Li *et al. J Wireless Com Network* (2021) 2021:56
https://doi.org/10.1186/s13638-021-01941-3

EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

Energy-aware task offloading with deadline constraint in mobile edge computing

Zhongjin Li[1], Victor Chang[2*], Jidong Ge[3], Linxuan Pan[3], Haiyang Hu[1] and Binbin Huang[1]

*Correspondence:
V.Chang@tees.ac.uk
[2] School of Computing and Digital Technologies, Teesside University, Middlesbrough, UK
Full list of author information is available at the end of the article

Abstract
With the development of the wireless network, increasing mobile applications are emerging and receiving great popularity. These applications cover a wide area, such as traffic monitoring, smart homes, real-time vision processing, objective tracking, and so on, and typically require computation-intensive resources to achieve a high quality of experience. Although the performance of mobile devices (MDs) has been continuously enhanced, running all the applications on a single MD still causes high energy consumption and latency. Fortunately, mobile edge computing (MEC) allows MDs to offload their computation-intensive tasks to proximal eNodeBs (eNBs) to augment computational capabilities. However, the current task offloading schemes mainly concentrate on average-based performance metrics, failing to meet the deadline constraint of the tasks. Based on the deep reinforcement learning (DRL) approach, this paper proposes an Energy-aware Task Offloading with Deadline constraint (DRL-E2D) algorithm for a multi-eNB MEC environment, which is to maximize the reward under the deadline constraint of the tasks. In terms of the actor-critic framework, we integrate the action representation into DRL-E2D to handle the large discrete action space problem, i.e., using the low-complexity k-nearest neighbor as an approximate approach to extract optimal discrete actions from the continuous action space. The extensive experimental results show that DRL-E2D achieves better performance than the comparison algorithms on all parameter settings, indicating that DRL-E2D is robust to the state changes in the MEC environment.

Keywords: Mobile edge computing, Task offloading, Energy and deadline-aware, Deep reinforcement learning (DRL), Actor-critic framework, K-nearest neighbor (K-NN)

1 Introduction

Many mobile applications are emerging with the development of a wireless network for mobile users, which are computation-intensive and time-sensitive, such as traffic monitoring, smart homes, real-time video analysis, and object tracking. Recently, mobile devices (MDs) from Apple, Samsung, and Huawei companies have more powerful computation capability [1]. For example, the up-to-date Apple iPhone 12 is equipped with a neural network engine and multi-core CPUs, which can process various computation-intensive mobile applications. Unfortunately, due to the constraint of the physical size of MDs, the computation resource and battery capacity are still limited. That is, running all the mobile applications on a single MD may cost comparatively battery capacity, i.e., the energy consumption of MD's battery.

One of the solutions to reduce energy consumption is to migrate tasks to the cloud computing platform by a wireless network, which is the concept of mobile cloud computing (MCC) [2]. In MCC, a MD can exploit the powerful distant centralized cloud computing resources but introducing a long transmission time that cannot meet the demand of delay-sensitive mobile applications.

With the development of 5G techniques, more and more eNodeBs (eNBs) will be deployed in a smart city. These eNBs form the service provisioning of computing, storage, and network for mobile users, contributing to the concept of mobile edge computing (MEC). MEC pushes the resources to the edge node, called eNB, and consolidates the functions of the base station and physical servers [3–5]. Thus, MDs can access the network, storage, and computation resources from the proximal eNBs. However, a technique report from Cisco predicts that by 2021, about 11.6 billion MDs will utilize the wireless network worldwide [6]. Therefore, deploying high-dense eNBs is required to cope with a large number of MDs presenting simultaneously [7, 8]. Thus, in the MEC environment, each MD can offload its computation-intensive and latency-critical tasks to multiple proximal eNBs. However, improper task offloading schemes result in more energy consumption and higher latency.

Task or computation offloading in the MEC environment has already been extensively studied. Traditional offloading schemes are model-based, i.e., usually assume that the mobile signal between MD and eNB is well modeled [7, 9–11]. However, the MEC environment is very complicated, and the users' mobilities are highly dynamic, making the mobile models hard to construct and predict. Hence, a novel model-free approach should be developed to control well the mobile network, instead of an accurate mathematical model.

In 2015, Mnih et al. [12] integrated the mechanism of reinforcement learning into deep learning, yielding the concept of deep reinforcement learning (DRL), which has been successfully applied in playing Atari games by making use of deep Q-network (DQN). Then, in 2016, Google's DeepMind team developed a famous DRL application, AlphaGo program, defeating the most professional Go players [13]. Hence, DRL has made a breakthrough in model-free learning. It has recently become a promising approach to implement task offloading in MEC and works well in practice [14–19].

The benefits of using the DRL method to implement task offloading in MEC are threefold: (1) DRL is a model-free optimization method, i.e., it does not need any model-based mathematical knowledge; (2) it can tackle the optimization problem in a highly dynamic time-varying system; and (3) it can handle the large state and action space problems (such as Atari games [12]). The above features indicate that DRL is the ideal way to implement task offloading in MEC.

However, applying the DRL technique for MEC task offloading should consider and solve the following problems. First, our proposed task offloading problem in MEC with high-dense eNBs is a large discrete action space problem. For instance, with five eNBs in MEC for an MD to offload 20 tasks, there exist more than 50 thousand offloading actions. In this case, DQN based DRL cannot work well, as it only has the potential to cope with the small action space problem. Second, task offloading is a discrete control problem, and hence, the continuous control methods, such as deep deterministic policy gradient (DDPG) [20], cannot work.

This paper proposes a DRL-based model-free task offloading algorithm, called DRL-E2D, with a large discrete action space for the multi-eNB MEC environment. In such an environment, an MD can offload its tasks to multiple eNBs with the deadline constraint of the tasks. The aim of DRL-E2D is to optimize the long-term MD's energy consumption by jointly learning the neural network from the unknown environment and making optimal decisions under the current system state. We summarize our contributions as below:

- In contrast with the model-based method, we study the energy-aware task offloading problem with the deadline constraint and propose a novel model-free DRL task offloading algorithm, called (DRL-E2D) a multi-eNB MEC environment.
- DRL-E2D deals with the task offloading problem by maximizing the well-designed reward, which is the weighted sum of the utility of processed tasks, energy consumption, and the penalty of task dropping. Thus, in each time slot, according to the system status, it makes the optimal actions on how many tasks should be processed by MD and how many tasks should be offloaded to eNBs.
- As discussed above, the task offloading in a multi-eNB environment is a large discrete action space problem. We first adopt an actor-critic framework to implement the learning process, and then, k-nearest neighbor (K-NN)-based action representation is used to extract optimal and discrete action from the continuous action space.
- By the extensive simulation, we first find that the K-NN approach indeed makes the DRL-E2D converge to a better reward. Compared with load balancing, remote execution, local execution, and DQN, DRL-E2D outperforms these baseline algorithms on the ultimate reward.

This paper is organized as follows: In Sect. 2, we summarize the existing work about task or computation offloading in MEC. In Sect. 3, we introduce our devised system architecture, present the task's parameter computation, and formulate the optimization problem. In Sect. 4, we elaborate on the methods and experimental design, including the preliminary of the DRL, DRL-E2D algorithm, and experimental simulation. In Sect. 5, we present and discuss the experiment results. Finally, in Sect. 6, we conclude this paper and propose future research directions.

2 Related work

There have been many existing studies on task or computation offloading in MEC. This section classifies the corresponding algorithms as general approaches and DRL-based approaches for task offloading.

2.1 General approaches for task offloading

The general approaches summarized in this paper mean using model-based optimization methods (such as game theory, queuing theory, Lyapunov, and so on) to implement task or computation offloading. For example, Chen et al. [11] put forward the multi-user offloading problem in mobile-edge cloud computing and achieve efficient offloading by leveraging the game theory. Beraldi et al. [21] consider a cooperative computation offloading problem between two eNBs. This problem is formulated as an M/M/2-2K queuing model and applies the trade-off policy to reduce data exchange while minimizing system congestion. However, this cooperative scheme cannot be used for our task offloading problem, but it may be extended to the D2D communication environment. Moreover, the above two works need to rely on an accurate mathematical model to implement task offloading.

Wang et al. [22] develop a framework of combing computation offloading with interference management, where graph coloring is used for the MEC server to make the offloading decision in terms of local computation overhead and offloading overhead. To address the conflict between MEC and Cloud radio access network (C-RAN), Li et al. [10] investigate the control decision of task offloading in C-RANs environments. A multi-phase heuristic method is introduced through the Gale-Shapley Matching Theory to minimize the rejection rate of offloading requests. However, the above two offloading methods are applied for MEC servers instead of MD. Dinh et al. [8] propose a computational offloading framework to minimize the joint cost of energy and delay and devise two approximate approaches to solve the generated mixed-integer linear programming (MILP) problem. Chen et al. [7] investigate task offloading problems in ultra-dense eNBs environment to minimize the delay and energy consumption and employ KKT condition and iterative algorithm to implement resource allocation and task placement, respectively. However, they mainly focus on deriving a one-shot offloading strategy that is not appropriate for long-term performance optimization.

Miao et al. [23] propose a prediction-based computation offloading and task migration algorithm to reduce the processing delay of users' applications. However, it is impossible to predict future values accurately, especially in such a dynamic and random MEC environment. Yan et al. [24] research the joint task offloading and resource allocation problem by considering both the energy consumption and execution time. In order to tackle this problem, an effective bisection search policy is used to compute optimal solutions. However, it considers offloading the tasks with dependency relations and cannot be used for independent tasks directly. To reduce the pressure on UAV-aided MEC servers, Wang et al. [25] propose an agent-enabled task offloading framework to perform the optimum offloading. However, it is difficult and unrealistic to obtain environmental information to model an agent in a dynamic and ultra-dense MEC environment.

Lyapunov optimization framework has been extensively applied in MEC task offloading. Liu et al. [26, 27] study the power-delay trade-off problem with multi-user task offloading situation, and Lyapunov stochastic optimization is used to solve this optimization problem under the constraint of delay and reliability. Lyu et al. [28] propose an asymptotically optimal offloading schedule under partial network knowledge based on the perturbed Lyapunov function to maximize the network utility. Sun et al. [29] develop energy-aware mobility management (EMM), aiming to utilize Lyapunov and multi-armed bandit theories to minimize the total delay of communication, computation, and handover. Li et al. [30] investigate the multi-user offloading problem in WP-MEC and propose a Lyapunov-based online computation rate maximization (OCRM) algorithm to achieve optimal bandwidth allocation and minimum energy consumption of data transmission. Lyapunov optimization-based approaches are suited for solving the problem with long-term performance optimization. However, each performance in objective function must be the time-average metric, e.g., average energy or power, average delay, and average cost. So, it cannot be applied in the optimization problem with deterministic constraints, such as the deadline constraint of each task.

2.2 DRL-based approaches for task offloading

DRL-based task offloading has also been considered in many works. Li et al. [15] research the service provision of task offloading in the UAV-based MEC environment, which is modeled as a semi-Markov decision process, and then is solved by the DRL method to maximize the system throughput. However, the size of the action space is the number of perceptual access points, which can only solve the problem with a small action space. Huang et al. [31] propose a DRL-based online offloading (DROO) framework for offloading users' tasks and allocating eNBs' network resources. The DROO only considers a binary offloading policy, i.e., a task can be executed locally or offloaded to the eNB. Hence, it cannot be applied in the multi-eNB MEC environment. Lei et al. [32] study the joint optimization problem of computation offloading and resource scheduling by using the DRL technique to optimize the long-term average performance of delay and energy consumption. However, the solved task offloading and rescheduling problem only considers the small action space with three offloading actions and $N + 1$ scheduling actions.

The DQN-based optimization method can achieve significant performance in high-dimensional state space problems, which has also been extensively applied in MEC task offloading. For example, Le et al. [16] propose a DQN-based optimal offloading policy to minimize the overall performance of energy, delay, payment, and task loss in the ad-hoc mobile cloud. However, the task delay is the sum of average values of the waiting time, communication time, and processing time. This kind of average delay cannot be applied to time-critical mobile applications. Li et al. [17] consider the multi-user optimization problem of computation offloading and resource allocation and employ a DQN-based optimization algorithm to minimize the sum of delay and energy consumption. However, in this multi-user offloading case, each mobile user only processes one task at every time slot. Hence, it only needs to consider two actions, i.e., local computing or offloading computing, which is only applicable to small action space problems. In [18], a deep Q-learning-based autonomic offloading framework is introduced to mitigate the service delay by making the optimal decisions according to current resource requirement, resource state, and network state. However, the autonomic offloading framework considers an MD can offload tasks to three different MEC sites, and hence only ten actions that the agent can take. Lu et al. [19] propose a DRL-based lightweight task offloading strategy for large-scale heterogeneous MEC, which integrates the LSTM network into DQN to improve the learning performance. However, the reward function is defined as the weighted sum of energy consumption, cost, and load balancing of all devices, neglecting the time performance, which is very critical for mobile task execution. To guarantee data security, Huang et al. [33] propose a security and cost-aware computation offloading (SCACO) strategy. The offloading problem is formulated as an MDP problem, and a DQN-based optimal offloading policy is devised to minimize the joint cost under the risk constraint. Similar to [16], SCACO takes the average delay, which is the average value of execution time, processing time, and security time, into account, not the deadline constraint.

DQN-based algorithms work well for the optimal problem with small action space, but not for the large discrete action space problem [34]. However, all the studies mentioned above take the task processing time as the average performance demand. That is not quite reasonable for mobile applications, which take the time as the most concern. This paper also considers the task execution time with deadline constraints.

3 System architecture, model, and problem formulation

This section first presents the system architecture of task offloading in the MEC environment. Then, the task execution model is introduced. Finally, we give the problem formulation. The basic notations used in this paper are given in Table 1.

3.1 System architecture

Figure 1 shows the architecture of task offloading for multi-eNB MEC environment, which consists of single MD and n eNBs denoted as a set $\mathcal{N} = \{1, 2, \ldots, \mathrm{n}\}$. The system time is divided into slots of equal duration; thus,$\tau \in \{0, 1, 2, \ldots\}$. At the beginning of each time slot τ, MD generates mobile applications, such as real-time video processing, smart home applications, and some artificial intelligence (AI) applications. The number of tasks arriving at MD is represented by $z(\tau) \in Z = \{0, 1, \ldots, z_{\max}\}$. The task arrival process $\{z(\tau)\}$ is assumed to be an independent and identically distributed (i.i.d.) sequence with $\mathbb{E}\{z(\tau)\} = \lambda$. We assume that each arrived task has a constant data size D and execution workload W.

In our task offloading scenario, each task can be arbitrarily offloaded to an eNB by a wireless network or processed locally. Let $\eta_i(\tau)$ denote the data transmission rate (i.e., signal strength) from MD to eNB$_i$ in time slot τ, which is calculated by [7, 8, 29]

Table 1 Notations

Name	Description
n	The number of eNBs in the MEC environment
T_{slot}	The duration of each time slot
W	The task workload
D	The data size of the task
λ	The task arrival rate at MD
T_{DL}	The task's deadline constraint
$z(\tau)$	The number of arrived tasks at MD at time slot τ
$\eta_i(\tau)$	The data transmission rate from MD to eNB$_i$ in time slot τ
$\alpha_i(\tau)$	The number of tasks offloaded to eNB$_i$ in time slot τ
$\beta_i(\tau)$	The number of tasks processed in time slot τ
$c_i(\tau)$	The computation capacity of MD or eNB$_i$
$T_i^{\text{tx}}(\tau)$	The data transmission time of offloading task from MD to eNB$_i$
$T_i^{\text{ex}}(\tau)$	The execution time of a task on MD or eNB$_i$
$E(\tau)$	The total energy consumption of MD at time slot τ
$U(\tau)$	The total utility at time slot τ
$P(\tau)$	The total penalty of task dropping at time slot τ
$R(\tau)$	The total reward at time slot τ

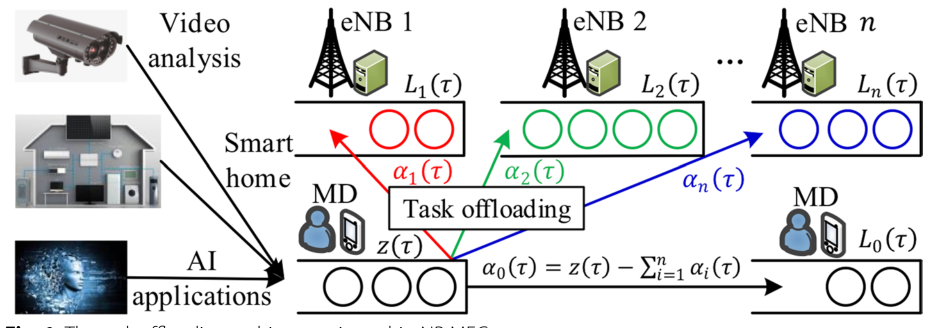

Fig. 1 The task offloading architecture in multi-eNB MEC

$$\eta_i(\tau) = B_i \log_2 [1 + \mathrm{SNR}_i(\tau)] \tag{1}$$

where B_i is the bandwidth eNB$_i$ allocates for the MD, and $\mathrm{SNR}_i(\tau) = p_0^{\mathrm{tx}} g_i(\tau)/\sigma^2$ denotes the signal-to-noise ratio (SNR), where p_0^{tx} represents the transmission power of MD, σ^2 is the Gaussian white noise power, and $g_i(\tau) = \varphi[d_i(\tau)]^{-\theta}$ denotes the channel gain, where φ and θ are the path-loss constant and path-loss exponent, respectively, and $d_i(\tau)$ is the path distance. We assume that during each time slot τ, the MD does not move much and hence $\eta_i(\tau)$ is a constant.

Let $\alpha_i(\tau)$ denote the number of tasks offloaded to eNB$_i$, which must be taken from the feasible set, i.e., $\alpha_i(\tau) \leq z(\tau), \forall i \in \mathcal{N}$. Let $\beta_i(\tau)$ denote the number of tasks processed by eNB$_i$ at time slot τ, and $\beta_i(\tau) \in \{0, 1, \ldots, \beta_{\max}\}$. Then, the evolution equation of the task processing queue of eNB$_i$ is expressed as

$$L_i(\tau + 1) = \max \{L_i(\tau) - \beta_i(\tau), 0\} + \alpha_i(\tau) \tag{2}$$

After offloading $\sum_{i=1}^{n} \alpha_i(\tau)$ tasks to eNBs, $z(\tau) - \sum_{i=1}^{n} \alpha_i(\tau)$ tasks will be put into the task processing queue of MD. Then, the corresponding evolution equation is

$$L_0(\tau + 1) = \max \{L_0(\tau) - \beta_0(\tau), 0\} + \alpha_0(\tau) \tag{3}$$

where $\alpha_0(\tau) = z(\tau) - \sum_{i=1}^{n} \alpha_i(\tau)$. Note that we use 0 to represent the index of MD and use i to represent the index of eNB.

3.2 Task execution model

A task can be handled locally or be offloaded to the eNBs. Here, we discuss the processes of task local execution and task eNB execution, respectively. Typically, MD is equipped with M-core CPU ($M \geq 2$). The state-of-the-art MD adopts an advanced DVFS technique, allowing automatic CPU frequency regulation. Suppose the working frequency of mth CPU core is $F_m(\tau)$, and then, the computing power of this MD can be expressed as [31, 35]

$$p_0^{\mathrm{ex}}(\tau) = \delta \sum_{m=1}^{M} F_m^3(\tau) \tag{4}$$

where δ is a constant related to the chip architecture. In practice, the level of CPU frequency is discrete and is bounded by a reasonable range, i.e., $F_m(\tau) \in \left\{F^{\min}, \ldots, F^{\max}\right\}$.

However, in [36], a conclusion has been justified that homogeneous CPU cores should have the same operating frequency to reach the optimal power consumption, i.e., $F_m(\tau) = F(\tau)$. So, Eq. (4) can be rewritten as

$$p_0^{\mathrm{ex}}(\tau) = \delta M F^3(\tau) \tag{5}$$

The computational performance is also controlled by the CPU frequency. Besides, we have the relation between the computation capacity and CPU frequency, i.e., $c = F$. Then, we have

$$c_0(\tau) = MF(\tau) \tag{6}$$

Thanks to the DFVS technique, the computation capacity $c_0(\tau)$ of MD changes with $F(\tau)$ variation. That means MD can adjust $c_0(\tau)$ according to the computation requirement at time slot τ to optimize the energy consumption. Thus, if a task is executed on MD, the execution time $T_0^{\mathrm{ex}}(\tau)$ is calculated by

$$T_0^{\mathrm{ex}}(\tau) = W/c_0(\tau) \tag{7}$$

Then, the energy consumption $E_0^{\mathrm{ex}}(\tau)$, resulting from task execution on MD, is expressed as

$$E_0^{\mathrm{ex}}(\tau) = P_0^{\mathrm{ex}}(\tau) T_0^{\mathrm{ex}}(\tau) \tag{8}$$

However, if MD offloads a task to eNB, the offloading process will be divided into two steps. First, MD needs to select an eNB and transmit it by uploading corresponding input data through the wireless access network. Then, the data transmission time $T_i^{\mathrm{tx}}(\tau)$ of offloading task from MD to eNB$_i$ is computed by

$$T_i^{\mathrm{tx}}(\tau) = D/\eta_i(\tau) \tag{9}$$

Meanwhile, the energy consumption of the data transmission is represented by

$$E_i^{\mathrm{tx}}(\tau) = p_0^{\mathrm{tx}} T_i^{\mathrm{tx}}(\tau) \tag{10}$$

Second, after receiving the tasks, eNB$_i$ put them into the processing queue $Q_i(\tau)$ and process them by the first-come-first-serve (FCFS) rule. Let $c_i(\tau)$ denote the processing capacity that eNB$_i$ allocates for MD. Then, the processing time of a task on the eNB$_i$ is given as

$$T_i^{\mathrm{ex}}(\tau) = W/c_i(\tau) \tag{11}$$

Note that as the result of each task is the small data, we neglect the transmission time and energy consumption by returning the execution results from eNBs to MD.

3.3 Problem formulation

This section formulates the task offloading as an optimization problem. The objective is to maximize the joint reward, including the utility of the finished task, the energy consumption of MD, and the penalty of task dropping.

At the beginning of each time slot τ, MD will generate $z(\tau)$ tasks. Then, it will decide how many tasks should be offloaded to each eNB, i.e., $(\alpha_1(\tau), \ldots, \alpha_i(\tau), \ldots, \alpha_n(\tau))$, and how many tasks should be processed locally, i.e., $\beta_0(\tau)$. According to Eqs. (8) and (10), the total energy consumption at time slot τ is computed as follows:

$$E(\tau) = \beta_0(\tau)E_0^{\mathrm{ex}}(\tau) + \sum_{i=1}^{n} \alpha_i(\tau)E_i^{\mathrm{tx}}(\tau) \tag{12}$$

where $\beta_0(\tau)E_0^{\mathrm{ex}}(\tau)$ is the execution energy for processing $\beta_0(\tau)$ tasks, and $\sum_{i=1}^{n} \alpha_i(\tau)E_i^{\mathrm{tx}}(\tau)$ is the transmission energy for offloading $\sum_{i=1}^{n} \alpha_i(\tau)$ tasks to eNBs.

The task's deadline constraint is considered. Let T_{DL} denote the deadline constraint for all the tasks. If the waiting time and execution time of task t_j is less than and equal to the deadline, i.e., $T\left(t_j\right) \leq T_{\mathrm{DL}}$, the MD gains the u utility of successfully finished task; otherwise, MD obtain zero utility. The above representation can be expressed as Eq. (13).

$$u\left(t_j\right) = \begin{cases} u, & T\left(t_j\right) \leq T_{\mathrm{DL}} \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

Note that the deadline constraint is measured by the number of the time slot. For example, we can set $T_{\mathrm{DL}} = N_{\mathrm{DL}}T_{\mathrm{slot}}$, which means the task should be executed within N_{DL} time slots once it is generated. Also, it is worth pointing out that if a task is generated at time slot τ, the start execution time is τ if it is processed by MD; the start execution time is $\tau + 1$ if it is offloaded to an eNB. Then, the total utility at time slot τ is represented by

$$U(\tau) = \sum_{i=0}^{n} \sum_{j=1}^{\beta_k(\tau)} u\left(t_j\right) \tag{14}$$

However, if a task misses the deadline, this task will be dropped, incurring the penalty. Then, the penalty for dropping a task at time slot τ is computed by

$$P(\tau) = \sum_{i=0}^{n} d_i(\tau) \tag{15}$$

where $d_i(\tau)$ is the number of dropped tasks of each eNB and MD. We can see from Eqs. (12)–(15) that MD wants to maximize the utility, incurring less the penalty of dropped tasks but more energy consumption. So, there exists the performance trade-off among the utility, energy consumption, and penalty.

Under the deadline constraints, the optimization problem of task offloading is formulated as follows:

$$\max : R(\tau) = U(\tau) - E(\tau) - P(\tau) \tag{16a}$$

$$\text{subject to} : \sum_{i=0}^{n} \alpha_i(\tau) \leq z(\tau) \tag{16b}$$

$$\alpha_i(\tau) \leq \eta_i(\tau)/D, \forall i \in \{0\} \cup \mathcal{N} \tag{16c}$$

$$\sum_{i=1}^{n}[\alpha_i(\tau)/\eta_i(\tau)] \leq T_{\text{slot}} \tag{16d}$$

$$\beta_i(\tau) \leq c_i(\tau)/W, \forall i \in \{0\} \cup \mathcal{N} \tag{16e}$$

where Eq. (16a) is the objective function of task offloading. Equation (16b) is the constraint of the number of offloaded tasks. Equation (16c) is the constraint of transmission capacity for each link between MD and eNB$_i$. Equation (16d) is the time constraint for offloading tasks, and Eq. (16e) is the computing capacity constraint.

4 Methods/experimental

Directly solving the optimization problem requires a set of a priori information from the multi-eNB MEC system, such as $\eta_i(\tau)$ and $z(\tau)$; however, which cannot be obtained in advance. Hence, we propose a model-free learning strategy DRL-E2D to tackle this kind of stochastic optimization problem. DRL-E2D is based on reinforcement learning (RL), so we first introduce the knowledge of DL. Then, we present the DRL-based task offloading process with the actor-critic framework. Moreover, we also design an experimental simulation to evaluate the performance of the proposed method and present the process in detail.

4.1 Preliminary of RL

RL system has an agent responsible for making optimal decisions, and anything outside the agent is called the environment. The interactions between the agent and the environment are described via three essential elements: state, action, and reward. Specifically, the agent and environment interact at each sequence of discrete-time steps, $\tau = \{0, 1, 2, 3, \ldots\}$. At each time step τ, the agent observes the system state s_τ from the environment and then learns the knowledge, and on that basis selects an action a_τ. One time step later, in part as a consequence of its action, the agent receives a numerical reward $r_{\tau+1}$, and finds itself in a new state $s_{\tau+1}$. The learning process of the RL system can be found in Fig. 2 [37]. The interaction between the agent and environment is generally achieved by the Markov Decision Process (MDP), a powerful dynamic optimization theory. RL has already been considered as a very popular model-free decision method.

This paper integrates RL into task offloading to form a fast and effective offloading policy for the multi-eNB MEC environment. In this case, the agent makes the task offloading strategy over a sequence of time steps with a stochastic system state to maximize

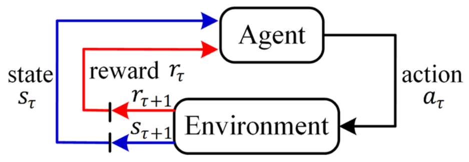

Fig. 2 The agent–environment interaction in MDP

the cumulative reward. We first model our optimal offloading problem as an MDP, consisting of four components: a state-space S, an action space A, a transition probability distribution Γ, and a reward function R [37, 38].

The state at each time slot τ is represented as s_τ. The agent observes s_τ and makes an offloading action a_τ accordingly. Then, the environment changes the current state s_τ to the next state $s_{\tau+1}$ and offers a corresponding reward $r_{\tau+1}$. Typically, the interactions between agent and environment are viewed as a series of states, actions, and rewards: $s_0, a_0, r_1, s_1, a_1, \ldots, r_\tau, s_\tau$. Note that the agent interacts with the environment producing a series of states, actions, and rewards from an initial state to the terminal state, which is called an episode.

A policy π is the mapping function from state s to action a. The objective is to find a deterministic policy π to maximize the discounted cumulative reward $R_0 = \sum_{\tau=0}^{\mathrm{T}} \gamma^\tau r(s_\tau, a_\tau)$, where $r(\cdot)$ is the reward function, and $\gamma \in [0,1]$ is the discount rate. Then, the state-action value function $Q^\pi(s,a) = \mathbb{E}[R_1 | s_0 = s, a_0 = a, \pi]$ is the expected return with state s, action a, and policy π. $Q^\pi(s,a)$ can be calculated by the Bellman optimality equation, which expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state.

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) Q^\pi\left(s', \pi\left(s'\right)\right) \tag{17}$$

where both Q and π are approximated by parametrized functions. Then, the optimal policy $\pi_Q(s)$ is the policy that maximizes the $Q^\pi(s,a)$.

$$\pi_Q(s) = \arg\max_{a \in A} Q(s,a) \tag{18}$$

This paper considers a case of task offloading from a single MD to multiple eNBs. In this case, the MD is fully aware of the observation and will select actions by influencing state changes. In other words, this is in line with the MDP. Next, we will present the actor-critic approach for implementing the task offloading problem with a large discrete action space.

4.2 DRL-based task offloading

The RL method is short for the problem with large discrete state space, i.e., it faces the well-known curse-of-dimensionality problem. Fortunately, Mnih et al. [12] have successfully combined RL with deep learning, called DRL, yielding promising effects. We use the DRL-based method to cope with task offloading for large discrete action space. The system architecture builds upon the actor-critic framework [34], where the actor is a reference to the learned policy, and critic refers to the learned value function, usually a state-value function. In the actor-critic framework, the policy is represented as $\pi_\theta : S \to A$, where π_θ is the function approximator, which is the multi-layer deep neural network. DDPG is adopted to train the optimal offloading policy. However, it is extensively used in the RL problem with continuous action space. So, we should extract the discrete actions from the continuous action space, which is solved by the K-NN approach introduced next. The architecture of DRL-E2D is shown in Fig. 3. We first design the state space, action space, and reward function of the multi-eNB task offloading problem.

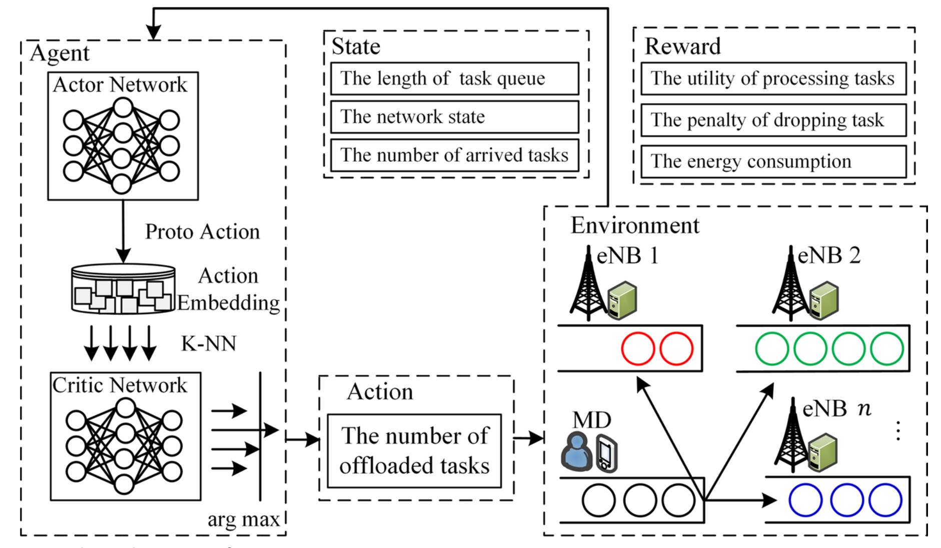

Fig. 3 The architecture of DRL-E2D

- *State Space* The state consists of three components: the length of the task processing queue, the data transmission rate, and the number of arrived tasks at time slot τ. Formally, the state vector $s_\tau = [L_0(\tau), L_1(\tau), \ldots, L_n(\tau), \eta_1(\tau), \ldots, \eta_n(\tau), z(\tau)]$, where $L_i(\tau)$ represents the queue length of the tasks to be processed in eNB$_i$, $L_0(\tau)$ represents the queue length of the tasks to be processed in MD, $\eta_i(\tau)$ represents the network state between MD and eNB$_i$, and $z(\tau)$ is the number of tasks arriving at MD.
- *Action Space* Under the system status, MD chooses an action to perform. An action is defined as the solution to task offloading. Formally, the action vector $a_\tau = [\alpha_0(\tau), \ldots, \alpha_n(\tau), c_0(\tau)]$, where each action contains the number of tasks $\alpha_i(\tau)$ that MD offloads to each eNB, the number of tasks reserved in local $\alpha_0(\tau)$, and the computing capacity of MD regulated by DVFS. Note that $\alpha_0(\tau)$ is also related to $F(\tau)$.
- *Reward* The reward is described in Eq. (16a), which is the integration of utility, energy, and penalty.

Due to the well-known dimension disaster problem, which generally refers to the exponential growth of state space with the change of a certain quantity, it will be very difficult to solve this kind of MDP model. With the increase in the number of connected eNBs and the maximum number of tasks, the state space will explode. This situation will lead to traditional reinforcement learning, such as Q-learning, SARSA, and so on, unable to apply. Because the Q table will be too large, the storage resources will take up too much, and the query is complex. It is not suitable for mobile devices, so we use DRL to solve this problem. We can use the neural network to perceive and understand the state and then decide for action. In this way, the problem of dimension explosion of state space can be alleviated to some extent.

However, it is not only the state space that causes dimensional disaster, but also the fact that an environment can have a large number of actions. In contrast to existing studies, our task offloading problem has a large number of discrete actions. This leads to the poor performance of the traditional DRL algorithm, DQN, which is usually used to solve small discrete action space problems. So, we combine the actor-critic framework with the K-NN algorithm to cope with the situation of high-dimensional state space and action space at the same time [34].

As can be seen from Fig. 3, first, the action space is embedded in advance. Embedding tries to find the relationship between discrete values and express it as the distance in continuous space. Here, embedding can be based on the complexity of action. If the action is relatively simple, it can be completed by using simple mapping or coding. If the action is relatively complex, it will be more convenient to use a neural network. We can get the "distance" between actions by embedding. So, we can get a set of actions by the K-NN method. All of them are then input into a critical network to get the Q value of each nearest neighbor and select the maximum Q value to execute. Through the operation of K-NN, it is equivalent to reduce the dimension of action, which can effectively improve the training effect and efficiency. Of course, it is also important to choose the k value. Too large or too small may lead to training failure or too long.

The pseudo-code of DRL-E2D is shown in Algorithm 1. First, the network parameters and experience pool are initialized, and we start from line 4 for each episode (lines 1–4). Then, we initialize the environment and the random model for exploration (lines 5–6). Next, MD collects a certain number of tasks and then obtains the observations from the environment (lines 7–8). For each time slot τ (lines 9–17), we first input the current state into the actor-network, obtain an initial action, and then get proto-action with the additional random process to explore. Then, k nearest neighbors are extracted through the action embedding. Next, these selected actions are, respectively, inputted into the critical network to get the Q value of each action, and the action with the largest Q value will be selected. The traditional DDPG process starts from lines 13 to 16 with performing tasks, calculating returns, and observing new states. Then, it stores the quadruples in the experience pool and randomly takes a mini-batch from the experience pool for network update.

Algorithm 1. DRL-E2D Algorithm

```
1. Initialize actor-critic network with weights $\theta^{\mu}$ and $\theta^{Q}$;
2. Initialize target network with weights $\theta^{\mu'}$, $\theta^{Q'}$ by $\theta^{\mu}$, $\theta^{Q}$;
3. Initialize replay buffer $\Omega$;
4. for episode = 1 to MAX_EPISODE do
5.     Initialize the environment;
6.     Initialize a random process $N$ for action exploration;
7.     $z(\tau)$ tasks arrive at MD;
8.     Receive the state $s_0$ by MD observing from the environment;
9.     for $\tau$ = 1 to MAX_STEP do
10.        Select proto-action $a_p = \pi_{\theta^{\mu}}(s_\tau) + N_\tau$;
11.        Search k nearest neighbors of proto-cation $A_k = k(a_p)$ in action embedding;
12.        Select action $a_\tau = \mathrm{argmax}_{a_x \in A_k} Q(s_\tau, a_x | \theta^{Q})$;
13.        MD execute action $a_\tau$, calculate reward $r_\tau$ and observe new state $s_{\tau+1}$;
14.        Store $(s_\tau, a_\tau, r_\tau, s_{\tau+1})$ in replay buffer $\Omega$;
15.        Sample a random mini-batch of transitions from $\Omega$;
16.        update critic, actor policy and target networks as DDPG;
17.    end for
18. end for
```

4.3 Experimental simulation

We consider a multi-eNB MEC environment with n eNBs, which are homogeneous and allocate the same computing capacity $c_i(\tau) = 10\,\text{GHz}$ for MD's task processing. At each time slot τ, a batch of tasks arrives at the MD with the arrival rate $\lambda = 10$. Suppose each arrived task has the same workload $W = 2.5\,\text{GHz s}$, data size $D = 10\,\text{MB}$, and deadline constraint $T_{\text{DL}} = 3T_{\text{slot}}$ (i.e., $N_{\text{DL}} = 3$). So, if a task is finished within the deadline constraint, MD will receive the utility $u = 1$. For the task offloading, the transmission power of MD is $p_0^{\text{tx}} = 250\,\text{mW}$ [39], the wireless network bandwidth is $B = 100\,\text{MHz}$, and the other relative parameters are set as follows: $\sigma^2 = -174\,\text{dbm/Hz}$, $\varphi = 0.01$, and $\theta = 4$ [40].

We assume that the number of CPU cores of MD is $M = 4$, and the working frequency of each CPU core is 2.0 GHz and can be adjusted in the set $\{1, 1.5, 2\}$ by the DVFS technique. Suppose the computational power is 900 mW, and according to the equation $p_0^{\text{ex}} = \delta M F^3$, δ is computed as $900/2^3 = 112.5\,\text{mW}/(\text{GHz})^3$ [8]. We set the length of each time slot $T_{\text{slot}} = 1$ second, and the number of time slots $N_{\text{slot}} = 1000$, i.e., each episode includes 1000 iterations, which is long enough for obtaining the stable results.

In addition, three algorithms used to compare with our DRL-E2D are introduced below.

Load Balance (LB) At each time slot τ, the arrived tasks are evenly distributed to the MD and all the eNBs according to their computing capacity, i.e.,

$$\alpha_i(\tau) = \frac{C_i(\tau)}{\sum_{j=0}^{n} C_j(\tau)} z(\tau) \tag{19}$$

where $c_0(\tau) = c_0^{\max}$. Note that $\alpha_i(\tau)$ is an integer, and the residual tasks $z(\tau) - \sum_{i=0}^{n} \alpha_i(\tau)$ will be executed on MD.

Remote Execution (Remote) All the tasks arrived at MD are routed to the eNBs, and the number of allocated tasks of each eNB is computed by the data transmission rate $\eta_i(\tau)$, i.e.,

$$\alpha_i(\tau) \leftarrow \frac{\eta_i(\tau)}{\sum_{j=1}^{n} \eta_j(\tau)} z(\tau) \tag{20}$$

Note that if $\alpha_i(\tau) > \eta_i(\tau)/D$, the number of $z(\tau) - \sum_{i=1}^{n} \max\{\alpha_i(\tau) - \eta_i(\tau)/D, 0\}$ tasks will be dropped due to insufficient transmission capacity.

Local Execution (Local) All the tasks are executed on MD. So, there does not exist the task offloading in this case.

DQN [12] DQN is a classic deep reinforcement learning method, which has been widely used in many areas, e.g., task offloading and resource allocation [16–19, 33]. However, it can only address the problem with a small action space and cannot work well in the problem of large action space [34].

Our proposed DRL-E2D algorithm is implemented by Tensorflow on Python. We run and train the neural network model on the server with GPU: NVIDIA Tesla K40m @ 12 GB and CPU: Intel Xeon E5-2620 v4 @ 2.10 GHz.

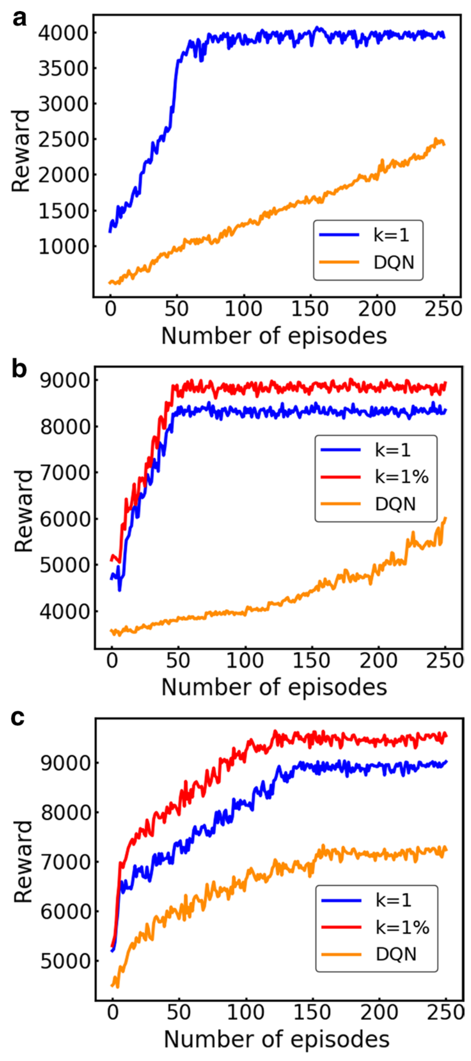

Fig. 4 The convergence of DRL-E2D on different n. **a** $n = 1$. **b** $n = 3$. **c** $n = 5$

5 Results and discussion

This section focuses on results analysis and the discussion of simulation experiments, where we compare the DRL-E2D algorithm with other comparison algorithms and analyze the experimental results.

5.1 The convergence analysis

As a description of algorithm design, the K-NN method is chosen to map continuous action space to a discrete set [41]. This section focuses on the convergence analysis of DRL-E2D and DQN. For DRL-E2D, we consider two cases: $k = 1$ and $k = 1\%$, where $k = 1$ means we only extract one action from the proto-action, and $k = 1\%$ indicates we extract 1% of actions from the proto-action.

We first discuss the convergence of DRL-E2D and DQN on the different number of eNBs n. Figure 4 shows the learning curves representing the total rewards, where we keep the other parameters fixed, i.e., $\lambda = 10$, $W = 2.5\,\text{GHz s}$, and $D = 10\,\text{MB}$. We can see from Fig. 4a, b that the total reward increases steadily from 0 to 50 episodes. With the larger action space, the convergence of DRL-E2D is more slowly, requiring 150 episodes to reach the maximum reward, as shown in Fig. 4c. Moreover, the DRL-E2D with $k = 1\%$ always performs better than the case of $k = 1$. This is because the larger k makes value-based policies use action features to reason about unseen better actions. However, we find that when $n = 1$, the action space is less than 100, and the number of $k = 1\%$ is less than 1. So, we only give the result of $k = 1$ in Fig. 4a. The DQN method always learns slower than DRL-E2D. For example, the ultimate rewards of DQN under the case of $n = 1$, $n = 3$, and $n = 5$ are about 2500, 6000, and 7000, respectively. This is because DQN only works well in the small action space problem and cannot be applied in the problem with large action space. The curve of $n = 1$ increases faster than that of $n = 3$ and $n = 5$, as the action space of $n = 1$ is much smaller. Hence, Fig. 4 indicates that DQN cannot converge to the stable state, and the larger n, the worse of the DQN convergence with the same number of episodes.

Figure 5 plots the convergence of the DRL-E2D and DQN algorithm on different task arrival rate λ, where we fix $n = 3$, $W = 2.5\,\text{GHz s}$, and $D = 10\,\text{MB}$. It can be seen that with the fixed number of eNBs n, $k = 1\%$ and $k = 1$ converge to their maximum rewards almost at 100 episodes. We also find that with the increase of λ, the ultimate reward goes up. For example, for $k = 1\%$, the rewards are about 4500, 10,000, and 10,500 when λ = 5, 12, and 20, respectively. The rationale is that with more tasks arrived at MD, executing more tasks will incur higher utility. However, for $k = 1\%$, the reward of $\lambda = 12$ is slightly larger than that of $\lambda = 20$. This is because the system resource almost becomes saturated when λ is further increased ($\lambda > 10$), making more tasks drop and, therefore, lowering the total reward. We can also observe that the $k = 1\%$ always outperforms the $k = 1$, demonstrating again that K-NN is a good method for extracting actions from proto-action. So, we set $k = 1\%$ for the following simulations. Similar to the varying of n, DQN always learns much slower than DRL-E2D in all cases of λ. For example, the rewards of DQN when $\lambda = 5$, $\lambda = 12$, and $\lambda = 20$ are about 3700, 5600, and 7200, respectively. This is because DQN cannot work well in the problem of large action space problem. With much smaller action space, the curve of $\lambda = 5$ increases much faster than that of $\lambda = 12$ and $\lambda = 20$. Hence, Fig. 5 shows DQN cannot converge to the stable state with 250 episodes.

5.2 The impact of a number of eNBs *n*

The simulation results on three performance metrics, such as reward, energy consumption, and the penalty of task dropping, are included. By varying the number of eNBs n from 1 to 5, are given in Fig. 6, where we keep the other parameters fixed, i.e., $\lambda = 10$, $W = 2.5\,\text{GHz s}$, and $D = 10\,\text{MB}$.

As shown in Fig. 6a, Local is independent of n, as it executes all the tasks on MD. However, the rewards of DRL-E2D, Remote, LB, and DQN increase when n goes up. This is because these algorithms can benefit from offloading tasks to eNBs. Moreover, with more eNBs, MD obtain higher rewards, resulting from finishing more tasks and consuming less energy. Figure 6b shows the energy consumption of all the algorithms.

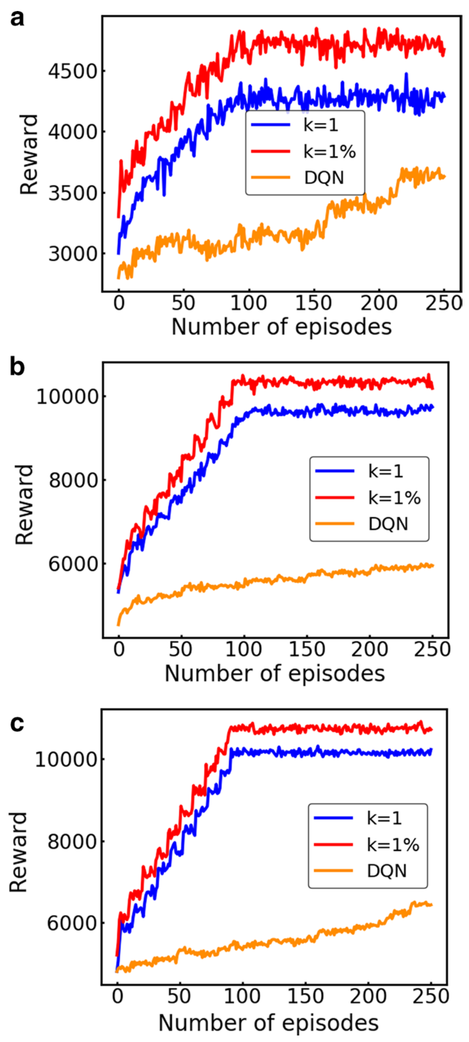

Fig. 5 The convergence of DRL-E2D on different λ. **a** $\lambda = 5$. **b** $\lambda = 12$. **c** $\lambda = 20$

The energy consumption of Local is independent of n. The energy consumption of LB, Remote, and DQN decreases when n increases. This is because the more eNBs, the fewer tasks should be processed by the MD or each eNB. However, the energy consumption of DRL-E2D is almost stable by varying n. The rationale is that to maximize the reward, MD tends to drop tasks instead of executing them. The penalty of all the algorithms (except Local) decreases with the increase of n, which is shown in Fig. 6c. With the fixed task arrival rate λ, fewer tasks will be dropped by offloading them to eNBs.

5.3 The impact of task arrival rate λ

This section inspects the performance impact of task arrival rate λ. We conduct the experiments with λ varying from 2 to 20, where we keep the other parameters fixed,

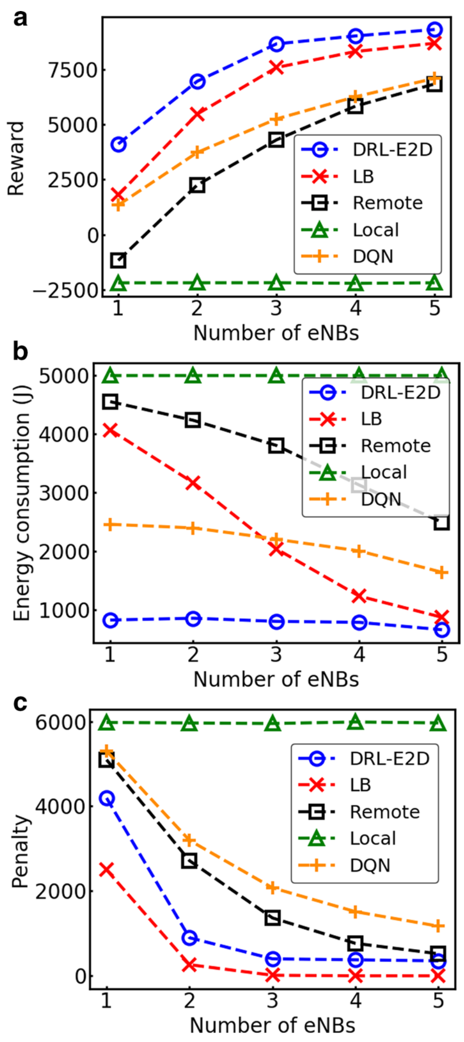

Fig. 6 The performance impact of *n*. **a** Reward. **b** Energy consumption. **c** Penalty

i.e., $n = 3$, $W = 2.5\,\text{GHz s}$, and $D = 10\,\text{MB}$. The related results are given in Fig. 7. Note that the larger λ, the more tasks will have arrived at MD.

From Fig. 7a, we find that DRL-E2D has the highest reward. We can also see that the reward of DRL-E2D, LB, Remote, and DQN goes up with λ. This is because the more arrived tasks, the more tasks can be processed, incurring higher utility. However, failing to offload tasks, the reward of Local decreases when λ increases. This is reflected by the fact that Local prefers to drop tasks rather than execute them, which can reduce energy consumption and bring high reward. For energy consumption in Fig. 7b, all the algorithms go up with an increase of λ. This is because of the larger λ, the more tasks will be processed, leading to higher energy consumption. Particularly, we find that the energy consumption of Local first increases and then keeps stable because it selects to drop tasks instead of process tasks when $\lambda \geq 6$. Figure 7c shows that the penalty of all the algorithms increases with λ. This indicates that the arrived tasks under the existing parameter setting cannot be finished completely when λ is

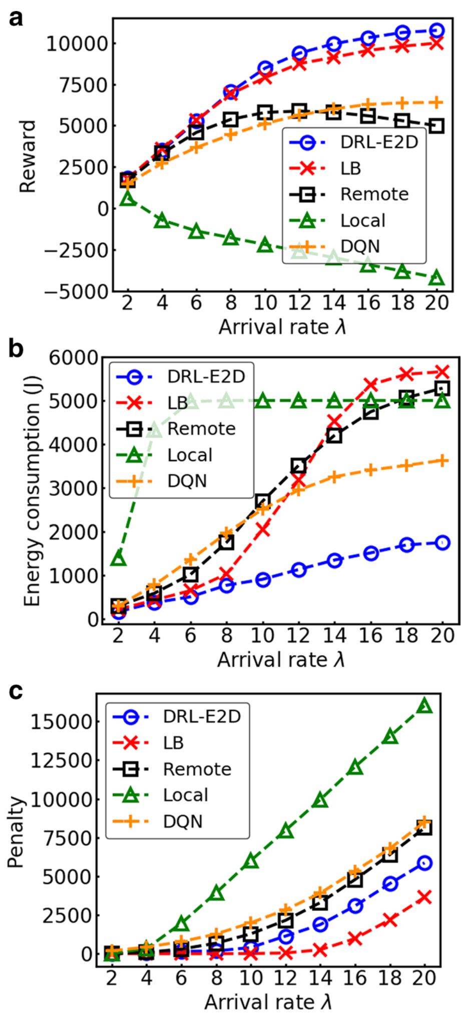

Fig. 7 The performance impact of λ. **a** Reward. **b** Energy consumption. **c** Penalty

too large due to insufficient computing resources. Notably, the increased speed of Local is obviously faster than others, suggesting that it drops more tasks. Although the penalty of DRL-E2D performs worse than that of LB, it still has a higher reward because of the less energy consumption.

5.4 The impact of workload W

We examine the performance impact of Workload W. The results with W varying from 0.5 to 3 GHz s are shown in Fig. 8, where we keep the other parameters fixed, for example, $n = 3$, $\lambda = 10$, and $D = 10$ MB. Note that the larger W means we need more computing resources to finish a task.

As observed from Fig. 8a, with the increase of W, the reward of all the algorithms goes down. This is due to the fact that with the fixed task arrival rate λ, the larger W requires more computing resources, which leads to higher energy consumption, less finished tasks and lower reward. Moreover, we find that the reward of Local decreases faster than the other algorithms because of dropping more tasks. DRL-E2D always performs better

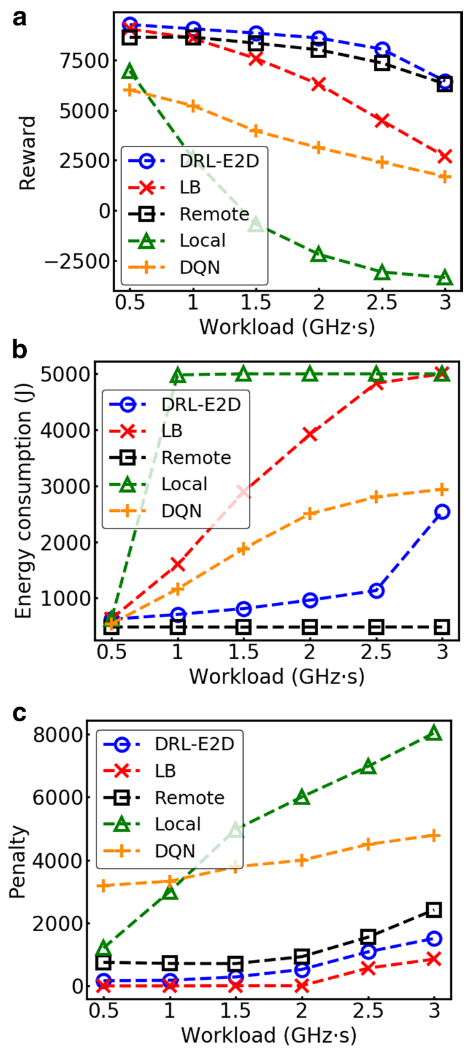

Fig. 8 The performance impact of W. **a** Reward. **b** Energy consumption. **c** Penalty

than others, indicating that it can achieve better performance and adapt the W varying. We also find that the rewards of DQN are always less than that of DRL-E2D because DQN cannot converge to an optimal value. Figure 8b gives the results of the energy consumption of all the algorithms. When W varies from 1 to 3, the curve of the energy consumption of the Local algorithm is flat, as it decides to drop tasks. This is because of only considering the data transmission energy, the energy consumption of Remote has the lowest energy consumption, and it is independent of W. The energy consumption of LB, DRL-E2D, and DQN increases with W, as the larger W needs more computing resources and time, thus incurring higher energy consumption. Figure 8c shows the results of the penalty. We can see that with the increase of W, the penalty of all the algorithms goes up. In particular, the penalty of Local increases rapidly with W, as it drops more tasks than others.

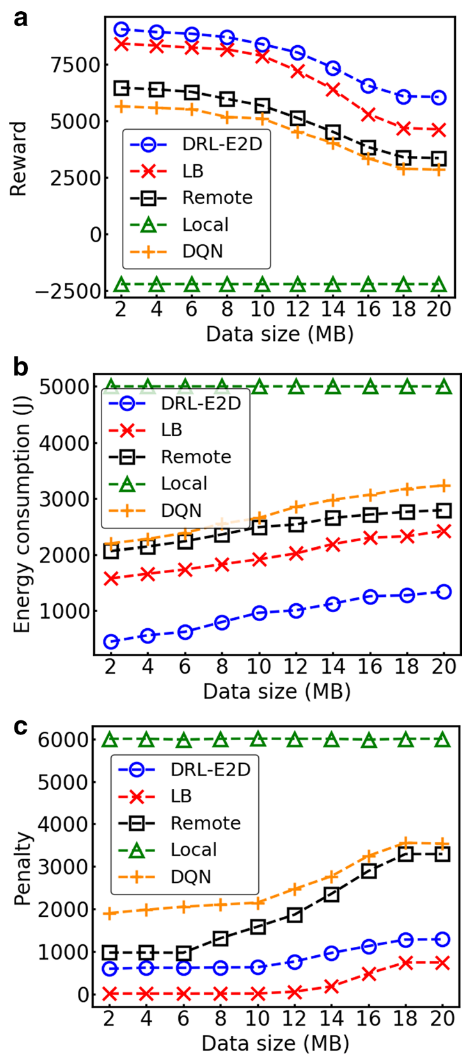

Fig. 9 The performance impact of *D*. **a** Reward. **b** Energy consumption. **c** Penalty

5.5 The impact of data size *D*

To reveal the performance impact of data size D, we vary D from 2 to 20 MB with an increment of 2 MB. The related simulation results are given in Fig. 9, where we keep the other parameters fixed, for example, $\lambda = 10$, $n = 3$, and $W = 2.5$ GHz s. Note that the larger D means needing more time to transmission data under the fixed transmission rate.

We can observe from Fig. 9a that the reward of all the algorithms except Local decreases when D goes up. This is because DRL-E2D, LB, Remote, and DQN apply the task offloading strategy. Thus, MD will cost much more energy consumption to offload tasks to eNBs in the case of larger D. Nevertheless, the Local does not adopt the task offloading, the reward of which is independent of D. The same reason can be explained for the energy consumption and penalty of Local in Fig. 9b, c. Figure 9b shows that the energy consumption of DRL-E2D, LB, Remote, and DQN slightly increases with D, since the task offloading incurs more transmission energy with the larger D. Due to the same reason, we also find that the penalty of DRL-E2D, LB, Remote, and DQN increase with D. Especially, the penalty of Remote increases more rapidly than other algorithms. This is because all the tasks should be transmitted to eNBs according to the Remote algorithm, resulting in much more energy consumption of data transmission.

6 Conclusions and future work

This paper proposes a DRL-E2D algorithm for task offloading in a multi-eNB MEC environment. Its goal is to maximize the reward (the weighted sum of utility, energy consumption, and penalty of tasks dropping) under the task's deadline constraint. The architecture of DRL-E2D is based on the actor-critic framework, and the action representation with the K-NN approach is developed to handle the large action space problem. The extensive simulation results have shown that: (1) the K-NN, as an approximate approach of finding the nearest neighbor optimal actions, is indeed able to generalize over the set of actions; (2) compared to several widely used baseline algorithms, DRL-E2D always have better behaviors and significantly outperforms the LB, Remote, Local, and DQN algorithms; (3) DRL-E2D is robust to system parameters, e.g., the number of eNBs, task arrival rate, task workload, and the data size.

In the future work, we consider two aspects in task offloading: (1) security and privacy problems, which are very critical for mobile users and applications, as the data is the important intellectual property; and (2) fault-tolerant scheduling problem, which is also common in the MEC environment, as both the MEC server and MD will experience the failure of task execution due to hardware and software faults.

Abbreviations

MD: Mobile device; QoE: Quality of experience; MEC: Mobile edge computing; eNB: ENodeB; DRL: Deep reinforcement learning; K-NN: K-nearest neighbor; MCC: Mobile cloud computing; DQN: Deep Q-network; DDPG: Deep deterministic policy gradient; AI: Artificial intelligence; MDP: Markov decision process.

Authors' contributions

ZL, VC, JG, LP, HH, and BH conceived and designed the study. ZL, VC, and LP performed the simulations. ZL wrote the paper. All authors reviewed and edited the manuscript. All authors read and approved the final manuscript.

Funding

This work was supported by the National Natural Science Foundation of China (Nos. 61802095, 61572162), the Zhejiang Provincial Key Science and Technology Project Foundation (No. 2018C01012), the Zhejiang Provincial National Science Foundation of China (No. LQ19F020011), the Open Foundation of State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) (No. SKLNST-2019-2-15), and the VC Research (No. VCR 0000057).

Availability of data and materials

The details of the experimental parameters are given in Sect. 4.3.

Declarations

Competing interests

The authors declare that they have no competing interests.

Author details

[1] School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China. [2] School of Computing and Digital Technologies, Teesside University, Middlesbrough, UK. [3] State Key Laboratory for Novel Software Technology, Software Institute, Nanjing University, Nanjing, China.

Received: 27 August 2020 Accepted: 9 March 2021
Published online: 17 March 2021

References

1. F. Jameel, Z. Hamid, F. Jabeen, S. Zeadally, M.A. Javed, A survey of device-to-device communications: research issues and challenges. IEEE Commun. Surv. Tutor. **20**(3), 2133–2168 (2018)
2. A.R. Khan, M. Othman, S.A. Madani, S.U. Khan, A survey of mobile cloud computing application models. IEEE Commun. Surv. Tutor. **16**(1), 393–413 (2014)
3. P. Mach, Z. Becvar, Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun. Surv. Tutor. **19**(3), 1628–1656 (2017)
4. X. Xu, X. Zhang, H. Gao, Y. Xue, L. Qi, W. Dou, BeCome: blockchain-enabled computation offloading for IoT in mobile edge computing. IEEE Trans. Ind. Inform. **16**(6), 4187–4195 (2020)
5. K. Peng, M. Zhu, Y. Zhang, L. Liu, J. Zhang, V.C.M. Leung, L. Zheng, An energy- and cost-aware computation offloading method for workflow applications in mobile edge computing. EURASIP J. Wirel. Commun. Netw. **19**, 207 (2019)
6. Cisco Systems, Cisco visual networking index: global mobile data traffic forecast update. Technique Report (2019)
7. 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)
8. T.Q. Dinh, 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)
9. X. Huang, K. Xu, C. Lai, Q. Chen, J. Zhang, Energy-efficient offloading decision-making for mobile edge computing in vehicular networks. EURASIP J. Wirel. Commun. Netw. **2020**(1), 35 (2020)
10. T. Li, C. M. S. Magurawalage, K. Wang, K. Xu, K. Yang, H. Wang, in *37th IEEE International Conference on Distributed Computing Systems (ICDCS)*. On efficient offloading control in cloud radio access network with mobile edge computing (IEEE, 2017), pp. 2258–2263
11. X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans. Netw. **24**(5), 2795–2808 (2016)
12. V. Mnih et al., Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
13. D. Silver et al., Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
14. Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, D. Yang, in *IEEE Conference on Computer Communications (INFOCOM)*. Experience-driven networking: A deep reinforcement learning based approach (IEEE, 2018), pp. 1871–1879
15. J. Li, Q. Liu, P. Wu, F. Shu, S. Jin, in *IEEE/CIC International Conference on Communications in China (ICCC)*. Task offloading for UAV-based mobile edge computing via deep reinforcement learning, (IEEE, 2018), pp. 798–802
16. D. V. Le, C. K. Tham, in *IEEE Conference on Computer Communications Workshops (INFOCOM Workshops)*. A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds (IEEE, 2018), pp. 760–765
17. J. Li, H. Gao, T. Lv, Y. Lu, in *IEEE Wireless Communications and Networking Conference (WCNC)*. Deep reinforcement learning based computation offloading and resource allocation for MEC (IEEE, 2018), pp. 1–6
18. M.G.R. Alam, M.M. Hassan, M.Z. Uddin, A. Almogren, G. Fortino, Autonomic computation offloading in mobile edge for IoT applications. Future Gener. Comput. Syst. **90**, 149–157 (2019)
19. H. Lu, C. Gu, F. Luo, W. Ding, X. Liu, Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. Future Gener. Comput. Syst. **102**, 847–861 (2020)
20. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, in *International Conference on Learning Representations (ICLR)*. Continuous control with deep reinforcement learning (2016)
21. R. Beraldi, A. Mtibaa, H. M. Alnuweiri, in *International Conference on Fog and Mobile Edge Computing (FMEC)*. Cooperative load balancing scheme for edge computing resources (2017), pp. 94–100
22. C. Wang, F. R. Yu, Q. Chen, L. Tang, in *IEEE International Conference on Communications (ICC)*. Joint computation and radio resource management for cellular networks with mobile edge computing (IEEE, 2017), pp. 1–6
23. Y. Miao, G. Wu, M. Li, A. Ghoneim, M. Al-Rakhami, M.S. Hossain, Intelligent task prediction and computation offloading based on mobile-edge cloud computing. Future Gener. Comput. Syst. **102**, 925–931 (2020)
24. J. Yan, S. Bi, Y.J. Zhang, M. Tao, Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency. IEEE Trans. Wirel. Commun. **19**(1), 235–250 (2020)
25. R. Wang, Y. Cao, A. Noor, T.A. Alamoudi, R. Nour, Agent-enabled task offloading in UAV-aided mobile edge computing. Comput. Commun. **149**, 324–331 (2020)
26. C. F. Liu, M. Bennis, H. V. Poor, in *Global Communications Conference Workshops (GLOBECOM Workshops)*. Latency and reliability-aware task offloading and resource allocation for mobile edge computing (2017), pp. 1–7
27. C.F. Liu, M. Bennis, M. Debbah, H.V. Poor, Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing. IEEE Trans. Commun. **67**(6), 4132–4150 (2019)
28. X. Lyu, W. Ni, H. Tian, R.P. Liu, X. Wang, G.B. Giannakis, A. Paulraj, Optimal schedule of mobile edge computing for internet of things using partial information. IEEE J. Sel. Areas Commun. **35**(11), 2606–2615 (2017)
29. Y. Sun, S. Zhou, J. Xu, EMM: energy-aware mobility management for mobile edge computing in ultra dense networks. IEEE J. Sel. Areas Commun. **35**(11), 2637–2646 (2017)
30. C. Li, J. Tang, Y. Luo, Dynamic multi-user computation offloading for wireless powered mobile edge computing. J. Netw. Comput. Appl. **131**, 1–15 (2019)
31. L. Huang, S. Bi, Y.J. Zhang, Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Trans. Mob. Comput. **19**, 2581–2593 (2019)
32. L. Lei, H. Xu, X. Xiong, K. Zheng, W. Xiang, X. Wang, Multi-user resource control with deep reinforcement learning in IoT edge computing. arXiv:1906.07860 (2019)
33. B. Huang, Y. Li, Z. Li, L. Pan, S. Wang, Y. Xu, H. Hu, Security and cost-aware computation offloading via deep reinforcement learning in mobile edge computing. Wirel. Commun. Mob. Comput. **2019**, 3816237:1–3816237:20 (2019)
34. G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, B. Coppin. Deep reinforcement learning in large discrete action spaces. arXiv:1512.07679 (2015)
35. 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)

36. Y. Yao, L. Huang, A.B. Sharma, L. Golubchik, M.J. Neely, Power cost reduction in distributed data centers: a two-time-scale approach for delay tolerant workloads. IEEE Trans. Parallel Distrib. Syst. **25**(1), 200–211 (2014)
37. R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction* (MIT press, Cambridge, 2018).
38. N.C. Luong, D.T. Hoang, S. Gong, D. Niyato, P. Wang, Y.C. Liang, D.I. Kim, Applications of deep reinforcement learning in communications and networking: a survey. IEEE Commun. Surv. Tutor. **21**(4), 3133–3174 (2019)
39. W. Jiang, G. Feng, S. Qin, T.S. Peter Yum, G. Cao, Multi-agent reinforcement learning for efficient content caching in mobile d2d networks. IEEE Trans. Wirel. Commun. **18**(3), 1610–1622 (2019)
40. Y. Mao, J. Zhang, S.H. Song, K.B. Letaief, Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems. IEEE Trans. Wirel. Commun. **16**(9), 5994–6009 (2017)
41. M. Muja, D.G. Lowe, Scalable nearest neighbor algorithms for high dimensional data. IEEE Trans. Pattern Anal. Mach. Intell. **36**(11), 2227–2240 (2014)

Publisher's Note