The Journal of Supercomputing (2024) 80:10275–10300
https://doi.org/10.1007/s11227-023-05816-w

Computing offloading and resource scheduling based on DDPG in ultra-dense edge computing networks

Ruizhong Du[1,2] · Jingya Wang[1,2] · Yan Gao[3]

Accepted: 11 November 2023 / Published online: 21 December 2023

Abstract

To address the current challenge of smart devices in healthcare Internet of things (IoT) struggling to efficiently process intensive applications in real-time, a collaborative cloud-edge offloading model tailored for ultra-dense edge computing (UDEC) networks is developed. While numerous studies have delved into the optimization of offloading in mobile edge computing (MEC), it is imperative to consider non-orthogonal multiple access (NOMA) as a physical technology when addressing the offloading optimization process in MEC. The multiuser sharing of spectrum resources in NOMA can enhance the network spectrum utilization and reduce the computational delay when users transmit computing tasks. Consequently, a model for NOMA-assisted UDEC systems is proposed. The model takes into account joint offloading decisions, computational resources, and sub-channel resources and is modeled as a complex nonlinear mixed-integer programming problem. The aim is to decrease the task execution delay and energy consumption of smart devices while ensuring that users' maximum acceptable delay for processing medical computational tasks is met efficiently and in a timely manner. Deep deterministic policy gradient (DDPG), a deep reinforcement learning method, is employed to solve the joint optimization problem. The final simulation results show that the algorithm converges well. The proposed offloading scheme can reduce the system cost by 54.5 and 69.9% in comparison with scenarios where users solely perform local computations and offload their tasks to the base station (BS). The application of NOMA communication in our offloading scheme boosts network spectrum utilization and trims down the system cost by 87.09% when contrasted with orthogonal multiple access (OMA).

Keywords Mobile edge computing · Ultra-dense network · Offloading · Non-orthogonal multiple access · Deep reinforcement learning

Extended author information available on the last page of the article

1 Introduction

With the rise of cutting-edge technologies like artificial intelligence [1], Internet of things (IoT), and sensors, we are seeing a proliferation of smart IoT devices that are becoming increasingly common in the medical field. These devices are poised to revolutionize healthcare services, taking them in a direction that is more intelligent and precise, thereby expanding the possibilities for human health improvement. For instance, we have wearable electrocardiogram detectors, wearable blood pressure detectors, and wearable blood glucose detectors. They are connected to mobile devices (MDs) and can monitor physiological parameters such as electrocardiogram signals, blood pressure changes, and patients' blood glucose levels in real-time and accurately [2, 3]. By thoroughly analyzing this data, doctors can gain a better understanding of their patients' health status and promptly identify potential health risks. This, in turn, enables them to offer tailor-made medical care and treatment plans. However, MDs have their limitations when it comes to battery life and processing power. Power-hungry applications like sleep analysis, motion tracking, epilepsy detection, and blood pressure monitoring can drain MDs quickly. Some applications, like epilepsy detection warnings and tracking blood pressure fluctuations, are time sensitive. Due to insufficient computing resources, computing in MDs will consume a lot of time. Mobile edge computing (MEC) can move computationally intensive applications from MDs to edge servers (ESs), effectively relieving computational pressure and energy consumption on MDs [4, 5].

MEC represents a cutting-edge computing architecture with the goal of shifting data processing and storage capabilities away from conventional centralized data centers to edge devices or nodes situated closer to the data source. Offloading computing tasks from MDs to nearby base stations (BSs) equipped with ESs reduces data transmission latency and network congestion. It meets user demand for services such as low latency and high bandwidth [6]. However, a single MEC server typically has certain limitations in terms of computing power. It can only fulfill the computing requirements of a limited number of MDs. Ultra-dense network (UDN), on the other hand, brings about an efficient and flexible large-scale wireless connectivity solution by deploying numerous small BSs to cater to the demands of a multitude of MDs accessing the network. This is particularly vital in densely populated settings like schools, hospitals, and residential areas with abundant greenery. To address these challenges, this paper introduces a fusion of UDN and MEC, referred to as ultra-dense edge computing (UDEC), within the 5 G architecture. In this framework, mobile network operators deploy numerous micro-BSs, each equipped with an ES. It can reuse the spectrum resources of the macro-BSs [7, 8]. In this way, the surge in demand for computational and spectral resources by MDs can be addressed, and the latency and energy consumption of task offloading can be reduced [9].

In a UDN collaborative network, MDs in a healthcare system can be more flexible in choosing their offloading methods, and tasks based on different types of computation can be processed on local devices, micro-BSs, or a macro-BS [10, 11]. The literature on computational offloading [12–14] reduces the total cost of the system by optimizing offloading and computational resource allocation. There is also some literature on optimizing the performance of MEC networks through advanced network access techniques. Liu and Yang [15] used orthogonal frequency division multiple access (OFDMA) architecture to access the unmanned aerial vehicle (UAV) cloud network, ensuring latency awareness in UAV-assisted MEC systems. Xing et al. [16] used time division multiple access (TDMA) transport protocols during user offloading of tasks to minimize the computational latency of tasks by optimizing the time, rate, resultant download of user task assignment, and computational offloading. However, this literature needs to pay more attention to the application of non-orthogonal multiple access (NOMA) technology in UDEC. It can make multiuser access in wireless communication systems. It is a promising multi-access technology for next-generation wireless networks [17, 18]. NOMA allows multiple users to transmit over the same time and frequency resources, enhancing spectral efficiency and system capacity. Consequently, it effectively boosts wireless network capacity and coverage. Moreover, NOMA proves advantageous in managing high-density user scenarios and enhancing system energy efficiency [19]. NOMA protocols, as a promising radio access technology, have been employed in [20] to enhance the efficiency of MEC offloading. In our pursuit of intelligently managing computational tasks in a healthcare system, we have merged UDN and NOMA to provide users with a more efficient and comprehensive service. The key contributions are summarized below:

1. In the system model of the NOMA-based UDN, each micro-BS deploys an MEC server and employs NOMA technology to cater to its associated users. Subsequently, the paper introduces an optimization problem that addresses joint decisions on offloading, computational resources, and sub-channel allocation. This optimization problem aims to minimize the weighted sum of energy consumption and computational task latency for all users, all while ensuring that the maximum tolerated user latency is met.
2. The objective problem involves mixed-integer nonlinear programming, a type of problem that is (non-deterministic polynomial) NP-hard and cannot be optimally solved in polynomial time. To tackle the original mixed problem, the neural network within deep deterministic policy gradient (DDPG) is employed to approximate the policy function. This approach utilizes empirical replay to train the neural networks, resulting in improved training outcomes.
3. The scheme we put forth is evaluated through simulation experiments, comparing it with various offloading methods and orthogonal multiple access (OMA) techniques. The results from the simulations indicate that our proposed scheme exhibits a quicker convergence when compared to other offloading methods. Furthermore, the utilization of NOMA techniques, as presented in this paper, results in a lower overall system cost in contrast to OMA.

The rest of this article is arranged as follows. Section 2 introduces the research status of UDEC and NOMA technology, and the significance of the combination of UDEC and NOMA. Section 3 describes the intelligent medical system model and objective optimization. In Sect. 4, a Markov decision process (MDP) decision-making process is constructed by deep reinforcement learning (DRL). In Sect. 5, the advantages and disadvantages of each unloading strategy are analyzed by comparing the system cost and algorithm convergence performance. Finally, in Sect. 6 to the full text of the summary. Table 1 contains the list of acronyms used in the research.

2 Related work

Numerous scholars have conducted extensive research on the computational offloading challenge within MEC. Their primary focus is on reducing energy consumption delays and enhancing quality of service (QoS) for users by jointly optimizing offloading, transmission power, and resource allocation. In [21], research explores how, within MEC, mobile network operators address the challenge of accommodating a large number of users accessing the network while dealing with increasing response times. They achieve this by deploying numerous edge-side micro-BSs and a cloud-side macro-BS equipped with ample computational resources to enhance network coverage. This strategic deployment reduces transmission delays by diverting users' computational tasks to the densely positioned edge-side micro-BSs or the cloud-side BS. Gao et al. [9] introduced the deployment of multiple edge-side macro-BSs in

Table 1 List of acronyms

Abbreviation	Definition
IoT	Internet of things
MDs	Mobile devices
MEC	Mobile edge computing
ESs	Edge servers
BSs	Base stations
UDN	Ultra-dense network
UDEC	Ultra-dense edge computing
OFDMA	Orthogonal frequency division multiple access
UAV	Unmanned aerial vehicle
TDMA	Time division multiple access
NOMA	Non-orthogonal multiple access
DDPG	Deep deterministic policy gradient
OMA	Orthogonal multiple access
MDP	Markov decision process
DRL	Deep reinforcement learning
QoS	Quality of service
DQN	Deep Q-leaning network
FDMA	Frequency division multiple access
RL	Reinforcement learning

UDN. These macro-BSs are supported by MEC servers, facilitating the transfer of user computational tasks to ESs located in closer proximity. This approach effectively reduces transmission delays and lowers energy consumption, thereby minimizing system overhead. Lu et al. [22] demonstrated that in the context of UDEC, system delay and energy consumption can be significantly reduced through coordinated resource scheduling, task offloading, and BS selection. Lin et al. [23] and Ahmed and Elmokashfi [24] delved into the development of a model-free offloading mechanism within the framework of MEC-supported UDN. This mechanism efficiently minimizes the computational latency of user tasks while meeting energy consumption requirements.

Various communication methods, such as TDMA and frequency division multiple access (FDMA), have seen extensive use in resource allocation within MEC. These methods effectively address the issue of transmission delays during the offloading of MEC computing tasks. Utilizing NOMA technology in UDN proves highly effective in reducing both system energy consumption and the latency of computational tasks [25]. Sun et al. [26] and Gupta et al. [27] applied NOMA technology to the uplink and downlink of MEC networks and confirmed that the introduction of NOMA technology can effectively reduce the latency and energy consumption during user offloading. In [28], the research delves into the MEC-NOMA system within an UDN. In this setup, all users of each micro-BS service are organized into separate clusters, with these users communicating with micro-BSs through NOMA technology. The study introduces a comprehensive iterative optimization algorithm aimed at minimizing the overall system cost. This algorithm achieves its goal by combining aspects such as user clustering, power allocation, and computational resource allocation.

In MEC environments, complex joint optimization challenges arise, encompassing offloading decisions, resource allocation, and energy management. These complexities emerge from the collaborative efforts involving multiple MDs, ESs, and central cloud resources. To tackle this issue, some researchers have turned to DRL algorithms for solving intricate joint optimization problems within MEC. In [29], the authors introduced a DRL algorithm based on Asynchronous Advantage Actor–Critic, tailored for addressing the two-layer architecture outlined in this paper. This algorithm demonstrates strong performance in managing expansive decision spaces, demands fewer computational resources, and achieves quicker convergence compared to strategies based on deep Q-leaning network (DQN). Wang et al. [30] presented an algorithm grounded in DDPG to handle high-dimensional state spaces and the non-convex nature of sequential actions, which characterize the problem in UAV-assisted MEC. The algorithm effectively reduces task processing latency by determining the optimal task offloading decisions. In [31], researchers focus on the challenge of joint computational offloading and resource scheduling, considering that user computational tasks are randomly generated and the environment is dynamically changing. Given the multiple optimization goals involved, the authors propose a double DQN approach based on reinforcement learning (RL) to tackle this issue. Jiang et al. [32] delved into the matter of ES placement within a dynamic MEC setup, where it is crucial to meet user computational task requirements. Since the network exhibits time-varying characteristics, ES placement requires dynamic adjustments. The researchers propose a proximal policy optimization algorithm to enable a limited number of ESs to efficiently serve all BSs, with results indicating a noticeable reduction in system costs. While these studies showcase the applicability of DRL for offloading optimization and resource scheduling, they primarily introduce improved algorithms for specific environments and lack generalizability. Therefore, this article adopts a DDPG algorithm to more accurately model and optimize the system's processing latency and energy consumption, aiming for more efficient task processing and resource utilization.

These works ignore the scenario of collaborative computing at the micro-BS, where computing tasks for users are transferred to other micro-BSs for collaborative execution. Collaborative execution among BSs enhances resource utilization and evens out the load on BSs across various regions, ensuring they remain in a relatively stable state. This, in turn, enables efficient processing of each user's computing tasks within a reasonable time frame. While some researches have explored task offloading in UDN and MEC settings, the majority has overlooked the influence of communication methods between mobile users and BSs. With the widespread implementation of UDN and the evolution of IoT devices, conventional communication methods are insufficient for managing the extensive connectivity of devices and data transmission within hospitals. Consequently, we will explore a UDEC network based on NOMA, constructing a novel offloading model to strike a harmonious equilibrium between energy consumption and latency.

3 System model

A UDEC system model based on NOMA is illustrated in Fig. 1. The model include multiple micro-BSs, and assuming that each BS is outfitted with a ES to provide computing services to its associated users and one macro-BS (assuming that the macro-BS is a remote cloud). It is assumed that users that are in the coverage area of BS are associated with only one, where the ESs can be either physical servers with computational power or virtual machines. As in [33], we assume that the macro-BS acts as the central controller responsible for collecting information about the tasks of the mobile users, information about the computing resources of the BS, and information about the network status. Assume that the system has S BSs and K sub-channels, denoted by the sets $S = \{1, 2, \dots, S\}$ and $K = \{1, 2, \dots, K\}$, respectively, where the set of users covered by each BS is denoted by $N_s = \{1, 2, \dots, N\}$. We assume that user n in BS s generates one computation request $\psi_{n,s}$ at a time as denoted by the $< D_{n,s}, T_{n,s}, C_{n,s} >$, where $D_{n,s}$ denotes the size of the task data requested by the user for computation, $T_{n,s}$ denotes the maximum tolerable delay of the task, and $C_{n,s}$ denotes the computational resources required to complete the request. These data assume that each user's task cannot be further divided into sub-tasks and that the user will offload

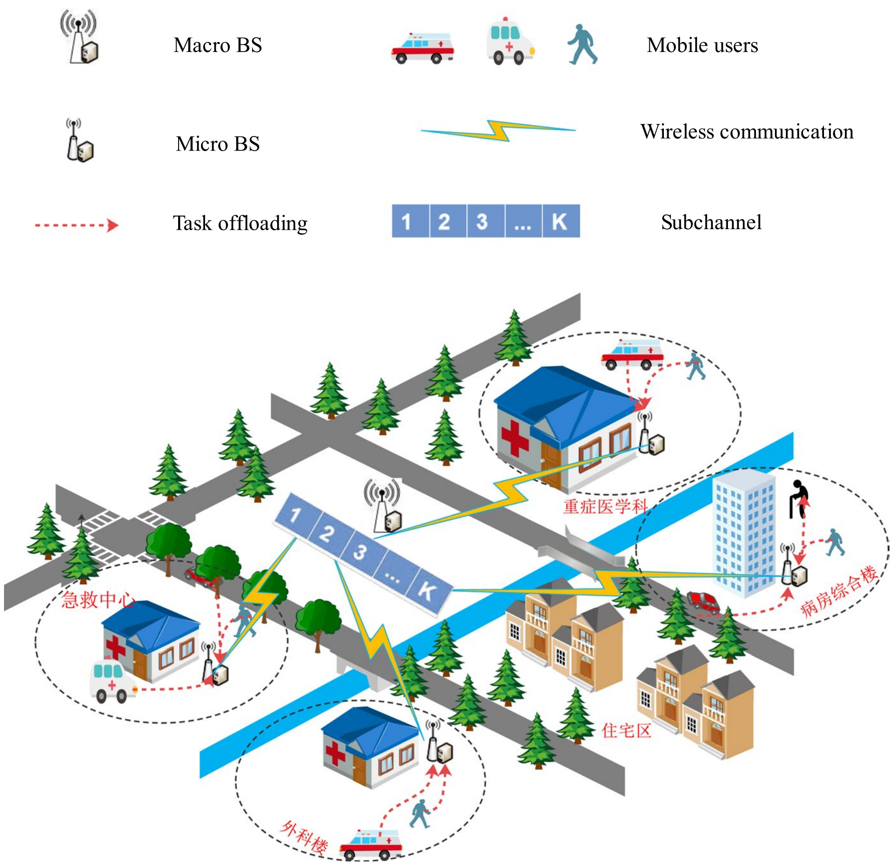

Fig. 1 IoT-enabled healthcare system model diagram

the task based on an offload decision. $x_{n,s}^{k}$ is an offload decision variable, which indicates whether or not user n offloads a task to its covered BS s.

$$x_{n,s}^{k} = \begin{cases} 1, & \text{if } \psi_{n,s} \text{ is offloaded from user n to MEC,} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

3.1 Communication model

MDs incur a certain communication cost (i.e., bandwidth) when generating computational tasks to offload to the edge server, so the communication scenario for user task offloading is shown in Fig. 2. In scenario (a), user n1 transmits its computation task to its associated MEC server s1 using NOMA technology.

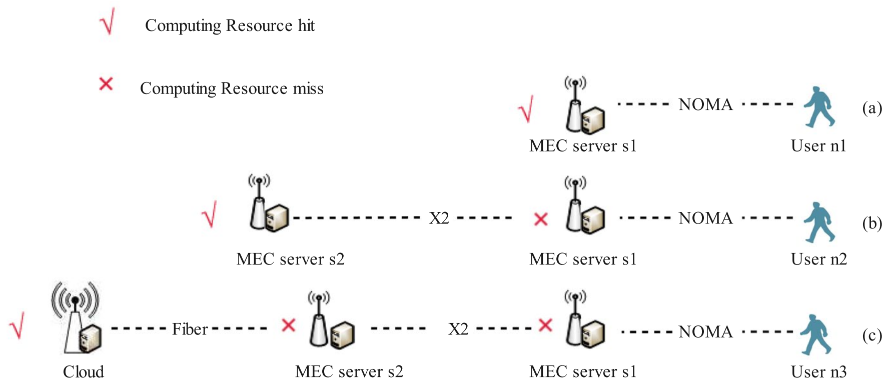

Fig. 2 Offloading decisions for mobile devices

In scenario (b), when MEC server s1 cannot provide the required service to its associated user n2, the computation task is transmitted to the neighboring MEC server s2 via the x2 link. In scenario (c), if the ESs cannot provide the service to user n3, the user's computation task will be transmitted to the edge cloud for processing via the fiber link. The computational task transfer delay is calculated as follows in different scenarios.

Scenario (a): assuming that NOMA technology is used in BS $s \in S$ to support different users $n \in N_s$ occupying the same channel, mobile users in the same area sacrifice interference to transmit data to the BS at the same time, and different BSs can multiplex the same channel. The sub-channel occupied by the BS define as c_s^k. When $c_s^k = 1$, it means that the BS occupies sub-channel k, otherwise $c_s^k = 0$. In this case, interference between users may cause a degradation of the uplink transmission rate, so the maximum number of BSs connected to a sub-channel is $M_{\max}$. On sub-channel k, for the set of users N_s served by BS s, the channel gain is assumed to follow the order of $g_{1s}^k \geq g_{2s}^k \geq \ldots \geq g_{ns}^k$. Thus, the signal-to-interference-noise ratio (SINR) of the n-th user in BS s on sub-channel k is expressed as follows:

$$\gamma_{n,s}^k = \frac{p_{n,s}^k g_{n,s}^k}{I_{n,s}^k + I_{s,s'}^k + \sigma^2}, \tag{2}$$

where $p_{n,s}^k$ denotes the transmit power of the n-th user in BS s on sub-channel k, $I_{n,s}^k = \sum_{i=n+1}^{N} x_i^s p_{n,s}^k g_{n,s}^k$ denotes the interference between users within the BS, $I_{s,s'}^k = \sum_{r \in S} \left(c_r^k \sum_{n \in U_r} x_n^r p_{n,r}^k g_{n,s}^k \right)$ denotes the interference between BSs, σ^2 denotes the power of Gaussian white noise, and B denotes the bandwidth of the uplink system. The data rate of the transmission of user n on sub-channel k is calculated according to Shannon's formula as follows:

$$R_{n,s}^{k} = B\log_2\left(1 + \gamma_{n,s}^{k}\right), \tag{3}$$

Therefore, the transmission time for the user device task to offload to BS s on sub-channel k is:

$$t_{n,s}^{\text{tran}} = \frac{x_{n,s}^{k} D_{n,s}}{R_{n,s}^{k}}, \quad \forall n \in N_s. \tag{4}$$

Scenario (b): Assume that when BS s receives a user request from within its area, but its computational resources are insufficient to process it, it can forward the associated request via the $X2$ link to another BS s' for processing. We use the binary variable $y_{n,s}^{s\to s'}$ to indicate whether the user's computational task $\psi_{n,s}$ is offloaded at its associated BS s:

$$y_{n,s}^{s\to s'} = \begin{cases} 1, & \text{if } \psi_{n,s} \text{ of user n is offloaded} \\ & \text{from BS s to a neighbor BS s}', \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Therefore, the offload delay from BS s to BS s' is:

$$t_{s,s'}^{\text{tran}} = \frac{\sum_{n\in N_s} y_{n,s^s}^{s\to s'} D_{n,s}}{\Gamma_s^{s'}}, \quad \forall s, s' \in S. \tag{6}$$

Scenario (c): when other BSs around BS s also have no remaining computing resources, the user's tasks will be offloaded to macro-BS via a wired backhaul link. $y_{n,s}^{s\to c}$ is a decision variable to indicate that the user's computational task is offloaded to macro-BS by BS s:

$$y_{n,s}^{s\to c} = \begin{cases} 1, & \text{if } \psi_{n,s} \text{ is offloaded from BS s} \\ & \text{to macro-BS,} \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Therefore, the offload delay from the BS s to macro-BS is:

$$t_{s,c}^{\text{tran}} = \frac{\sum_{n\in N_s} y_{n,s}^{s\to c} \psi_{n,s}}{\Omega_s^{c}}, \quad \forall s \in S. \tag{8}$$

3.2 Computation offloading model

3.2.1 Local execution

When user's task $\psi_{n,s}$ is executed locally, computing delay $T_{n,s}^{\text{loc}}$ and local power consumption $E_{n,s}^{\text{loc}}$ of the MD can be expressed as follows:

$$T_{n,s}^{\mathrm{loc}} = \frac{C_{n,s}}{f_{n,s}^{l}}, \tag{9}$$

$$E_{n,s}^{\mathrm{loc}} = \kappa \cdot \left(f_{n,s}^{l}\right)^{2} \cdot T_{n,s}^{\mathrm{loc}}. \tag{10}$$

where $f_{n,s}^{l}$ represents the CPU frequency of user n in BS s, the size of κ is determined by the chip structure of the MD [13, 34].

3.2.2 Edge execution

In our model, if BS s associated with the user has enough resources, it must perform the user's task $\psi_{n,s}$. In this paper, we define a decision variable $y_{n,s}^{s} \in \{0, 1\}$ to indicate whether BS s calculates the user's unloaded task.

$$y_{n,s}^{s} = \begin{cases} 1, & \text{if } \psi_{n,s} \text{ offloaded by user n} \\ & \text{is computed at BS s,} \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

The execution delay of task $\psi_{n,s}$ on BS s is:

$$\mathrm{t}_{n,s}^{\mathrm{comp}} = \frac{C_{n,s}}{f_{n,s}^{\mathrm{mec}}}, \tag{12}$$

Similar to other studies [31, 35], the transmission delay of BS s sending the calculation results back to the MD is ignored because the size of the output data is much smaller than that of the input data. Therefore, the total execution time of the tasks uninstalled by user n on BS s is:

$$T_{n,s}^{\mathrm{mec}} = t_{n,s}^{\mathrm{tran}} + t_{n,s}^{\mathrm{comp}}, \tag{13}$$

The energy consumed by transmitting the data of user computing task $\psi_{n,s}$ to its associated BS s can be calculated as:

$$E_{n,s}^{\mathrm{tran}} = p_{n,s} t_{n,s}^{\mathrm{tran}}. \tag{14}$$

When BS s associated with the user does not have enough computing resources to meet the user's computing requirements, BS s needs to unload the task to any BS $s^{'}$ that has enough computing resources to meet the user's requirements. The computing delay of the task on BS $s^{'}$ is expressed as:

$$\mathrm{t}_{n,s'}^{\mathrm{comp}} = \frac{C_{n,s}}{f_{n,s'}^{\mathrm{mec}}}, \tag{15}$$

Therefore, the total execution time of user n unloading to BS $s^{'}$ is:

$$T^{\mathrm{mec}}_{s,s'} = t^{\mathrm{tran}}_{n,s} + t^{\mathrm{tran}}_{s,s'} + \mathrm{t}^{\mathrm{comp}}_{n,s'}. \tag{16}$$

3.2.3 Cloud execution

When no other BS is available, the user's computing tasks will be executed at macro-BS, where the computing delay is expressed as:

$$T^{c}_{n} = t^{\mathrm{tran}}_{n,s} + t^{\mathrm{tran}}_{s,c} + \frac{C_{n,s}}{f^{c}_{n}}, \tag{17}$$

In summary, the total latency of user's tasks $\psi_{n,s}$ calculation is:

$$\begin{aligned} T^{\mathrm{total}}_{n,s} = \left(1 - x^{k}_{n,s}\right)T^{\mathrm{loc}}_{n,s} + x^{k}_{n,s}\Big(y^{s}_{n,s}T^{\mathrm{mec}}_{n,s} \\ + \sum_{n\in N_s} y^{s\to s'}_{n,s}T^{mec}_{s,s'} + y^{s\to c}_{n,s}T^{c}_{n}), \end{aligned} \tag{18}$$

The energy consumed by the user to complete the task $\psi_{n,s}$ is:

$$\begin{aligned} E^{\mathrm{total}}_{n,s} =& \left(1 - x^{k}_{n,s}\right)E^{\mathrm{loc}}_{n,s} + x^{k}_{n,s}\left(y^{s}_{n,s} + \sum_{n\in N_s} y^{s\to s'}_{n,s}\right. \\ &\left. + y^{s\to c}_{n,s}\right)E^{\mathrm{tran}}_{n,s}. \end{aligned} \tag{19}$$

To ensure that the user's computing tasks are performed in only one location, namely on the user's device, BS, or macro-BS, the following constraints are imposed:

$$\left(1 - x^{k}_{n,s}\right) + x^{k}_{n,s}\left(y^{s}_{n,s} + \sum_{n\in Ns} y^{s\to s'}_{n,s} + y^{s\to c}_{n,s}\right) = 1, \tag{20}$$

$$\max\left\{y^{s}_{n,s}, y^{s\to s'}_{n,s}, y^{s\to c}_{n,s}, \forall n\right\} \le x^{k}_{n,s}, \quad \forall n \in Ns. \tag{21}$$

3.3 Problem formation

The optimization problem is expressed as the following ("CORS" means computational offloading and resource scheduling)

$$CORS : \min \sum_{s\in S}\sum_{n\in N_s}\left(\lambda T^{\mathrm{total}}_{n,s} + (1-\lambda)E^{\mathrm{total}}_{n,s}\right) \tag{22a}$$

$$\text{s.t. } T^{\mathrm{total}}_{n,s} \leqslant T_{n,s}, \quad s \in S, \quad n \in N_s, \tag{22b}$$

$$E_{n,s}^{\text{total}} \leqslant E_{\max}, \quad \forall s \in S, \quad n \in N_s, \tag{22c}$$

$$0 \leqslant x_{n,s}^{k} p_{n,s}^{k} \leqslant P_{n,s}^{k}, \tag{22d}$$

$$\sum_{k \in K} c_s^k \leqslant 1, \quad \forall s \in S, \tag{22e}$$

$$\sum_{s \in S} c_s^k \leqslant M, \quad \forall k \in K, \tag{22f}$$

$$c_s^k \in \{0,1\}, \quad \forall s \in S, \quad k \in K, \tag{22g}$$

$$\left(1 - x_{n,s}^{k}\right) + x_{n,s}^{k}\left(y_{n,s}^{s} + \sum_{n \in Ns} y_{n,s}^{s\to s'} + y_{n,s}^{s\to c}\right) = 1, \tag{22h}$$

$$\max\left\{y_{n,s}^{s}, y_{n,s}^{s\to s'}, y_{n,s}^{s\to c}, \forall n\right\} \le x_{n,s}^{k}, \quad \forall n \in Ns. \tag{22i}$$

where λ is the weight of energy consumption and calculation delay. Constraint (22b) ensures that the computing task delay of the user's MD cannot exceed the maximum delay. Constraint (22c) is the energy consumption for users. Constraint (22d) is the power constraint for all unloading users. Constraint (22e) ensures that sub-channel allocation is a binary decision variable, and constraint (22f) ensures each BS multiplexes at most one sub-channel, and constraint (22g) ensures each sub-channel can only be multiplexed by M BSs. Constraints (22h) and (22i) ensure that the user's computing tasks can only be performed in one location. Table 2 lists the important symbols used in the text.

4 Algorithm for solving problem

RL can make optimal decisions in a specific environment through self-study. It models all realistic problems as an interaction between an agent and its environment. In every period of interaction, the agent can receive the state of the environment and choose appropriate actions according to the state. Agents can obtain a reward value and a new state based on the feedback of the environment [31, 36]. Although RL has a lot of strengths, it is not scalable and the problems it deals with are limited to low-dimensional state space [37].

Different from RL, DRL involves the perceptual capabilities of RL and the decision-making capabilities of RL to solve environmental problems with high-dimensional state space and action space [38]. In this section, the MDP model of this paper is built in actual MEC scenario. In system, the current state of the BS is only

Table 2 Symbolic representation and description

Notion	Description
$\psi_{n,s}$	Computing tasks for devices
$D_{n,s}$	Data size of the device computing task
$T_{n,s}$	The maximum delay of the device
$C_{n,s}$	CPU cycles required by the device
S	The number of BS
N_s	The number of users within the range of BS
K	The number of sub-channels in the system
$M_{\max}$	The maximum number of BSs occupying sub-channel
c_s^k	$c_s^k = 1$ if BS occupies sub-channel k. Otherwise, $c_s^k = 0$
$y_{n,s}^s$	$y_{n,s}^s = 1$ if BS computes the user's computing tasks. Otherwise, $y_{n,s}^s = 0$
$y_{n,s}^{s\to s'}$	$y_{n,s}^{s\to s'} = 1$ if the user offloads the computing task from its associated BS to other nearby BSs. Otherwise, $y_{n,s}^{s\to s'} = 0$
$y_{n,s}^{s\to c}$	$y_{n,s}^{s\to c} = 1$ if the user's computing tasks are offloaded to macro-BS. Otherwise, $y_{n,s}^{s\to c} = 0$
B	The bandwidth of the uplink system channel
σ^2	Gaussian white noise
$g_{n,s}^k$	Channel gain
$P_{n,s}^k$	The maximum transmission power of the user device
$\Gamma_s^{s'}$	The $X2$ link between BS s and BS s'
Ω_s^c	Link capacity between BS s and macro-BS
κ	Effective switching capacity of the user's mobile device
$f_{n,s}^l$	Computing power on mobile devices
$f_{n,s}^{\mathrm{mec}}$	The computing power of BS s
f_n^c	Computing power of macro-BS
λ	Weight between energy consumption and processing delay in system cost

related to the state and actions of the previous moment (for example, whether the remaining resources of the BS are available, whether the user's tasks are offloaded to the BS). The DDPG algorithm is improved to solve the unloading and resource scheduling problems of computing tasks for mobile users. Although the number of MDs and ESs will constantly change, the process of task processing will not change. Computing tasks are randomly generated at the mobile user end, some of which are processed locally on the MD, and the other part is unloaded to the BS associated with the user for processing through policy. Finally, the result is returned to the MD.

4.1 Problem formulation based on DDPG

MDP [29] is a mathematical framework describing the discrete time stochastic control process. Part of the results generated by it is random and controlled by an agent or decision maker. It is usually made up of five tuples $(\mathcal{S}, \mathcal{A}, p(.,.), R, \gamma)$ [39, 40], where $\mathcal{S}$ denotes a finite set of states, $\mathcal{A}$ represents the finite action set, and is the description of the behavior of the agent. $p(s_{i+1} \mid s_i, a_i)$ denotes the transitional probabilities of the system state from state $s_i \in \mathcal{S}$ to state $s_{i+1} \in \mathcal{S}$ after the execution of action $a_i \in \mathcal{A}$, where $R : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ is the instantaneous reward function, and γ represents the discount factor used to calculate the cumulative income of the whole process. When the agent interacts with the environment, the cumulative return at state $s_t \in \mathcal{S}$ is:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{n} \gamma^k R_{t+k+1}, \tag{23}$$

To evaluate the value of state s and the optimal strategy, the state value function is introduced $v_\pi(s)$ and state-behavior value function $Q_\pi(s, a)$. In practical application, the Behrman equation is adopted:

$$v_\pi(s) = E_\pi\left(R_{t+1} + \gamma v_\pi S_{t+1} \mid S_t = s\right), \tag{24}$$

$$Q_\pi(s,a) = E_\pi\left[R_{t+1} + \gamma Q\left(S_{t+1},\ \mathrm{A}_{t+1}\right) \mid S_t = s,\ \mathrm{A}_t = a\right], \tag{25}$$

State space: we use state $S_t = \left(D_{n,s}, T_{n,s}, C_{n,s}, \phi, U_1, U_2, \dots U_{2+s}\right)$ to characterize the tasks generated by users in slot t and the computational resources available to the ES. Where $D_{n,s}$ represents the size of the data amount requested by the MD; $T_{n,s}$ represents the maximum tolerated delay of computing tasks on the user's MD; $C_{n,s}$ indicates the computing resources required by the MD to complete the requested computing task; ϕ_i represents the remaining available computing resources on the ES; Among them $\phi = \left(\phi_1, \phi_2, \dots, \phi_i, \dots, \phi_s\right)$, $\phi_i = f_{n,s}^{\mathrm{mec}} - \sum_{i=1}^{S} x_{n,i}^{k} f_{n,s}^{\mathrm{mec}}$. U_i indicates whether the I-th computing device in time slot t is used. Meanwhile, to ensure that the tasks of MDs can only be computed locally or offloaded, only $2 + s$ computing devices, including a macro-BS, a user equipment, and s BSs, need to be considered for computing tasks generated by MDs.

Action space: to unload computing tasks of MDs onto appropriate computing devices, the action space set in DRL is corresponding to the collection of available computing devices. $(0/1)_i^j$ is used to indicate whether the computing tasks of user i are unloaded on device j. So it has an action space of $\mathbf{A} = \left(a_1, a_2, \dots, a_{2+s}\right)$.

Algorithm 1 computational offloading and resource scheduling (CORS)-DDPG algorithm

```
Input: Input state space S, action space A, different workloads $\psi_{n,s}$ of the
    MD at time t;
Output: Optimal action selection to achieve minimum system cost;
 1: Initialize actor network policy network parameters and target network
    policy network parameters: $\theta^{Q}$, $\theta^{Q\prime}$;
 2: Initialize critic network policy network parameters and target network
    policy network parameters: $\theta^{\mu}$, $\theta^{\mu\prime}$;
 3: Initialize experience pool, random noise, attenuation factor and soft
    renewal factor: $D$, $\mathcal{N}$, $\gamma$, $\tau$;
 4: for each episode do
 5:     Initialize the current state $s_t$;
 6:     for each step of episode do
 7:         Select an action $a_t = \mu(s_t \mid \theta^{\mu}) + \mathcal{N}_t$ by the current policy and
    random noise;
 8:         Obtain the reward $r_t$ and next state $s_{t+1}$ according to the action $a_t$;
 9:         Store the $\{s_t, a_t, r_t, s_{t+1}\}$ into experience pool D;
10:         The actor and critic networks are trained by randomly drawing a
    batch of $\{s_i, a_i, r_i, s_{i+1}\}$ from the experience pool;
11:         Set $y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) \mid \theta^{Q'}\right)$.
12:         Update all parameters of the current network of critic by the
    equation(28);
13:         Update all parameters of the current network of actor by the
    equation (29);
14:         Update the critic target network and the actor target network
    parameters:
15:         $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$;
16:         $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$.
17:     end for
18: end for
```

Reward: after one step, the agent will receive the corresponding reward R after executing action A. In general, the reward function and the objective function are positively correlated. However, since the objective of this paper is to minimize the cost of the system, the target value can be defined as, $R = -\left(\lambda T_{n,s}^{\text{total}} + (1-\lambda)E_{n,s}^{\text{total}}\right) = Z$.

4.2 Computational offloading and resource scheduling algorithm based on DDPG

The DDPG algorithm consists of a learning strategy function (Actor) and a learning action value function (Critic). The actor network defines a parameterized strategy according to the observed environment state and generates an action, while the critic is responsible for evaluating the rewards obtained through the current strategies [30, 41]. The critic network in DDPG uses experiential playback technology which uses $\{s_t, a_t, r_t, s_{t+1}\}$ tuple to save its track as a record and uses a small number of tuples to update network parameters. To minimize the loss function, the critic network calculates the Q value of the current network and the Q value of the target network based on the current state and action. Next, the policy function is updated through the current network using the policy gradient. Then, update the target network parameters. The present action a_t can be calculated using the following formula

$$a_t = \mu\left(s_t \mid \theta^{\mu}\right) + \mathcal{N}_t, \tag{26}$$

Then, the target Q value is given by

$$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) \mid \theta^{Q'}\right), \tag{27}$$

The loss function in the evaluation process is calculated as follows:

$$L = \frac{1}{N} \sum_i \left(y_i - Q\left(s_i, a_i \mid \theta^{Q}\right)\right)^2, \tag{28}$$

Next, the actor updates the current policy using policy gradient with the help of θ^Q and sample tuples.

$$\nabla_{\theta^{\mu}} J|_{s_i} \approx \frac{1}{N} \sum_i \nabla_a Q\left(s, a|_{\theta^Q}\right)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu\left(s|_{\theta^{\mu}}\right)|_{s_i}, \tag{29}$$

In contrast to DQN, DDPG increases the stability of learning by using soft updates to refresh the parameters. The process of updating target Q network parameter $\theta^{Q'}$ in the actor network and target Q network parameter $\theta^{\mu'}$ in the critic network can be represented as

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}, \tag{30}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}. \tag{31}$$

This section presents a collaborative computing offloading algorithm that leverages DRL techniques to facilitate efficient offloading. The proposed algorithm, CORS-DDPG, aims to identify an optimal offloading destination for users, determining whether to process tasks locally or offload them to an ES or a macro-BS. CORS-DDPG stores previous experiences in an experience replay buffer, allowing the mechanism to learn from past decisions and make appropriate offloading choices tailored to the computational requirements of different users. The goal is to strike a balance between energy consumption and processing latency. The step-by-step operation of the CORS-DDPG algorithm is as follows:

1. The agent acquires actions based on the current state of the environment using the policy.
2. For each step of the episode, MDs within the coverage area of the micro-BS first perform NOMA channel occupancy based on cooperative offloading and resource allocation policies to accomplish data transmission and task execution.
3. Next, energy consumption and transmission delay are calculated based on the current action and state. Based on these calculations, the reward for the step is obtained, and the environment state is updated.
4. Store the reward and the next state in the experience pool. Periodically, a portion of the samples from the experience pool is used to train the actor and critic networks.
5. When the execution step satisfies the policy update condition, specific equation (28) and equation (29) are used to update the current critic network and current actor network.

5 Performance evaluation

First of all, the simulation parameter setting and different experimental methods are recommended. Then, the effects of different parameter settings on the algorithm convergence performance are compared. Lastly, the scheme's feasibility is demonstrated by a comparison with the benchmark offloading method and the communication method.

5.1 Simulation setup

In this section, we verify the performance of the proposed joint optimization algorithm and the advantages of the NOMA communication method through simulation experiments. In simulation, we use Python to set the coverage area of the macro-BS based on our system model, which is designated as an area with a radius of 1000 m. Micro-BSs are randomly placed in this area. Based on Wang et al. [30], the number of micro-BSs is from 4 to 8. Each NOMA cluster contains two MDs. The number of MDs is 8 to 16, and sub-channels are 4. The size of the user task is between 600 and 1000 Kbits. The rest of the parameters for the experiments are shown in Table 3.

5.2 Contrast experiment

5.2.1 Comparison of offloading methods

To compare the impact of the unload methods in this article with the total cost of the system, we compare them with other offloading methods, all three of which are described in detail below.

1. Local-Execution-only: the user's computational tasks are all processed locally without requesting a computational offload to the MEC server.

Table 3 Simulation parameter setting

Parameter	Value
The number of Micro-BS	$[4, 8]$
The number of MDs	$\{8, 10, 12, 14, 16\}$
Number of sub-channels	4
The data size of task	600–1000 Kbits
Computing resources required by user tasks	1000–2000 MHZ
Maximum tolerated latency of a user computing task	0.5-–0.75 s
Computing power of MDs	1–300 Kbits/s
Macro-BS computing power	5 Mbits/s
Channel bandwidth	20 MHz
Bandwidth between BS	20–25 MHz
Bandwidth between BS and macro-BS	50–120 Mbps
Noise power	−100 dBm

2. Edge-Execution-only: User-generated computing tasks are executed only on the ES associated with them.
3. The scheme proposed in this paper (CORS-DDPG): the user's computing tasks can be processed locally, and offloaded to BS associated with it or to macro-BS for processing.

5.2.2 Comparison of communication methods

To compare the impact of the communication approaches presented in this article on overall system cost, we compare OMA communication methods, and the comparison method is described in detail below.

1. Local-Execution-only: the user's computational tasks are all processed locally without requesting a computational offload to BS.
2. Edge-Execution-only: all users' computing assignments are unloaded to BS for execution. (communication method using NOMA).
3. The scheme proposed in this paper (CORS-DDPG): all user's task can be executed locally, offloaded to its associated BS, or offloaded to macro-BS, where the communication method uses NOMA and orthogonal multiple access (OMA), respectively (CORS-DDPG-NOMA) (CORS-DDPG-OMA).

Next simulation in this paper, according to the number of iterations, evaluates different offloading approaches based on DDPG convergence performance. The metric used in the article is system cost which is the sum of the completion latency and user energy consumption.

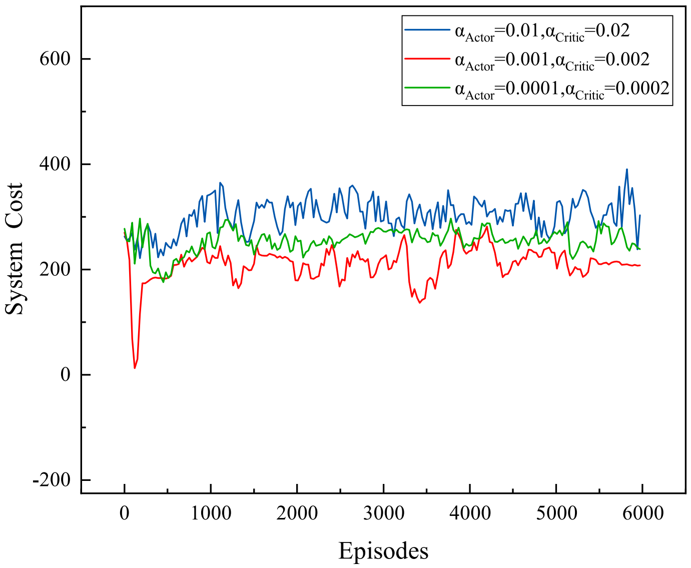

Fig. 3 Convergence performance of CORS-DDPG algorithm at different learning rates

5.3 Algorithm performance analysis

In Fig. 3, we illustrate the impact of various network hyperparameters on the algorithm's convergence performance. We assess these hyperparameters based on the system cost. Precisely, at the instance where $\alpha_{\text{Actor}} = 0.01, \alpha_{\text{Critic}} = 0.02$, the algorithm fails to converge within a specific time frame. This failure can be attributed to the neural network's large update step and the high learning rate of both the behavioral and critic networks, making it impossible to decide the behavior that minimizes the system cost. Upon resetting $\alpha_{\text{Actor}} = 0.0001, \alpha_{\text{Critic}} = 0.0002$, the algorithm still does not converge. The absence of convergence results from the reduced learning parameter, which decelerates the network updates, thereby requiring an increased number of iterations to attain convergence. This process demands additional computational resources, escalating the system's cost. Then, we reset $\alpha_{\text{Actor}} = 0.001, \alpha_{\text{Critic}} = 0.002$, which allows the algorithm to converge within a specific time interval, reducing system cost.

As shown in Fig. 4, we explore the impact of different discount rate factors γ on the algorithm's performance. We use the system cost as an evaluation metric to determine the optimal discount rate setting. The system cost reaches its lowest point when $\gamma = 0.95$. The computational offloading strategy is optimized at this point, and the best offloading decision can be found within a time interval. However, in the other two sets of experiments, the set γ values do not converge quickly and are ineffective in reducing the system cost. When larger or smaller values of γ are employed, the algorithm treats the data gathered in the current period as having a prolonged influence on the entire training process. This results in time intervals that are either excessively extended or too brief, subsequently impacting the generalization effect.

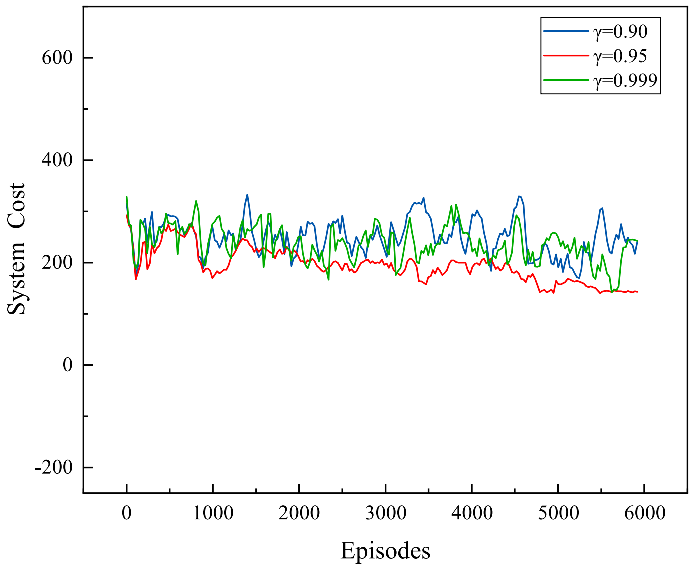

Fig. 4 Effect of different discount factors on the performance of CORS-DDPG algorithm

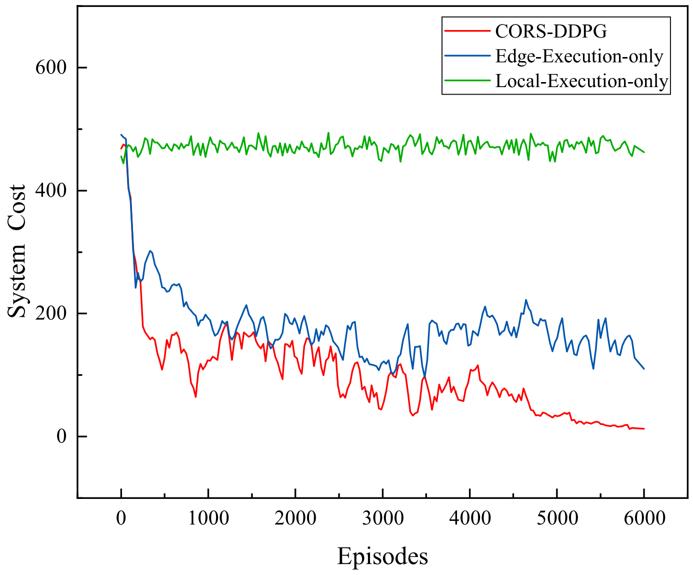

Fig. 5 Convergence performance of the CORS-DDPG algorithm

Consequently, the algorithm fails to reduce the system cost-effectively. To improve the training strategy, we will set $\gamma = 0.95$ in the next experiments.

As shown in Fig. 5, CORS-DDPG and Local-Execution-only can converge over time, the other offloading methods have large fluctuations, but Local-Execution-only does not lead to a minimization of the total cost of the system and it can be seen that the CORS-DDPG offloading method is better.

5.4 Performance Comparison

Figure 6 shows the system cost obtained through different offloading methods. It can be found that the system cost increases with the number of BSs. As shown in Fig. 6, CORS-DDPG method obtains lower system cost than the other two baseline methods (Edge-Execution-only, Local-Execution-only), this is because these two methods cannot handle the dynamic environmental changes.

Figure 7 displays the quantitative relationship established between the system cost and BS when the number of sub-channels varies. Therefore, the sub-channels are different. In Fig. 7, the system cost is gradually increasing. As the number of sub-channels decreases the system cost is increases, this is because when there are fewer sub-channels in the system the quantity of BSs occupying the same channel increases, which leads to severe BS interference and in turn increases the overall cost of the user offloading process.

Figure 8 shows the system cost with the amount of users under the CORS-DDPG algorithm. From Fig 8, the total system cost is gradually growing as the amount of

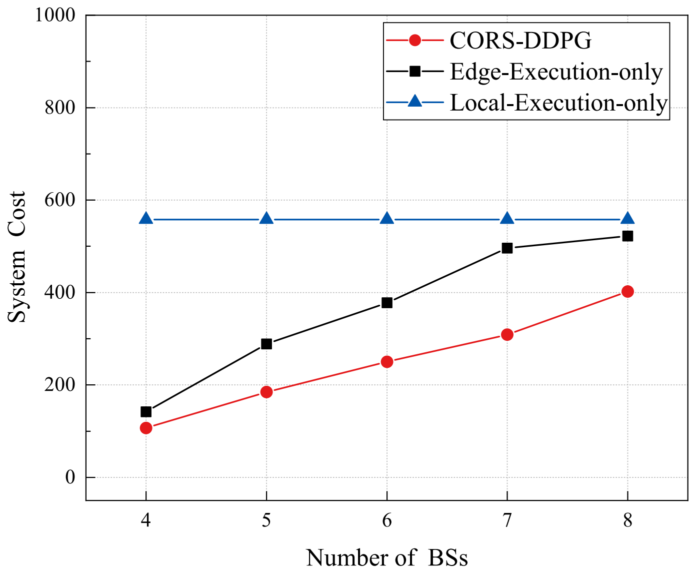

Fig. 6 Total system reward in relation to the number of BSs

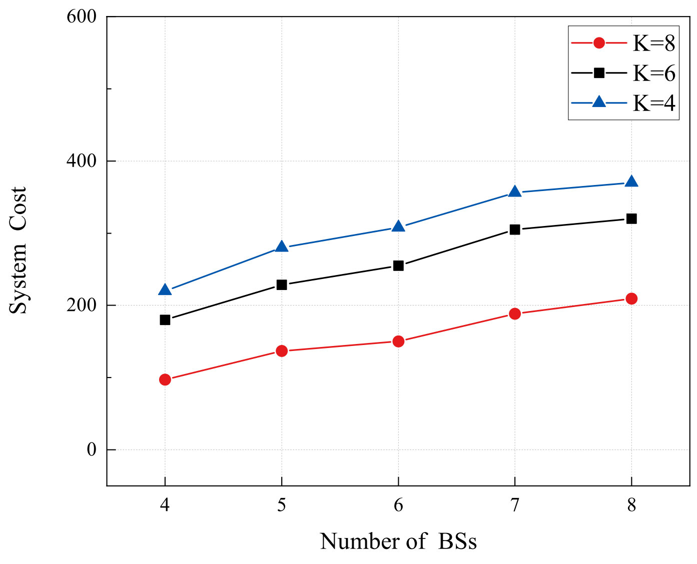

Fig. 7 Total system reward in relation to the number of BSs

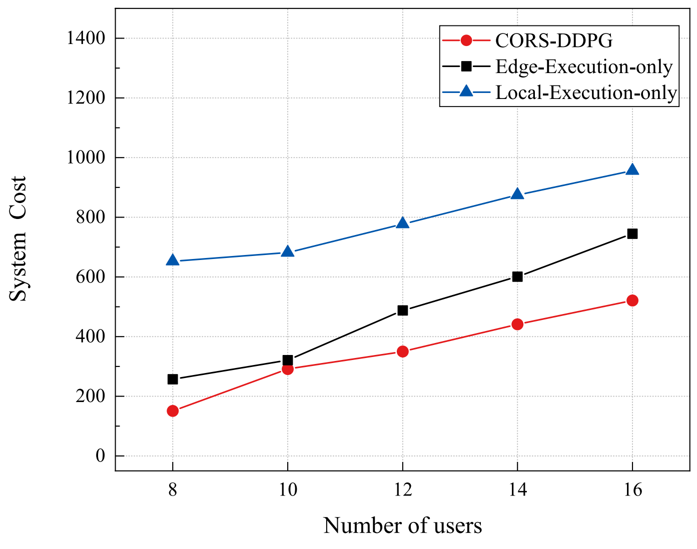

Fig. 8 Total system reward in relation to the number of users

users increases, but the proposed CORS-DDPG scheme allows for a lower system cost than other offloading schemes.

Figure 9 displays the correlation between the system cost and the size of the input data for user computing tasks. In Fig. 9, the total system cost increases with the volume of data increases, this is because a larger user computing task load will lead to

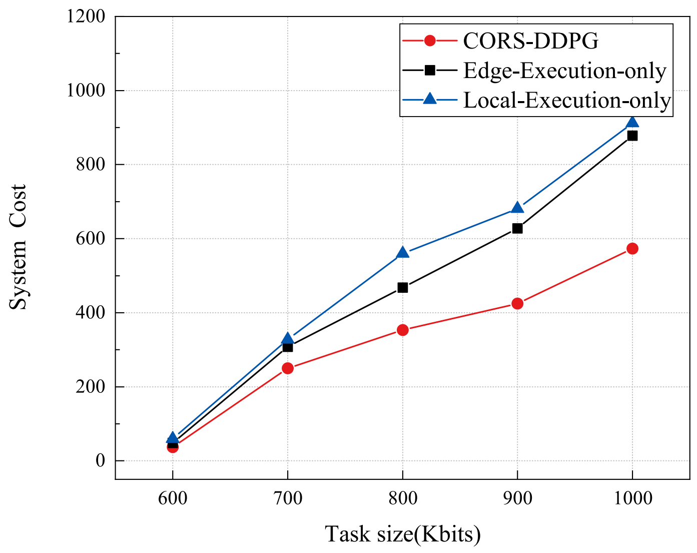

Fig. 9 Total system reward in relation to task size

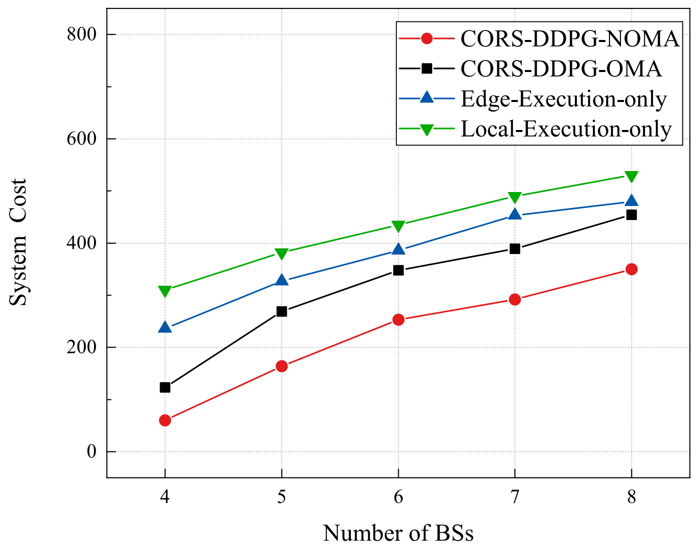

Fig. 10 Total system reward in relation to the number of BSs

a significant increase in the cost of local computing and also, more users will choose offload their tasks to BS.

In Fig. 10, the impact of two different communication approaches is compared on the system cost. The total system cost of CORS-DDPG-NOMA is lower than the other solutions, which shows the effectiveness of the communication method we have applied.

6 Conclusion

In this paper, we propose an offloading model for cloud-edge-end cooperative offloading by combining UDEC and NOMA to achieve a balance between energy consumption and latency of MDs in the offloading process of computing tasks. The model has important applications in the field of medical IoT, which can provide energy-efficient and low-latency services according to the different medical needs of patients. The study takes into account energy consumption, latency, and QoS, aiming to minimize the total cost incurred during user offloading tasks by means of joint offloading decisions, computational resources, and sub-channel allocation. Despite the objective problem being non-convex, this paper proposes a DRL cooperative offloading algorithm to determine the optimal offloading strategy for users. Simulation results show that the proposed offloading scheme achieves lower total system cost compared to schemes that process only locally or offload only to ESs. In addition, MOMA can lead to a near-optimal system performance compared to OMA.

However, the research in this paper has some shortcomings and needs to be considered. For example, ESs cannot handle the offloading of users' computing tasks in a dynamic environment in real-time. In future research, the mobility of users will be considered, such as the sudden movement of a user from the coverage area of micro-BS to that of another.

Author contributions The authors contributed equally to this work.

Funding The authors did not receive support from any organization for the submitted work.

Availability of data and materials The raw/processed data required to reproduce these findings cannot be shared at this time as the data also forms part of an ongoing study.

Declarations

Conflict of interest The authors declare that they have no competing interests to this work.

Ethical approval Not applicable.

References

1. Shukla PP, Pandey S (2023) Maa: multi-objective artificial algae algorithm for workflow scheduling in heterogeneous fog-cloud environment. J Supercomput. https://doi.org/10.1007/s11227-023-05110-9
2. Rao AR, Clarke DJB (2020) Perspectives on emerging directions in using IoT devices in blockchain applications. Internet Things 10:100079
3. Chen X, Xie H, Li Z, Cheng G, Leng M, Wang FL (2023) Information fusion and artificial intelligence for smart healthcare: a bibliometric study. Inf Process Manag 60:103113
4. Wan Z, Dong X (2022) Computation power maximization for mobile edge computing enabled dense network. Comput Netw 220:109458
5. Yang S (2020) A joint optimization scheme for task offloading and resource allocation based on edge computing in 5g communication networks. Comput Commun 160:759–768

6. Liao L, Lai Y, Yang F, Zeng W (2022) Online computation offloading with double reinforcement learning algorithm in mobile edge computing. J Parallel Distrib Comput 171:28–39
7. Ding Z, Fan P, Poor HV (2018) Impact of non-orthogonal multiple access on the offloading of mobile edge computing. IEEE Trans Commun 67:375–390
8. Feng S, Zhang R, Xu W, Hanzo LH (2019) Multiple access design for ultra-dense VLC networks: orthogonal vs non-orthogonal. IEEE Trans Commun 67:2218–2232
9. Gao Y, Zhang H, Yu F, Xia Y, Shi Y (2022) Joint computation offloading and resource allocation for mobile-edge computing assisted ultra-dense networks. J Commun Inf Netw 7:96–106
10. Du R, Liu C, Gao Y, Hao P, Wang Z (2022) Collaborative cloud-edge-end task offloading in noma-enabled mobile edge computing using deep learning. J Grid Comput. https://doi.org/10.1007/s10723-022-09605-2
11. Shukla PP, Pandey S, Hatwar P, Pant A (2023) Fat-eto: fuzzy-ahp-topsis-based efficient task offloading algorithm for scientific workflows in heterogeneous fog-cloud environment. Proc Natl Acad Sci, India, Sect A 93:339–353
12. Nath S, Li Y, Wu J, Fan P (2020) Multi-user multi-channel computation offloading and resource allocation for mobile edge computing. In: ICC 2020—2020 IEEE International Conference on Communications (ICC), pp 1–6
13. Liu J, Guo S, Wang Q, Pan C, Yang L (2022) Optimal multi-user offloading with resources allocation in mobile edge cloud computing. Comput Netw 221:109522
14. Qi J, Liu Y, Ling Y, Xu B, Dong Z, Sun Y (2022) Research on an intelligent computing offloading model for the internet of vehicles based on blockchain. IEEE Trans Netw Serv Manage 19:3908–3918
15. Liu S, Yang T (2020) Delay aware scheduling in UAV-enabled OFDMA mobile edge computing system. IET Commun 14:3203–3211
16. Xing H, Liu L, Xu J, Nallanathan A (2019) Joint task assignment and resource allocation for d2d-enabled mobile-edge computing. IEEE Trans Commun 67:4193–4207
17. Khan MJ, Chauhan RCS, Singh I (2022) Energy-efficient multiple cooperative moving relay selection for heterogeneous nonorthogonal-multiple access systems. Int J Commun Syst. https://doi.org/10.1002/dac.5408
18. Khan MJ, Chauhan RCS, Singh I (2022) Outage probability and throughput of cooperative non-orthogonal multiple access with moving relay in heterogeneous network. Trans Emerging Telecommun Technol. https://doi.org/10.1002/ett.4616
19. Ke F, Lin Y, Liu Y, Zhou H, Wen M, Zhang Q (2023) Task offloading, caching and matching in ultra-dense relay networks. IEEE Trans Veh Technol 72:4010–4025
20. Long K, Leung VCM, Zhang H, Feng Z, Li Y, Zhang Z (2017) 5g for future wireless networks. In: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering
21. Hu S, Li G (2020) Dynamic request scheduling optimization in mobile edge computing for IoT applications. IEEE Internet Things J 7:1426–1437
22. Lu Y, Chen X, Zhang Y, Chen Y (2022) Cost-efficient resources scheduling for mobile edge computing in ultra-dense networks. IEEE Trans Netw Serv Manage 19:3163–3173
23. Lin Z-H, Gu B, Zhang X, Yi D, Han Y (2022) Online task offloading in UDN: a deep reinforcement learning approach with incomplete information. In: 2022 IEEE Wireless Communications and Networking Conference (WCNC), pp 1236–1241
24. Ahmed AH, Elmokashfi AM (2022) Icran: intelligent control for self-driving ran based on deep reinforcement learning. IEEE Trans Netw Serv Manage 19:2751–2766
25. Sowjanya K, Porwal A, Pandey S, Mishra PK (2022) Tlbo-based resource allocation scheme in 5g h-CRAN. In: 2022 14th International Conference on Communication Systems and Networks (COMSNETS), pp 646–651
26. Sun W, Zhang H, Wang R, Zhang Y (2020) Reducing offloading latency for digital twin edge networks in 6g. IEEE Trans Veh Technol 69:12240–12251
27. Gupta S, Rajan D, Camp JD (2022) Noma-enabled computation and communication resource trading for a multi-user MEC system. IEEE Trans Veh Technol 71:7532–7547
28. Li L, Cheng Q, Tang X, Bai T, Chen W, Ding Z, Han Z (2021) Resource allocation for NOMA-MEC systems in ultra-dense networks: a learning aided mean-field game approach. IEEE Trans Wirel Commun 20:1487–1500
29. Zou J, Hao T, Yu C, Jin H (2021) A3c-do: a regional resource scheduling framework based on deep reinforcement learning in edge scenario. IEEE Trans Comput 70:228–239

30. Wang Y, Fang W, Ding Y, Xiong NN (2021) Computation offloading optimization for UAV-assisted mobile edge computing: a deep deterministic policy gradient approach. Wirel Netw 27:2991–3006
31. Liu T, Zhang Y, Zhu Y, Tong W, Yang Y (2021) Online computation offloading and resource scheduling in mobile-edge computing. IEEE Internet Things J 8:6649–6664
32. Jiang X, Hou P, Zhu H, Li B, Wang Z, Ding H (2023) Dynamic and intelligent edge server placement based on deep reinforcement learning in mobile edge computing. Ad Hoc Netw 145:103172
33. Yu B, Pu L, Xie Q, Xu J (2018) Energy efficient scheduling for IoT applications with offloading, user association and bs sleeping in ultra dense networks. In: 2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), pp 1–6
34. Wen Y, Zhang W, Luo H (2012) Energy-optimal mobile application execution: taming resource-poor mobile devices with cloud clones. In: 2012 Proceedings IEEE INFOCOM, pp 2716–2720
35. Chen X, Zhang H, Wu C, Mao S, Ji Y, Bennis M (2018) Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. IEEE Internet Things J 6:4005–4018
36. Sun J, Lu Y, Cui L, Fu Q, Wu H, Chen J (2022) A method of optimizing weight allocation in data integration based on q-learning for drug-target interaction prediction. Front Cell Dev Biol. https://doi.org/10.3389/fcell.2022.794413
37. Chen L, Gong G, Jiang K, Zhou H, Chen R (2022) Ddpg-based computation offloading and service caching in mobile edge computing. In: IEEE INFOCOM 2022—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp 1–6
38. Ale L, King SA, Zhang N, Sattar AR, Skandaraniyam J (2021) D3pg: Dirichlet DDPG for task partitioning and offloading with constrained hybrid action space in mobile-edge computing. IEEE Internet Things J 9:19260–19272
39. Sadiki A, Bentahar J, Dssouli R, En-Nouaary A, Otrok H (2021) Deep reinforcement learning for the computation offloading in mimo-based edge computing. Ad Hoc Netw 141:103080
40. Zhang Y, Zhang Z, Chen L, Wang X (2021) Reinforcement learning-based opportunistic routing protocol for underwater acoustic sensor networks. IEEE Trans Veh Technol 70:2756–2770
41. He J (2023) 5g communication resource allocation strategy for mobile edge computing based on deep deterministic policy gradient. J Eng. https://doi.org/10.1049/tje2.12250

Authors and Affiliations

Ruizhong Du[1,2] · Jingya Wang[1,2] · Yan Gao[3]

✉ Jingya Wang
wjy1583616@163.com

Ruizhong Du
durz@hbu.edu.cn

Yan Gao
gymorsiback@gmail.com

[1] School of Cyber Security and Computer, Hebei University, Baoding 071000, Hebei, China

[2] Hebei Province Key Laboratory of High Confidence Information System, Hebei University, Baoding 071000, Hebei, China

[3] School of New Media and Communication, Tianjin University, Tianjin 300000, China