading with fuzzy neural network in edge computing. IEEE Trans. Fuzzy Syst. **30**(11), 4593–4604 (2022)
19. R. Zhao, J. Xia, Z. Zhao, S. Lai, L. Fan, D. Li, Green MEC networks design under UAV attack: a deep reinforcement learning approach. IEEE Trans. Green Commun. Netw. **5**(3), 1248–1258 (2021)
20. S. Bi, Y.J. Zhang, Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. IEEE Trans. Wirel. Commun. **17**(6), 4177–4190 (2018)
21. F. Wang, J. Xu, X. Wang, S. Cui, Joint offloading and computing optimization in wireless powered mobile-edge computing systems. IEEE Trans. Wirel. Commun. **17**(3), 1784–1797 (2018)
22. C. You, K. Huang, H. Chae, Energy efficient mobile cloud computing powered by wireless energy transfer. IEEE J. Sel. Areas Commun. **34**(5), 1757–1771 (2016)
23. W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, D.O. Wu, Energy-optimal mobile cloud computing under stochastic wireless channel. IEEE Trans. Wirel. Commun. **12**(9), 4569–4581 (2013)
24. M.-H. Chen, B. Liang, M. Dong, Joint offloading decision and resource allocation for multi-user multi-task mobile cloud. In: *2016 IEEE International Conference on Communications (ICC)* (2016), p. 1–6
25. T.Q. Thinh, J. Tang, Q.D. La, T.Q.S. Quek, Offloading in mobile edge computing: task allocation and computational frequency scaling. IEEE Trans. Commun. **65**(8), 3571–3584 (2017)
26. C. You, K. Huang, H. Chae, B.-H. Kim, Energy-efficient resource allocation for mobile-edge computation offloading. IEEE Trans. Wirel. Commun. **16**(3), 1397–1411 (2017)
27. Y. Wang, M. Sheng, X. Wang, L. Wang, J. Li, Mobile-edge computing: partial computation offloading using dynamic voltage scaling. IEEE Trans. Commun. **64**(10), 4268–4282 (2016)
28. T. Liu, S. Ni, X. Li, Y. Zhu, L. Kong, Y. Yang, Deep reinforcement learning based approach for online service placement and computation resource allocation in edge computing. IEEE Trans. Mob. Comput. **22**(7), 3870–3881 (2023)
29. S. Joo, H. Kang, J. Kang, CoSMoS: Cooperative sky-ground mobile edge computing system. IEEE Trans. Veh. Technol. **70**(8), 8373–8377 (2021)
30. H. Xing, L. Liu, J. Xu, A. Nallanathan, Joint task assignment and resource allocation for D2D-enabled mobile-edge computing. IEEE Trans. Commun. **67**(6), 4193–4207 (2019)
31. X. Zhu, Y. Luo, A. Liu, N.N. Xiong, M. Dong, S. Zhang, A deep reinforcement learning-based resource management game in vehicular edge computing. IEEE Trans. Intell. Transp. Syst. **23**(3), 2422–2433 (2022)
32. L. He, J. Zhao, X. Sun, D. Zhang, Dynamic task offloading for mobile edge computing in urban rail transit. In *Presented at the 2021 13th International Conference on Wireless Communications and Signal Processing (WCSP)* (2021)
33. N. Irtija, I. Anagnostopoulos, G. Zervakis, E.E. Tsiropoulou, H. Amrouch, J. Henkel, Energy efficient edge computing enabled by satisfaction games and approximate computing. IEEE Trans. Green Commun. Netw. **6**(1), 281–294 (2022)
34. Y. Zou, F. Shen, F. Yan, L. Tang, Task-oriented resource allocation for mobile edge computing with multi-agent reinforcement learning. In *Presented at the 2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)* (2021)
35. Q. Li, X. Ma, A. Zhou, X. Luo, F. Yang, S. Wang, User-oriented edge node grouping in mobile edge computing. IEEE Trans. Mob. Comput. **22**(6), 3691–3705 (2023)
36. Y. Zhang, X. Dong, Y. Zhao, Decentralized computation offloading over wireless-powered mobile-edge computing networks. In: *IEEE International Conference on Artificial Intelligence and Information Systems (ICAIIS)* (2020). pp. 137–140
37. P. Zhou, B. Yang, C. Chen: Joint computation offloading and resource allocation for NOMA-enabled industrial internet of things. In: *39th Chinese Control Conference (CCC)* (2020). pp. 5241–5246

38. Z. Song, Y. Liu, X. Sun, Joint task offloading and resource allocation for NOMA-enabled multi-access mobile edge computing. IEEE Trans. Commun. **69**(3), 1548–1564 (2021)
39. Z. Wan, D. Xu, D. Xu et al., Joint computation offloading and resource allocation for NOMA-based multi-access mobile edge computing systems. Comput. Netw. **196**, 108256 (2021)
40. N.C. Luong, D.T. Hoang, S. Gong, D. Niyato, I.K. Dong, Applications of deep reinforcement learning in communications and networking: A survey. IEEE Commun. Surv. Tuts **21**(4), 3133–3174 (2019)
41. M. Chen, Y. Hao, Task offloading for mobile edge computing in software defined ultra-dense network. IEEE J. Sel. Areas Commun. **36**(3), 587–597 (2018)
42. H. Zhou, K. Jiang, X. Liu, X. Li, V.C.M. Leung, Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing. IEEE Internet Things J. **9**(2), 1517–1530 (2022)