A DRL Agent for Jointly Optimizing Computation Offloading and Resource Allocation in MEC

Juan Chen, Huanlai Xing, *Member, IEEE*, Zhiwen Xiao, *Student Member, IEEE*, Lexi Xu, *Member, IEEE*, and Tao Tao

***Abstract*—This article studies the joint optimization problem of computation offloading and resource allocation (JCORA) in mobile-edge computing (MEC). Deep reinforcement learning (DRL) is one of the ideal techniques for addressing the dynamic JCORA problem. However, it is still challenging to adapt traditional DRL methods for the problem since they usually lead to slow and unstable convergence in model training. To this end, we propose a temporal attentional deterministic policy gradient (TADPG) to tackle JCORA. Based on the deep deterministic policy gradient (DDPG), TADPG has two significant features. First, a temporal feature extraction network consisting of a 1-D convolution (Conv1D) residual block and an attentional long short-term memory (LSTM) network is designed, which is beneficial to high-quality state representation and function approximation. Second, a rank-based prioritized experience replay (rPER) method is devised to accelerate and stabilize the convergence of model training. Experimental results demonstrate that the decentralized TADPG-based mechanism can achieve more efficient JCORA performance than the centralized one, and the proposed TADPG outperforms a number of state-of-the-art DRL agents in terms of the task completion time and energy consumption.**

***Index Terms*—Computation offloading, deep deterministic policy gradient (DDPG), deep reinforcement learning (DRL), mobile-edge computing (MEC), resource allocation.**

I. Introduction

In MOBILE-EDGE computing (MEC), one of the most critical challenges is to decide how mobile devices (MDs) offload their tasks to edge servers, which is referred to as the computation offloading problem [1], [2]. Task offloading reduces the computational burden and prolongs the battery lifetime for MDs. However, it leads to additional transmission delay and power consumption when offloading task data. These conflicting factors need to be carefully dealt with to obtain a compromised solution. In addition, efficient resource allocation is essential to support computation offloading in MEC systems [3]. With limited computing and radio resources, a MEC system aims to accommodate as many computation offloading requests from different MDs as possible. Without efficient coordination of resource allocations, the system may lead to slow response time and high-power consumption, which, in turn, affects the overall computation offloading performance. Therefore, to maximize system efficiency, it is crucial to study the joint optimization problem of computation offloading and resource allocation (JCORA) since the two issues are closely related [4].

Traditional approaches, such as convex approximation [5], [6], game theory [7]–[9], and metaheuristics [4], [10], are not suitable for addressing dynamic JCORA problems in MEC. These methods usually suffer from exponentially increasing search space and heavy computational burdens, especially in large-scale scenarios. Adopting deep reinforcement learning (DRL) to handle JCORA problems has attracted increasingly more research interests because of the powerful deep neural networks (DNNs) [3], [12]–[20]. Deep deterministic policy gradient (DDPG) [16], [18] shows excellent potential when coping with dynamic optimization problems with a continuous action space. More details can be found in Section II.

In the traditional DDPG, both the actor and critic networks are based on fully connected networks (FCNs). However, FCNs generally suffer from two drawbacks. One is the huge number of trainable parameters, which increases the difficulty for training and significantly consumes computing resources. The other is that FCNs only extract the global discriminative state and action policy features, ignoring the temporal variation of task sequences that may contain useful shapelets for function approximation. Moreover, experience transitions are uniformly sampled from the replay buffer. This approach treats all experiences equally, regardless of their significance. However, ignoring the importance of valuable experiences usually leads to poor stability and slow convergence during training.

To overcome the disadvantages above, this article proposes temporal attentional deterministic policy gradient (TADPG), an improved DDPG agent for tackling the decentralized JCORA problem in a dynamic MEC environment. Running a TADPG agent on each MD requires fewer control costs between this MD and its corresponding MEC server. It is thus more suitable for large-scale scenarios, compared with those centralized JCORA methods. Our main contributions are summarized as follows.

Manuscript received January 3, 2021; revised March 21, 2021; accepted May 4, 2021. Date of publication May 19, 2021; date of current version December 7, 2021. This work was supported in part by the Fundamental Research Funds for the Central Universities and in part by the China Scholarship Council, China. *(Corresponding authors: Huanlai Xing; Lexi Xu.)*

Juan Chen, Huanlai Xing, and Zhiwen Xiao are with the School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu 611756, China (e-mail: hxx@home.swjtu.edu.cn).

Lexi Xu is with the Research Institute, China United Network Communications Corporation, Beijing 100048, China (e-mail: davidlexi@hotmail.com).

Tao Tao is with the Management Department, China United Network Communications Group Corporation, Beijing 100033, China (e-mail: taotao3@chinaunicom.cn).

Digital Object Identifier 10.1109/JIOT.2021.3081694

1) A JCORA problem in the single-cell multi-MD MEC system is studied. The average task completion time and the average energy consumption of MD are aggregated into one objective function for minimization. This article allows partial task offloading, where a proportion of an arbitrary task can be offloaded to the corresponding MEC server for processing. Each MD runs a TADPG agent, making independent decisions according to the local JCORA information only. We derive the optimal JCORA policy for each task under different network conditions.
2) TADPG is featured with a temporal feature extraction network (TFEN) and a rank-based prioritized experience replay (rPER). TFEN consists of a 1-D convolution (Conv1D) residual block and an attentional long short-term memory (LSTM) network. Instead of FCNs, both the actor and critic networks in TADPG are based on TFEN, taking advantage of local and historical states with special focuses to capture rich temporal features. rPER ensures that more important experiences are replayed at higher probabilities. It helps to not only speed up the training process but also improves its stability.
3) We evaluate the centralized and decentralized TADPG-based JCORA mechanisms in terms of convergence, optimization performance, and complexity. In addition, experimental results show that TADPG achieves better convergence in training, compared with four other DDPG variants. TADPG overweighs eight state-of-the-art DRL agents in most test scenarios with different mean data arriving rates and different numbers of MDs in terms of the average task completion time and average energy consumption of MD. The TADPG agent can achieve decent decentralized JCORA performance.

The remainder of this article is organized as follows. Section II reviews the related work. The JCORA problem is modeled in Section III. The proposed TADPG is introduced in Section IV. Section V analyzes and discusses the experimental results, and Section VI concludes this article.

II. Related Work

Various techniques have been devised to tackle computation offloading and resource allocation in MEC. We classify them into two research streams, including optimization-based and machine-learning-based techniques, respectively.

A. Optimization-Based Techniques

A static JCORA problem can be formulated as a mixed-integer nonlinear program. For example, Bi and Zhang [5] studied the binary computation offloading problem in multiuser MEC networks. They proposed a joint optimization method based on the alternating direction method of multipliers (ADMMs) decomposition to maximize the MD computation rate. Ahani and Yuan [6] developed an algorithm based on Lagrangian duality to handle task offloading. These exact approaches often involve too many complex calculation operations, e.g., matrix inversion and singular value decomposition, resulting in a slow response to the offloading decision making.

Some researchers adopted game theory to solve the multiuser JCORA optimization problem in a static scenario. For example, Jošilo and Dán [7] modeled the interaction process between MDs and the operator as a Stackelberg game and presented a decentralized algorithm to minimize the completion time of tasks. Hong *et al.* [8] applied a distributed game theory approach to the multihop computation offloading problem in an IoT-edge-cloud system to ensure the Quality of Service (QoS). Zheng *et al.* [9] studied the dynamic computation offloading in MEC-aided networks and used game theory to allocate wireless channels to users, where Nash equilibrium was achieved. However, these methods were computationally expensive, especially in large-scale scenarios.

Metaheuristics have been recognized as a good candidate when the network environment was static or quasistatic, e.g., wireless channel states never changed over time. For example, Guo *et al.* [10] investigated the energy-efficient computation offloading management scheme in MEC with small cell networks. A hierarchical algorithm based on the genetic algorithm and particle swarm optimization was devised. Tran and Pompili [4] were concerned with JCORA in multiserver MEC networks, where the task offloading decision, uplink transmission power of MD, and computing resource allocation at MEC servers were jointly optimized. The authors applied convex and quasiconvex optimization to solve the resource allocation subproblem and designed a heuristic to address the task offloading subproblem. However, Metaheuristic-based methods are not suitable for real MEC networks. The reason is that real MEC networks are dynamic and require a rapid response to offloading requests, but metaheuristics are usually time consuming to obtain a satisfactory solution.

In order to address dynamic JCORA problems, researchers focused on Lyapunov optimization techniques. For example, Wu *et al.* [11] designed an adaptive offloading decision algorithm based on Lyapunov optimization to minimize the overall energy consumption. However, Lyapunov optimization needs prior information on environment statistics, which may not be practically available in dynamic MEC systems.

B. Machine-Learning-Based Techniques

Dynamic offloading decision making is complicated since its influential factors are multidimensional and time varying. Recently, more and more research efforts have been dedicated to machine-learning-based techniques, including deep-learning (DL)-based and DRL-based methods.

1) Deep-Learning-Based Methods: Due to its excellent prediction and reasoning performance, DL-based methods [21] have been used to assist efficient offloading and resource allocation. Pradhan *et al.* [22] formulated a computation offloading problem for IoT applications in an uplink xL-MIMO C-RAN. A joint optimization scheme with supervised DL was deployed to minimize the total transmission power of IoT devices. Zhao *et al.* [23] constructed a multi-LSTM-based DL model to predict the real-time traffic of SBS and devised a cross-entropy (CE)-based mobile data offloading strategy. Liu *et al.* [24] integrated DL-based classification algorithms into an edge-computing-based real-time computing system, where CNN was used to perform data analysis. However, unpredictable training time, long response delay, and massive labeled data were the main challenges.

2) Deep Reinforcement-Learning-Based Methods: DRL learns optimal policies and makes quick decisions by interacting with a time-varying environment, which is suitable for dynamic MEC systems. There are mainly two research streams, including value-based and policy-based methods.

Value-Based DRLs: Huang *et al.* [12] addressed the online computation offloading in wireless-powered MEC networks, with a DRL algorithm designed for binary offloading decision making and resource allocation. Li *et al.* [13] applied the deep Q-network (DQN) to jointly handle the computation offloading and resource allocation in multiuser MEC. The sum cost of delay and energy consumption was minimized. Xiong e*t al.* [3] formulated a resource allocation problem in an IoT edge computing system and devised an improved DQN algorithm with multiple replay memories to increase resource utilization efficiency. Chen *et al.* [14] modeled the computation offloading problem as a Markov decision process. A deep state action–reward–state–action-based RL algorithm (Deep-SARL) was developed to maximize the system's long-term utility performance. Lu *et al.* [15] studied the computation offloading problem in large-scale heterogeneous MEC with multiple service nodes, where a DQN algorithm based on LSTM and candidate networks was put forward. The above DRL-based methods achieved decent performance without requiring prior knowledge of environment statistics. However, the value-based methods are incapable of handling continuous action space as the dynamic JCORA problem is featured. In DQNs, the degree of discretization needs to be set carefully. If it is too coarse, a large amount of behavioral information may be lost; if it is too small, a rapid increase in dimensions could be resulted, which leads to higher complexity.

Policy-Based DRLs: Lu *et al.* [16] proposed a double-dueling deterministic policy gradient (D^{3}PG) to jointly optimize the service latency, energy consumption, and task success rate. Zhang *et al.* [17] developed two DRL algorithms for dynamic computation offloading to provide low-latency computing services. The authors devised a hybrid-decision-based actor–critic learning (hybrid-AC) for the single-device scenario and a multidevice hybrid AC (md-Hybrid-AC) for the multidevice scenario. Chen and Wang [18] proposed a decentralized DDPG-based JCORA mechanism for a multiuser MEC system. Each DDPG agent learned its offloading policy independently to minimize the average long-term cost. Liu and Liao [19] designed a discrete and continuous actor–critic (DAC) approach to optimize resource allocation and offloading decisions. Qiu *et al.* [20] proposed a distributed and collective DRL algorithm called DC-DRL to handle the computation offloading problem for resource-intensive and deadline-sensitive applications.

Policy-based methods are more suitable to address the dynamic JCORA problem than value-based ones. Especially, the decentralized DDPG-based JCORA mechanism has shown

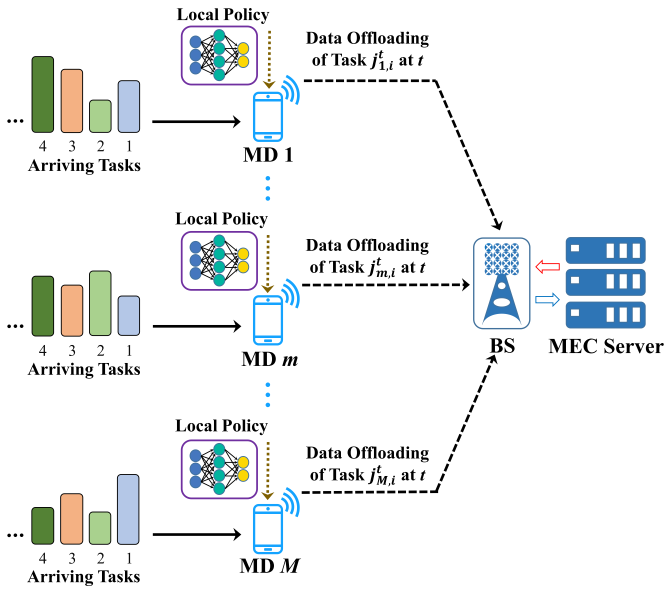

Fig. 1. Example MEC system.

promising potential to handle large-scale MEC scenarios [18]. Nevertheless, the FCNs-based networks in traditional DDPG could not extract the temporal information among task data as they were aware of global discrete discriminative features only. The uniform-sampling-based experience replay slowed the convergence and caused fluctuations to the training process. Motivated by the above, we aim at adapting DDPG for the decentralized JCORA mechanism for multi-MD MEC networks, with appropriate actor and critic networks and high-quality experience replay designed. That is the rationale behind the introduction of TADPG.

III. System Model and Problem Formulation

This section introduces the system and task queue models first. After that, the processing delay and energy consumption in the computing models are described. Then, the transmission delay and energy consumption in the communication model are discussed. Finally, this section gives the problem formulation.

A. System Model

This article considers a MEC system with a single cell and multiple MDs [25]. This system consists of a MEC server, a base station (BS) equipped with massive MIMO antenna arrays, and a set of MDs, as shown in Fig. 1. Each MD has a single antenna and connects to the BS via wireless channels. The MEC server connects to the BS through optic fibers or copper wires. Note that the number of antennas in the BS is significantly larger than the number of MDs.

Each MD maintains a task queue storing local computation-intensive tasks for processing and decides each task's offloading ratio. It involves a local JCORA decision-making process, supported by the TADPG agent in Section IV. Each MD independently learns and determines an appropriate offloading policy for each task in its queue, e.g., a proportion of input data to be offloaded to the MEC server.

B. Task Queue Model

This article considers a stochastic task arrival model. For an arbitrary MD, only one task arrives in each time slot, and its data size follows a Poisson distribution [26]. Each task, after its arrival, is stored in the task queue in the corresponding MD. Each MD processes the tasks in its queue one by one according to the first-in–first-out rule. Any two tasks are independent of each other.

Assume there are M MDs in the system, where M is a positive integer. In the task queue of MD m, $m = 1, 2, \ldots, M$, tasks dynamically change since those already completed are removed, and new arriving ones join in. Once the queue is full, any newly arrived task is discarded. Let $q(t)$ denote the number of tasks that have been processed by time slot t. Let $j^t_{m,i}$, $i = 1, 2, \ldots, q(T)$, be the ith task arriving at MD m, which is to be processed in t, where T is a sufficiently large value representing the last time slot elapsed in the MEC system. This article defines $j^t_{m,i}$ as a 4-tuple set $\{\pi^{\text{arr}}_{m,i}, d_{m,i}, \tau^{\max}_{m,i}, c_{m,i}\}$. $\pi^{\text{arr}}_{m,i}$ denotes the time slot when task $j^t_{m,i}$ arrives. $d_{m,i}$ is the input data size of $j^t_{m,i}$, which is independently generated and satisfies a Poisson process with the mean data arriving rate, λ_m. $\tau^{\max}_{m,i}$ stands for the maximum tolerable delay of completing $j^t_{m,i}$, which is a practical constraint of the JCORA problem, especially for those delay-sensitive tasks. In addition, $c_{m,i}$ is the number of CPU cycles needed to process per input data bit. Note that $d_{m,i}$, $\tau^{\max}_{m,i}$, and $c_{m,i}$ are easily obtained by a task profiler [13]. The main notations used in this article are summarized in Table I.

TABLE I
TABLE OF MAIN NOTATIONS

Notation	Description
$\alpha_{m,i}(t)$	the offloading ratio of $j^t_{m,i}$
$c_{m,i}$	the CPU cycles needed to process per input data bit
$C^t_{m,i}$	the weight sum cost of $\tau_{m,i}(t)$ and $E_{m,i}(t)$
C_{lterm}	the average long-term cost of all tasks
$d_{m,i}$	the size of input data of $j^t_{m,i}$
$\mathcal{E}$	the tradeoff weight between $\tau_{m,i}(t)$ and $E_{m,i}(t)$
$E^{\text{loc}}_{m,i}(t)$	the energy consumption incurred by processing the remaining data of $j^t_{m,i}$ on MD m
$E^{\text{tra}}_{m,i}(t)$	the energy consumption incurred by offloading the partial input data of $j^t_{m,i}$ to the MEC server
$E_{m,i}(t)$	the total energy consumption incurred by processing $j^t_{m,i}$ on MD m
$f_{m,i}(t)$	the CPU cycles per second for $j^t_{m,i}$ on MD m
f_{ser}	the CPU cycles per second on the MEC server
F_m	the maximum computing capacity of MD m
$\boldsymbol{g}_m(t)$	the channel gain vector between MD m and BS
$\hat{g}_m(t)$	the normalized effective channel gain for MD m
$j^t_{m,i}$	the i-th task arriving at MD m, which is to be processed in t
M	the total number of MDs
$P^{\text{tra}}_{m,i}(t)$	the uplink transmission power of MD m in time slot t
$P^{\max}_m$	the maximum uplink transmission power of MD m
$q(t)$	the number of tasks that have been processed by time slot t
$v^{\text{tra}}_{m,i}(t)$	the uplink transmission rate of $j^t_{m,i}$
T	the last time slot elapsed in the MEC system
$\tau^{\max}_{m,i}$	the maximum tolerable delay of completing $j^t_{m,i}$
$\tau^{\text{loc}}_{m,i}(t)$	the processing delay incurred by processing the remaining data of $j^t_{m,i}$ on MD m
$\tau^{\text{ser}}_{m,i}(t)$	the processing delay incurred by executing the partial input data of $j^t_{m,i}$ on the server
$\tau^{\text{tra}}_{m,i}(t)$	the transmission delay incurred by offloading the partial input data of $j^t_{m,i}$ to the MEC server
$\tau_{m,i}(t)$	the task completion time of $j^t_{m,i}$
$\pi^{\text{arr}}_{m,i}$	the arrival time slot of $j^t_{m,i}$
ϑ	the time duration each time slot spans
λ_m	the mean data arriving rate of MD m

C. Computing Models

This section introduces delay- and energy-related issues in the context of local and edge computing.

1) Local Computing: Let F_m and $f_{m,i}(t)$ denote the maximum computing capacity of MD m and the number of CPU cycles per second on MD m for processing task $j^t_{m,i}$, respectively. We assume $f_{m,i}(t) \le F_m$ is flexibly controlled via chip voltage adjustment using the dynamic voltage and frequency scaling (DVFS) technique [2]. Assume the input data of $j^t_{m,i}$, $d_{m,i}$, is fine-grained and can be arbitrarily divided into two parts, namely, one to be executed on MD m and the other to be offloaded and processed on the MEC server. Let the offloading ratio of $j^t_{m,i}$ denoted by $\alpha_{m,i}(t)$, indicating how much data in $d_{m,i}$ is offloaded to the edge. For $j^t_{m,i}$, the processing delay incurred on MD m, $\tau^{\text{loc}}_{m,i}(t)$, is calculated in [27]

$$\tau^{\text{loc}}_{m,i}(t) = \frac{\big(1 - \alpha_{m,i}(t)\big) \cdot d_{m,i} \cdot c_{m,i}}{f_{m,i}(t)}. \tag{1}$$

When processing the remaining data of $j^t_{m,i}$ on MD m, the incurred energy consumption $E^{\text{loc}}_{m,i}(t)$ is defined in

$$E^{\text{loc}}_{m,i}(t) = \kappa \cdot \big(f_{m,i}(t)\big)^2 \cdot \big(1 - \alpha_{m,i}(t)\big) \cdot d_{m,i} \cdot c_{m,i} \tag{2}$$

where $\kappa = 10^{-27}$ is an effectively switching capacitance constant. As $d_{m,i}$ and $c_{m,i}$ are determined once $j^t_{m,i}$ arrives, $\alpha_{m,i}(t)$ and $f_{m,i}(t)$ are the only variables directly affecting $\tau^{\text{loc}}_{m,i}(t)$ and $E^{\text{loc}}_{m,i}(t)$.

2) Edge Computing: As aforementioned, a proportion of the input data of task $j^t_{m,i}$, $\alpha_{m,i}(t) \cdot d_{m,i}$, is offloaded to the MEC server for processing. The processing delay incurred for executing the partial input data on the server $\tau^{\text{ser}}_{m,i}(t)$ can be defined in [26]

$$\tau^{\text{ser}}_{m,i}(t) = \frac{\alpha_{m,i}(t) \cdot d_{m,i} \cdot c_{m,i}}{f_{\text{ser}}}. \tag{3}$$

This article assumes the MEC server has sufficient computing resources, e.g., high-performance multicore CPUs. Thereby, this server can handle a large number of tasks offloaded from different MDs in parallel. In other words, once the server receives the partial input data of a task, we assume the data can be executed immediately, and thus no queuing delay incurs. As MDs have limited batteries while the rest of the system has a continuous and sufficient power supply, this article only considers MDs' energy consumption.

D. Communication Model

Computation offloading involves the delivery of task data and its result between MDs and the MEC server. To be specific, offloading the partial input data of a task to the server incurs not only transmission delay from the corresponding

MD to the server but also energy consumption of transmitting the data from MD. Note that we ignore the transmission delay between the BS and the MEC server since the wirelines between them ensure much faster data delivery than wireless channels.

Compared with the partial input data offloaded to the MEC server, its execution result is much smaller. Thus, this article ignores the transmission delay and energy consumption incurred for sending the result back to the corresponding MD. In this article, massive MIMO is used as the air interface technique between MDs and the BS.

Note that we assume the channel state information (CSI) is known in advance, i.e., the perfect CSI scenario. Let $\boldsymbol{G}(t) = [\boldsymbol{g}_1(t), \ldots, \boldsymbol{g}_M(t)]$ denote the $N \times M$ channel matrix between the BS and all MDs, where $\boldsymbol{g}_m(t)$ is the $N \times 1$ channel gain vector of MD m, and N is the number of antennas. Denote by $\boldsymbol{G}^H(t)$ the conjugate transpose of $\boldsymbol{G}(t)$. The zero-forcing detection matrix is written as $\boldsymbol{A}(t) = \boldsymbol{G}^H(t)(\boldsymbol{G}(t)\boldsymbol{G}^H(t))^{-1}$. We obtain the normalized effective channel gain for MD m as $\hat{g}_m(t) = [|\boldsymbol{A}_m(t) \cdot \boldsymbol{g}_m(t)|^2/(|\boldsymbol{A}_m(t)|^2 \cdot \varrho^2)]$, where $\boldsymbol{A}_m(t)$, $\boldsymbol{g}_m(t)$, and ϱ^2 are the mth row of $\boldsymbol{A}(t)$, the mth column of $\boldsymbol{G}(t)$, and the noise power, respectively. Let $P^{\text{tra}}_{m,i}(t)$ and $P^{\max}_m$ denote the uplink transmission power of MD m when sending data $\alpha_{m,i}(t) \cdot d_{m,i}$ to the BS and the maximum transmission power of MD m, where $P^{\text{tra}}_{m,i}(t) \leq P^{\max}_m$. The uplink transmission rate for MD m to offload $\alpha_{m,i}(t) \cdot d_{m,i}$, $\upsilon^{\text{tra}}_{m,i}(t)$, is defined in [28]

$$\upsilon^{\text{tra}}_{m,i}(t) = W \cdot \log_2\left(1 + P^{\text{tra}}_{m,i}(t) \cdot \hat{g}_m(t)\right) \tag{4}$$

where W is the system bandwidth [18].

The transmission delay and energy consumption incurred for offloading the partial input data of $j^t_{m,i}$, $\alpha_{m,i}(t) \cdot d_{m,i}$, from MD m to the MEC server, $\tau^{\text{tra}}_{m,i}(t)$ and $E^{\text{tra}}_{m,i}(t)$, are defined in the following equations:

$$\tau^{\text{tra}}_{m,i}(t) = \frac{\alpha_{m,i}(t) \cdot d_{m,i}}{\upsilon^{\text{tra}}_{m,i}(t)} \tag{5}$$

$$E^{\text{tra}}_{m,i}(t) = P^{\text{tra}}_{m,i}(t) \cdot \tau^{\text{tra}}_{m,i}(t). \tag{6}$$

E. Problem Formulation

Assume task $j^t_{m,i}$ arriving in time slot $\pi^{\text{arr}}_{m,i}$ is processed in time slot t. For $j^t_{m,i}$, its task completion time $\tau_{m,i}(t)$ is defined as the time interval between when $j^t_{m,i}$ arrives at MD m and when the processing of $j^t_{m,i}$ is over, as written in

$$\tau_{m,i}(t) = \vartheta \cdot \left(t - \pi^{\text{arr}}_{m,i}\right) + \max\left(\tau^{\text{loc}}_{m,i}(t), \tau^{\text{tra}}_{m,i}(t) + \tau^{\text{ser}}_{m,i}(t)\right) \tag{7}$$

where ϑ is the time duration each time slot spans. $\vartheta \cdot (t - \pi^{\text{arr}}_{m,i})$ is the time $j^t_{m,i}$ spends on queuing before it is processed. $\max(\tau^{\text{loc}}_{m,i}(t), \tau^{\text{tra}}_{m,i}(t) + \tau^{\text{ser}}_{m,i}(t))$ is the task processing delay of $j^t_{m,i}$, where the details of $\tau^{\text{loc}}_{m,i}(t)$, $\tau^{\text{ser}}_{m,i}(t)$, and $\tau^{\text{tra}}_{m,i}(t)$ can be found in (1), (3), and (5).

The total energy consumption of MD m for processing $j^t_{m,i}$, $E_{m,i}(t)$, is written in (8). Obviously, $E_{m,i}(t)$ consists of the energy consumption for processing $j^t_{m,i}$ on and that for transmitting $j^t_{m,i}$ from MD m

$$E_{m,i}(t) = E^{\text{loc}}_{m,i}(t) + E^{\text{tra}}_{m,i}(t). \tag{8}$$

For each $j^t_{m,i}$, we define its JCORA decision-making cost in (9). Actually, $C^t_{m,i}$ is a compromised tradeoff between the task completion time of $j^t_{m,i}$ and energy consumption of MD m, where $\mathcal{E}$ is the weight

$$C^t_{m,i} = \mathcal{E} \cdot \tau_{m,i}(t) + (1 - \mathcal{E}) \cdot E_{m,i}(t). \tag{9}$$

As T is the last time slot elapsed in the MEC system, $q(T)$ represents the number of tasks that have been processed by T. The dynamic JCORA problem concerned in this article is to minimize the average long-term cost of all tasks in the MEC system C_{lterm} as formulated in

$$\min_{\alpha_{m,i}(t), f_{m,i}(t), P^{\text{tra}}_{m,i}(t)} \left(\lim_{T\to\infty} \frac{1}{M} \sum_{m=1}^{M} \left(\frac{1}{q(T)} \sum_{i=1}^{q(T)} C^t_{m,i}\right)\right) \tag{10}$$

s.t.

$$C1\colon \tau_{m,i}(t) \leq \tau^{\max}_{m,i}, m = 1, \ldots, M, i = 1, \ldots, q(T) \tag{10a}$$

$$C2\colon 0 \leq \alpha_{m,i}(t) \leq 1, m = 1, \ldots, M, i = 1, \ldots, q(T) \tag{10b}$$

$$C3\colon 0 \leq f_{m,i}(t) \leq F_m, m = 1, \ldots, M, i = 1, \ldots, q(T) \tag{10c}$$

$$C4\colon 0 \leq P^{\text{tra}}_{m,i}(t) \leq P^{\max}_m, m = 1, \ldots, M, i = 1, \ldots, q(T). \tag{10d}$$

Constraint (10a) defines that for an arbitrary task, its actual task completion time cannot exceed its associated maximum tolerable delay. Constraint (10b) specifies that for each task, its associated offloading ratio is a variable between 0 and 1. Constraint (10c) states that for each MD, the number of CPU cycles per second cannot exceed its associated maximum computing capacity. Constraint (10d) defines that for each MD, the power for transmitting its tasks to the BS cannot be larger than its associated maximum power. Note that, $\alpha_{m,i}(t)$, $f_{m,i}(t)$, and $P^{\text{tra}}_{m,i}(t)$ are the decision variables associated with MD m. This article aims at obtaining promising JCORA solutions by jointly optimizing the three variables above for each MD via our TADPG agent.

IV. Proposed TADPG Agent

Based on DDPG, our TADPG has two significant features, including TFEN and rPER methods. This section introduces the basic elements, overall framework, TFEN, rPER, and training process.

A. Three Basic Elements

Like DDPG [18], a TADPG agent runs on each MD, maintaining three basic elements, including the state, action, and reward. A TADPG agent obtains the current MD state $\boldsymbol{S}_t$ by periodically interacting with the corresponding MD, e.g., MD m. This agent then generates an action $\boldsymbol{a}_t$ based on $\boldsymbol{S}_t$ via a strategy guidance, e.g., deterministic policy. After that, action $\boldsymbol{a}_t$ is carried out in MD m, which results in an immediate reward $r(\boldsymbol{S}_t, \boldsymbol{a}_t)$. By finding an optimal action policy, the goal of the TADPG agent is to maximize the long-term reward R_t over a decision episode. The state space, action space, and reward function are introduced as follows.

State: The state of MD m in time slot t $\boldsymbol{S}_t$ contains seven parameters, including four task-related attributes and three environment-related parameters. The four attributes include the

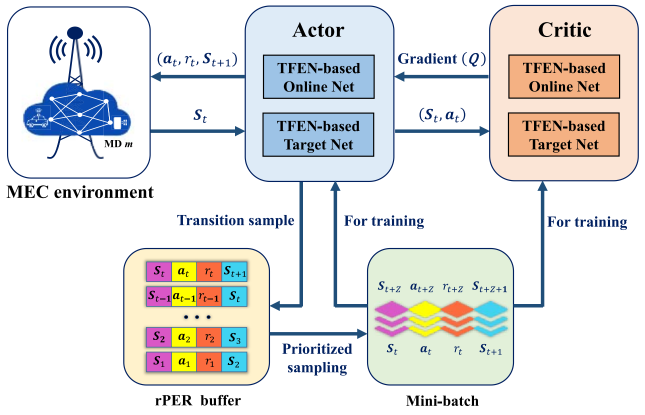

Fig. 2. Structure of the TADPG agent.

task arrival time slot $\pi_{m,i}^{\text{arr}}$, the input data size $d_{m,i}$, the maximum tolerable delay $\tau_{m,i}^{\max}$, and the number of CPU cycles needed to process per input data bit of $j_{m,i}^t$, $c_{m,i}$. The three parameters are the maximum computing capacity F_m, the uplink transmission power $P_m^{\max}$, and the uplink channel gain vector between MD m and the BS $\boldsymbol{g}_m(t)$. In other words, we have $\boldsymbol{S}_t = \{\pi_{m,i}^{\text{arr}}, d_{m,i}, \tau_{m,i}^{\max}, c_{m,i}, F_m, P_m^{\max}, \boldsymbol{g}_m(t)\}$.

Action: As aforementioned, an action is generated based on the current MD state. Let $\boldsymbol{a}_t = \{\alpha_{m,i}(t), f_{m,i}(t), P_{m,i}^{\text{tra}}(t)\}$ be the action for MD m to take in time slot t, where $\alpha_{m,i}(t)$ is the offloading ratio of task $j_{m,i}^t$, $f_{m,i}(t)$ is the number of CPU cycles per second for $j_{m,i}^t$ on MD m, and $P_{m,i}^{\text{tra}}(t)$ is the uplink transmission power of MD m. Clearly, $\boldsymbol{a}_t$ is an explicit policy for the JCORA problem associated with MD m. The action space of $\boldsymbol{a}_t$ consists of all possible actions for MD m, which is continuous.

Reward: The long-term reward value of MD m, R_t, is defined in

$$R_t = \sum_{t=1}^{T} \gamma^t \cdot r(\boldsymbol{S}_t, \boldsymbol{a}_t) \tag{11}$$

where the discount factor $0 \le \gamma \le 1$ indicates the importance of each immediate reward $r(\boldsymbol{S}_t, \boldsymbol{a}_t)$ obtained over time. $r(\boldsymbol{S}_t, \boldsymbol{a}_t)$ is set to $C_0 - C_{m,i}^t$, where $C_{m,i}^t$ is the cost of $j_{m,i}^t$ in (9) and C_0 is a constant greater than the theoretical upper bound of $C_{m,i}^t$. This makes sure a better cost results into a larger $r(\boldsymbol{S}_t, \boldsymbol{a}_t)$. The goal of our TADPG agent is to maximize R_t over a long time span.

B. Framework Outline

The structure of the TADPG agent is shown in Fig. 2. This agent is a variant of DDPG, with two performance-enhancing features, including TFEN and rPER.

Both the actor and critic networks contain an online net and a target net, respectively, whose architectures are exactly the same. Instead of FCNs, we use the proposed TFEN for feature extraction and representation learning. To be specific, identical TFEN is adopted as the online and target nets in both the actor and critic networks. The actor network is responsible for calculating the continuous actions based on the observed MD states. Besides, the critic network evaluates the deterministic policy according to the action-value function $Q(\boldsymbol{S}_t, \boldsymbol{a}_t)$, which is used to update the policy parameters.

The rPER buffer stores historical experiences with different importance, which is used to train the TFEN-based nets. The mini-batch consists of Z samples selected from the buffer, where Z is a positive integer. The details of TFEN and rPER are described as follows.

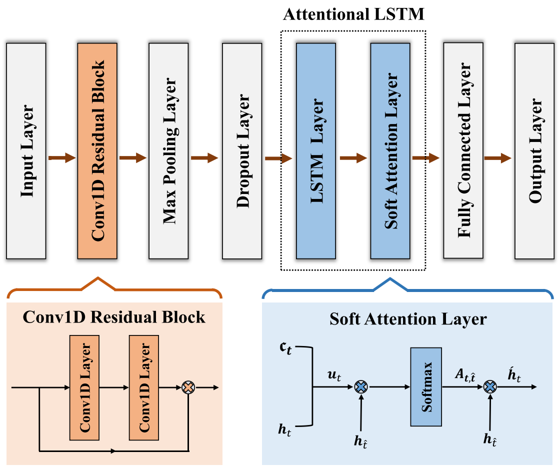

Fig. 3. Structure of TFEN.

C. TFEN

In the traditional DDPG, FCNs are usually used as the feature extraction networks in both the actor and critic networks [18]. However, FCNs are often criticized for their huge numbers of trainable weights, resulting in high computational complexity. Also, they can only capture global discriminative task features, with the temporal variation of task sequences ignored, which does not help to analyze and process sequential data. The computation-intensive tasks, however, are complex temporal data in nature.

To overcome the drawbacks above, this article proposes TFEN as the temporal feature extraction network. TFEN is composed of a Conv1D residual block and an attentional LSTM network. The first part stacks Conv1D networks using a residual structure, responsible for consistently learning the local features of input data while alleviating the declining-accuracy phenomenon in the training process. The second part combines the LSTM network and soft attention mechanism, which is used to capture long-distance sequential patterns and pay attention to mining the important reasoning information hidden in these patterns. Hence, TFEN can achieve promising performance in state representation and function approximation when handling sequential data.

The structure of TFEN is illustrated in Fig. 3. Let $\boldsymbol{X} = [\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_T]$ denote the set of feature maps, where T is the last time slot elapsed in the system. $\boldsymbol{X}_t \in \boldsymbol{X}$ is the input vector of the TFEN network in time slot t. The main layers of TFEN are introduced one by one.

1) Conv1D Residual Block: In general, the depth of a neural network has a significant impact on prediction and classification accuracy. Nevertheless, deeper networks are more likely to lead to model degradation [29]. A common way to overcome the weakness above is to combine convolution layers in a residual manner. Inspired by ResNet [30], we design a Conv1D residual block with two Conv1D layers, aiming at learning the correlations among local features of each $\boldsymbol{X}_t$.

Because of the sparse connections with shared weights, the two Conv1D layers significantly reduce the number of trainable parameters, compared with FCNs. Besides, there is a shortcut connection bypassing the two Conv1D layers to obtain the linear superposition of each $\boldsymbol{X}_t$ itself and its nonlinear transformation, helping to facilitate the backpropagation of gradients and fit the expected function better than FCNs.

Let $\boldsymbol{X}_{\text{Conv1D}} = \sigma(\boldsymbol{w}_1\boldsymbol{X}_t)$ be the output after $\boldsymbol{X}_t$ passes the two Conv1D layers, where σ is the ReLU activation function, and the biases of $\boldsymbol{X}_t$ are omitted for simplifying notations. The residual operation $\boldsymbol{X}_{\text{Conv1D}} + \boldsymbol{X}_t$ is performed via elementwise addition, where ReLU is used again. Its output $\boldsymbol{O}_t$ is a residual mapping defined in (12), where $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are the two learnable weight matrices

$$\boldsymbol{O}_t = \sigma(\boldsymbol{w}_2 \cdot \sigma(\boldsymbol{w}_1 \cdot \boldsymbol{X}_t) + \boldsymbol{X}_t). \tag{12}$$

2) LSTM Layer: Due to the Conv1D layer, the Conv1D residual block easily leads to a dilemma that when $\boldsymbol{X}_t$ is being processed, this block cannot make use of the relevant temporal information captured earlier from $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{t-1}$. In other words, the temporal dependencies over long and short periods cannot be well explored and utilized.

As a recurrent neural network, LSTM introduces a memory cell to learn long short-term temporal dependencies [31]. It discovers those underlying representations hidden in temporal shapelets. That is why an LSTM layer is used in TFEN to mine the long short-term changes of the input features from the Conv1D residual block.

As aforementioned, the attentional LSTM network combines LSTM and attention mechanism. It is realized by cascading an LSTM layer and a soft attention layer in this article. In the LSTM layer, the input vector $\boldsymbol{O}_t$, the previous hidden state $\boldsymbol{h}_{t-1}$, and the previous cell state of the network $\mathfrak{c}_{t-1}$ are used to update $\mathfrak{c}_t$ and $\boldsymbol{h}_t$, as can be described in

$$(\mathfrak{c}_t, \boldsymbol{h}_t) = \text{LSTM}(\boldsymbol{O}_t, \mathfrak{c}_{t-1}, \boldsymbol{h}_{t-1}). \tag{13}$$

3) Soft Attention Layer: As we know, the traditional LSTM only accumulates long-range temporal representations contained in its hidden states to extract representative features. It is, however, not able to capture meaningful information in $\boldsymbol{X}$ at certain moments. On the other hand, soft attention is an effective weight distribution mechanism. It can automatically focus on the information at certain moments that has a decisive effect on prediction [32]. That is why an attention layer is usually added after an LSTM layer along the temporal direction. This form of combination, also called the attentional LSTM, has been widely applied in various application areas, e.g., action recognition [33], object detection [34], and routing optimization [35], [36]. Therefore, we also add a soft attention layer right after the LSTM layer, automatically emphasizing important temporal relationships of the input at critical moments.

In the soft attention layer, $\boldsymbol{h}_t$ and $\mathfrak{c}_t$ are merged as a set of state summary vectors $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_T]$, where $\boldsymbol{u}_t$ is the feature vector in time slot t. Let the temporal attention probability distribution in t denoted by $A_{t,\hat{t}}$, where $\hat{t}$ is a time slot earlier than t. Then, $A_{t,\hat{t}}$ is computed based on $\boldsymbol{u}_t$ and $\boldsymbol{h}_{\hat{t}}$, as defined in

$$A_{t,\hat{t}} = \frac{\exp(\boldsymbol{u}_t^T \cdot \boldsymbol{h}_{\hat{t}})}{\sum_{\hat{t}=1}^{T} \exp(\boldsymbol{u}_t^T \cdot \boldsymbol{h}_{\hat{t}})} \tag{14}$$

where $\boldsymbol{u}_t^T$ is the transpose of $\boldsymbol{u}_t$. $A_{t,\hat{t}}$ indicates to what extent the input in time slot $\hat{t}$ contributes to the output in time slot t. Note that $\acute{\boldsymbol{h}}_t$ is the output hidden state, updated by

$$\acute{\boldsymbol{h}}_t = \sum_{\hat{t}=1}^{T} A_{t,\hat{t}} \cdot \boldsymbol{h}_{\hat{t}}. \tag{15}$$

D. rPER

In the traditional DDPG, uniform sampling is widely adopted to randomly select a mini-batch of transition samples from the experience replay buffer to train the actor and critic networks in each time slot [18]. However, transition samples are considered equally important, with their significance ignored. That is not helpful for DDPG to rapidly sense the negative consequences of wrong behaviors in the corresponding states and avoid taking the wrong actions in these conditions again. So, researchers introduce PER-based experience replay methods to overcome the difficulty above, where more valuable experiences are replayed at higher probabilities [37], [38]. To be specific, each experience is associated with a priority value. The replay probability is calculated based on all the priority values in the replay buffer. In PER, those successful attempts and painful lessons are more likely to be selected to the mini-batch than others, which helps to shorten the learning time and increase training stability.

How to define the priority value is a crucial issue when studying PER techniques. The absolute temporal-difference (TD) error is commonly used as the indicator to evaluate the experiences [37], [39]. Experiences with large positive TD errors are regarded as successful attempts, while those with large negative TD errors are viewed as failed attempts. In particular, failed attempts are also important experiences, helping to prevent an agent from frequently taking bad actions. Hence, both successful or failed attempts are assigned larger priority values than others.

Most studies on PER only consider one indicator, namely, the absolute TD error, which is used to update the trainable weights of the critic network. Nevertheless, the actor network is also critical. Ignoring it may decrease the decision-making accuracy. Besides, the commonly defined priority value is proportional to the absolute TD error, which may lead to dramatic fluctuations when updating trainable weights of the critic network.

To overcome the drawback above and improve the robustness of TADPG, we propose an rPER method. Different from the existing PER methods, rPER has two features. The first one is to consider two indicators for experience evaluation, including the TD error δ_t and the deterministic strategy gradient $\nabla_{\boldsymbol{\theta}^\mu}\boldsymbol{J}_t$. Indicator $\nabla_{\boldsymbol{\theta}^\mu}\boldsymbol{J}_t$ is related to the training of the actor network [40], where $\boldsymbol{\theta}^\mu$ is the weight vector of the actor network. The second feature is the introduction of "rank," where all transition samples are classified into different ranks, according to which their priority values are calculated. To be specific, in the rPER buffer, all transition samples are evaluated based on δ_t and $\nabla_{\boldsymbol{\theta}^\mu}\boldsymbol{J}_t$, and then sorted in descending order according to their scores. Samples with higher scores are assigned lower rank numbers. Each sample is assigned a priority value according to its rank number. The following describes the details of the key features.

Let score_t^φ be the score of an arbitrary transition sample φ in the rPER buffer in time slot t, which is a weighted sum of the two indicators above, as defined in

$$\text{score}_t^\varphi = \varpi \cdot \left|\delta_t^\varphi\right| + (1-\varpi)\cdot \overline{\left|\nabla_{\boldsymbol{\theta}^\mu}\boldsymbol{J}_t^\varphi\right|} \tag{16}$$

where $|\delta_t^\varphi|$ and $\overline{|\nabla_{\boldsymbol{\theta}^\mu}\boldsymbol{J}_t^\varphi|}$ are the absolute TD error and the average absolute deterministic strategy gradient, respectively. Besides, $0 \le \varpi \le 1$ is constant reflecting the relative importance of $|\delta_t^\varphi|$ against $\overline{|\nabla_{\boldsymbol{\theta}^\mu}\boldsymbol{J}_t^\varphi|}$.

All the transition samples are sorted in descending order according to their scores and then divided into $\mathcal{N}$ ranks, according to a set of predefined lower bound score values of ranks, $\{\text{Score}_{rk,1}, \text{Score}_{rk,2}, \ldots, \text{Score}_{rk,\mathcal{N}-1}\}$. For example, a transition sample with a score value larger than $\text{Score}_{rk,1}$ belongs to rank 1 while one with a score value smaller than $\text{Score}_{rk,\mathcal{N}-1}$ belongs to rank $\mathcal{N}$. In this way, transition samples with higher scores belong to lower ranks. Let rank_t^φ and value_t^φ be the rank number and priority value of transition sample φ in time slot t, respectively. We have $\text{value}_t^\varphi = 1/\text{rank}_t^\varphi$ for each φ. The replay probability of φ, p_t^φ, is expressed in

$$p_t^\varphi = \frac{\left(\text{value}_t^\varphi\right)^\ell}{\sum_{\varphi=1}^{B}\left(\text{value}_t^\varphi\right)^\ell} \tag{17}$$

where ℓ is a constant controlling the prioritization contribution, and B is the actual number of transition samples in the rPER buffer no larger than the maximum buffer size β.

This article introduces a compensation weight w_t^φ in (18), to correct the bias caused by rPER, where $\zeta \ge 1$ is an annealing variable [37]

$$w_t^\varphi = \frac{1}{\left(B\cdot p_t^\varphi\right)^\zeta}. \tag{18}$$

Given a transition sample φ, the accumulated weight change of the critic network, $\Delta_{\boldsymbol{\theta}^Q}$, and that of the actor network, $\Delta_{\boldsymbol{\theta}^\mu}$, are updated by using the following equations [41]:

$$\Delta_{\boldsymbol{\theta}^Q} = \Delta_{\boldsymbol{\theta}^Q} + w_t^\varphi\cdot\delta_t^\varphi\cdot\nabla_{\boldsymbol{\theta}^Q}Q\left(\boldsymbol{S}_t^\varphi, \boldsymbol{a}_t^\varphi\right) \tag{19}$$

$$\Delta_{\boldsymbol{\theta}^\mu} = \Delta_{\boldsymbol{\theta}^\mu} + w_t^\varphi\cdot\nabla_{\boldsymbol{\theta}^\mu}\boldsymbol{J}_t^\varphi \tag{20}$$

where $\nabla_{\boldsymbol{\theta}^Q}Q(\boldsymbol{S}_t^\varphi, \boldsymbol{a}_t^\varphi)$ is the Q gradient used to update the critic network, and $\boldsymbol{\theta}^Q$ is the weight vector of the critic network.

E. Training Process

As aforementioned, there are M TADPG agents in the MEC system, each running on an MD. By interacting with a local MD, each agent makes independent JCORA decisions that are taken actions in that MD. Without loss of generality, we introduce the training process of the TADPG agent on MD m, $m = 1, \ldots, M$, in Algorithm 1. For the critic and actor networks, their target nets, $Q'(\boldsymbol{S}, \boldsymbol{a}|\boldsymbol{\theta}^Q)$ and $\mu'(\boldsymbol{s}|\boldsymbol{\theta}^\mu)$, are clone of their online nets, respectively (see step 2). Steps 10–20 show the details related to the rPER method. Steps 21 and 22 update the weight vectors of the TFEN-based online and target nets in both the critic and actor networks with update rate η, respectively. The training stops once a predefined number of episodes $K_{\max}$ are completed.

Algorithm 1 Training Process of the TADPG Agent on MD m

```
1: Initialize the TFEN-based online nets in the critic and actor networks, i.e.
   $Q(\boldsymbol{S},\boldsymbol{a}|\boldsymbol{\theta}^Q)$ and $\mu(\boldsymbol{s}|\boldsymbol{\theta}^\mu)$, with random weights $\boldsymbol{\theta}^Q$ and $\boldsymbol{\theta}^\mu$, respectively;
2: Initialize the TFEN-based target nets in the critic and actor networks, i.e.
   $Q'(\boldsymbol{S},\boldsymbol{a}|\boldsymbol{\theta}^Q)$ and $\mu'(\boldsymbol{s}|\boldsymbol{\theta}^\mu)$, with $\boldsymbol{\theta}^{Q'} = \boldsymbol{\theta}^Q$ and $\boldsymbol{\theta}^{\mu'} = \boldsymbol{\theta}^\mu$, respectively;
3: Set the rPER buffer $\boldsymbol{Buf} = \emptyset$;
4: for episode = 1 to $K_{\max}$ do
5:     for $t = 1$ to $T$ do
6:         Obtain state $\boldsymbol{S}_t$ from MD $m$;
7:         Generate action $\boldsymbol{a}_t$ by the actor network and carry out it in MD $m$;
8:         Obtain immediate reward $r_t$ and state $\boldsymbol{S}_{t+1}$ from MD $m$;
9:         Form transition sample $\varphi_t = (\boldsymbol{S}_t, \boldsymbol{a}_t, r_t, \boldsymbol{S}_{t+1})$;
10:        Calculate $score_t^\varphi$ according to Eq. (16);
11:        if the rPER buffer is full then do
12:            Find the least replayed sample in $\boldsymbol{Buf}$, e.g. $\varphi'$;
13:            Remove $\varphi'$ from $\boldsymbol{Buf}$;
14:        Place $\varphi_t$ in the right position in $\boldsymbol{Buf}$ according to $score_t^\varphi$;
15:        Assign a rank value, $rank_t^\varphi$, to $\varphi_t$ and calculate its $p_t^\varphi$.
           // Mini-batch sample selection and weigh change update
16:        for $j = 1$ to $Z$ do
17:            Probabilistically select a sample from $\boldsymbol{Buf}$ as the $j$-th sample
               in the mini-batch, i.e., $\varphi(j)$;
18:            Calculate $w_t^{\varphi(j)}$, according to Eq. (18);
19:            Update $\Delta_{\boldsymbol{\theta}^Q}$ and $\Delta_{\boldsymbol{\theta}^\mu}$ according to Eqs. (19) and (20);
20:            Update the replay probability of $\varphi(j)$ according to Eq. (17);
21:        Update the weight vectors of the TFEN-based online nets in the
           critic and actor networks, respectively:
           Set $\boldsymbol{\theta}^Q = \boldsymbol{\theta}^Q + \boldsymbol{\theta}^Q \cdot \Delta_{\boldsymbol{\theta}^Q}$, and then reset $\Delta_{\boldsymbol{\theta}^Q} = 0$;
           Set $\boldsymbol{\theta}^\mu = \boldsymbol{\theta}^\mu + \boldsymbol{\theta}^\mu \cdot \Delta_{\boldsymbol{\theta}^\mu}$, and then reset $\Delta_{\boldsymbol{\theta}^\mu} = 0$;
22:        Update the weight vectors of the TFEN-based target nets in the
           critic and actor networks, respectively:
           Set $\boldsymbol{\theta}^{Q'} = \eta\boldsymbol{\theta}^Q + (1-\eta)\boldsymbol{\theta}^{Q'}$ and $\boldsymbol{\theta}^{\mu'} = \eta\boldsymbol{\theta}^\mu + (1-\eta)\boldsymbol{\theta}^{\mu'}$.
```

V. Optimal JCORA Policy

The dynamic JCORA problem concerned in this article is to minimize the average long-term cost of all tasks in the MEC system, C_{lterm}, as formulated in (10). If each $C_{m,i}^t$, $i = 1, \ldots, q(T)$, $m = 1, \ldots, M$, $t = 1, \ldots, T$, in (10) is optimal, the objective C_{lterm} is optimal. So, the dynamic problem above can be converted to a series of static optimization subproblems. Each subproblem optimizes $C_{m,i}^t$ for a specific task $j_{m,i}^t$. We regard the JCORA decision making for each task, e.g., $j_{m,i}^t$, $i = 1, \ldots, q(T)$, $m = 1, \ldots, M$, $t = 1, \ldots, T$, as an independent static optimization subproblem.

For the static optimization subproblem of task $j_{m,i}^t$, the objective function is defined in

$$\min_{\alpha_{m,i}(t), f_{m,i}(t), P_{m,i}^{\text{tra}}(t)} C_{m,i}^t = \mathcal{E}\cdot\tau_{m,i}(t) + (1-\mathcal{E})\cdot E_{m,i}(t) \tag{21}$$

s.t.

$$C1\colon \tau_{m,i}(t) \le \tau_{m,i}^{\max}, m = 1, \ldots, M, i = 1, \ldots, q(T) \tag{21a}$$

$$C2\colon 0 \le \alpha_{m,i}(t) \le 1, m = 1, \ldots, M, i = 1, \ldots, q(T) \tag{21b}$$

$$C3\colon 0 \le f_{m,i}(t) \le F_m, m = 1, \ldots, M, i = 1, \ldots, q(T) \tag{21c}$$

$$C4\colon 0 \le P_{m,i}^{\text{tra}}(t) \le P_m^{\max}, m = 1, \ldots, M, i = 1, \ldots, q(T). \tag{21d}$$

Lemma 1: When

$$\frac{(1-\mathcal{E})\cdot P^{\rm tra}_{m,i}(t)}{\mathcal{E}\cdot\frac{c_{m,i}}{f_{m,i}(t)}+(1-\mathcal{E})\cdot\kappa\cdot\left(f_{m,i}(t)\right)^2\cdot c_{m,i}} < \upsilon^{\rm tra}_{m,i}(t) < \frac{\mathcal{E}+(1-\mathcal{E})\cdot P^{\rm tra}_{m,i}(t)}{-\mathcal{E}\cdot\frac{c_{m,i}}{f_{\rm ser}}+(1-\mathcal{E})\cdot\kappa\cdot\left(f_{m,i}(t)\right)^2\cdot c_{m,i}}$$

MD m tends to offload partial data of task $j^t_{m,i}$ to the server to reduce $\tau_{m,i}(t)$. When $\tau^{\rm loc}_{m,i}(t)$ is equal to $\tau^{\rm tra}_{m,i}(t)+\tau^{\rm ser}_{m,i}(t)$, we obtain the minimum task completion time of $j^t_{m,i}$. The optimal JCORA policy $(\alpha^*_{m,i}(t), f^*_{m,i}(t), P^{\rm tra^*}_{m,i}(t))$ for $j^t_{m,i}$ is defined in

$$\begin{aligned}
\alpha^*_{m,i}(t) &= \frac{c_{m,i}}{\frac{f_{m,i}(t)}{\upsilon^{\rm tra}_{m,i}(t)}+\frac{f_{m,i}(t)\cdot c_{m,i}}{f_{\rm ser}}+c_{m,i}}\\
f^*_{m,i}(t) &= \min\left\{\sqrt[3]{\frac{\mathcal{E}}{2(1-\mathcal{E})\cdot\kappa}}, F_m\right\}\\
P^{\rm tra^*}_{m,i}(t) &= \min\left\{\frac{e^{\mathbb{W}\left(\left(\frac{\mathcal{E}\cdot\hat{g}_m(t)}{1-\mathcal{E}}-1\right)\cdot\frac{1}{e}\right)+1}-1}{\hat{g}_m(t)}, P^{\max}_m\right\}.
\end{aligned}\tag{22}$$

Proof: Refer to the Appendix. ■

Lemma 2: When

$$0<\upsilon^{\rm tra}_{m,i}(t)<\frac{(1-\mathcal{E})\cdot P^{\rm tra}_{m,i}(t)}{\mathcal{E}\cdot\frac{c_{m,i}}{f_{m,i}(t)}+(1-\mathcal{E})\cdot\kappa\cdot\left(f_{m,i}(t)\right)^2\cdot c_{m,i}}$$

MD m tends to process task $j^t_{m,i}$ locally. The optimal JCORA policy for $j^t_{m,i}$ is written as

$$\begin{aligned}
\alpha^*_{m,i}(t) &= 0\\
f^*_{m,i}(t) &= \min\left\{\sqrt[3]{\frac{\mathcal{E}}{2(1-\mathcal{E})\cdot\kappa}}, F_m\right\}.
\end{aligned}\tag{23}$$

Proof: Similar to Lemma 1, when

$$0<\upsilon^{\rm tra}_{m,i}(t)<\frac{(1-\mathcal{E})\cdot P^{\rm tra}_{m,i}(t)}{\mathcal{E}\cdot\frac{c_{m,i}}{f_{m,i}(t)}+(1-\mathcal{E})\cdot\kappa\cdot\left(f_{m,i}(t)\right)^2\cdot c_{m,i}}$$

we have $[\partial C^t_{m,i}/\partial\alpha_{m,i}(t)]>0$. Equation (21) is an increasing function for $\alpha_{m,i}(t)$. Thus, $C^t_{m,i}$ is minimal at $\alpha^*_{m,i}(t)=0$. ■

Lemma 3: When

$$\upsilon^{\rm tra}_{m,i}(t)>\frac{\mathcal{E}+(1-\mathcal{E})\cdot P^{\rm tra}_{m,i}(t)}{-\mathcal{E}\cdot\frac{c_{m,i}}{f_{\rm ser}}+(1-\mathcal{E})\cdot\kappa\cdot\left(f_{m,i}(t)\right)^2\cdot c_{m,i}}$$

MD m tends to execute task $j^t_{m,i}$ remotely. The optimal JCORA policy for $P^{\rm tra}_{m,i}(t)$ is written as

$$\begin{aligned}
\alpha^*_{m,i}(t) &= 1\\
P^{\rm tra^*}_{m,i}(t) &= \min\left\{\frac{e^{\mathbb{W}\left(\left(\frac{\mathcal{E}\cdot\hat{g}_m(t)}{1-\mathcal{E}}-1\right)\cdot\frac{1}{e}\right)+1}-1}{\hat{g}_m(t)}, P^{\max}_m\right\}.
\end{aligned}\tag{24}$$

Proof: Similar to Lemma 1, when

$$\upsilon^{\rm tra}_{m,i}(t)>\frac{\mathcal{E}+(1-\mathcal{E})\cdot P^{\rm tra}_{m,i}(t)}{-\mathcal{E}\cdot\frac{c_{m,i}}{f_{\rm ser}}+(1-\mathcal{E})\cdot\kappa\cdot\left(f_{m,i}(t)\right)^2\cdot c_{m,i}}$$

we have $[\partial C^t_{m,i}/\partial\alpha_{m,i}(t)]>0$. Equation (21) is a decreasing function for $\alpha_{m,i}(t)$. Hence, $C^t_{m,i}$ is minimal at $\alpha^*_{m,i}(t)=1$. ■

Combining the optimal policies for all tasks results in an optimal solution to the dynamic JCORA problem. The resulting objective value can be used as a lower bound for evaluating the performance of the proposed TADPG agent.

VI. Simulation Results

This section evaluates the performance of the TADPG agent. First, the experimental setup is described. Then, the convergence and effectiveness of TADPG are analyzed. Finally, we compare TADPG with eight state-of-the-art DRL methods in terms of the long-term cost, task completion time, and energy consumption of MD.

A. Experimental Setup

Parameter Settings of MEC: In our MEC system, a BS with multiantennas is located in the center, and a number of MDs are randomly scattered around the BS with a radius of 100 m, i.e., ${\rm rad}_m = 100$ m. The number of MDs ranges from 1 to 40. The number of antennas N is set to 128.

For MD m, $m=1,2,\ldots,M$, the channel gain vector is initialized as $\boldsymbol{g}_m(0)\sim\mathcal{CN}(0, g_0\cdot({\rm rad}_0/{\rm rad}_m)^e\cdot\boldsymbol{I}_{N\times1})$, where the path-loss constant g_0, the reference distance ${\rm rad}_0$, the distance between BS and MD m ${\rm rad}_m$, and the path-loss exponent e are set to -30 dB, 1 m, 100 m, and 3 [18], respectively. $\boldsymbol{I}_{N\times1}$ denotes an $N\times1$ identity matrix. Let ρ and $\boldsymbol{e}_m(t)$ be the normalized channel correlation coefficient between time slots $t-1$ and t and the error vector in t, respectively. In order to characterize the temporal correlation of the CSIs between two consecutive time slots for each MD m, e.g., $t-1$ and t, we update $\boldsymbol{g}_m(t)$ according to the Gaussian Markov block fading autoregressive model [18], as defined in

$$\boldsymbol{g}_m(t)=\rho\cdot\boldsymbol{g}_m(t-1)+\sqrt{1-\rho}\cdot\boldsymbol{e}_m(t)\tag{25}$$

where ρ is set to 0.95, and $\boldsymbol{e}_m(t)$ follows a complex Gaussian distribution $\mathcal{CN}(0, g_0\cdot({\rm rad}_0/{\rm rad}_m)^e\cdot\boldsymbol{I}_{N\times1})$.

For an arbitrary task $j^t_{m,i}$ of MD m, $i=1,\ldots,q(T)$, the mean data arriving rate λ_m varies from 1 to 7 Mb/s in different scenarios; the number of CPU cycles needed to process per input data bit $c_{m,i}$ is randomly generated and its values are uniformly distributed in the range of [200, 500] cycles/bit; the maximum tolerable delay of completing $j^t_{m,i}$, $\tau^{\max}_{m,i}$, is randomly generated and its values are uniformly distributed in the range of [20, 50] ms. Besides, we set time slot duration $\vartheta = 10$ ms, the maximum computing capacity $F_m = 1.26$ GHz, the number of CPU cycles per second on the MEC server $f_s = 41.8$ GHz [42], the system bandwidth $W = 10$ MHz [19], the noise power spectral density $\varrho^2 = -174$ dBm/Hz [25], and the maximum transmission power $P^{\max}_m = 2$ W [19], respectively. Besides, we set the tradeoff weight $\mathcal{E} = 0.5$ in (9), which means the task completion time and the energy consumption of MD are equally important.

Hyperparameter Settings of TFEN: TFEN consists of a Conv1D residual block, a max-pooling layer, a dropout layer, an LSTM layer, a soft attention layer, a fully connected layer, and an output layer. The residual block contains two Conv1D layers and each layer owns 32 kernels. For each kernel, its

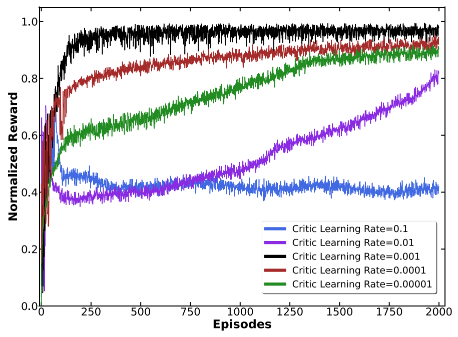

Fig. 4. Normalized reward values obtained versus critic network learning rate.

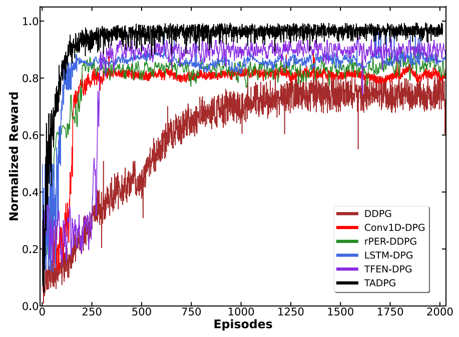

Fig. 5. Normalized reward values obtained during training.

kernel and stride sizes are set to 3 and 1, respectively. In the max-pooling layer, the pooling kernel and stride sizes are set to 3 and 1, respectively. The dropout operation is used as a regularization function with a dropout rate of 0.5. Both the LSTM and output layers have 64 hidden units. The TFEN-based nets in both the actor and critic networks are implemented via TensorFlow.

Hyperparameter Settings of Training: The learning rates of the actor and critic networks are initialized as 0.0001 and 0.001, respectively. We set the maximum rPER buffer size $\beta = 2.5 \times 10^5$, the mini-batch size $Z = 32$, the target net update rate $\eta = 0.001$, and the discount factor $\gamma = 0.99$, respectively. The Adam optimizer is used to optimize the loss function during training.

We run all experiments on a workstation with Intel Xeon E5-2667V4 8Core CPU×2 @3.2 GHz, 128-GB RAM, and 4×NVIDIA GTX Titan V 12G GPU. All agents are implemented using Python 3.6. For training, the predefined number of episodes is set to 2000, and each episode contains 1000 time slots. It takes a TADPG agent around 100 s to run an episode on average.

B. Convergence and Effectiveness

1) Convergence With Different Critic Network Learning Rates: For simplicity purpose, we define the normalized reward R_t^{nor} in (26), where R_t, $R_t^{\min}$, and $R_t^{\max}$ denote the current, minimum, and maximum rewards during the training process, respectively

$$R_t^{\text{nor}} = \frac{R_t - R_t^{\min}}{R_t^{\max} - R_t^{\min}}. \tag{26}$$

We study the impact of the learning rate of the critic network on convergence performance, shown in Fig. 4. When the critic learning rate increases from 0.00001 to 0.001, a higher learning rate results in faster convergence. When the critic learning rate reaches 0.01 and above, the agent cannot well converge because a too large learning rate leads to our agent easily getting trapped into local optima. It is the rationale behind that the critic learning rate is set to 0.001.

2) Effectiveness of the Two Features: To evaluate the effectiveness of the two new features in the proposed agent, namely, TFEN in Section IV-C and rPER in Section IV-D, we compare the following six agents.

1) *DDPG:* The existing DDPG for addressing the dynamic JCORA problem [18]. An architecture with two-layer FCNs is used in the actor and critic networks. Uniform sampling-based experience replay is adopted.
2) *Conv1D-DPG:* DDPG with Conv1D residual blocks. inspired by [43], the FCNs networks in the actor and critic networks are replaced by identical Conv1D residual blocks introduced in Section IV-C.
3) *LSTM-DPG:* DDPG with LSTM networks. Note that the FCNs networks in the actor and critic networks are replaced by identical one-layer LSTM network.
4) *TFEN-DPG:* DDPG with TFEN. Note that we only replace the FCNs networks with TFEN-based networks in the actor and critic networks.
5) *rPER-DDPG:* DDPG with rPER. Note that we only replace the uniform sampling-based experience replay with rPER.
6) *TADPG:* The proposed agent in this article.

In this experiment, we consider a MEC system with only one MD, which helps to get an intuitive sense of the performance of the six agents. The mean data arriving rate λ_m is set to 2 Mb/s. In both DDPG and rPER-DDPG, the numbers of neurons in the two hidden layers of FCNs are set to 400 and 300, respectively. In the Conv1D-DPG, LSTM-DPG, TFEN-DPG, and TADPG, the hyperparameters for the DNNs networks in the actor and critic networks are shown in Section VI-A. For the six agents, the rest of the parameter settings, such as those related to state, action, and reward, are exactly the same.

Also, we define the normalized training loss, which is obtained based on the current, minimum, and maximum losses, similar to (26).

Fig. 5 shows the normalized reward curves of the six agents during training. In the beginning, the six curves grow up quickly as each agent consistently interacts with its corresponding environment, indicating the six DDPG variants can acquire efficient JCORA policies without any prior environment knowledge. When the number of training episodes is large enough, e.g., 1000, the trends of the six curves become stable. A similar phenomenon can be found in Fig. 6 that illustrates the normalized training loss curves of the six agents during training. After around 1250 episodes, the normalized training loss of each agent gradually converges. The curves above, to some extent, indicate that DDPG-based

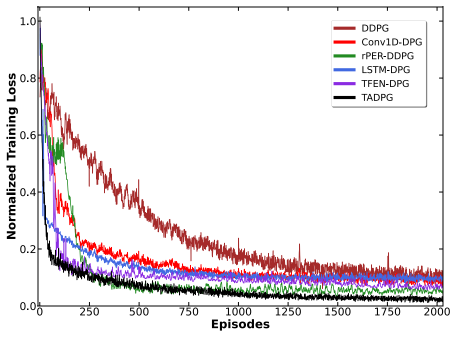

Fig. 6. Normalized training loss values obtained during training.

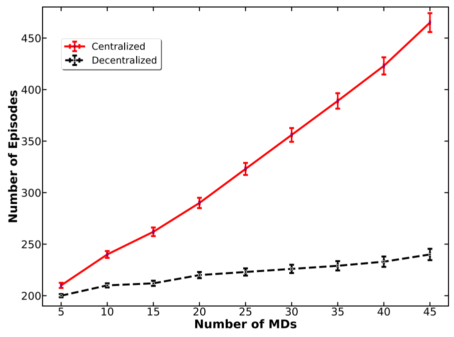

Fig. 7. Episodes needed by the decentralized and centralized JCORA mechanisms.

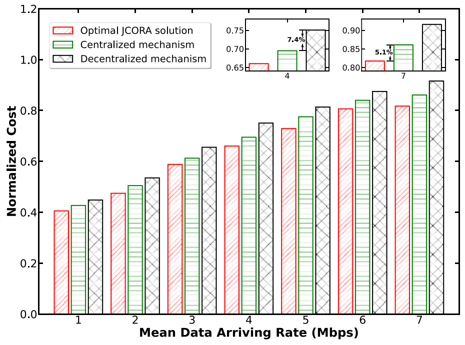

Fig. 8. Normalized cost versus mean data arriving rate.

agents are suitable for addressing the JCORA problem concerned.

Then, we analyze the effectiveness of the six agents. Clearly, these Conv1D-DPG, LSTM-DPG, TFEN-DPG, and rPER-DDPG result in better convergence than the original DDPG in terms of the normalized reward and training loss. The DDPG agent is more likely to be stuck at local optima. That is because the 2-layer FCNs in DDPG are not capable of effectively extracting local shapelets nor long-term dependencies from temporal data, affecting the training stability and effectiveness. In addition, uniform sampling is adopted in the experience replay, which may lead to dramatic fluctuations in the training process. On the other hand, the Conv1D residual block is good at extracting local features from sequential data, while the LSTM network does well in exploring long-range variations among temporal data. That is why Conv1D-DPG and LSTM-DPG perform better than the original DDPG. TFEN can well handle local and long-term temporal feature extraction, which favors efficient training. That is why TFEN-DDPG performs better than the original DDPG, Conv1D-DPG, and LSTM-DPG. Meanwhile, the rPER method makes better use of those valuable experiences than others, helping to stabilize and shorten the training process. That is why the original DDPG is beaten by rPER-DDPG. Moreover, it is no doubt the TADPG agent achieves the best performance with respect to the normalized reward and training loss. With both TFEN and rPER, the proposed agent is enhanced in value function prediction and useful experience exploitation. That is the reason behind its superiority over the other five agents.

C. Decentralized and Centralized JCORA Mechanisms

First, we evaluate the decentralized and centralized JCORA mechanisms based on TADPG in terms of the convergence characteristic. The centralized JCORA gathers the global environment and task information, while the decentralized one requires each MD to make JCORA decisions locally. We plot the results of the number of episodes (needed for an agent to converge) versus the number of MDs in Fig. 7. With the number of MDs increasing, more episodes are needed for both mechanisms to converge. The centralized one has a steeper slope than the decentralized one. The reason behind it is the former is trained with much more environmental and task information, imposing heavier pressure on the system management, especially when more MDs exist. So, the decentralized JCORA is more robust to the problem scale.

Second, we compare the decentralized and centralized TADPG-based JCORA mechanisms with respect to the average cost. The number of MDs is set to 20, and each mechanism is run 100 times. We plot the results of the average normalized long-term cost versus the mean data arriving rate in Fig. 8, showing the performance loss incurred by the decentralized mechanism. Note that the optimal JCORA solutions are obtained according to Section IV. The centralized JCORA mechanism outperforms the decentralized one in each case since the former's cost is closer to the optimal JCORA solution. That is because the centralized JCORA mechanism is trained by using the global environment and task information, where all MDs are well coordinated to achieve decent JCORA performance. The decentralized mechanism makes JCORA decisions locally, lacking the coordination between MDs. Compared with the centralized JCORA mechanism, the decentralized mechanism suffers a performance loss of less than 10%. Yet, the decentralized one still reaches over 90.5% of the optimal JCORA solution on average, demonstrating the effectiveness of the proposed agent.

Third, we compare the decentralized and centralized JCORA mechanisms regarding the average CPU computation time of making a JCORA decision for a task, as shown in Table II. The results, to a certain extent, reflect the computational complexity of the decentralized and centralized versions of our JCORA mechanism with different numbers of MDs. It is seen that the decentralized JCORA mechanism

TABLE II
AVERAGE CPU COMPUTATION TIME (S) OF MAKING A JCORA DECISION FOR A TASK

TADPG-Based JCORA Mechanisms	$M = 10$	$M = 20$	$M = 30$	$M = 40$
Decentralized	$5.56{\times}10^{-4}$	$8.82{\times}10^{-4}$	$9.97{\times}10^{-4}$	$1.13{\times}10^{-3}$
Centralized	$6.73{\times}10^{-4}$	$1.14{\times}10^{-3}$	$1.89{\times}10^{-3}$	$2.76{\times}10^{-3}$

outputs a JCORA decision for an average task around 0.9 ms, which is on average 1.7 times faster than the centralized TADPG. Meanwhile, the decentralized JCORA mechanism is less sensitive to the number of MDs than the centralized JCORA mechanism. That is because the time needed for outputting a JCORA decision depends largely on the action space of an agent. Compared with the decentralized one, the centralized JCORA mechanism has a much larger action space to handle, especially when more MDs exist in the MEC system. In other words, the decentralized JCORA mechanism has lower complexity and thus responds more quickly to each task offloading request. That is why we develop the decentralized JCORA mechanism.

D. Overall Performance Evaluation

To thoroughly study the performance of the TADPG agent, we compare it with eight state-of-the-art DRL-based JCORA agents against the long-term cost, task completion time, and energy consumption of MD.

1) *DQN [13]:* The DQN agent adapted for the dynamic JCORA problem in multiuser MEC environment, where a specially devised binary action space is used.
2) *MRM-DQN [3]:* The DQN agent to address the resource allocation problem in the IoT edge computing system. This agent has multiple replay memories to improve the efficiency of resource utilization.
3) *Deep-SARL [14]:* The double DQN agent with Q-function decomposition proposed for stochastic computation offloading in the virtual MEC system, where a Deep-SARL scheme is used.
4) *IDRQN [15]:* The DQN agent based on the LSTM and candidate networks for task offloading in MEC.
5) *DDPG [18]:* The only DDPG in the literature developed for tackling the dynamic JCORA problem in MEC.
6) *Conv1D-DPG:* DDPG with Conv1D residual blocks used in Section VI-B.
7) *LSTM-DPG:* DDPG with LSTM networks used in Section VI-B.
8) *D^{3}PG [16]:* The double dueling DPG agent applied to the QoE-based computation offloading. The critic network is based on double Q-learning and dueling networks.
9) *TADPG:* The proposed agent in this article.

In the literature, the Q-networks in DQN [13], MRM-DQN [3], and Deep-SARL [14] are based on 1- or 2-layer FCNs, while those in IDRQN [15] are based on 1-layer LSTM networks. To make a fair comparison, we

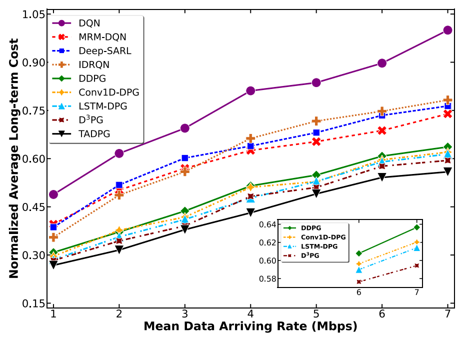

Fig. 9. Results of the average C_{lterm} versus λ_m.

set the Q-networks in DQN, MRM-DQN, and Deep-SARL exactly the same as the 2-layer FCNs used in Section V-B, where the numbers of neurons in the two layers are 400 and 300, respectively. Meanwhile, the Q-networks in IDRQN consist of two 1-layer LSTM networks, each with 64 neurons. In addition, the actor and critic networks in both DDPG [18] and D^{3}PG [16] are all based on 2-layer FCNs. Hence, the same FCNs networks used in DQN, MRM-DQN, and Deep-SARL are also adopted in DDPG and D^{3}PG. Each Conv1D residual block in Conv1D-DPG has two Conv1D layers while each LSTM network in LSTM-DPG is just one layer. On the other hand, the action spaces of the DQN-based agents, namely, DQN, MRM-DQN, Deep-SARL, and IDRQN, are discrete. In reality, a continuous action space is more appropriate to reflect the nature of the JCORA problem. To avoid the curse of high dimensionality, each DQN-based agent adopts a coarse-grained discretization method to quantify its action space. As aforementioned, there are three decision variables associated with the action space, i.e., the task offloading ratio $\alpha_{m,i}(t)$, the number of CPU cycles per second $f_{m,i}(t)$, and the uplink transmission power $P^{\text{tra}}_{m,i}(t)$. We equally divide the range of each decision variable into eight levels. Therefore, each DQN-based agent corresponds to an action space with 8^3 solutions. In the testing stage, the results obtained in 100 runs are averaged.

1) Comparisons Against λ_m: We compare the performance of the nine DRL agents in scenarios with different mean data arriving rate λ_m. To focus on the impact of λ_m, we fix the number of MDs M to 10. The results of the average long-term cost of all tasks in (10), the average completion time of each task, and the average energy consumption of each MD with different λ_m are shown in Figs. 9–11, respectively.

In Fig. 9, with λ_m increasing, the normalized average long-term cost values of all DRL agents gradually grow up. That is because a larger λ_m means more amount of data arrives in each time slot, which usually leads to longer task completion time and heavier energy consumption of MD. Meanwhile, the DDPG-based agents perform better than the DQN-based ones. In particular, the TADPG agent overweighs the others. It is because TFEN is able to mine local correlations and rich long-range temporal changes while rPER frequently replays significant experiences, which helps to ensure the effectiveness and stability of the TADPG. D^{3}PG is the second-best agent. The reason is that D^{3}PG is featured with a modified

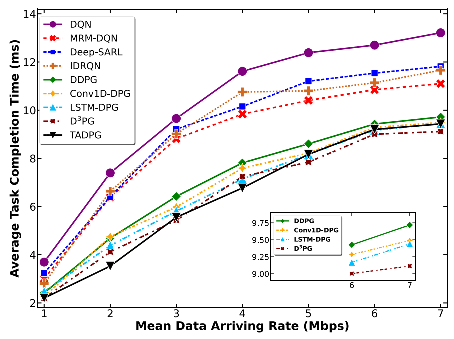

Fig. 10. Results of the average $\tau_{m,i}(t)$ versus λ_m.

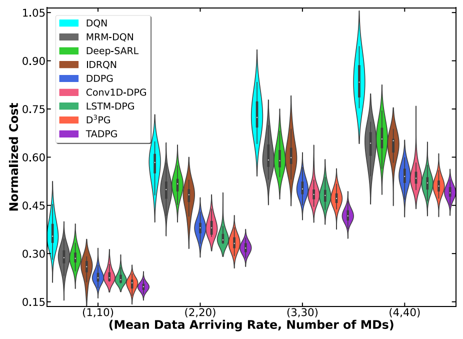

Fig. 12. Results of the average C_{lterm} versus λ_m and M.

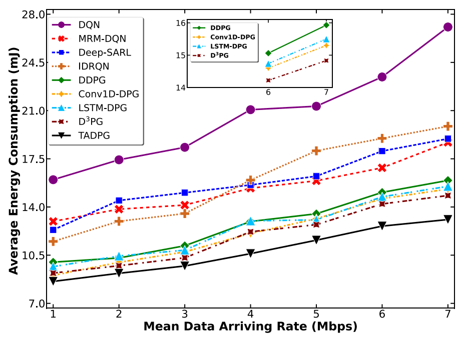

Fig. 11. Results of the average $E_{m,i}(t)$ versus λ_m.

critic network based on the double Q-learning and dueling network, which eliminates the overestimation phenomenon in the original DDPG and thus improves the stability of the training process. Besides, both Conv1D-DPG and LSTM-DPG achieve better normalized cost than the original DDPG. That is because FCNs are designed to capture global discrete discriminative features only. They are not as good as CNN or LSTM when handling temporal data. CNN is capable of extracting important local features from the sequence data via special inherent filters while LSTM performs well in long-term dependency extraction thanks to its memory cell. On the other hand, MRM-DQN, Deep-SARL, and IDRQN have similar performance and they are better than the original DQN in terms of the average long-term cost. The following explains why. In the original DQN, FCNs networks are used as feature extractors to approximate the state–actions Q-function by extracting the global discriminative features. However, the long-term temporal dependencies of sequential data are not considered. To overcome this difficulty, IDRQN adopts LSTM and candidate networks for action selection and evaluation. Deep-SARL applies a Q-function decomposition technique to a double DQN structure, reducing the problem-solving difficulty. Both IDRQN and Deep-SARL reduce the prediction error by improving their state–actions Q-function estimation methods. MRM-DQN uses multiple replay memories to speed up and stabilize its learning process.

As mentioned in Section III-E, the long-term cost comes from the task completion time and the energy consumption of MD. More data arriving in each time slot leads to heavier computation and transmission burdens for task offloading, subject to the limited computing capacity of MD. In Figs. 10 and 11, for each agent, the trend of the average task completion time and that of the average energy consumption both go up, with λ_m getting larger and larger. This finding is consistent with the one observed from Fig. 9.

2) Comparisons Against λ_m and M: We compare the performance of the nine DRL agents in scenarios with different λ_m and M, denoted by (λ_m, M). To be specific, λ_m varies from 1 to 4 Mb/s and M is from 10 to 40. For example, (1, 10) stands for a scenario with $\lambda_m = 1$ Mb/s and $M = 10$.

The violin plot is a combination of a box plot and kernel density plot [44]. We adopt it to exhibit both the summary statistics and the probability density distribution of the normalized task costs obtained by each agent in each (λ_m, M) case, as shown in Fig. 12. In particular, the outer shape of a violin implies the density estimation of the corresponding cost values. We have two findings as follows. First, with both λ_m and M increasing, the normalized cost values get bigger and bigger. That makes sense as we have similar findings in the first experiment above. It is simply because both λ_m and M affect the task completion time and the energy consumption of MD. Increasing both λ_m and M leads to growth in both the task completion time and energy consumption. Second, with both λ_m and M going up, for each DRL agent, the distribution of the normalized task cost values becomes increasingly more dispersed. It is because larger λ_m and M introduce more irregular fluctuations with larger amplitudes to the task costs.

In each (λ_m, M) case, the DDPG-based agents achieve better performance than the DQN-based ones, as shown in Fig. 12. Similar to the reasons explained in Section VI-D1, it is mainly because the former navigates through continuous action space via the fine control offered by the actor and critic networks while the latter is more suitable to handle discrete action space. It is no doubt TADPG performs the best among the nine. That is because the two features, TFEN and rPER, can significantly improve the state representation and function approximation of the original DDPG. Meanwhile, D^{3}PG is the second-best agent, thanks to the double Q-learning and dueling network designed to alleviate overestimation and stabilize the training process. In D^{3}PG, when carrying out the action space exploration, the FCNs in the actor network are not sensitive to potential temporal feature representation and relationship

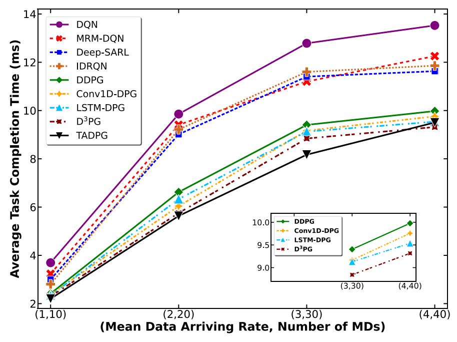

Fig. 13. Results of the average $\tau_{m,i}(t)$ versus λ_m and M.

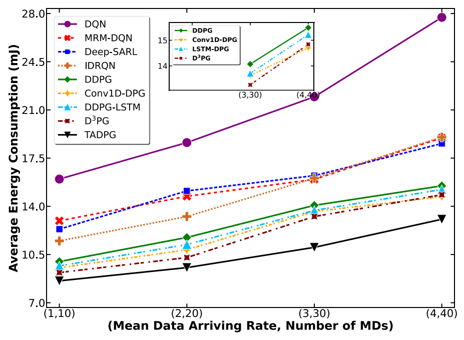

Fig. 14. Results of the average $E_{m,i}(t)$ versus λ_m and M.

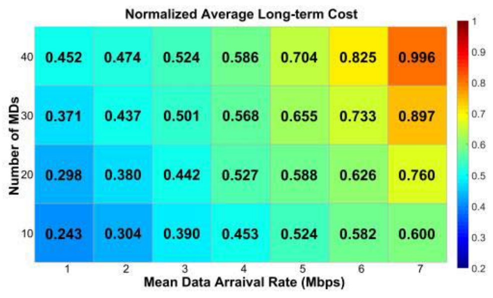

Fig. 15. Average C_{lterm} versus λ_m and M obtained by the TADPG agent.

extraction. That is why D³PG performs slightly worse than the proposed TADPG.

The results of the average completion time of each task and the average energy consumption of each MD are shown in Figs. 13 and 14, respectively. They both increase, as λ_m and M become bigger and bigger. TADPG and D³PG are the best and second best, while the original DQN is the worst among all.

To get a more insightful understanding of the TADPG agent, we plot the normalized average long-term cost values with different (λ_m, M) combinations in Fig. 15. No matter which one is fixed, λ_m or M, the normalized average long-term cost gradually increases as the other becomes larger and larger. Also, this cost grows up faster when both of them increase at the same time. The observations above also support the findings in the previous experiments.

VII. Conclusion

This article jointly optimizes the problem of computation offloading and resource allocation in MEC with multiple MDs. To address this problem, we propose a TADPG. Each MD runs a TADPG agent to make dynamic decisions on partial task offloading and resource allocation. Each TADPG agent is featured with a TFEN and an rPER. Experimental results show that TADPG achieves faster and more stable convergence than the original DDPG. The decentralized TADPG-based JCORA mechanism is more suitable for large-scale MEC scenarios than the centralized one, provided that the CSI is known in advance. Besides, TADPG performs better than eight DQN- and DDPG-based agents in almost all test instances, with respect to the average long-term cost, average task completion time, and average energy consumption of MD.

Our work assumes the BS can obtain the perfect CSI for each MD. However, the CSI estimation is usually imperfect in multiuser massive MIMO systems. In the future, we will investigate the dynamic JCORA problem under the imperfect CSI scenario. This article does not consider problem parameter variation during training, such as the cases that new users may enter, and old users may leave the system freely. We will address this problem by designing appropriate online training schemes. This article assumes there are sufficient spectrum and computing resources in the MEC system. In the future, we will consider a resource-constrained MEC system, where MDs compete for the limited spectrum and computing resources. To tackle this complicated problem, we plan to develop more sophisticated methods, such as multiagent DRL.

Appendix
Proof of Lemma 1

First, we find the optimal value for $\alpha_{m,i}(t)$.
When

$$0 \le \alpha_{m,i}(t) \le \frac{c_{m,i}}{\frac{f_{m,i}(t)}{\upsilon_{m,i}^{\text{tra}}(t)} + \frac{f_{m,i}(t)\cdot c_{m,i}}{f_{\text{ser}}} + c_{m,i}}$$

i.e., $\tau_{m,i}^{\text{loc}}(t) > (\tau_{m,i}^{\text{tra}}(t) + \tau_{m,i}^{\text{ser}}(t))$, (21) is rewritten as

$$C_{m,i}^{t} = \mathcal{E}\cdot\left(\vartheta\cdot\left(t-\pi_{m,i}^{\text{arr}}\right) + \frac{\left(1-\alpha_{m,i}(t)\right)\cdot d_{m,i}\cdot c_{m,i}}{f_{m,i}(t)}\right) + (1-\mathcal{E})\cdot\left(\kappa\cdot\left(f_{m,i}(t)\right)^{2}\cdot\left(1-\alpha_{m,i}(t)\right)\cdot d_{m,i}\cdot c_{m,i} + P_{m,i}^{\text{tra}}(t)\cdot\frac{\alpha_{m,i}(t)\cdot d_{m,i}}{\upsilon_{m,i}^{\text{tra}}(t)}\right). \quad (27)$$

The first-order derivative for (27) with respect to $\alpha_{m,i}(t)$ is defined in

$$\frac{\partial C_{m,i}^{t}}{\partial \alpha_{m,i}(t)} = -\mathcal{E}\cdot\frac{d_{m,i}\cdot c_{m,i}}{f_{m,i}(t)} - (1-\mathcal{E})\cdot\left(\kappa\cdot\left(f_{m,i}(t)\right)^{2}\cdot d_{m,i}\cdot c_{m,i} - P_{m,i}^{\text{tra}}(t)\cdot\frac{d_{m,i}}{\upsilon_{m,i}^{\text{tra}}(t)}\right). \quad (28)$$

When

$$\frac{c_{m,i}}{\frac{f_{m,i}(t)}{\upsilon_{m,i}^{\text{tra}}(t)}+\frac{f_{m,i}(t)\cdot c_{m,i}}{f_{\text{ser}}}+c_{m,i}} \leq \alpha_{m,i}(t) \leq 1$$

i.e., $\tau_{m,i}^{\text{loc}}(t) < (\tau_{m,i}^{\text{tra}}(t)+\tau_{m,i}^{\text{ser}}(t))$, (21) is rewritten as

$$C_{m,i}^{t} = \mathcal{E}\cdot\left(\vartheta\cdot\left(t-\pi_{m,i}^{\text{arr}}\right)+\frac{\alpha_{m,i}(t)\cdot d_{m,i}}{\upsilon_{m,i}^{\text{tra}}(t)}+\frac{\alpha_{m,i}(t)\cdot d_{m,i}\cdot c_{m,i}}{f_{\text{ser}}}\right) + (1-\mathcal{E})\cdot\left(\kappa\cdot\left(f_{m,i}(t)\right)^{2}\cdot\left(1-\alpha_{m,i}(t)\right)\cdot d_{m,i} \cdot c_{m,i}+P_{m,i}^{\text{tra}}(t)\cdot\frac{\alpha_{m,i}(t)\cdot d_{m,i}}{\upsilon_{m,i}^{\text{tra}}(t)}\right). \tag{29}$$

The first-order derivative for (29) with respect to $\alpha_{m,i}(t)$ is defined in

$$\frac{\partial C_{m,i}^{t}}{\partial\alpha_{m,i}(t)} = \mathcal{E}\cdot\left(\frac{d_{m,i}}{\upsilon_{m,i}^{\text{tra}}(t)}+\frac{d_{m,i}\cdot c_{m,i}}{f_{\text{ser}}}\right)-(1-\mathcal{E}) \cdot\left(\kappa\cdot\left(f_{m,i}(t)\right)^{2}\cdot d_{m,i}\cdot c_{m,i}-P_{m,i}^{\text{tra}}(t)\cdot\frac{d_{m,i}}{\upsilon_{m,i}^{\text{tra}}(t)}\right). \tag{30}$$

When

$$\frac{(1-\mathcal{E})\cdot P_{m,i}^{\text{tra}}(t)}{\mathcal{E}\cdot\frac{c_{m,i}}{f_{m,i}(t)}+(1-\mathcal{E})\cdot\kappa\cdot\left(f_{m,i}(t)\right)^{2}\cdot c_{m,i}} < \upsilon_{m,i}^{\text{tra}}(t) < \frac{\mathcal{E}+(1-\mathcal{E})\cdot P_{m,i}^{\text{tra}}(t)}{-\mathcal{E}\cdot\frac{c_{m,i}}{f_{\text{ser}}}+(1-\mathcal{E})\cdot\kappa\cdot\left(f_{m,i}(t)\right)^{2}\cdot c_{m,i}}$$

we have (28) < 0 and (30) > 0. On this basis, the corresponding static optimization subproblem is a convex optimization problem regarding $\alpha_{m,i}(t)$. Equation (21) is a decreasing function when

$$\alpha_{m,i}(t) \leq \frac{c_{m,i}}{\frac{f_{m,i}(t)}{\upsilon_{m,i}^{\text{tra}}(t)}+\frac{f_{m,i}(t)\cdot c_{m,i}}{f_{\text{ser}}}+c_{m,i}}$$

and it is an increasing function when

$$\alpha_{m,i}(t) \geq \frac{c_{m,i}}{\frac{f_{m,i}(t)}{\upsilon_{m,i}^{\text{tra}}(t)}+\frac{f_{m,i}(t)\cdot c_{m,i}}{f_{\text{ser}}}+c_{m,i}}.$$

Thus, the objective value $C_{m,i}^{t}$ is minimal at

$$\alpha_{m,i}^{*}(t) = \frac{c_{m,i}}{\frac{f_{m,i}(t)}{\upsilon_{m,i}^{\text{tra}}(t)}+\frac{f_{m,i}(t)\cdot c_{m,i}}{f_{\text{ser}}}+c_{m,i}}$$

where $\tau_{m,i}^{\text{loc}}(t)=\tau_{m,i}^{\text{tra}}(t)+\tau_{m,i}^{\text{ser}}(t)$.

Then, we find the optimal value for $f_{m,i}(t)$. The first-order derivative for (27) with respect to $f_{m,i}(t)$ is defined in

$$\frac{\partial C_{m,i}^{t}}{\partial f_{m,i}(t)} = \left(1-\alpha_{m,i}(t)\right)\cdot d_{m,i}\cdot c_{m,i}\cdot f_{m,i}(t) \cdot\left(-\mathcal{E}\cdot\left(f_{m,i}(t)\right)^{-3}+2(1-\mathcal{E})\cdot\kappa\right). \tag{31}$$

We have (31) < 0 when $f_{m,i}(t) > \sqrt[3]{(\mathcal{E}/[2(1-\mathcal{E})\cdot\kappa])}$, and (31) > 0 when $f_{m,i}(t) < \sqrt[3]{(\mathcal{E}/[2(1-\mathcal{E})\cdot\kappa])}$, respectively. The objective value $C_{m,i}^{t}$ is minimal when $[\partial C_{m,i}^{t}/\partial f_{m,i}(t)]=0$, where $f_{m,i}(t) = \sqrt[3]{(\mathcal{E}/[2(1-\mathcal{E})\cdot\kappa])}$. Note that for any $f_{m,i}(t)$, we have $0 \leq f_{m,i}(t) \leq F_m$. Thus, the optimal $f_{m,i}(t)$ is

$$f_{m,i}^{*}(t) = \min\left\{\sqrt[3]{\frac{\mathcal{E}}{2(1-\mathcal{E})\cdot\kappa}}, F_m\right\}.$$

Finally, we find the optimal value for $P_{m,i}^{\text{tra}}(t)$ when sending data $\alpha_{m,i}(t)\cdot d_{m,i}$ to the BS. Note that $\upsilon_{m,i}^{\text{tra}}(t) = W\cdot\log_2(1+P_{m,i}^{\text{tra}}(t)\cdot\hat{g}_m(t))$. The first-order derivative for (29) with respect to $P_{m,i}^{\text{tra}}(t)$ is defined in

$$\begin{aligned}\frac{\partial C_{m,i}^{t}}{\partial P_{m,i}^{\text{tra}}(t)} &= -\frac{\mathcal{E}\cdot\alpha_{m,i}(t)\cdot d_{m,i}}{\left[\upsilon_{m,i}^{\text{tra}}(t)\right]^{2}}\cdot\frac{\partial\upsilon_{m,i}^{\text{tra}}(t)}{\partial P_{m,i}^{\text{tra}}(t)}+(1-\mathcal{E}) \\ &\quad\cdot\frac{\alpha_{m,i}(t)\cdot d_{m,i}\cdot\upsilon_{m,i}^{\text{tra}}(t)-P_{m,i}^{\text{tra}}(t)\cdot\alpha_{m,i}(t)\cdot d_{m,i}\cdot\frac{W\cdot\hat{g}_m(t)}{1+P_{m,i}^{\text{tra}}(t)\cdot\hat{g}_m(t)}\cdot\frac{1}{ln2}}{\left[\upsilon_{m,i}^{\text{tra}}(t)\right]^{2}} \\ &= \frac{\alpha_{m,i}(t)\cdot d_{m,i}}{\left[\upsilon_{m,i}^{\text{tra}}(t)\right]^{2}}\cdot\left((1-\mathcal{E})\cdot W\cdot\log_2^{\left(1+P_{m,i}^{\text{tra}}(t)\cdot\hat{g}_m(t)\right)} - \frac{(1-\mathcal{E})\cdot P_{m,i}^{\text{tra}}(t)\cdot W\cdot\hat{g}_m(t)+\mathcal{E}\cdot W\cdot\hat{g}_m(t)}{\left(1+P_{m,i}^{\text{tra}}(t)\cdot\hat{g}_m(t)\right)\cdot ln2}\right).\end{aligned} \tag{32}$$

Similar to the derivative process of $f_{m,i}^{*}(t)$, the objective value $C_{m,i}^{t}$ is minimal when $[\partial C_{m,i}^{t}/\partial P_{m,i}^{\text{tra}}(t)]=0$, as expressed in

$$(1-\mathcal{E})\cdot W\cdot\log_2^{\left(1+P_{m,i}^{\text{tra}}(t)\cdot\hat{g}_m(t)\right)} - \frac{(1-\mathcal{E})\cdot P_{m,i}^{\text{tra}}(t)\cdot W\cdot\hat{g}_m(t)+\mathcal{E}\cdot W\cdot\hat{g}_m(t)}{\left(1+P_{m,i}^{\text{tra}}(t)\cdot\hat{g}_m(t)\right)\cdot ln2} = 0. \tag{33}$$

After some algebraic operations, $P_{m,i}^{\text{tra}}(t)$ is obtained by

$$P_{m,i}^{\text{tra}}(t) = \frac{e^{\mathbb{W}\left(\left(\frac{\mathcal{E}\cdot\hat{g}_m(t)}{1-\mathcal{E}}-1\right)\cdot\frac{1}{e}\right)+1}-1}{\hat{g}_m(t)} \tag{34}$$

where $\mathbb{W}(x)$ denotes the Lambert-W function, an inverse function of $f(z)=z\cdot e^{z}=x$, i.e., $z=\mathbb{W}(x)$. Note that for any $P_{m,i}^{\text{tra}}(t)$, we have $0 \leq P_{m,i}^{\text{tra}}(t) \leq P_m^{\max}$. Hence, the optimal $P_{m,i}^{\text{tra}}(t)$ is

$$P_{m,i}^{\text{tra}^{*}}(t) = \min\left\{\frac{e^{\mathbb{W}\left(\left(\frac{\mathcal{E}\cdot\hat{g}_m(t)}{1-\mathcal{E}}-1\right)\cdot\frac{1}{e}\right)+1}-1}{\hat{g}_m(t)}, P_m^{\max}\right\}.$$

REFERENCES

[1] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[3] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in IoT edge computing," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1133–1146, Jun. 2020.

[4] T.-X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[5] A. S. Bi and Y. J. A. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.

[6] G. Ahani and D. Yuan, "BS-assisted task offloading for D2D networks with presence of user mobility," in *Proc. IEEE 89th Veh. Technol. Conf. (VTC-Spring)*, Kuala Lumpur, Malaysia, 2019, pp. 1–5.

[7] S. Jošilo and G. Dán, "Wireless and computing resource allocation for selfish computation offloading in edge computing," in *Proc. IEEE 33th Int. Conf. Comput. Commun. (INFOCOM)*, Paris, France, 2019, pp. 2467–2475.

[8] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial IoT–edge–Cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2759–2774, Dec. 2019.

[9] J. Zheng, Y. Cai, Y. Wu, and X. Shen, "Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach," *IEEE Trans. Mobile Comput.*, vol. 18, no. 4, pp. 771–786, Apr. 2019.

[10] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE ACM/Trans. Netw.*, vol. 26, no. 6, pp. 2651–2664, Dec. 2018.

[11] H. Wu, Y. Sun, and K. Wolter, "Energy-efficient decision making for mobile cloud offloading," *IEEE Trans. Cloud Comput.*, vol. 8, no. 2, pp. 570–584, Jun. 2020.

[12] L. Huang, S. Bi, and Y. J. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020.

[13] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2018, pp. 1–6.

[14] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[15] H. Lu, C. Gu, F. Luo, W. Ding, and X. Liu, "Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning," *Future Gener. Comput. Syst.*, vol. 102, pp. 847–861, Jan. 2020.

[16] H. Lu, X. He, M. Du, X. Ruan, Y. Sun, and K. Wang, "Edge QoE: Computation offloading with deep reinforcement learning for Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9255–9265, Oct. 2020.

[17] J. Zhang, J. Du, Y. Shen, and J. Wang, "Dynamic computation offloading with energy harvesting devices: A hybrid-decision-based deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9303–9317, Oct. 2020.

[18] Z. Chen, and X. Wang, "Decentralized computation offloading for multiuser mobile edge computing: A deep reinforcement learning approach," *EURASIP J. Wireless Commun.*, vol. 188, no. 2020, pp. 1–21, Sep. 2020, doi: 10.1186/s13638-020-01801-6.

[19] K. Liu and W. Liao, "Intelligent offloading for multi-access edge computing: A new actor–critic approach," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Dublin, Ireland, 2020, pp. 1–6.

[20] X. Qiu, W. Zhang, W. Chen, and Z. Zheng, "Distributed and collective deep reinforcement learning for computation offloading: A practical perspective," *IEEE Trans. Parallel. Distrib. Syst.*, vol. 32, no. 5, pp. 1085–1101, May 2021.

[21] H. He and H. Jiang, "Deep learning based energy efficiency optimization for distributed cooperative spectrum sensing," *IEEE Wireless Commun. Mag.*, vol. 26, no. 3, pp. 32–39, Jun. 2019.

[22] C. Pradhan, A. Li, C. She, Y. Li, and B. Vucetic, "Computation offloading for IoT in C-RAN: Optimization and deep learning," *IEEE Trans. Commun.*, vol. 68, no. 7, pp. 4565–4579, Jul. 2020.

[23] X. Zhao *et al.*, "Deep learning based mobile data offloading in mobile edge computing systems," *Future Gener. Comput. Syst.*, vol. 99, pp. 346–355, Oct. 2019.

[24] C. Liu *et al.*, "A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure," *IEEE Trans. Service Comput.*, vol. 11. no. 2, pp. 249–261, Mar./Apr. 2018.

[25] M. Zeng, W. Hao, O. A. Dobre, Z. Ding, and H. V. Poor, "Massive MIMO-assisted mobile edge computing: Exciting possibilities for computation offloading," *IEEE Veh. Technol. Mag.*, vol. 15, no. 2, pp. 31–38, Jun. 2020.

[26] W. Zhan *et al.*, "Deep-reinforcement-learning-based offloading scheduling for vehicular edge computing," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5449–5465, Jun. 2020.

[27] Y. Hao, Q. Ni, H. Li, and S. Hou, "Energy-efficient multi-user mobile-edge computation offloading in massive MIMO enabled HetNets," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Shanghai, China, 2019, pp. 1–6.

[28] M. Zeng, W. Hao, O. A. Dobre, and H. V. Poor, "Delay minimization for massive MIMO assisted mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 6, pp. 6788–6792, Jun. 2020.

[29] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 6, pp. 85–117, Jan. 2015.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770–778.

[31] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[32] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 5998–6008.

[33] W. Du, Y. Wang, and Y. Qiao, "Recurrent spatial–temporal attention network for action recognition in videos," *IEEE Trans. Image Process.*, vol. 27, no. 3, pp. 1347–1360, Mar. 2018.

[34] Y. Zhao, C. Guo, J. Wang, X. Zhang, and H. Lu, "Attention CoupleNet: Fully convolutional attention coupling network for object detection," *IEEE Trans. Image Process.*, vol. 28, no. 1, pp. 1170–1175, Jan. 2019.

[35] J. Chen *et al.*, "STDPG: A spatio-temporal deterministic policygradient agent for dynamic routing in SDN," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Dublin, Ireland, 2020, pp. 1–6.

[36] Y. Ji, R. Gu, Z. Yang, J. Li, H. Li, and M. Zhang "Artificial intelligence-driven autonomous optical networks: 3S architecture and key technologies," *Sci. China Inf. Sci.*, vol. 63, May 2020, Art. no. 160301.

[37] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, San Juan, Puerto Rico, May 2016, pp. 1–9.

[38] A. W. Moore and C G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Mach. Learn.*, vol. 13, no. 1, pp. 103–130, 1993.

[39] Z. Ni, N. Malla, and X. Zhong, "Prioritizing useful experience replay for heuristic dynamic programming-based learning systems," *IEEE Trans. Cybern.*, vol. 49, no. 11, pp. 3911–3922, Nov. 2019.

[40] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. 31th Int. Conf. Mach. Learn. (ICML)*, 2014, pp. 387–395.

[41] Z. Xu *et al.*, "Experience-driven networking: A deep reinforcement learning based approach," in *Proc. IEEE 33th Int. Conf. Comput. Commun. (INFOCOM)*, Honolulu, HI, USA, Apr. 2018, pp. 1871–1879.

[42] J. L. D. Neto, S. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, "ULOOF: A user level online offloading framework for mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 17, no. 11, pp. 2660–2674, Nov. 2018.

[43] L. Xiao *et al.*, "Reinforcement learning-based downlink interference control for ultra-dense small cells," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 423–434, Jan. 2020.

[44] J. L. Hintze and R. D Nelson, "Violin plots: A box plot-density trace synergism," *Amer. Stat.*, vol. 52, no. 2, p. 181–184, 1998.

Juan Chen received the M.S. degree from Southwest Jiaotong University, Chengdu, China, in 2012, where she is currently pursuing the Ph.D. degree with the School of Computing and Artificial Intelligence.

From 2012 to 2016, she was an LTE Vehicular Network Engineer with China Energy Investment Corporation, Ltd., Beijing, China. Her research interests include software-defined networking, mobile-edge computing, wireless communication, and deep reinforcement learning.

Huanlai Xing (Member, IEEE) received the Ph.D. degree in computer science from the University of Nottingham, Nottingham, U.K., in 2013.

He is an Associate Professor with the School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu, China. His research interests include mobile-edge computing, data mining, machine learning, network function virtualization, software-defined networking, and evolutionary computation.

Zhiwen Xiao (Student Member, IEEE) received the B.Eng. degree in network engineering from Chengdu University of Information Technology, Chengdu, China, in 2019. He is currently pursuing the Ph.D. degree in computer science with Southwest Jiaotong University, Chengdu.

His research interests are deep learning, representation learning, data mining, and computer vision.

Lexi Xu (Member, IEEE) received the B.S. degree from Southwest Jiaotong University, Chengdu, China, in 2006, the M.S. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2009, and the Ph.D. degree from the Queen Mary University of London, London, U.K., in 2013.

He is currently working as a Senior Engineer with China United Network Communications Corporation, Beijing. His research interests include telecom operation and network communications.

Tao Tao received the B.S. degree from the North China University of Technology, Beijing, China, in 2004, and the M.S. degree from Beijing University of Technology, Beijing, in 2008.

He is currently working with China United Network Communications Corporation, Beijing. His research interests include telecom operation and network communications.