Chen and Wang *EURASIP Journal on Wireless Communications and Networking* (2020) 2020:188
https://doi.org/10.1186/s13638-020-01801-6

EURASIP Journal on Wireless Communications and Networking

RESEARCH **Open Access**

Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach

Zhao Chen[1*] and Xiaodong Wang[2]

*Correspondence:
zhao_chen@tsinghua.edu.cn
[1]Beijing National Research Center for Information Science and Technology, Tsinghua University, 100084 Beijing, China
Full list of author information is available at the end of the article

Abstract
Mobile edge computing (MEC) emerges recently as a promising solution to relieve resource-limited mobile devices from computation-intensive tasks, which enables devices to offload workloads to nearby MEC servers and improve the quality of computation experience. In this paper, an MEC enabled multi-user multi-input multi-output (MIMO) system with stochastic wireless channels and task arrivals is considered. In order to minimize long-term average computation cost in terms of power consumption and buffering delay at each user, a deep reinforcement learning (DRL)-based dynamic computation offloading strategy is investigated to build a scalable system with limited feedback. Specifically, a continuous action space-based DRL approach named deep deterministic policy gradient (DDPG) is adopted to learn decentralized computation offloading policies at all users respectively, where local execution and task offloading powers will be adaptively allocated according to each user's local observation. Numerical results demonstrate that the proposed DDPG-based strategy can help each user learn an efficient dynamic offloading policy and also verify the superiority of its continuous power allocation capability to policies learned by conventional discrete action space-based reinforcement learning approaches like deep Q-network (DQN) as well as some other greedy strategies with reduced computation cost. Besides, power-delay tradeoff for computation offloading is also analyzed for both the DDPG-based and DQN-based strategies.

Keywords: Mobile edge computing, Deep reinforcement learning, Computation offloading, Multi-user MIMO

1 Introduction

As the popularity of smart mobile devices in the coming 5G era, mobile applications, especially for computation-intensive tasks such as online 3D gaming, face recognition, and location-based augmented or virtual reality (AR/VR), have been greatly affected by the limited on-device computation capability [1]. Meanwhile, for the large number of low-power and resource-constrained wireless terminals serving in the emerging Internet of Things (IoT) [2] and Intelligent Transport Systems (ITS) [3], a huge amount of sensory data also needs to be pre-processed and analyzed. As a result, to meet the quality of experience (QoE) of these mobile applications, the technology of mobile edge computing (MEC) [4] has been proposed as a promising solution to bridge the gap between the limited resources on mobile devices and the ever-increasing demand of computation requested by mobile applications.

Unlike the remote public clouds in conventional cloud computing systems such as Amazon Web Services and Microsoft Azure, MEC enhances radio access networks (RANs), which is in close proximity to mobile users, with computing capability [5]. Thus, mobile devices can offload computation workloads to the MEC server associated with a base station (BS), and mobile applications can be improved with considerably reduced latency and power consumption. Nevertheless, computation offloading highly depends on the efficiency of wireless data transmission, which requires MEC enabled systems to manage radio resources along with computation resources and complete computation tasks efficiently.

To achieve higher energy efficiency or better computation experience, computation offloading strategies for MEC have been widely investigated recently. For short-term optimization over quasi-static channels, some algorithms have been studied in [6–12]. In [6], optimal offloading selection and radio resource allocation for mobile tasks was studied to minimize the overall execution time. Moreover, by using dynamic voltage and frequency (DVFS) techniques, CPU-cycle frequency was flexibly controlled in [7] to reduce the system cost which is defined as weighted sum of energy consumption and execution time. Besides, energy-latency tradeoff has been discussed in [8] with jointly optimized communication and computation resource allocation under the limited energy and sensitive latency. Also, performance of MEC have been further improved with adopting some other emerging technologies such as wireless power transfer [9] and non-orthogonal multiple access (NOMA) [10–12]. Particularly, physical layer security is studied in NOMA-based MEC networks in [11], where security of computation offloading is improved in terms of secrecy outage probability and user connectivity is also enhanced.

To cope with stochastic task arrivals and time-varying wireless channels, strategies for dynamic joint control of radio and computation resources in MEC enabled systems become even challenging [13–19]. In [13], dynamic policies for offloading decision, clock speed and network interface control were considered to minimize energy consumption with given delay constraints. Joint optimization of multiple-input multiple-output (MIMO) beamforming and computational resource allocation for a multi-cell MEC system is designed in [14]. Additionally, a green MEC enabled system with energy harvesting devices is studied in [15], where the delay cost addressing both the execution delay and task failure is minimized. For multi-user scenarios, power-delay tradeoff [16], network utility maximization balancing throughput and fairness with reduced feedback [17], and stochastic admission control and scheduling for multi-user multi-task computation offloading [18] were discussed, respectively. Markov decision process (MDP) can be also applied to the analysis and design of dynamic control of computation offloading [19]. Furthermore, [20] demonstrated how dynamic computation offloading policies can be learned by reinforcement learning (RL)-based algorithm with no prior knowledge of the system.

Conventional RL algorithms cannot scale well as the number of agents increases, since the explosion of state space makes traditional tabular methods infeasible [21].

Nevertheless, by exploiting deep neural networks (DNNs) for function approximation, deep reinforcement learning (DRL) has been demonstrated to be able to efficiently approximate Q-values of RL [22] and more scalable. There have been some attempts to adopt DRL in the design of online resource allocation and scheduling for computation offloading in MEC [23–27]. Specifically, in [23], system sum cost of a multi-user network is minimized in terms of execution delay and energy consumption by computational resource allocation. Similarly, the authors in [24] considered an online offloading algorithm to maximize the weighted sum computation rate in a wireless powered system. In [25], a DRL-based computation offloading strategy of an IoT device is learned to choose a MEC server to offload and determine the offloading rate. Besides, double deep Q-network (DQN)-based strategic computation offloading algorithm was proposed in [26], where an mobile device learned the optimal task offloading and energy allocation to maximize the long-term utility based on the task queue state, the energy queue state as well as the channel qualities. What is more, a DQN-based vehicle-assisted offloading scheme is studied in [27] to maximize the long-term utility of the vehicle edge computing network by considering the delay of the computation task. Overall, existing works on DRL-based dynamic computation offloading only consider centralized algorithms for either single user cases or multi-user scenarios. It requires BS to collect global information of the environment from all users and makes decision for them, which leads to high system overhead and considerable latency to collect such informations at BS and then distribute the decisions to each user. Thus, decentralized DRL-based dynamic task offloading algorithms are desired for a multi-user system, which still remains unknown.

In this paper, we consider a system consisting of one BS attached with an MEC server and multiple mobile users, where tasks arrive stochastically and channel condition is time-varying at each user. Without any prior knowledge of the system, i.e., the number of users, statistical modeling of task arrivals and wireless channels, each mobile user can learn a dynamic computation offloading policy independently based on its local observations of the system. Moreover, different from conventional discrete action space-based DRL policies, we adopt a continuous action space-based algorithm named deep deterministic policy gradient (DDPG) to derive better power control of local execution and task offloading. Specifically, major contributions of this paper can be summarized as follows:

- By considering a MIMO enabled multi-user MEC system, a long-term average computation cost minimization problem is formulated under stochastic task arrivals and wireless channels, which aims to optimize local execution and computation offloading powers for each user.
- A decentralized dynamic computation offloading framework based on the DDPG algorithm has been built, which enables each user to independently learn efficient offloading policies from only local observations for dynamic power allocation in a continuous domain.
- Performance of the decentralized policies learned by DDPG and the power-delay trade-off is illustrated by numerical simulations, which demonstrates the DDPG-based continuous power control outperforms the DQN-based discrete control and some other greedy strategies.

The rest of this paper is organized as follows. In Section 2, system model and problem formulation of dynamic computation offloading is presented. In Section 3, design of the decentralized DDPG-based dynamic computation offloading framework is proposed. Numerical results are illustrated in Section 4. Finally, Section 5 concludes this paper.

2 System model and problem formulation

As shown in Fig. 1, a multi-user MIMO system is considered, which consists of an N-antenna BS, an MEC server and a set of single-antenna mobile users $\mathcal{M} = \{1, 2, \ldots, M\}$. Given limited computational resources on the mobile device, each user $m \in \mathcal{M}$ has computation-intensive tasks to be completed. To improve user computation experience, an MEC server is deployed in proximity to BS to enable users to offload part of computation needs to the MEC server via wireless links [28]. In the proposed system, a discrete-time model is adopted, where the operating period is slotted with equal length τ_0 and indexed by $\mathcal{T} = \{0, 1, \ldots\}$. For each slot $t \in \mathcal{T}$, each user's channel condition and task arrival varies, which requires the ratio of local execution and computation offloading to be elaborately selected to balance the average energy consumption and task processing delay. Moreover, as the number of mobile users increases, decentralized task scheduling is more favorable, which reduces system overhead between the users and the MEC server and thus improves scalability of the proposed system. In the following parts, the modeling of networking and computing of the system will be introduced in detail. The summary of notations are presented in Table 1.

2.1 Network model

For each slot t, if the channel vector of each mobile user $m \in \mathcal{M}$ is represented by $\boldsymbol{h}_m(t) \in \mathbb{C}^{N\times 1}$, the received signal of BS from all users in the uplink can be written as

$$\boldsymbol{y}(t) = \sum_{m=1}^{M} \boldsymbol{h}_m(t)\sqrt{p_{o,m}(t)}s_m(t) + \boldsymbol{n}(t), \tag{1}$$

where $p_{o,m}(t) \in [0, P_{o,m}]$ is the transmission power of user m to offload task data bits with $P_{o,m}$ being the maximum value, $s_m(t)$ is the complex data symbol with unit variance, and $\boldsymbol{n}(t) \sim \mathcal{CN}(\mathbf{0}, \sigma_R^2 \boldsymbol{I}_N)$ is a vector of additive white Gaussian noise (AWGN) with variance σ_R^2. Note that $\boldsymbol{I}_N$ denotes an $N \times N$ identity matrix. In order to characterize the temporal correlation between time slots for each mobile user m, the following Gaussian-Markov block fading autoregressive model [29] is adopted:

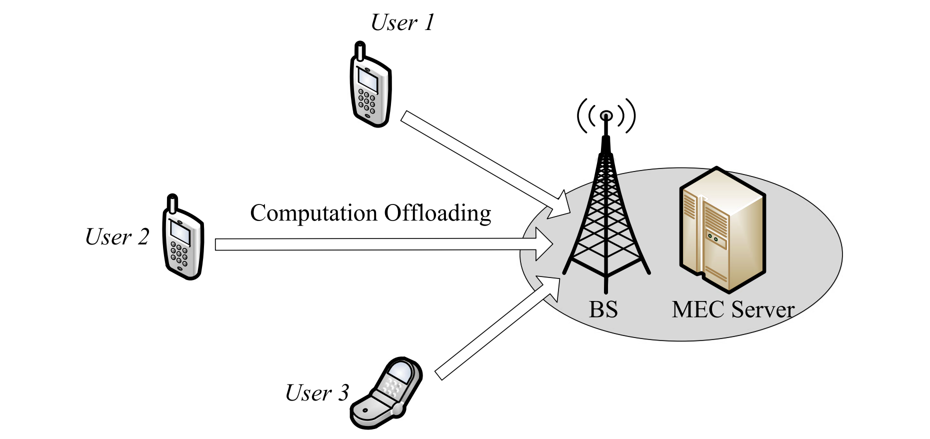

Fig. 1 Computation offloading in an MEC enabled multi-user MIMO system

Table 1 Summary of notations

Notation	Description
$\mathcal{M}$	The set of mobile users
$\mathcal{T}$	Index set of time slots
$\boldsymbol{h}_m(t)$	Channel vector between user *m* and BS at slot *t*
$\boldsymbol{y}(t)$	Received signal of BS at slot *t*
ρ_m	Normalized temporal channel correlation coefficient of user *m*
$\boldsymbol{H}(t)$	Channel matrix from all the users to BS at slot *t*
$\boldsymbol{g}_m(t)$	User *m*'s ZF detection vector used by BS at slot *t*
$\gamma_m(t)$	User *m*'s receiving SINR at BS of slot *t*
$B_m(t)$	Queue length of user *m*'s task buffer at slot *t*
$a_m(t)$	Amount of task arrivals of user *m* in bit at slot *t*
$f_m(t)$	CPU frequency scheduled for local execution of user *m* at slot *t*
L_m (F_m)	CPU cycles required per one task bit (allowable CPU-cycle frequency) at user *m*
$p_{o,m}(t)$	Transmission power of user *m* for computation offloading at slot *t*
$d_{o,m}(t)$	Data transmitted by user *m* for computation offloading at slot *t*
$p_{l,m}(t)$	Power consumption of user *m* for local execution at slot *t*
$d_{l,m}(t)$	Data processed by user *m* via local execution at slot *t*
$P_{o,m}$ ($P_{l,m}$)	Maximum transmission power (local execution power) of user *m*
τ_0	Length of one time slot

$$\boldsymbol{h}_m(t) = \rho_m \boldsymbol{h}_m(t-1) + \sqrt{1-\rho_m^2}\boldsymbol{e}(t), \tag{2}$$

where ρ_m is the normalized channel correlation coefficient between slots t and $t-1$, and the error vector $\boldsymbol{e}(t)$ is complex Gaussian and uncorrelated with $\boldsymbol{h}_m(t)$. Note that $\rho_m = J_0(2\pi f_{d,m}\tau_0)$ according to Jake's fading spectrum, where $f_{d,m}$ is the Doppler frequency of user m, t_0 is the slot length, and $J_0(\cdot)$ is the Bessel function of the first kind [30].

Denoting $\boldsymbol{H}(t) = [\boldsymbol{h}_1(t), \ldots, \boldsymbol{h}_M(t)]$ as the $N \times M$ channel matrix between BS and all the users, the linear zero-forcing (ZF) detector at BS[1] can be written by its pseudo inverse $\boldsymbol{H}^\dagger(t) = \left(\boldsymbol{H}^H(t)\boldsymbol{H}(t)\right)^{-1}\boldsymbol{H}^H(t)$. The mth row of $\boldsymbol{H}^\dagger(t)$, which we referred to as $\boldsymbol{g}_m^H(t)$, is used by BS to detect signal of user m. Thus, the detected signal can be written by

$$\boldsymbol{g}_m^H(t)\boldsymbol{y}(t) = \sqrt{p_{o,m}(t)}s_m(t) + \boldsymbol{g}_m^H(t)\boldsymbol{n}(t), \tag{3}$$

since we have $\boldsymbol{g}_i^H(t)\boldsymbol{h}_j(t) = \delta_{ij}$ for ZF detection by definition [31]. Here, $\delta_{ij} = 1$ when $i = j$ and 0 otherwise. Then, the corresponding signal-to-interference-plus-noise (SINR) at BS can be derived by

$$\gamma_m(t) = \frac{p_{o,m}(t)}{\sigma_R^2 \|\boldsymbol{g}_m(t)\|^2}, \tag{4}$$

$$= \frac{p_{o,m}(t)}{\sigma_R^2 \left[\left(\boldsymbol{H}^H(t)\boldsymbol{H}(t)\right)^{-1}\right]_{mm}}, \tag{5}$$

where $[\boldsymbol{A}]_{mn}$ is the (m, n)th element of matrix $\boldsymbol{A}$. From (4), it can be verified that each user's SINR becomes worse as the number of users M increases, which generally makes each user to spend more power on task offloading. In the sequel, we will show how the user learns to adapt to the environment from the SINR feedbacks.

[1] Here, the number of antennas at BS is larger than the number of users, i.e., $N > M$. ZF is adopted for multiuser detection for its low complexity and efficiency, especially for with large antenna arrays [31]. It is demonstrated that as N increases, the SINR performance of ZF detection approaches that of the MMSE detection.

2.2 Computation model

Without loss of generality, we use $a_m(t)$ to quantify the amount of task arrivals of user m in bit during slot t, which is assumed to be independent and identically distributed (i.i.d) over all slots. Upon arrival, task bits will be firstly stored in the user's queuing buffer and then processed in upcoming slots starting from $t+1$. Besides, we assume computation applications are fine-grained [13]. Thus, during slot t, part of buffered task bits denoted by $d_{l,m}(t)$ will be processed locally on the mobile device and another part of buffered task bits denoted by $d_{o,m}(t)$ will be offloaded to and executed by the MEC server. Note that the amount of all executed task bits, i.e., $d_{l,m}(t)+d_{o,m}(t)$, should not exceed the queue length of task buffer. If $B_m(t)$ stands for the queue length of user m's task buffer at the beginning of slot t, it will evolve as follows:

$$B_m(t+1) = \left[B_m(t) - \left(d_{l,m}(t) + d_{o,m}(t)\right)\right]^+ + a_m(t), \forall t \in \mathcal{T}, \tag{6}$$

where $B_m(0) = 0$ and $[x]^+ = \max(x, 0)$.

2.2.1 Local execution

For user m, if $p_{l,m}(t) \in [0, P_{l,m}]$ is the allocated local execution power at slot t, the local processed bits can be calculated by

$$d_{l,m}(t) = \tau_0 f_m(t) L_m^{-1}, \tag{7}$$

where $P_{l,m}$ is the maximum local execution power and $f_m(t) = \sqrt[3]{p_{l,m}(t)/\kappa}$ is the CPU frequency scheduled by using DVFS techniques [32] to adjust chip voltage. Note that κ is the effective switched capacitance depending on the chip architecture, and thus $f_m(t) \in [0, F_m]$ with $F_m = \sqrt[3]{P_{l,m}(t)/\kappa}$ being the maximum allowable CPU-cycle frequency of user m's device. Besides, L_m denotes the number of CPU cycles required to process one task bit which can be estimated through off-line measurement [33].

2.2.2 Edge computing

MEC servers are usually equipped with sufficient computational resources such as high-frequency multi-core CPUs. Thus, it can be assumed that different applications can be handled in parallel with a negligible processing latency. Moreover, feedback delay is ignored for the small sized computation output by considering applications like video stream analysis and services for connected vehicles. In this way, all the task data bits offloaded to the MEC server via BS will be processed. Therefore, given the uplink transmission power $p_{o,m}(t)$, the amount of offloaded data bits of user m during slot t can be derived by

$$d_{o,m}(t) = \tau_0 W \log_2\left(1 + \gamma_m(t)\right), \tag{8}$$

where W is the system bandwidth and $\gamma_m(t)$ is the SNR obtained from (4).

2.2.3 Computation cost

To take both energy cost and latency into account, the overall computation cost for each user agent is quantified in term of a weighted sum of energy consumption and task buffering delay.[2] According to the Little's Theorem [34], average queue length of the task buffer

[2]In this paper, we focus on applications such as multimedia streaming or file backup, where the long-term average performance metrics are relevant to consider. For other applications with more strict delay requirement, the proposed computation cost model can be further extended to include more metrics.

is proportional to buffering delay. Thus, the long-term average computation cost for each user m is defined by

$$C_m = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} w_{m,1}\left(p_{l,m}(t) + p_{o,m}(t)\right) + w_{m,2}B_m(t), \tag{9}$$

where $w_{m,1}$ and $w_{m,2}$ are both nonnegative weighted factors. By setting different values for these weighted factors, there will be a tradeoff between energy consumption and buffering delay exists for dynamic computation offloading.

In contrast to make centralized decisions at BS that requires full observation of the environment, we take advantage of local observation to minimize the computation cost at each user m by decentralized decisions, i.e., dynamically allocating powers for local execution and computation offloading of each slot t as follows,

$$\min_{p_{l,m}(t), p_{o,m}(t)} C_m$$

$$s.t. \quad p_{l,m}(t) \in [0, P_{l,m}], \forall t \in \mathcal{T}; \tag{10}$$

$$p_{o,m}(t) \in [0, P_{o,m}], \forall t \in \mathcal{T}. \tag{11}$$

It is worth noting that, full observation requires global information such as uplink channel vectors and task buffer's queue lengths of all users, as well as the receiving SINRs of each user at BS. To collect such informations at BS and then distribute to each user will introduce high system overhead and considerable latency in practice, which even grows exponentially as the number of users increases. To this end, we assume that each user m makes decentralized decisions only depending on its local observation of the environment. That is, at the beginning of each slot t, user m will be only provided with the arriving bits $a_m(t)$ to update task buffer, be acknowledged with the previous receiving SINR at BS, i.e., $\gamma_m(t-1)$, and be able to estimate the current channel vector $\boldsymbol{h}_m(t)$ by using channel reciprocity. After that, the learned policy at each user can decide $p_{l,m}(t)$ and $p_{o,m}(t)$ independently from other user agents.

3 Decentralized dynamic computation offloading method

In this section, decentralized dynamic computation offloading is proposed to minimize long-term average computation cost of each user. Specifically, by leveraging the DDPG [35] algorithm, dynamic computation offloading policies can be independently learned at each user, where an continuous action, i.e., powers allocated for local execution and computation offloading in a continuous domain, will be selected for each slot based on local observation of the environment. Each user has no prior knowledge of the system environment, which means the number of users M, and statistics of task arrivals and wireless channels are all unknown to each user agent and thus the online learning process is totally model-free. In the sequel, we will firstly introduce some basics of DRL technology. Then, the DDPG-based framework for decentralized dynamic computation offloading will be built, where the state space, action space and reward function are defined. Finally, training and testing of decentralized policies in the framework is also presented.

3.1 Preliminaries on DRL

The standard setup for DRL follows from traditional RL, which consists of an agent, an environment E, a set of possible state $\mathcal{S}$, a set of available action $\mathcal{A}$, and a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$, where the agent continually learns and makes decisions from the interaction with the environment in discrete time steps. In each step t, the agent observes current state of the environment as $s_t \in \mathcal{S}$, and chooses and executes a action $a_t \in \mathcal{A}$ according to a policy π. The policy π is generally stochastic, which maps the current state to a probability distribution over the actions, i.e., $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. Then, the agent will receive a scalar reward $r_t = r(s_t, a_t) \in \mathcal{R} \subseteq \mathbb{R}$ and transition to the next state s_{t+1} according to the transition probability of the environment $p(s_{t+1}|s_t, a_t)$. Thus, it can be known that state transition of the environment E depends on the agent's action a_t executed in the current state s_t. In order to find the optimal policy, we define the return from a state as the sum of discounted reward in the future as $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$, where $T \to \infty$ represents the total number of time steps and $\gamma \in [0, 1]$ stands for the discounting factor. The goal of RL is to find the policy that maximizes the long-term expected discounted reward from the start distribution, i.e.,

$$J = \mathbb{E}_{s_i \sim E, a_i \sim \pi} [R_1] . \tag{12}$$

Under a policy π, the state-action function $Q^{\pi}(s_t, a_t) : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$, also known as the critic function that will be introduced later in this paper, is defined as the expected discounted return starting from state s_t with the selected action a_t,

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{i>t} \sim E, a_{i>t} \sim \pi} [R_t | s_t, a_t)] , \tag{13}$$

where expectation is over the transition probability $p(s_{t+1}|s_t, a_t)$ and the policy π.

RL algorithms attempt to learn the optimal policies from actual interactions with the environment and adapts its behavior upon experiencing the outcome of its actions. This is due to the fact that there may not be an explicit model of the environment E's dynamics. That is, the underlying transition probability $p(s_{t+1}|s_t, a_t)$ is unknown and even non-stationary. Fortunately, it has been proved [21] that under the policy π^* that maximizes the expected discounted rewards, the state-action function satisfies the following Bellman optimality equation,

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim E} \left[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \right], \tag{14}$$

from which the optimal policy π^* can be derived by choosing action in any state $s \in \mathcal{S}$ as follows[3],

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a). \tag{15}$$

In order to learn the value of state-action function from raw experience, the temporal-difference (TD) method can be leveraged to update the state-action function from an agent's experience tuple (s_t, a_t, r_t, s_{t+1}) at each time step t,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right], \tag{16}$$

where α is the learning rate and the value $\delta_t := r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$ is referred to as the TD error. Note that the well-known Q-learning algorithm

[3]Note that this gives an special case for deterministic policies, which can be readily extended to stochastic policies. Specifically, (14) still holds for stochastic policies. If there are ties for different actions that maximized the Q-value, each maximizing action can be given a portion of probability to be selected, while other actions is selected with zero probability.

[36] is based on (16) and thus the state-action function is also known as Q-value. Besides, it has been proved that Q-learning algorithm converges with probability one [21], which indicates that an estimated $Q^*(s,a)$ will be obtained eventually.

Thanks to the powerful function approximation properties of DNNs, DRL algorithms are able to learn low-dimensional representations of RL problems efficiently. For instance, the DQN algorithm [22] utilizes a DNN parameterized by θ^Q to approximate Q-value as $Q(s_t, a_t|\theta^Q)$. Moreover, an experience replay buffer $\mathcal{B}$ is employed to store the agent's experience tuple $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time step t, which can be then used to resolve the problem of instability of using function approximation in RL. Specifically, for each time slot, a mini-batch of samples $(s,a,r,s') \sim U(\mathcal{B})$ will be drawn uniformly at random from $\mathcal{B}$ to calculate the following loss function:

$$L(\theta^Q) = \mathbb{E}_{(s,a,r,s')\sim U(\mathcal{B})}\left[\left(r + \gamma \max_{\hat{a}\in\mathcal{A}} Q(s', \hat{a}|\theta^{Q'}) - Q(s,a|\theta^Q)\right)^2\right]. \tag{17}$$

Such loss function will be used to update the network parameter by $\theta^Q \leftarrow \theta^Q - \alpha_Q \cdot \nabla_{\theta^Q} L(\theta^Q)$ with a learning rate α_Q. In order to further improve stability, the so-called soft update strategy is adopted by DQN, where a target network $\theta^{Q'}$ in (17) is used to derive the TD error for the agent. Note that the target network tracks the weights of the learned network by $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$ with $\tau \ll 1$. Besides, from (15), we know that the action taken by the agent at each time step t is obtained by $a_t = \arg\max_{a\in\mathcal{A}} Q(s_t, a|\theta^Q)$.

3.2 DDPG-based dynamic computation offloading

Although problems in high-dimensional state spaces can be solved by DQN, only discrete and low-dimensional action spaces is supported. To this end, DDPG has been proposed [35] to extend conventional DRL algorithms to continuous action space. As shown in Fig. 2, an actor-critic approach is adopted by using two separate DNNs to approximate the Q-value network $Q(s,a|\theta^Q)$, i.e., the critic function, and the policy network $\mu(s|\theta^\mu)$, i.e., the actor function, respectively[4]. Specifically, the critic $Q(s,a|\theta^Q)$ is similar to DQN and can be updated according to (17). On the other hand, the actor $\mu(s|\theta^\mu)$ deterministically maps state s to a specific continuous action. As derived in [37], policy gradient of the actor can be calculated as follows,

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_{(s,a,r,s')\sim U(\mathcal{B})}\left[\nabla_a Q(s,a|\theta^Q)|_{a=\mu(s|\theta^\mu)} \cdot \nabla_{\theta^\mu}\mu(s|\theta^\mu)\right], \tag{18}$$

which is actually the averaged gradient of the expected return from the start distribution J with respect to the actor parameter θ^μ over the sampled mini-batch $U(\mathcal{B})$. Note that the chain rule is applied here since the action $a_t = \mu(s|\theta^\mu)$ is taken as the input of the critic function. Thus, with (17) and (18) in hand, network parameters of the actor and the critic can be updated by $\theta^Q \leftarrow \theta^Q - \alpha_Q \cdot \nabla_{\theta^Q} L(\theta^Q)$ and $\theta^\mu \leftarrow \theta^\mu - \alpha_\mu \cdot \nabla_{\theta^\mu} J$, respectively. Here, α_Q and α_μ are the learning rates.

It is worth noting that the soft update strategy is also needed for DDPG. Thus, target networks parameterized by $\theta^{\mu'}$ and $\theta^{Q'}$ will be used to calculate the loss function different from (17),

$$L(\theta^Q) = \mathbb{E}_{(s,a,r,s')\sim U(\mathcal{B})}\left[\left(r + \gamma Q(s', \mu(s|\theta^{\mu'})|\theta^{Q'}) - Q(s,a|\theta^Q)\right)^2\right], \tag{19}$$

[4]Note that similar to DQN, both the original DNNs in DDPG also have its own target network parameterized by $\theta^{Q'}$ and $\theta^{\mu'}$, respectively, which use soft update strategy and slowly track the weights of the learned networks in the same way.

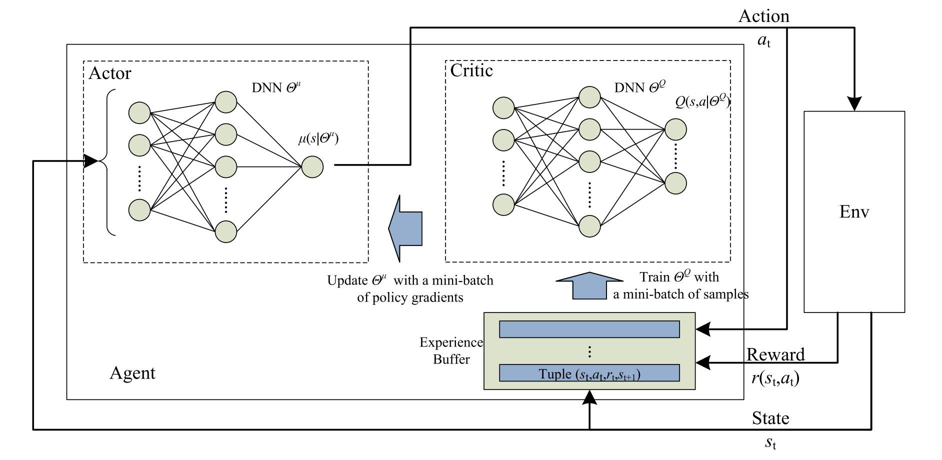

Fig. 2 Diagram of the DDPG algorithm

where the target networks track the weights of the learned networks by $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1 - \tau)\theta^{\mu'}$ and $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$ with $\tau \ll 1$, respectively.

In order to minimize the computation cost at each user, a DDPG-based framework is proposed to train decentralized policies to elaborately controlling power allocation for local execution and computation offloading. More specifically, the learned policy only depends on local observations of the environment and can make power allocation decisions in the continuous domain. In the sequel, definitions of state space, action space, and reward function in the framework will be introduced in detail.

State space: In the proposed framework, state space defines the representation of the environment from each user m's perspective, which will then used to make power allocation decisions. For decentralized policies, each user's state space only depends on local observations as shown in Fig. 3. Specifically, at the start of time slot t, the queue length of its data buffer $B_m(t)$ will be updated according to (6). Meanwhile, the user can receive the feedback from BS with its last receiving SINR $\gamma_m(t-1)$. Moreover, the channel vector $\boldsymbol{h}_m(t)$ for uplink transmission can be estimated by using channel reciprocity.

To define the state space based on the local observations mentioned above, we will firstly leverage $B_m(t)$ to indicate the amount of task data bits waiting to be executed. To reflect the impact of the interference from other mobile users' uplink signals under ZF detection, we denote the normalized SINR of slot t as follows,

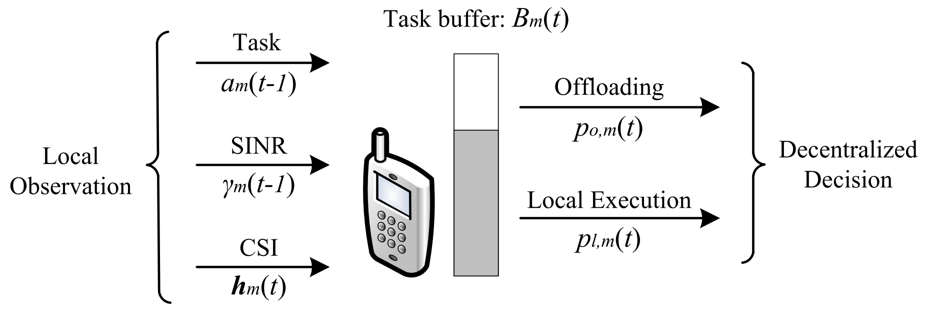

Fig. 3 Local observation and decentralized decision of user agent at slot *t*

$$\phi_m(t) = \frac{\gamma_m(t)\sigma_R^2}{p_{o,m}(t)\|\boldsymbol{h}_m(t)\|^2} \tag{20}$$

$$= \frac{1}{\|\boldsymbol{h}_m(t)\|^2\left[\left(\boldsymbol{H}^H(t)\boldsymbol{H}(t)\right)^{-1}\right]_{mm}}, \tag{21}$$

which represents the projected power ratio after ZF detection at BS. In fact, in order to decode user *m*'s symbol without inter-stream interference, ZF detection will project the received signal $\boldsymbol{y}(t)$ to the subspace orthogonal to the one spanned by the other users' channel vectors [38], after which the interference from other users has already removed for user *m*. Thus, $\phi_m(t)$ can be interpreted as the normalized SINR of user *m*, since it excludes the influence of its transmission power $p_{o,m}(t)$ and channel vector $\boldsymbol{h}_m(t)$ and is able to accurately reflect the impact of other users' interference on unit received power for user *m*. Practically, for slot *t*, the last normalized SINR $\phi_m(t-1)$ can be estimated from $\gamma_m(t-1)$ and then included in the state space. Besides, channel quality can be quantified by the channel vector $\boldsymbol{h}_m(t)$. To summarize, from the perspective of each user *m*, the state space of slot *t* can be defined as

$$s_{m,t} = [B_m(t), \phi_m(t-1), \boldsymbol{h}_m(t)]\,. \tag{22}$$

Action space: Based on the current state $s_{m,t}$ of the system observed by each user agent *m*, an action $a_{m,t}$ including the allocated powers for both local execution and computation offloading will be selected for each slot $t \in \mathcal{T}$ as below:

$$a_{m,t} = \left[p_{l,m}(t), p_{o,m}(t)\right]. \tag{23}$$

It is worth noting that, by applying the DDPG algorithm, either power allocation can be elaborately optimized in a continuous action space, i.e., $p_{l,m}(t) \in [0, P_{l,m}]$ and $p_{o,m}(t) \in [0, P_{o,m}]$, to minimize the average computation cost, unlike other conventional DRL algorithms to select from several predefined discrete power levels. Consequently, the high dimension of discrete action spaces can be significantly reduced.

Reward function: As mentioned in Section 3.1, the behavior of each user agent is reward-driven, and thus, the reward function plays a key role in the performance of DRL algorithms. In order to learn an energy-aware dynamic computation offloading policy which minimizes the long-term computation cost defined in (9), the reward function $r_{m,t}$ that each user received after time step *t* can be defined as

$$r_{m,t} = -w_{m,1}\cdot\left(p_{l,m}(t) + p_{o,m}(t)\right) - w_{m,2}\cdot B_m(t), \tag{24}$$

which is the negative weighted sum of the instantaneous total power consumption and the queue length of task buffer.

Remark 1. *For each user agent, the value function is defined as the expected discounted return starting from a state, which can be maximized by the optimal policy* [21]. *Specifically, in the proposed MEC model, the policy learned by the DDPG-based algorithm maximizes the value function of user m which starts from the initial state $s_{m,1}$ under policy μ_m, i.e.,*

$$V^{\mu_m}(s_{m,1}) = \mathbb{E}\left[\sum_{t=1}^{\infty}\gamma^{t-1}r_{m,t}|s_{m,1}\right], \tag{25}$$

which can be used to approximate the real expected infinite-horizon undiscounted return at each user agent when $\gamma \to 1$ [39]. *That is, the long-term average computation cost*

$$\bar{C}_m(s_{m,t}) = \mathbb{E}\left[\lim_{T\to\infty}\frac{1}{T}\sum_{i=t}^{T} w_{m,1}\left(p_{l,m}(i)+p_{o,m}(i)\right)+w_{m,2}B_m(i)|s_{m,t}\right], \tag{26}$$

will be minimized by applying the learned computation offloading policy μ_m^*.

3.3 Training and testing

To learn and evaluate the learned policies for decentralized computation offloading, a simulated environment with a group of user agents is constructed to conduct training and testing. Particularly, training and testing data can be generated from the interactions between the user agents and the simulated environment, which accepts the decision of each user agent and returns the local observations. Also notice that the interaction

Algorithm 1 Training Stage for the DDPG based Framework

```
1:  for each user agent $m \in \mathcal{M}$ do
2:      Randomly initialize the actor network $\mu(s|\theta_m^\mu)$ and the critic network $Q(s,a|\theta_m^Q)$;
3:      Initialize the associated target networks with weights $\theta_m^{\mu'} \leftarrow \theta_m^\mu$ and $\theta_m^{Q'} \leftarrow \theta_m^Q$;
4:      Initialize an empty experience replay buffer $\mathcal{B}_m$;
5:  end for
6:  for each episode $k = 1,2,\ldots,K_{\max}$ do
7:      Reset simulation parameters for the multi-user MEC model environment;
8:      Randomly generate an initial state $s_{m,1}$ for each user agent $m \in \mathcal{M}$;
9:      for each time slot $t = 1,2,\ldots,T_{\max}$ do
10:         for each user agent $m \in \mathcal{M}$ do
11:             Determine the power for local execution and computation offloading by selecting an action $a_{m,t} = \mu(s_{m,t}|\theta_m^\mu) + \Delta\mu$ using running the current policy network $\theta_m^\mu$ and generating exploration noise $\Delta\mu$;
12:             Execute action $a_{m,t}$ independently at the user agent, and then receive reward $r_{m,t}$ and observe the next state $s_{m,t+1}$ from the environment simulator;
13:             Collect and save the tuple $(s_{m,t}, a_{m,t}, r_{m,t}, s_{m,t+1})$ into the replay buffer $\mathcal{B}_m$;
14:             Randomly sample a mini-batch of $I$ tuples $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^I$ from $\mathcal{B}_m$;
15:             Update the critic network $Q(s,a|\theta_m^Q)$ by minimizing the loss $L$ with the samples:
                    $L = \frac{1}{I}\sum_{i=1}^{I}\left(r_i + Q(s'_i, \mu(s_i|\theta_m^{\mu'})|\theta_m^{Q'}) - Q(s_i, a_i|\theta_m^Q)\right)^2;$
16:             Update the actor network $\mu(s|\theta_m^\mu)$ by using the sampled policy gradient:
                    $\nabla_{\theta_m^\mu} J \approx \frac{1}{I}\sum_{i=1}^{I}\nabla_a Q(s_i, a|\theta_m^Q)|_{a=\mu(s_i|\theta_m^\mu)} \cdot \nabla_{\theta_m^\mu}\mu(s_i|\theta_m^\mu);$
17:             Update the target networks by $\theta_m^{\mu'} \leftarrow \tau\theta_m^\mu + (1-\tau)\theta_m^{\mu'}$ and $\theta_m^{Q'} \leftarrow \tau\theta_m^Q + (1-\tau)\theta_m^{Q'}$;
18:         end for
19:     end for
20: end for
```

between the user agents and the environment is generally continuing RL tasks [21], which does not break naturally into identifiable episodes[5]. Thus, in order to have better exploration performance, the interaction will be manually started with a random initial state $s_{m,1}$ for each user m and terminate at a predefined maximum steps $T_{\max}$ for each episode.

The detailed training stage is illustrated in Algorithm 1. At each time step t during an episode, each agent's experience tuple $(s_{m,t}, a_{a,t}, r_{m,t}, s_{m,t+1})$ will be stored in its own experience buffer $\mathcal{B}_m$. Meanwhile, the use agent's actor and critic network will be updated accordingly using a mini-batch of experience tuples $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^{I}$ randomly sampled from the replay buffer $\mathcal{B}_m$. In this way, after the training of $K_{\max}$ episodes, the dynamic computation offloading policy will be gradually and independently learned at each user agent. In order to improve the model with adequate exploration of the state space, one major challenge is the trade-off between exploration and exploitation, which is even more difficult for learning in continuous action spaces [35]. Since the exploration of DDPG are treated independently from the learning process, the exploration policy μ' can be constructed by adding noise $\Delta\mu$ sampled from a random noise process to the actor, i.e.,

$$\mu'(s) = \mu(s|\theta^{\mu}) + \Delta\mu, \tag{27}$$

where the random noise process needs to be elaborately selected. E.g., exploration noise sampled from a temporally correlated random process can better preserve momentum.

As for the testing stage, each user agent will firstly load its actor network parameters learned in the training stage. Then, the user agent will start with an empty data buffer and interact with a randomly initialized environment, where it selects actions according to the output of the actor network, using its local observation of the environment as current state.

4 Results and discussion

In this section, numerical simulations are presented to illustrate the proposed DRL framework for decentralized dynamic computation offloading in the proposed system. Performance of the DDPG-based framework is demonstrated and compared with some other baseline schemes in the scenarios of single user and multiple users, respectively.

4.1 Simulation setup

In the system, time is slotted by $\tau_0 = 1$ms. Task arrivals at each user m follow Poisson distribution as $a_m(t) \sim \text{Pois}(\lambda_m)$, where the mean λ_m is referred as the task arrival rate, i.e., $\lambda_m = \mathbb{E}[a_m(t)]$. For the beginning of every episode, each user m's channel vector is initialized as $\boldsymbol{h}_m(0) \sim \mathcal{CN}(0, h_0(d_0/d_m)^{\alpha}\boldsymbol{I}_N)$, where path-loss constant $h_0 = -30$dB, reference distance $d_0 = 1$m, path-loss exponent $\alpha = 3$, and d_m is the distance of user m to BS in meters. In the following slots, $\boldsymbol{h}_m(t)$ will be updated according to (2), where the channel correlation coefficient $\rho_m = 0.95$ and the error vector $\boldsymbol{e}(t) \sim \mathcal{CN}(0, h_0(d_0/d_m)^{\alpha}\boldsymbol{I}_N)$ with $f_{d,m} = 70$Hz. Additionally, we set the system bandwidth to be 1MHz, the maximum transmission power $P_{o,m} = 2$W, and the noise power $\sigma_R^2 = 10^{-9}$W. On the other hand, for local execution, we assume that $\kappa = 10^{-27}$, the required CPU cycles per bit $L_m = 500$ cycles/bit, and the maximum allowable CPU-cycle frequency $F_m = 1.26$GHz, from which we know that the maximum power required for local execution $P_{l,m} = 2$W.

[5]Note that for episodic RL tasks, each initial state of the user agent will terminate at a specific state.

To implement the DDPG algorithm, the actor and critic networks at each user agent m is a four-layer fully connected neural network with two hidden layers. The number of neurons in the two hidden layers are 400 and 300, respectively. The neural networks use the Relu, i.e., $f(x) = \max(0, x)$, as the activation function for all hidden layers, while the final output layer of the actor uses a sigmoid layer to bound the actions. Note that for the critic, actions are not included until the second hidden layer of the Q-network. Adaptive moment estimation (Adam) method [40] is used for learning the neural network parameters with a learning rate of 0.0001 and 0.001 for the actor and critic respectively. The soft update rate for the target networks is $\tau = 0.001$. Moreover, in order to explore well, the Ornstein-Uhlenbeck process [41] with $\theta = 0.15$ and $\sigma = 0.12$ is used to provide temporal correlated noise. The experience replay buffer size is set as $|\mathcal{B}_m| = 2.5 \times 10^5$, and the mini-batch size $M = 16$.

To better explore the tradeoff, a factor $w_m \in [0, 1]$ is introduced to rephrase the two nonnegative weighted factors as $w_{m,1} = 10w_m$ and $w_{m,2} = 1 - w_m$ for each use agent $m \in \mathcal{M}$. Note that the representation of $w_{m,1}$ indicates that energy consumption has more contribution to computation cost. Thus, the reward function $r_{m,t}$ in (24) can be written by

$$r_{m,t} = -10w_m \cdot \left(p_{l,m}(t) + p_{o,m}(t)\right) - (1 - w_m) \cdot B_m(t), \tag{28}$$

from which the tradeoff between energy consumption and buffering delay can be made by simply setting a single factor w_m. Moreover, the number of episodes is $K_{\max} = 2000$ in the training stage, and the maximum steps of each episode is $T_{\max} = 200$.

For comparison, some baseline strategies are introduced:

- Greedy Local Execution First (GD-Local): For each slot, the user agent firstly execute task data bits locally as many as possible and then the remaining buffered bits is offloaded.
- Greedy Computation Offloading First (GD-Offload): Each user agent firstly offloads data bits with best efforts and then execute the remaining buffered data bits locally.
- DQN-based Dynamic Offloading (DQN): The conventional discrete action space-based DQN algorithm [22] is adopted. In order to have a fair comparison between DDPG and DQN, we adopt the same neural network configurations as in DDPG. Besides, the number of discrete power levels are selected such that both DQN and DDPG can achieve the same convergence behavior. In this case, the power levels for local execution and computation offloading are defined as $\mathcal{P}_{l,m} = \{0, \frac{P_{l,m}}{L-1}, \ldots, P_{l,m}\}$ and $\mathcal{P}_{o,m} = \{0, \frac{P_{o,m}}{L-1}, \ldots, P_{o,m}\}$, i.e., the number of power levels is set as $L = 8$. The action space for each user agent to select from is $\mathcal{P}_{l,m} \times \mathcal{P}_{o,m}$. Besides, ϵ-greedy exploration and Adam method are adopted for training.

4.2 Single user scenario

The user is set to be randomly located in a distance of $d_1 = 100$ meters to BS.

4.2.1 Training

As shown in Figs. 4 and 5, the training process of single-user dynamic computation offloading is presented by setting $w_1 = 0.5$ and 0.8 respectively, where each curve is averaged from 10 runs of numerical simulations. In each figure, two different cases, i.e., task arrival rate $\lambda_1 = 2.0$Mbps and 3.0Mbps, are also illustrated. It can be observed that for both policies learned by DDPG and DQN, the average reward of each episode increases as

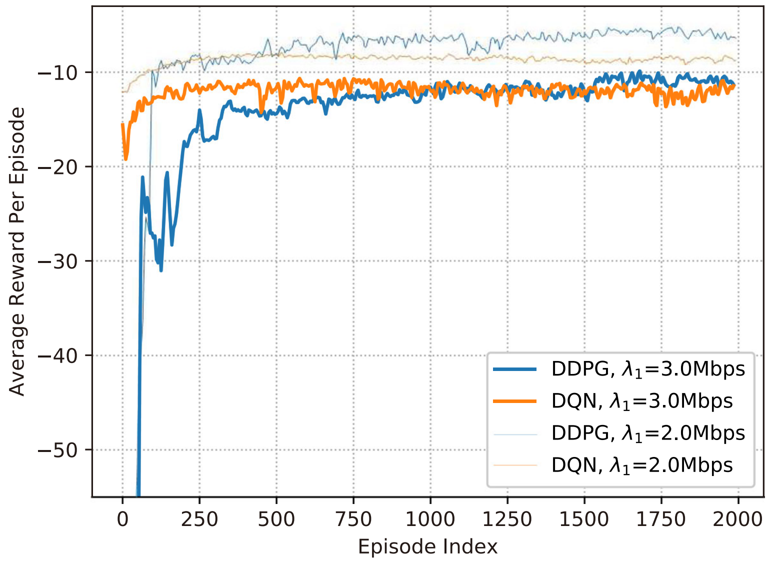

Fig. 4 Average reward per episode in the training process for a single user agent with $w_1 = 0.5$

the interaction between the user agent and the proposed system environment continues, which indicates that efficient computation offloading policies can be gradually learned. On the other hand, performance of each learned policy becomes stable after about 1500 episodes. In particular, the stable performance of the policy learned from DDPG is always better than DQN for different scenarios, which demonstrates that for continuous control problems, DDPG-based strategies can explore the action space more efficiently than DQN-based strategies.

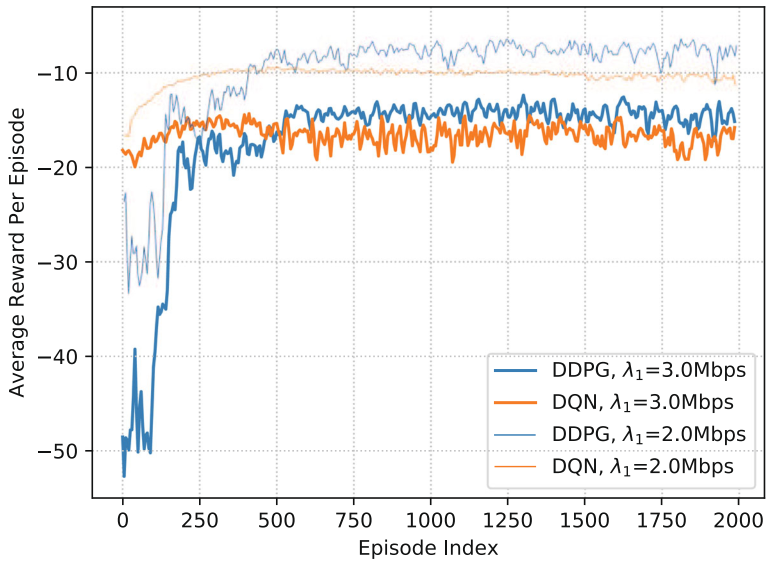

Fig. 5 Average reward per episode in the training process for a single user agent with $w_1 = 0.8$

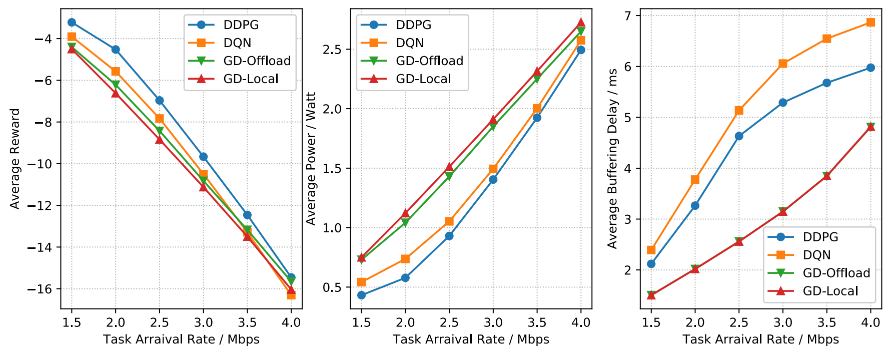

Fig. 6 Comparison of testing results for a single user agent with $w_1 = 0.5$

4.2.2 Testing

After $K_{\max} = 2000$ episodes' training, we have obtained dynamic computation offloading policies learned by the DDPG-based and DQN-based strategies, respectively. To compare the performance of different policies, tests are conducted under 100 runs of numerical simulations, where each run consists of 10000 steps. As shown in Figs. 6 and 7, averaged testing results are presented for $w_1 = 0.5$ and 0.8, respectively, each of which includes the performance of average reward, power consumption, and buffering delay. It can be observed from Fig. 6 that the average reward decreases as the task arrival rate grows, which indicates the computation cost becomes higher when facing a larger computation demand. Specifically, the increased computation cost results from a higher power consumption and a longer buffering delay. Also keep in mind that each user's reward is defined as the negative of its actual computation cost as in (24). Moreover, although the DDPG-based strategy outperforms both greedy strategies, which, however, slightly compromises the buffering delay to achieve the lowest energy consumption. It is worth noting that the averaged reward of the DQN-based strategy is lower than the greedy strategies for task arrival rates higher than 3.5Mbps, which is due to the limited number of discrete power levels of DQN-based strategy[6]. Meanwhile, the two greedy strategies, i.e., GD-Local and GD-Offload, have almost the same average buffering delay in Fig. 6, which is due to the fact that both strategies attempt to execute buffered task bits with best effort during each time slot, regardless of the factor w_1. Actually, for lower task arrival rates, e.g. $\lambda_1 = 1.5 \sim 3.0$Mbps, it is highly possible that task bits arrived in one slot can be completely executed during the next slot for either greedy strategy. The only difference of those two strategies is the priority of local execution and task offloading, which leads to different average power consumption as shown in Fig. 6. Therefore, both average buffering delay are very close to the average amount of task bits arrived in one slot, which will slightly increase as the task arrival rate λ_1 grows.

In Fig. 7, testing results for a larger tradeoff factor $w_1 = 0.8$ are also provided. From (28), we know that a larger w_1 will give more penalty on power consumption in the reward function. In this scenario, the average reward of the DDPG-based strategy outperforms all the other strategies again, and the gap is much larger than that in the case of $w_1 = 0.5$. Specifically, this is achieved by much lower power consumption and increased buffering

[6]Although finer grained discretization of the action space will potentially lead to better performance, the number of actions increases exponentially with the number of degrees of freedom, which makes it much more challenging to explore efficiently and in turn significantly decreases the performance.

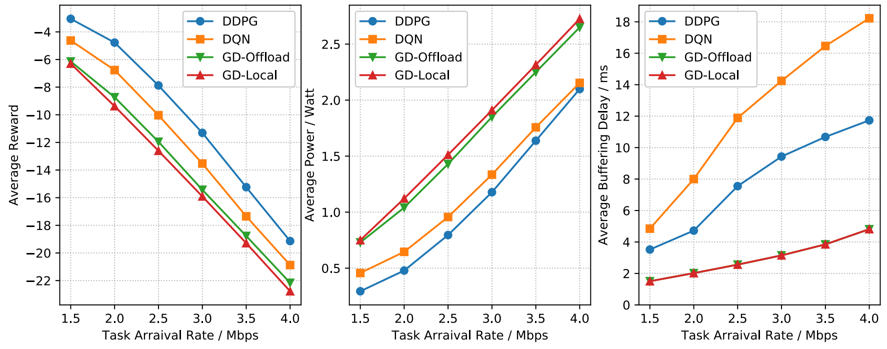

Fig. 7 Comparison of testing results for a single user agent with $w_1 = 0.8$

delay. As for the average buffering delay of GD-Local and GD-Offload in Fig. 7, they are the same to that in Fig. 6, since the two greedy strategies are not impacted by w_1 and their behaviors will not change as well.

4.2.3 Power-delay tradeoff

We also investigate testing results for the power-delay tradeoff by setting different values of w_1 in Fig. 8. It can be inferred from the curves that, there is a tradeoff between the average power consumption and the average buffering delay. Specifically, with a larger w_1, the power consumption will be decreased by sacrificing the delay performance, which indicates that in practice w_1 can be tuned to have a minimum power consumption with a given delay constraint. It is also worth noting that for each value of w_1, the policy learned form DDPG always has better performance in terms of both power consumption and buffering delay, which demonstrates the superiority of the DDPG-based strategy for continuous power control.

4.3 Multi-user scenario

There are $M = 3$ mobile users in the system, each of which is randomly located in a distance of $d_m = 100$ meters to BS, and the task arrival rate is $\lambda_m = m \times 1.0$Mbps, for $m \in \{1, 2, 3\}$.

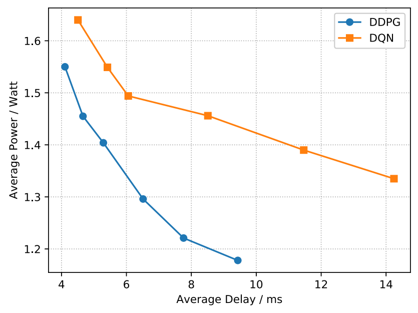

Fig. 8 Illustration of power-delay tradeoff for a single user agent with $\lambda_1 = 3.0$Mbps. Note that for the points on each curve from left to right, the tradeoff factor w_1 varies from $0.3 \sim 0.8$, correspondingly. Notice that there is no trade-off curve for these two greedy algorithms, since they are not impacted by the tradeoff factors. The power-delay points for GD-offload and GD-local are located at $(1.85, 3.148)$ and $(1.91, 3.147)$, respectively

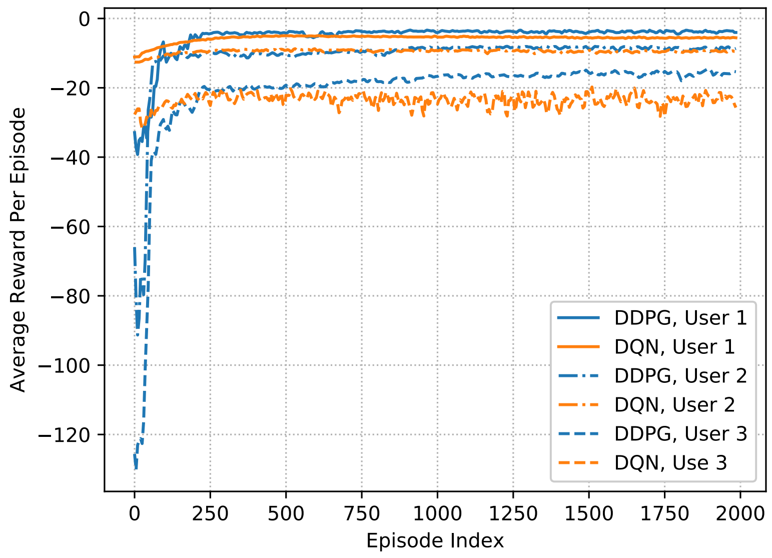

Fig. 9 Illustration of the average reward in the training process with $w_m = 0.5$ for all mobile users

4.3.1 Training

By setting $w_m = 0.5$ for all users, the training process has been shown in Fig. 9. It can be observed that for each mobile user, the average reward increases gradually when the mobile user interacts with the system environment after more episodes. Thus, we know that for both the DDPG-based and DQN-based strategies, efficient decentralized dynamic computation offloading policies can be learned at each mobile user, especially for heterogeneous users with different computation demands. Moreover, it can be inferred that the higher computation cost needs to be paid by the user with a higher computation demand. Meanwhile, compared with the single user scenario, the average reward obtained in the multi-user scenario is much lower for the same task arrival rate. It is due to the fact that the spectral efficiency of data transmission will be degraded when more mobile users are served by BS. Hence, more power will be consumed in computation offloading in the multi-user scenario.

4.3.2 Testing

Loading the neural network parameters learned by the DDPG-based and DQN-based algorithms after $K_{\max} = 2000$ episodes, testing results of different dynamic computation offloading policies are compared in Tables 2 and 3. From Table 2, we know that the average rewards of user 2 and user 3 adopting DDPG-based strategies are better than all other strategies under the scenario of $w_m = 0.5$. However, as for user 1, the DDPG-based strategy is slightly worse than the GD-Local strategy, which indicates that the exploration of

Table 2 Comparison of testing results with $w_m = 0.5$ for all mobile users

	Average reward			Average power (watt)			Average delay (ms)		
	User 1	*User 2*	*User 3*	*User 1*	*User 2*	*User 3*	*User 1*	*User 2*	*User 3*
DDPG	− 1.770	− **5.670**	− **12.782**	**0.205**	**0.774**	**1.939**	1.489	3.600	6.174
DQN	− 2.174	− 7.657	− 14.688	0.292	1.156	2.320	1.428	3.753	6.176
GD-Offload	− 2.514	− 7.597	− 18.690	0.402	1.309	3.143	**1.007**	**2.103**	**5.951**
GD-Local	− **1.633**	− 9.504	− 20.071	0.216	1.678	3.407	1.106	2.228	6.072

Table 3 Comparison of testing results with $w_m = 0.8$ for all mobile users

	Average reward			**Average power (watt)**			**Average delay (ms)**		
	User 1	***User 2***	***User 3***	***User 1***	***User 2***	***User 3***	***User 1***	***User 2***	***User 3***
DDPG	− **1.919**	− **6.366**	− **16.164**	**0.162**	**0.602**	**1.674**	3.114	7.752	13.861
DQN	− 2.780	− 8.915	− 18.675	0.284	0.915	1.954	2.539	7.973	15.216
GD-Offload	− 3.417	− 10.893	− 26.334	0.402	1.309	3.143	**1.007**	**2.103**	**5.951**
GD-Local	− 1.949	− 13.870	− 28.470	0.216	1.678	3.407	1.106	2.228	6.072

DDPG for a small allocated power needs to be further improved. Also, it can be observed that both the DDPG-based and DQN-based strategies can achieve much lower power consumption with a little compromised buffering delay.

By setting the tradeoff factor $w_m = 0.8$ as shown in Table 3, the DDPG-based strategies obtain the best average reward at each user agent. Moreover, the performance gaps between the DDPG-based strategies and other greedy strategies become larger. Besides, with more penalty given to the power consumption, the consumed power of each user is much lower than that of $w_m = 0.5$, which, however, results in a moderately increased buffering delay. Thus, we know that for the multi-user scenario, power consumption can be minimized with a satisfied average buffering delay, by selecting a proper value of w_m. Again, the DDPG-based strategies outperform the DQN-based strategies in terms of average reward for all users.

5 Conclusions and future directions

In this paper, we considered an MEC enabled multi-user MIMO system with stochastic tasks arrivals and wireless channels. In order to minimize the long-term average computation cost in terms of power consumption and buffering delay, the design of decentralized dynamic computation offloading algorithms has been investigated. Specifically, by adopting the continuous action space-based DDPG algorithm, an efficient computation offloading policy has been successfully learned at each user, which allocates powers for local execution and task offloading from its local observation of the proposed system environment. Numerical simulations have been performed to verify the superiority of the proposed DDPG-based strategy over some other baseline schemes. Besides, power-delay tradeoff for both the DDPG-based and DQN-based strategies has been also studied. For future directions, we would like firstly to investigate decentralized binary computation offloading strategies for indivisible tasks, i.e., the task can only be offloaded to the BS or executed locally as a whole. This may require a hierarchical DRL framework to give a binary offloading decision and then continuous power allocation decision. Moreover, collaboration among mobile users can also be introduced in the DDPG-based framework to improve performance of learned policies, since currently policy is trained independently at each user without considering other user agents' behaviors.

Abbreviations

MEC: Mobile edge computing, MIMO: Multi-input multi-output, RL: Reinforcement learning, DRL: Deep reinforcement learning, DDPG: Deep deterministic policy gradient, DQN: Deep Q-network, AR/VR: Augmented or virtual reality, ITS: Intelligent Transport Systems, IoT: Internet of Things, QoE: Quality of experience, RAN: Radio access network, BS: Base station, DVFS: Dynamic voltage and frequency, NOMA: Non-orthogonal multiple access, MDP: Markov decision process, DNN: Deep neural network, AWGN: Additive white Gaussian noise, ZF: Zero-forcing, SINR: Signal-to-interference-plus-noise, TD: Temporal-difference, Adam: Adaptive moment estimation, GD-Local: Greedy Local Execution First, GD-Offload: Greedy Computation Offloading First

Acknowledgements
The authors sincerely thank the anonymous referees for their technical suggestions and their advice on decorum.

Authors' contributions
Zhao Chen contributed to the system modeling, theoretical analysis, algorithm design, and numerical simulation as well as writing of this article. Xiaodong Wang supervised the work and proofread this article. The authors read and approved the final manuscript.

Availability of data and materials
Not applicable.

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

Author details
[1]Beijing National Research Center for Information Science and Technology, Tsinghua University, 100084 Beijing, China.
[2]Department of Electrical Engineering, Columbia University, 10027 New York, NY, USA.

Received: 4 January 2020 Accepted: 8 September 2020
Published online: 29 September 2020

References

1. W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
2. X. Sun, N. Ansari, EdgeIoT: mobile edge computing for the Internet of Things. IEEE Commun. Mag. **54**(12), 22–29 (2016)
3. K. Zhang, Y. Mao, S. Leng, Y. He, Y. Zhang, Mobile-edge computing for vehicular networks: a promising network paradigm with predictive off-loading. IEEE Veh. Technol. Mag. **12**(2), 36–44 (2017)
4. M. Satyanarayanan, The emergence of edge computing. Computer. **50**(1), 30–39 (2017)
5. Y. Mao, C. You, J. Zhang, K. Huang, K. B. Letaief, A survey on mobile edge computing: the communication perspective. IEEE Commun. Surv. Tuts. **19**(4), 2322–2358 (2017)
6. 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)
7. H. Guo, J. Liu, J. Zhang, W. Sun, N. Kato, Mobile-edge computation offloading for ultra-dense IoT networks. IEEE Internet Things J. **5**(6), 4977–4988 (2018)
8. J. Zhang, X. Hu, Z. Ning, E. C.-. Ngai, L. Zhou, J. Wei, J. Cheng, B. Hu, Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks. IEEE Internet Things J. **5**(4), 2633–2645 (2018)
9. S. Bi, Y. J. Zhang, Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. IEEE Trans. Wirel. Commun. **17**(6), 4177–4190 (2018)
10. Z. Ding, P. Fan, H. V. Poor, Impact of non-orthogonal multiple access on the offloading of mobile edge computing. IEEE Trans Commun. **67**(1), 375–390 (2018)
11. W. Wu, F. Zhou, R. Q. Hu, B. Wang, Energy-efficient resource allocation for secure noma-enabled mobile edge computing networks. IEEE Trans. Commun. **68**(1), 493–505 (2019)
12. J. Zhu, J. Wang, Y. Huang, F. Fang, K. Navaie, Z. Ding, Resource allocation for hybrid NOMA MEC offloading. IEEE Trans. Wirel. Commun. **19**(7), 4964–4977 (2020)
13. J. Kwak, Y. Kim, J. Lee, S. Chong, Dream: dynamic resource and task allocation for energy minimization in mobile cloud systems. IEEE J. Sel. Areas Commun. **33**(12), 2510–2523 (2015)
14. S. Sardellitti, G. Scutari, S. Barbarossa, Joint optimization of radio and computational resources for multicell mobile-edge computing. IEEE Trans. Signal Inf. Process. Over Netw. **1**(2), 89–103 (2015)
15. Y. Mao, J. Zhang, K. B. Letaief, Dynamic computation offloading for mobile-edge computing with energy harvesting devices. IEEE J. Sel. Areas Commun. **34**(12), 3590–3605 (2016)
16. Y. Mao, J. Zhang, S. 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)
17. 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)
18. W. Chen, D. Wang, K. Li, Multi-user multi-task computation offloading in green mobile edge cloud computing. IEEE Trans. Serv. Comput. **12**(5), 726–738 (2018)
19. J. Liu, Y. Mao, J. Zhang, K. B. Letaief, in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Delay-optimal computation task scheduling for mobile-edge computing systems (IEEE, Honolulu, 2016), pp. 1451–1455
20. T. Q. Dinh, Q. D. La, T. Q. Quek, H. Shin, Distributed learning for computation offloading in mobile edge computing. IEEE Trans. Commun. **66**(12), 6353–6367 (2018)
21. R. S. Sutton, A. G. Barto, et al., *Reinforcement learning: an introduction*. (MIT Press, Cambridge, MA, 1998)
22. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning. Nature. **518**(7540), 529 (2015)
23. J. Li, H. Gao, T. Lv, Y. Lu, in *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, Deep reinforcement learning based computation offloading and resource allocation for MEC (IEEE, Barcelona, 2018), pp. 1–6
24. L. Huang, S. Bi, Y.-J. A. Zhang, Deep reinforcement learning for online offloading in wireless powered mobile-edge computing networks. IEEE Trans Mob Comput, 1–1 (2019)
25. M. Min, D. Xu, L. Xiao, Y. Tang, D. Wu, Learning-based computation offloading for IoT devices with energy harvesting. IEEE Trans Veh Technol. **68**(2), 1930–1941 (2019)
26. X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, M. Bennis, Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. IEEE Int Things J. **6**(3), 4005–4018 (2018)
27. Y. Liu, H. Yu, S. Xie, Y. Zhang, Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks. IEEE Trans. Veh. Technol. **68**(11), 11158–11168 (2019)

28. P. Mach, Z. Becvar, Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun. Surv. Tuts. **19**(3), 1628–1656 (2017)
29. H. A. Suraweera, T. A. Tsiftsis, G. K. Karagiannidis, A. Nallanathan, Effect of feedback delay on amplify-and-forward relay networks with beamforming. IEEE Trans. Veh. Technol. **60**(3), 1265–1271 (2011)
30. M. Abramowitz, I. A. Stegun, et al., *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables, vol. 55*. (Dover publications, New York, 1972)
31. H. Q. Ngo, E. G. Larsson, T. L. Marzetta, Energy and spectral efficiency of very large multiuser MIMO systems. IEEE Trans. Commun. **61**(4), 1436–1449 (2013)
32. T. D. Burd, R. W. Brodersen, Processor design for portable systems. J. VLSI Signal Process. Syst. Signal Image Video Technol. **13**(2-3), 203–221 (1996)
33. A. P. Miettinen, J. K. Nurminen, Energy efficiency of mobile clients in cloud computing. HotCloud. **10**, 4–4 (2010)
34. J. F. Shortle, J. M. Thompson, D. Gross, C. M. Harris, *Fundamentals of Queueing Theory, vol. 399*. (Wiley, Hoboken, 2018)
35. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, in *Proc. International Conference on Learning Representations (ICLR)*, Continuous control with deep reinforcement learning, (San Juan, 2016)
36. C. J. C. H. Watkins, P. Dayan, Q-learning. Mach. Learn. **8**(3), 279–292 (1992)
37. D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, in *Proc. International Conference on Machine Learning (ICML)*, Deterministic policy gradient algorithms, (New York City, 2014), pp. 387–395
38. D. Tse, P. Viswanath, *Fundamentals of wireless communication*. (Cambridge university press, Cambridge, 2005)
39. D. Adelman, A. J. Mersereau, Relaxations of weakly coupled stochastic dynamic programs. Oper. Res. **56**(3), 712–727 (2008)
40. D. P. Kingma, J. Ba, in *Proc. International Conference on Learning Representations (ICLR), San Diego, CA, USA*, Adam: a method for stochastic optimization, (2015)
41. G. E. Uhlenbeck, L. S. Ornstein, On the theory of the Brownian motion. Phys. Rev. **36**(5), 823 (1930)

Publisher's Note