QECO: A QoE-Oriented Computation Offloading Algorithm based on Deep Reinforcement Learning for Mobile Edge Computing

Iman Rahmati , Hamed Shah-Mansouri , and Ali Movaghar

***Abstract*—In the realm of mobile edge computing (MEC), efficient computation task offloading plays a pivotal role in ensuring a seamless quality of experience (QoE) for users. Maintaining a high QoE is paramount in today's interconnected world, where users demand reliable services. This challenge stands as one of the most primary key factors contributing to handling dynamic and uncertain mobile environment. In this study, we delve into computation offloading in MEC systems, where strict task processing deadlines and energy constraints can adversely affect the system performance. We formulate the computation task offloading problem as a Markov decision process (MDP) to maximize the long-term QoE of each user individually. We propose a distributed QoE-oriented computation offloading (QECO) algorithm based on deep reinforcement learning (DRL) that empowers mobile devices to make their offloading decisions without requiring knowledge of decisions made by other devices. Through numerical studies, we evaluate the performance of QECO. Simulation results reveal that compared to the state-of-the-art existing works, QECO increases the number of completed tasks by up to 14.4%, while simultaneously reducing task delay and energy consumption by 9.2% and 6.3%, respectively. Together, these improvements result in a significant average QoE enhancement of 37.1%. This substantial improvement is achieved by accurately accounting for user dynamics and edge server workloads when making intelligent offloading decisions. This highlights QECO's effectiveness in enhancing users' experience in MEC systems.**

***Index Terms*—Mobile edge computing, computation task offloading, quality of experience, deep reinforcement learning.**

I. Introduction

MOBILE edge computing (MEC) [1] has emerged as a promising technological solution to overcome the challenges faced by mobile devices (MDs) when performing high computational tasks, such as real-time data processing and artificial intelligence applications [2] [3]. In spite of the MDs' technological advancements, their limited computing power and energy may lead to task drops, processing delays, and an overall poor user experience. By offloading intensive tasks to nearby edge nodes (ENs), MEC effectively empowers computation capability and reduces the delay and energy consumption. This improvement enhances the users' quality of experience (QoE), especially for time-sensitive computation tasks [4] [5].

Efficient task offloading in MEC is a complex optimization challenge due to the dynamic nature of the network and the variety of MDs and servers involved [6] [7]. In particular, determining the optimal offloading strategy, scheduling the tasks, and selecting the most suitable EN for task offloading are the main challenges that demand careful consideration. In addition to the dynamic network changes, the uncertain requirements and sensitive latency properties of computation tasks pose nontrivial challenges that can significantly impact the MEC systems.

To cope with the dynamic nature of the network, recent research has proposed several task offloading algorithms using machine learning methods. In particular, reinforcement learning (RL) [8] holds promises to determine optimal decision-making policies by capturing the dynamics of environments and learning strategies for accomplishing long-term objectives. However, the traditional RL methods are not efficient in handling environments with high-dimensional state spaces. Deep reinforcement learning (DRL) combines traditional RL with deep neural networks to intelligently respond to the unknown and dynamic system when addressing the limitations of RL-based algorithms. Despite these advancements, task offloading still faces significant challenges in real-world scenarios with multiple MDs and ENs. In such scenarios, it is essential for MDs to make offloading decisions independently without prior knowledge of other MDs' tasks and offloading models. However, existing works fail to adequately address this challenge. In addition, QoE is a time-varying performance measure that reflects user satisfaction and is not affected only by delay, but also by energy consumption. Albeit some existing works have investigated the trade-off between delay and energy consumption, they fail to properly estimate the QoE value and address the users' requirements. Although deep Q-networks (DQN) offer a valuable tool, a more comprehensive approach is required to accurately estimate the QoE while addressing the aforementioned challenges in practical scenarios where global information is limited.

In this study, we delve into the computation task offloading problem in MEC systems, where strict task processing deadlines and energy constraints can adversely affect the system performance. We propose a distributed QoE-oriented computation offloading (QECO) algorithm to efficiently handle task offloading under uncertain loads at ENs. This algorithm empowers MDs to make offloading decisions utilizing only locally observed information, such as task size, queue details, battery status, and historical workloads at the ENs. To capture the dynamic nature of the MEC environment, we employ the dueling double deep Q-network (D3QN), which is a refined improvement over standard DQN model. By integrating both

I. Rahmati and A. Movaghar are with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran (email:{iman.rahmati, movaghar}@sharif.edu).

H. Shah-Mansouri is with the Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran (email: hamedsh@sharif.edu).

double Q-learning [9] and dueling network architectures [10], D3QN reduces overestimation bias in action-value predictions and more accurately distinguishes the relative importance of states and actions. We also incorporate long short-term memory (LSTM) [11] into our D3QN model to continuously estimate dynamic workloads at ENs. This enables MDs to effectively handle the uncertainty of the MEC environment, where global information is limited, and proactively adjust the offloading strategies to improve their long-term QoE estimation. By adopting the appropriate policy based on each MD's specific requirements at any given time, the QECO algorithm significantly improves the QoE for individual users.

Our main contributions are summarized as follows:

- *Task Offloading Problem in the MEC System:* We consider queuing systems for MDs and ENs and formulate the task offloading problem taking into account the time-varying system environments (e.g., the arrival of new tasks, and the computational requirement of each task). We aim to maximize the long-term QoE, which reflects the task completion rate, task delay, and energy consumption of MDs. Our problem formulation effectively utilizes the resources and properly handles the dynamic fluctuations of workload at ENs.
- *DRL-based Offloading Algorithm:* To solve the problem of long-term QoE maximization in highly dynamic mobile environments, we propose the QECO algorithm based on DRL, which empowers each MD to make offloading decisions independently, without prior knowledge of other MDs' tasks and offloading models. Focusing on the MD's QoE preference, our approach leverages D3QN and LSTM to prioritize and strike an appropriate balance between QoE factors while accounting for workload uncertainty at the ENs. QECO empowers MDs to anticipate the EN's load level over time, leading to more accurate offloading strategies. We also analyze the training convergence and computational complexity of the proposed algorithm.
- *Performance Evaluation:* We conduct comprehensive experiments to evaluate QECO's performance and its training convergence against baseline schemes as well as two state-of-the-art existing works. The results demonstrate that our algorithm effectively utilizes the computing resources while taking the dynamic workloads at ENs into consideration. In particular, QECO converges more quickly than vanilla DQN and DDQN methods. QECO also improves average QoE by 37.1% and 29.8% compared to the potential game-based offloading algorithm (PGOA) [12] and distributed and collective DRL-based offloading algorithm (DCDRL) [13], respectively.

The structure of this paper is as follows. Section II reviews the related works. Section III presents the system model, followed by the problem formulation in Section IV. In Section V, we present the algorithm, while Section VI provides an evaluation of its performance. Finally, we conclude in Section VII.

II. Related Work

To effectively tackle the challenges of MEC arising from the ever-changing nature of networks, recent research highlights the effectiveness of RL in adapting to environmental changes and learning optimal strategies. In this section, we explore the current RL-based state-of-the-art works and discuss their strengths and limitations. Table I provides an intuitive comparison of these works.

For time-sensitive applications, minimizing task delay is the primary optimization objective, driving the development of delay-optimal approaches in MEC. To solve the delay minimization problem in MEC environment, Zhang *et al.* in [14] proposed an offline RL-based computation offloading algorithm, where the problem is modeled as a repeated game between two agents. Sun *et al.* in [15] explored both computation offloading and service caching problems in MEC. They formulated an optimization problem that aims to minimize the long-term average service delay. They then proposed a hierarchical DRL framework, which effectively handles both problems under heterogeneous resources. Liu *et al.* in [16] formulated the problem as MDP and proposed DRL-based approaches to minimize both communication and computation delay in MEC. To minimize total delay and reduce mobile vehicle task waiting time, Tang *et al.* in [17] developed a dynamic offloading model for multiple mobile vehicles, segmenting tasks into sequential subtasks for more precise offloading decisions. Li *et al.* in [18] proposed an integrated optimization framework, which leverages past experience and model knowledge to enable fast and resilient real-time offloading control. In [19], the authors investigated deadline-constrained time-sensitive tasks in vehicular MEC networks and proposed a multi-agent RL-based computation offloading algorithm based on POMDP to address joint delay and computation rate optimization. However, delay-optimal approaches [14]–[19] often sacrifice other key performance metrics and may cause greater energy consumption or reduce the overall computation capacity, to achieve their goal.

For energy-constrained MEC systems, reducing energy consumption becomes a key objective, motivating the creation of energy-efficient approaches. Munir *et al.* [20] investigated an energy dispatch problem of a self-powered wireless network with MEC capabilities and developed a semi-distributed approach using a multi-agent RL framework to minimize energy consumption. Zhou *et al.* in [21] proposed a Q-learning approach, which is an extension of RL to achieve optimal resource allocation strategies and computation offloading. Dai *et al.* in [22] introduced the integration of action refinement into DRL and designed an algorithm based on deep deterministic policy gradient (DDPG) to optimize resource allocation and computation offloading concurrently. To ensure that minimizing energy consumption does not lead to a reduction in the system's computing rate, researchers have expanded their work by incorporating the maximum tolerable delay for task models. In [23], Chouikhi *et al.* proposed a computation offloading approach based on DRL to minimize long-term energy consumption and maximize the number of tasks completed before their tolerant deadlines. To address the privacy-aware computation offloading problem, Wu *et al.* in [24] modeled the problem as an MDP and proposed a DQN-based method to optimize the computation rate and energy consumption in a queuing-based IIoT network. Huang *et al.* in [25] proposed a DRL-based method based on a partially observable MDP (POMDP), which guarantees the deadlines of real-time tasks

TABLE I
COMPARISON OF RELATED WORKS

Paper	Problem	Optimization target				Method	Limitation
		Delay	Energy	Drop	QoE		
[14]	Computation offloading	✓	✗	✗	✗	Offline RL	Overlooking energy consumption; Not suitable for energy-constrained MEC; May cause greater energy consumption or reduce the overall system capacity.
[15]	Joint offloading and service caching	✓	✗	✗	✗	Hierarchical DRL	
[16]	Computation offloading	✓	✗	✗	✗	DQN, DDPG	
[17]	Computation offloading	✓	✗	✗	✗	DDQN	
[18]	Online computation offloading	✓	✗	✗	✗	AC network	
[19]	Computation offloading	✓	✗	✓	✗	AC network	
[20]	Computation offloading	✗	✓	✗	✗	Multi-agent RL	Do not account for time-sensetive applications; Saving transmission energy consumption may reduce the transmission power of MDs.
[21]	Joint offloading and resource allocation	✗	✓	✗	✗	DDQN	
[22]	Joint offloading and resource allocation	✗	✓	✗	✗	DDPG	
[23]	Computation offloading	✗	✓	✓	✗	DRL	
[24]	Privacy aware computation offloading	✗	✓	✓	✗	AC network	
[25]	Computation offloading	✓	✓	✗	✗	DDPG	Not suitable for applications with strict delay requirements, as processing deadlines influence the load dynamics at the ENs, impacting the delay of offloaded tasks.
[26]	Online computation offloading	✓	✓	✗	✗	DDQN	
[27]	Computation offloading	✓	✓	✗	✗	Multi-agent PPO	
[28]	Joint offloading and resource allocation	✓	✓	✗	✗	DRL	
[29]	Joint offloading and resource allocation	✓	✓	✗	✗	Multi-agent AC	
[30]	Computation offloading	✓	✓	✗	✗	Multi-agent DDPG	
[31]	Computation offloading	✓	✓	✓	✗	Multi-agent AC	Overlooking personalized QoE; Treating all MDs uniformly in performance optimization fails to address the user satisfaction.
[32]	Joint offloading and power allocation	✓	✓	✓	✗	Multi-agent AC	
[33]	Energy harvesting offloading	✓	✓	✓	✗	DQN	
[34]	Joint offloading and resource allocation	✓	✓	✓	✗	DDPG	
[35]	Stochastic game resource allocation	✓	✓	✓	✗	Multi-agent RL	
QECO	QoE-oriented computation offloading	✓	✓	✓	✓	D3QN + LSTM	

while minimizing the total energy consumption of MDs. This algorithm effectively tackles the challenges of dynamic resource allocation in large-scale heterogeneous networks. However, energy-efficient approaches [20]–[25], even when considering deadline constraints, do not adequately address the complexities of time-sensitive applications, where reducing delay can have significant effects on the user's QoE. Particularly, saving transmission energy consumption may reduce the transmission power of MDs and lead to poor performance for real-time applications.

To address time-sensitive applications in energy-constrained MEC systems, solving the relationship between delay and energy consumption is the primary optimization objective. Liao *et al.* in [26] introduced a double DQN (DDQN)-based algorithm for performing online computation offloading in MEC. This algorithm optimizes transmission power and scheduling of CPU frequency when minimizing both task computation delay and energy consumption. To optimize delay and energy consumption, Wu *et al.* in [27] investigated the computation offloading problem in a queuing-based MEC IIoT system. They modeled the problem as a POMDP and proposed a multi-agent proximal policy optimization (PPO)-based method to obtain the optimal offloading strategy in dynamic environments. Gong *et al.* in [28] proposed a DRL-based network structure in the IIoT systems to jointly optimize task offloading and resource allocation to achieve lower energy consumption and decreased task delay. Liu *et al.* in [29] investigated a two-timescale computing offloading and resource allocation problem and proposed a resource coordination algorithm based on multi-agent DRL, which can generate interactive information along with resource decisions. She *et al.* in [30] investigated the computation offloading optimization problem and proposed a DRL-based approach to minimize both the transmission delay and energy, based on POMDP. Although delay-energy trade-off approaches [26]–[30] effectively address time-sensitive tasks, they are not suitable for applications with strict requirements on delay or energy consumption.

Incorporating task processing deadlines for time-sensitive applications in energy-constrained MEC systems requires achieving an effective balance among delay, energy consumption, and computation rate. In [31], Gao *et al.* introduced an attention-based multi-agent algorithm designed for decentralized computation offloading. To optimize privacy protection and quality of service, authors in [32] investigated the joint

computation offloading and power allocation problems for the IIoT network. They modeled the problem as an MDP and proposed a multi-agent DQN-based algorithm. In [33], the authors proposed an offloading algorithm using deep Q-learning for wireless-powered IoT devices in MEC. This algorithm aims to minimize the task drop rate while the devices solely rely on harvested energy for operation. Chen *et al.* in [34] addressed a joint optimization problem involving computation offloading and resource allocation, aiming to reduce both task processing delay and energy consumption across all MDs. Wu *et al.* in [35] introduced a stochastic game-based resource allocation in the SDN-based MEC. They used an MDP and proposed a multi-agent RL method to minimize both energy consumption and delay. While studies [31]–[35] have explored delay-sensitive applications with processing deadlines and addressed energy-constrained MEC, they overlook user-specific demands and fail to meet QoE requirements effectively.

In contrast to [14]–[25], which do not account for energy-constrained and time-sensitive MEC, we are motivated to address both issues to provide individual QoE requirements effectively. Although [26]–[30] taking delay and energy consumption into consideration as joint optimization objectives, they fail to address applications with strict processing deadlines. This is challenging to address, as processing deadlines impact the load dynamics at the ENs, affecting the delay of offloaded tasks. On the other hand, QoE is a time-varying performance measure that reflects user satisfaction, where each user may have unique QoE requirements. According to the MDs' individual preferences each key QoE factor may take precedence over others. Different from [31]–[35], which overlooked personalized user preferences, we designed an adaptive balance to enhance QoE for MEC users effectively.

In addition, the limited computing resourses of ENs necessitate careful consideration when allocating resources to MDs. The allocated resource to an MD depends on the dynamic workload of an EN at any given time. Some studies have failed to account for these challenges in dynamic network environments, resulting in reduced system performance. For example, in [34], it is assumed that MDs have sufficient transmission and computation resources from ENs, which may not reflect real-world conditions. While [22], [32], and [36] investigated limited resources in MEC, they did not consider dynamic workloads at ENs in decision-making processes, leading to potential performance issues. Some studies such as [15], [28], and [31] have addressed load levels at ENs and proposed task offloading algorithms. However, these algorithms often require global system information, resulting in high signaling overhead. In contrast to these works, we propose a distributed algorithm that effectively manages unknown load dynamics at ENs. This algorithm enables each MD to make offloading decisions independently, without requiring information (e.g., task models, offloading decisions) from other MDs, thereby reducing signaling overhead and improving overall system performance.

III. System Model

We investigate a MEC system consisting of a set of MDs denoted by $\mathcal{I} = \{1, 2, ..., I\}$, along with a set of ENs denoted by $\mathcal{J} = \{1, 2, ..., J\}$, where I and J represent the number of MDs and ENs, respectively. We regard time as a specific episode containing a series of T time slots denoted by $\mathcal{T} = \{1, 2, \ldots, T\}$, each representing a duration of τ seconds. As shown in Fig. 1, we consider two separate queues for each MD to organize tasks for local processing or dispatching to ENs, operating in a first-in-first-out (FIFO) manner. The MD's scheduler is responsible for assigning newly arrived tasks to each of the queues at the beginning of the time slot. On the other hand, we assume that each EN $j \in \mathcal{J}$ consists of I FIFO queues, where each queue corresponds to an MD $i \in \mathcal{I}$. When each task arrives at an EN, it is enqueued in the corresponding MD's queue.

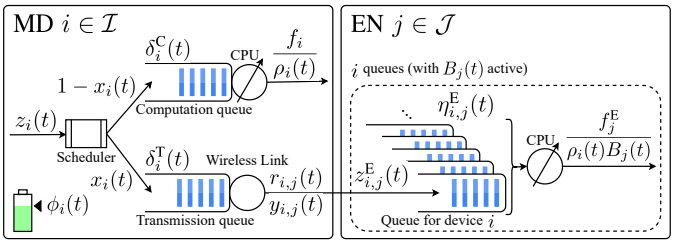


Fig. 1. An illustration of MD $i \in \mathcal{I}$ and EN $j \in \mathcal{J}$ in the MEC system.

We define $z_i(t)$ as the index assigned to the computation task arriving at MD $i \in \mathcal{I}$ in time slot $t \in \mathcal{T}$. Let $\lambda_i(t)$ denote the size of this task in bits. The size of task $z_i(t)$ is selected randomly and uniformly from a discrete set $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_\theta\}$, where θ represents the number of these values. Note that task sizes are drawn from a discrete set since, in many applications, tasks typically come with predefined sizes, [37], [38]. We consider $\lambda_i(t) \in \Lambda \cup \{0\}$ to include the case that no task has arrived. We also denote the task's processing density as $\rho_i(t)$ that indicates the number of CPU cycles required to complete the execution of a unit of the task. Furthermore, we denote the deadline of this task by $\Delta_i(t)$ which is the number of time slots that the task must be completed to avoid being dropped.

We define two binary variables, $x_i(t)$ and $y_{i,j}(t)$ for $i \in \mathcal{I}$ and $j \in \mathcal{J}$ to determine the offloading decision and offloading target, respectively. Specifically, $x_i(t)$ indicates whether task $z_i(t)$ is assigned to the computation queue ($x_i(t) = 0$) or to the transmission queue ($x_i(t) = 1$), and $y_{i,j}(t)$ indicates whether task $z_i(t)$ is offloaded to EN $j \in \mathcal{J}$. If the task is dispatched to EN j, we set $y_{i,j}(t) = 1$; otherwise, $y_{i,j}(t) = 0$.

A. Communication Model

We consider that the tasks in the transmission queue are dispatched to the appropriate ENs via the MD wireless interface. We denote the transmission rate of MD i's interface when communicating with EN $j \in \mathcal{J}$ in time t as $r_{i,j}(t)$. In time slot $t \in \mathcal{T}$, if task $z_i(t)$ is assigned to the transmission queue for computation offloading, we define $l_i^{\mathrm{T}}(t) \in \mathcal{T}$ to represent the time slot when the task is either dispatched to the EN or dropped. We also define $\delta_i^{\mathrm{T}}(t)$ as the number of time slots that task $z_i(t)$ should wait in the queue before transmission. It

should be noted that MD i computes the value of $\delta_i^{\mathrm{T}}(t)$ before making a decision. The value of $\delta_i^{\mathrm{T}}(t)$ is computed as follows:

$$\delta_i^{\mathrm{T}}(t) = \left[\max_{t^{'} \in \{0,1,\ldots,t-1\}} l_i^{\mathrm{T}}(t^{'}) - t + 1 \right]^+, \quad (1)$$

where $[\cdot]^+ = \max(0, \cdot)$ and $l_i^{\mathrm{T}}(0) = 0$ for the simplicity of presentation. Note that the value of $\delta_i^{\mathrm{T}}(t)$ only depends on $l_i^{\mathrm{T}}(t)$ for $t' < t$. If MD $i \in \mathcal{I}$ schedules task $z_i(t)$ for dispatching in time slot $t \in \mathcal{T}$, then it will either be dispatched or dropped in time slot $l_i^{\mathrm{T}}(t)$, which is

$$l_i^{\mathrm{T}}(t) = \min\left\{t + \delta_i^{\mathrm{T}}(t) + \lceil D_i^{\mathrm{T}}(t) \rceil - 1, t + \Delta_i(t) - 1\right\}, \quad (2)$$

where $D_i^{\mathrm{T}}(t)$ refers to the number of time slots required for the transmission of task $z_i(t)$ from MD $i \in \mathcal{I}$ to EN $j \in \mathcal{J}$. We have

$$D_i^{\mathrm{T}}(t) = \sum_{j \in \mathcal{J}} y_{i,j}(t) \frac{\lambda_i(t)}{r_{i,j}(t)\tau}. \quad (3)$$

Let $E_i^{\mathrm{T}}(t)$ denote the energy consumption of the transmission from MD $i \in \mathcal{I}$ to EN $j \in \mathcal{J}$. We have

$$E_i^{\mathrm{T}}(t) = D_i^{\mathrm{T}}(t) p_i^{\mathrm{T}}(t) \tau, \quad (4)$$

where $p_i^{\mathrm{T}}(t)$ represents the power consumption of the communication link of MD $i \in \mathcal{I}$ in time slot t.

B. Computation Model

The computation tasks can be executed either locally on the MD or on the EN. In this subsection, we provide a detailed explanation of these two cases.

1) Local Execution: We model the local execution by a queuing system consisting the computation queue and the MD processor. Let f_i denote the MD i's processing power (in cycle per second). When task $z_i(t)$ is assigned to the computation queue at the beginning of time slot $t \in \mathcal{T}$, we define $l_i^{\mathrm{C}}(t) \in \mathcal{T}$ as the time slot during which task $z_i(t)$ will either be processed or dropped. If the computation queue is empty, $l_i^{\mathrm{C}}(t) = 0$. Let $\delta_i^{\mathrm{C}}(t)$ denote the number of remaining time slots before processing task $z_i(t)$ in the computation queue. We have:

$$\delta_i^{\mathrm{C}}(t) = \left[\max_{t' \in \{0,1,\ldots,t-1\}} l_i^{\mathrm{C}}(t') - t + 1 \right]^+. \quad (5)$$

In the equation above, the term $\max_{t' \in \{0,1,\ldots,t-1\}} l_i^{\mathrm{C}}(t')$ denotes the time slot at which each existing task in the computation queue, which arrived before time slot t, is either processed or dropped. Consequently, $\delta_i^{\mathrm{C}}(t)$ denotes the number of time slots that task $z_i(t)$ should wait before being processed. We denote the time slot in which task $z_i(t)$ will be completely processed by $l_i^{\mathrm{C}}(t)$ if it is assigned to the computation queue for local processing in time slot t. We have

$$l_i^{\mathrm{C}}(t) = \min\left\{t + \delta_i^{\mathrm{C}}(t) + \lceil D_i^{\mathrm{C}}(t) \rceil - 1, t + \Delta_i(t) - 1\right\}. \quad (6)$$

The task $z_i(t)$ will be immediately dropped if its processing is not completed by the end of the time slot $t + \Delta_i(t) - 1$. In addition, we introduce $D_i^{\mathrm{C}}(t)$ as the number of time slots required to complete the processing of task $z_i(t)$ on MD $i \in \mathcal{I}$. It is given by:

$$D_i^{\mathrm{C}}(t) = \frac{\lambda_i(t)}{f_i \tau / \rho_i(t)}. \quad (7)$$

To compute the MD's energy consumption in the time slot $t \in \mathcal{T}$, we define $E_i^{\mathrm{L}}(t)$ as:

$$E_i^{\mathrm{L}}(t) = D_i^{\mathrm{C}}(t) p_i^{\mathrm{C}} \tau, \quad (8)$$

where $p_i^{\mathrm{C}} = 10^{-27}(f_i)^3$ represents the energy consumption of MD i's CPU frequency [39].

2) Edge Execution: We model the edge execution by the queues associated with MDs deployed at ENs. If computation task $z_i(t')$ is dispatched to EN j in time $t' < t$, we let $z_{i,j}^{\mathrm{E}}(t)$, $\lambda_{i,j}^{\mathrm{E}}(t)$ (in bits), and $\rho_{i,j}^{\mathrm{E}}(t)$ denote the unique index of the task, the task size, and the number of CPU cycles required per unit of the task in the i^{th} queue at EN j, respectively. We define $\eta_{i,j}^{\mathrm{E}}(t)$ (in bits) as the length of this queue at the end of time slot $t \in \mathcal{T}$. We refer to a queue as an active queue in a certain time slot if it is not empty. That being said, if at least one task is already in the queue from previous time slots or there is a task arriving at the queue, that queue is active. We define $\mathcal{B}_j(t)$ to denote the set of active queues at EN j in time slot t.

$$\mathcal{B}_j(t) = \left\{ i \,\middle|\, i \in \mathcal{I}, \lambda_{i,j}^{\mathrm{E}}(t) > 0 \ \text{ or } \ \eta_{i,j}^{\mathrm{E}}(t-1) > 0 \right\}. \quad (9)$$

We introduce $b_j(t) \triangleq |\mathcal{B}_j(t)|$ that represents the number of active queues in EN $j \in \mathcal{J}$ in time slot $t \in \mathcal{T}$. In each time slot $t \in \mathcal{T}$, the EN's processing power is divided among its active queues using a generalized processor sharing method [40]. Let variable f_j^{E} (in cycles per second) represent the computational capacity of EN j. Therefore, EN j can allocate computational capacity of $f_j^{\mathrm{E}}/(\rho_i(t) b_j(t))$ to each MD $i \in \mathcal{B}_j(t)$ during time slot t. To calculate the length of the computation queue for MD $i \in \mathcal{I}$ in EN $j \in \mathcal{J}$, we define $\omega_{i,j}(t)$ (in bits) to represent the number of bits from dropped tasks in that queue at the end of time slot $t \in \mathcal{T}$. The backlog of the queue, referred to as $\eta_{i,j}^{\mathrm{E}}(t)$ is given by:

$$\eta_{i,j}^{\mathrm{E}}(t) = \left[\eta_{i,j}^{\mathrm{E}}(t-1) + \lambda_{i,j}^{\mathrm{E}}(t) - \frac{f_j^{\mathrm{E}} \tau}{\rho_{i,j}^{\mathrm{E}}(t) b_j(t)} - \omega_{i,j}(t) \right]^+. \quad (10)$$

We also define $l_{i,j}^{\mathrm{E}}(t) \in \mathcal{T}$ as the time slot during which the offloaded task $z_{i,j}^{\mathrm{E}}(t)$ is either processed or dropped by EN j. Given the uncertain workload ahead at EN j, neither MD i nor EN j has information about $l_{i,j}^{\mathrm{E}}(t)$ until the corresponding task $z_{i,j}^{\mathrm{E}}(t)$ is either processed or dropped. Let $\hat{l}_{i,j}^{\mathrm{E}}(t)$ represent the time slot at which the execution of task $z_{i,j}^{\mathrm{E}}(t)$ starts. In mathematical terms, for $i \in \mathcal{I}$, $j \in \mathcal{J}$, and $t \in \mathcal{T}$, we have:

$$\hat{l}_{i,j}^{\mathrm{E}}(t) = \max\{t, \max_{t' \in \{0,1,\ldots,t-1\}} l_{i,j}^{\mathrm{E}}(t^{'}) + 1\}, \quad (11)$$

where $l_{i,j}^{\mathrm{E}}(0) = 0$. Indeed, the initial processing time slot of task $z_{i,j}^{\mathrm{E}}(t)$ at EN should not precede the time slot when the task was enqueued or when the previously arrived tasks were

processed or dropped. Therefore, $l^{\mathrm{E}}_{i,j}(t)$ is the time slot that satisfies the following constraints.

$$\sum_{t'=\hat{l}^{\mathrm{E}}_{i,j}(t)}^{l^{\mathrm{E}}_{i,j}(t)} \frac{f^{\mathrm{E}}_j \tau}{\rho^{\mathrm{E}}_{i,j}(t) b_j(t')} \mathbb{1}(i \in \mathcal{B}_j(t')) \geq \lambda^{\mathrm{E}}_{i,j}(t), \tag{12}$$

$$\sum_{t'=\hat{l}^{\mathrm{E}}_{i,j}(t)}^{l^{\mathrm{E}}_{i,j}(t)-1} \frac{f^{\mathrm{E}}_j \tau}{\rho^{\mathrm{E}}_{i,j}(t) b_j(t')} \mathbb{1}(i \in \mathcal{B}_j(t')) < \lambda^{\mathrm{E}}_{i,j}(t), \tag{13}$$

where $\mathbb{1}(z \in \mathbb{Z})$ is the indicator function. In particular, the total processing capacity that EN j allocates to MD i from the time slot $\hat{l}^{\mathrm{E}}_{i,j}(t)$ to the time slot $l^{\mathrm{E}}_{i,j}(t)$ should exceed the size of task $z^{\mathrm{E}}_{i,j}(t)$. Conversely, the total allocated processing capacity from the time slot $l^{\mathrm{E}}_{i,j}(t)$ to the time slot $l^{\mathrm{E}}_{i,j}(t)-1$ should be less than the task's size.

Additionally, we define $D^{\mathrm{E}}_{i,j}(t)$ to represent the quantity of processing time slots allocated to task $z^{\mathrm{E}}_{i,j}(t)$ when executed at EN j. This value is given by:

$$D^{\mathrm{E}}_{i,j}(t) = l^{\mathrm{E}}_{i,j}(t) - \hat{l}^{\mathrm{E}}_{i,j}(t). \tag{14}$$

We also define $E^{\mathrm{E}}_{i,j}(t)$ as the energy consumption of processing at EN j in time slot t by MD i. This can be calculated as:

$$E^{\mathrm{E}}_{i,j}(t) = \sum_{t'=\hat{l}^{\mathrm{E}}_{i,j}(t)}^{l^{\mathrm{E}}_{i,j}(t)} \frac{p^{\mathrm{E}}_j \tau}{b_j(t')} \mathbb{1}(i \in \mathcal{B}_j(t')), \tag{15}$$

where p^{E}_j is a constant value which denotes the energy consumption of the EN j's processor when operating at full capacity.

In addition to the energy consumed by EN j for task processing, we also take into account the energy consumed by the MD i's user interface in the standby state while waiting for task completion at the EN j. We define $E^{\mathrm{I}}_i(t)$ as the energy consumption associated with the user interface of MD $i \in \mathcal{I}$, which is given by

$$E^{\mathrm{I}}_i(t) = \sum_{j \in \mathcal{J}} y_{i,j}(t) D^{\mathrm{E}}_{i,j}(t) p^{\mathrm{I}}_i \tau, \tag{16}$$

where p^{I}_i is the standby energy consumption of MD $i \in \mathcal{I}$. Recall that $y_{i,j}(t)$ is the binary offloading indicator. Moreover, among all $j \in \mathcal{J}$, $y_{i,j} = 1$ only for one j which is the corresponding EN.

IV. Task Offloading Problem Formulation

Based on the introduced system model, we present the computation task offloading problem in this section. Our primary goal is to enhance each MD's QoE individually by taking the dynamic demands of MDs into account. To achieve this, we approach the optimization problem as an MDP, aiming to maximize the MD's QoE by striking a balance among key QoE factors, including task completion, task delay, and energy consumption. To prioritize QoE factors, we utilize the MD's energy level of the battery, which plays a crucial role in decision-making. Specifically, when an MD observes its state (e.g. task size, queue details, and battery status) and encounters a newly arrived task, it selects an appropriate action for that task. The selected action, based on the observed state, will result in enhanced QoE. Each MD strives to maximize its long-term QoE by optimizing the policy mapping from states to actions. In what follows, we first present the state space, action space, and QoE function, respectively. We then formulate the QoE maximization problem for each MD.

A. State Space

A state in our MDP represents a conceptual space that comprehensively describes the state of an MD facing the environment. We represent the MD i's state in time slot t as vector $\boldsymbol{s}_i(t)$ that includes the newly arrived task size, the queues information, the MD's energy level of the battery, and the workload history at the ENs. The MD observers this vector at the beginning of each time slot. The vector $\boldsymbol{s}_i(t)$ is defined as follows:

$$\boldsymbol{s}_i(t) = \Big(\lambda_i(t), \delta^{\mathrm{C}}_i(t), \delta^{\mathrm{T}}_i(t), \boldsymbol{\eta}^{\mathrm{E}}_i(t-1), \phi_i(t), \mathcal{H}(t)\Big), \tag{17}$$

where vector $\boldsymbol{\eta}^{\mathrm{E}}_i(t-1) = (\eta^{\mathrm{E}}_{i,j}(t-1))_{j \in \mathcal{J}}$ represents the queues length of MD i in ENs at the previous time slot and is computed by the MD according to (10). Since MDs are assumed to be battery-operated, their operation modes are aligned with those of real-world devices. Let $\phi_i(t)$ represent the battery level percentage of MD i at time slot t, where $\phi_i(t)$ is selected from the discrete set $\Phi = \{\phi_1, \phi_2, \phi_3\}$, corresponding to ultra power-saving, power-saving, and performance modes, respectively.

In addition, to predict future EN workloads, we define the matrix $\mathcal{H}(t)$ as historical data, indicating the number of active queues for all ENs. This data is recorded over T^{s} time slots, ranging from $t-T^{\mathrm{s}}$ to $t-1$, in $T^{\mathrm{s}} \times J$ matrix. For EN j workload history at i^{th} time slot from $T^{\mathrm{s}} - t$, we define $h_{i,j}(t)$ as:

$$h_{i,j}(t) = b_j(t - T^{\mathrm{s}} + i - 1). \tag{18}$$

EN $j \in \mathcal{J}$ broadcasts $b_j(t)$ at the end of each time slot.

We define vector $\mathcal{S}$ as the discrete and finite state space for each MD. The size of the set $\mathcal{S}$ is given by $\Lambda \times T^2 \times \mathcal{U} \times 3 \times I^{T^{\mathrm{s}} \times J}$, where $\mathcal{U}$ is the set of available queue length values at an EN over T time slots.

B. Action Space

The action space represents the agent's behavior and the decisions. In this context, we define $\boldsymbol{a}_i(t)$ to denote the action taken by MD $i \in \mathcal{I}$ in time slot $t \in \mathcal{T}$. These actions involve two decisions, (a) Offloading decision to determine whether or not to offload the task, and (b) Offloading target to determine the EN to send the offloaded tasks. Thus, the action of MD i in time slot t can be concisely expressed as the following action tuple:

$$\boldsymbol{a}_i(t) = (x_i(t), \boldsymbol{y}_i(t)), \tag{19}$$

where vector $\boldsymbol{y}_i(t) = (y_{i,j}(t))_{j \in \mathcal{J}}$ represents the selected EN for offloading this task. In Section V-B, we will discuss the size of this action space.

C. QoE Function

The QoE function reflects user satisfaction with task computation, whether using local or edge resources. QoE accounts for task completion rate, processing delay, and energy consumption. Each user may have unique QoE requirements [41]. Therefore, we define a multi-dimensional, adaptive structure to assess QoE, balancing these factors according to the MDs preferences, such as prioritizing reduced task delay or energy saving, which may vary over time [42], [43]. To capture personalized QoE requirements, we define QoE as a weighted sum of the above factors. Each MD dynamically adjusts these weights to reflect the importance of different factors based on its energy modes, which are performance, power-saving, and ultra-power-saving modes. We now calculate task delay and energy consumption and then introduce the associated cost and QoE function.

Given the selected action $\boldsymbol{a}_i(t)$ in the observed state $\boldsymbol{s}_i(t)$, we represent $\mathcal{D}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t))$ as the delay of task $z_i(t)$, which indicates the number of time slots from time slot t to the time slot in which task $z_i(t)$ is processed. It is calculated by:

$$\mathcal{D}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = (1 - x_i(t))\Big(l_i^{\mathrm{C}}(t) - t + 1\Big) + x_i(t)\Bigg(\sum_{j\in\mathcal{J}}\sum_{t'=t}^{T} \mathbb{1}\big(z_{i,j}^{\mathrm{E}}(t') = z_i(t)\big)\, l_{i,j}^{\mathrm{E}}(t') - t + 1\Bigg), \quad (20)$$

where $\mathcal{D}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = 0$ if task $z_i(t)$ is dropped. Correspondingly, we denote the energy consumption of task $z_i(t)$ when taking action $\boldsymbol{a}_i(t)$ in the observed state $\boldsymbol{s}_i(t)$ as $\mathcal{E}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t))$, which is:

$$\mathcal{E}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = (1 - x_i(t))E_i^{\mathrm{L}}(t) + x_i(t)\Bigg(E_i^{\mathrm{T}}(t) + E_i^{\mathrm{I}}(t) + \sum_{j\in\mathcal{J}}\sum_{t'=t}^{T} \mathbb{1}\big(z_{i,j}^{\mathrm{E}}(t') = z_i(t)\big) E_{i,j}^{\mathrm{E}}(t')\Bigg). \quad (21)$$

To define associated cost, we use a weighted sum of task delay $\mathcal{D}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t))$ and energy consumption $\mathcal{E}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t))$, where the MD dynamically adjusts the weights based on its energy level to reflect the preference for each factor. Given the delay and energy consumtion of task $z_i(t)$, we also define $\mathcal{C}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t))$ that denotes the assosiate cost of task $z_i(t)$ given the action $\boldsymbol{a}_i(t)$ in the state $\boldsymbol{s}_i(t)$.

$$\mathcal{C}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = \phi_i(t)\,\mathcal{D}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) + (1 - \phi_i(t))\,\mathcal{E}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)), \quad (22)$$

where $\phi_i(t)$ represents the MD i's energy level. When the MD is operating in performance mode, the primary focus is on minimizing task delays, thus the delay contributes more to the cost. On the other hand, when the MD switches to ultra power-saving mode, the main attention is directed toward reducing power consumption.

Finally, we define $\boldsymbol{q}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t))$ as the QoE associated with task $z_i(t)$ given the selected action $\boldsymbol{a}_i(t)$ and the observed state $\boldsymbol{s}_i(t)$. The QoE function is defined as follows:

$$\boldsymbol{q}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = \begin{cases} \mathcal{R} - \mathcal{C}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) & \text{if task } z_i(t) \text{ is processed,} \\ -\,\mathcal{E}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) & \text{if task } z_i(t) \text{ is dropped,} \end{cases} \quad (23)$$

where $\mathcal{R} > 0$ represents a constant reward for task completion. If $z_i(t) = 0$, then $\boldsymbol{q}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = 0$. Throughout the rest of this paper, we adopt the shortened notation $\boldsymbol{q}_i(t)$ to represent $\boldsymbol{q}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t))$.

D. Problem Formulation

We define the task offloading policy for MD $i \in \mathcal{I}$ as a mapping from its state to its corresponding action, denoted by i.e., $\pi_i : \mathcal{S} \to \mathcal{A}$. Especially, MD i determines an action $\boldsymbol{a}_i(t) \in \mathcal{A}$, according to policy π_i given the observed environment state $\boldsymbol{s}_i(t) \in \mathcal{S}$. The MD aims to find its optimal policy π_i^* which maximizes the long-term QoE,

$$\pi_i^* = \arg\ \max_{\pi_i} \mathbb{E}\left[\sum_{t\in\mathcal{T}} \gamma^{t-1} \boldsymbol{q}_i(t) \middle| \pi_i\right], \quad (24)$$

where $\gamma \in (0, 1]$ is a discount factor and determines the balance between instant QoE and long-term QoE. As γ approaches 0, the MD prioritizes QoE within the current time slot exclusively. Conversely, as γ approaches 1, the MD increasingly factors in the cumulative long-term QoE. The expectation $\mathbb{E}[\cdot]$ is taken into consideration of the time-varying system environments. Solving the optimization problem in (24) is particularly challenging due to the dynamic nature of the network. To address this challenge, we introduce a DRL-based offloading algorithm to learn the mapping between each state-action pair and their long-term QoE.

V. DRL-Based Offloading Algorithm

We now present QECO algorithm so as to address the distributed offloading decision-making of MDs. The aim is to empower MDs to identify the most efficient action that maximizes their long-term QoE. In the following, we introduce a neural network that characterizes the MD's state-action Q-values mapping, followed by a description of the information exchange between the MDs and ENs.

A. DQN-based Approach

We utilize the DQN technique to find the mapping between each state-action pair to Q-values in the formulated MDP. As shown in Fig. 2, each MD $i \in \mathcal{I}$ is equipped with a neural network comprising six layers. These layers include an input layer, an LSTM layer, two dense layers, an advantage-value (A&V) layer, and an output layer. The parameter vector θ_i of MD i's neural network is defined to maintain the connection weights and neuron biases across all layers. For MD $i \in \mathcal{I}$, we utilize the state information as the input of neural network. The state information $\lambda_i(t)$, $\delta_i^{\mathrm{C}}(t)$, $\delta_i^{\mathrm{T}}(t)$, $\phi_i(t)$, and $\boldsymbol{\eta}_i^{\mathrm{E}}(t-1)$ are directly passed to the dense layer, while the state information $\mathcal{H}(t)$ is first supplied to the LSTM layer and then the resulting output is sent to the dense layer. The role and responsibilities of each layer are detailed as follows.

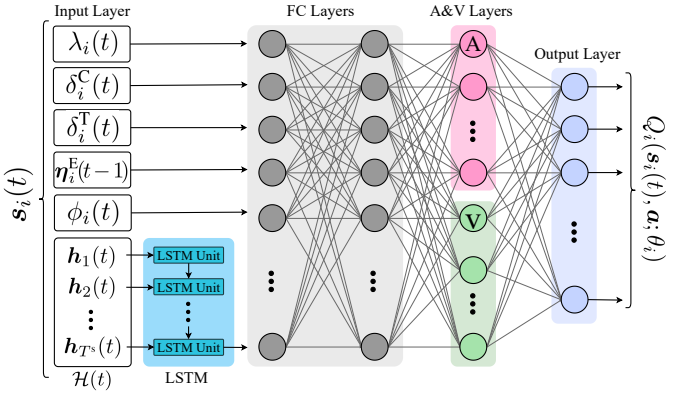


Fig. 2. The neural network of MD $i \in \mathcal{I}$, which characterize the Q-value of each action $\boldsymbol{a} \in \mathcal{A}$ under state $\boldsymbol{s}_i(t) \in \mathcal{S}$.

1) Predicting Workloads at ENs: In order to capture the dynamic behavior of workloads at the ENs, we employ the LSTM network [11]. This network maintains a memory state $\mathcal{H}(t)$ that evolves over time, enabling the neural network to predict future workloads at the ENs based on historical data. By taking the matrix $\mathcal{H}(t)$ as an input, the LSTM network learns the patterns of workload dynamics. The architecture of the LSTM consists of T^{s} units, each equipped with a set of hidden neurons, and it processes individual rows of the matrix $\mathcal{H}(t)$ sequentially. Through this interconnected design, MD tracks the variations in sequences from $\boldsymbol{h}_1(t)$ to $\boldsymbol{h}_{T^{\mathrm{s}}}(t)$, where vector $\boldsymbol{h}_i(t) = (h_{i,j}(t))_{j\in\mathcal{J}}$, thereby revealing workload fluctuations at the ENs across different time slots. The final LSTM unit produces an output that encapsulates the anticipated workload dynamics, and is then connected to the subsequent layer neurons for further learning.

2) State-Action Q-Value Mapping: The pair of dual dense layers plays a crucial role in learning the mapping of Q-values from the current state and the learned load dynamics to the corresponding actions. The dense layers consist of a cluster of neurons that employ rectified linear units (ReLUs) as their activation functions. In the initial dense layer, connections are established from the neurons in the input layer and the LSTM layer to each neuron in the dense layer. The resulting output of a neuron in the dense layer is connected to each neuron in the subsequent dense layer. In the second layer, the outputs from each neuron establish connections with all neurons in the A&V layers.

3) Dueling-DQN Approach for Q-Value Estimation: In the neural network architecture, the A&V layer and the output layer incorporate the principles of the dueling-DQN [10] to compute action Q-values. The fundamental concept of dueling-DQN involves two separate learning components: one for action-advantage values and another for state-value. This approach enhances Q-value estimation by separately evaluating the long-term QoE attributed to states and actions.

The A&V layer consists of two distinct dense networks referred to as network A and network V. Network A's role is to learn the action-advantage value for each action, while network V focuses on learning the state-value. For an MD $i \in \mathcal{I}$, we define $V_i(\boldsymbol{s}_i(t);\theta_i)$ and $A_i(\boldsymbol{s}_i(t),\boldsymbol{a};\theta_i)$ to denote the state-value and the action-advantage value of action $\boldsymbol{a} \in \mathcal{A}$ under state $\boldsymbol{s}_i(t) \in \mathcal{S}$, respectively. The parameter θ_i is responsible for determining these values, and it can be adjusted when training the QECO algorithm.

For an MD $i \in \mathcal{I}$, the A&V layer and the output layer collectively determine $Q_i(\boldsymbol{s}_i(t),\boldsymbol{a};\theta_i)$, representing the resulting Q-value under action $\boldsymbol{a} \in \mathcal{A}$ and state $\boldsymbol{s}_i(t) \in \mathcal{S}$, as follows:

$$Q_i(\boldsymbol{s}_i(t),\boldsymbol{a};\theta_i) = V_i(\boldsymbol{s}_i(t);\theta_i) + \left(A_i(\boldsymbol{s}_i(t),\boldsymbol{a};\theta_i) - \frac{1}{|\mathcal{A}|}\sum_{\boldsymbol{a}'\in\mathcal{A}}(A_i(\boldsymbol{s}_i(t),\boldsymbol{a}';\theta_i)\right), \quad (25)$$

where θ_i establishes a functional relationship that maps Q-values to pairs of state-action.

B. QoE-Oriented DRL-Based Algorithm

The QECO algorithm is meticulously designed to optimize the allocation of computational tasks between MDs and ENs. Since the training of neural networks imposes an extensive computational workload on MDs, we enable MDs to utilize ENs for training their neural networks, effectively reducing their computational workload. For each MD $i \in \mathcal{I}$, there is an associated EN, denoted as EN $j_i \in \mathcal{J}$, which assists in the training process. This EN possesses the highest transmission capacity among all ENs. We define $\mathcal{I}_j \subset \mathcal{I}$ as the set of MDs for which training is executed by EN $j \in \mathcal{J}$, i.e. $\mathcal{I}_j = \{i \in \mathcal{I} | j_i = j\}$. This approach is feasible due to the minimal information exchange and processing requirements for training compared to MD's tasks. The algorithms to be executed at MD $i \in \mathcal{I}$ and EN $j \in \mathcal{J}$ are given in Algorithms 1 and 2, respectively. The core concept involves training neural networks with MD experiences (i.e., state, action, QoE, next state) to map Q-values to each state-action pair. This mapping allows MD to identify the action in the observed state with the highest Q-value and maximize its long-term QoE.

In detail, EN $j \in \mathcal{J}$ maintains a replay buffer denotes as $\mathcal{M}_i$ with two neural networks for MD i: Net_i^{E}, denoting the evaluation network, and Net_i^{T}, denoting the target network, which have the same neural network architecture. However, they possess distinct parameter vectors θ_i^{E} and θ_i^{T}, respectively. Their Q-values are represented by $Q_i^{\mathrm{E}}(\boldsymbol{s}_i(t),\boldsymbol{a};\theta_i^{\mathrm{E}})$ and $Q_i^{\mathrm{T}}(\boldsymbol{s}_i(t),\boldsymbol{a};\theta_i^{\mathrm{T}})$ for MD $i \in \mathcal{I}_j$, respectively, associating the action $\boldsymbol{a} \in \mathcal{A}$ under the state $\boldsymbol{s}_i(t) \in \mathcal{S}$. The replay buffer records the observed experience $(\boldsymbol{s}_i(t),\boldsymbol{a}_i(t),\boldsymbol{q}_i(t),\boldsymbol{s}_i(t+1))$ of MD i. Moreover, Net_i^{E} is responsible for action selection, while Net_i^{T} characterizes the target Q-values, which represent the estimated long-term QoE resulting from an action in the observed state. The target Q-value serves as the reference for updating the network parameter vector θ_i^{E}. This update occurs through the minimization of disparities between the Q-values under Net_i^{E} and Net_i^{T}. In the following, we introduce the offloading decision algorithm of MD $i \in \mathcal{I}$ and the training process algorithm running in EN $j \in \mathcal{J}$.

1) Offloading Decision Algorithm at MD $i \in \mathcal{I}$: We analyze a series of episodes, where N^{ep} denotes the number of them. At the beginning of each episode, if MD $i \in \mathcal{I}$ receives a new task $z_i(t)$, it initializes the state $\mathcal{S}_i(1)$ and sends an *UpdateRequest*

Algorithm 1 Offloading Decision Algorithm at MD $i \in \mathcal{I}$

Input: state space $\mathcal{S}$, action space $\mathcal{A}$, parameters vector θ_i^{E}
Output: MD $i \in \mathcal{I}$ experience $(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t), \boldsymbol{q}_i(t), \boldsymbol{s}_i(t+1))$

for episode 1 to N^{ep} **do**
  Initialize $\boldsymbol{s}_i(1)$
  for time slot $t \in \mathcal{T}$ **do**
    if MD i receives a new task $z_i(t)$ **then**
      Send an *UpdateRequest* to EN j_i;
      Receive network parameter vector θ_i^{E};
      Select action $\boldsymbol{a}_i(t)$ based on (26);
    end if
    Observe a set of QoEs $\{\boldsymbol{q}_i(t'), t' \in \mathcal{F}_i^t\}$;
    Observe the next state $\boldsymbol{s}_i(t+1)$;
    for each task $z_i(t')$ where $t' \in \mathcal{F}_i^t$ **do**
      Send $(\boldsymbol{s}_i(t'), \boldsymbol{a}_i(t'), \boldsymbol{q}_i(t'), \boldsymbol{s}_i(t'+1))$ to EN j_i;
    end for
  end for
end for

to EN j_i. After receiving the requested vector θ_i^{E} of Net_i^{E} from EN j_i, MD i chooses the following action for task $z_i(t)$.

$$\boldsymbol{a}_i(t) = \begin{cases} \arg\max_{\boldsymbol{a} \in \mathcal{A}} Q_i^{\mathrm{E}}(\boldsymbol{s}_i(t), \boldsymbol{a}; \theta_i^{\mathrm{E}}), & \text{w.p. } 1-\boldsymbol{\epsilon}, \\ \text{pick a random action from } \mathcal{A}, & \text{w.p. } \boldsymbol{\epsilon}, \end{cases} \tag{26}$$

where w.p. stands for with probability, and $\boldsymbol{\epsilon}$ represents the random exploration probability. The value of $Q_i^{\mathrm{E}}(\boldsymbol{s}_i(t), \boldsymbol{a}; \theta_i^{\mathrm{E}})$ indicates the Q-value under the parameter θ_i^{E} of the neural network Net_i^{E}. Specifically, the MD with a probability of $1-\boldsymbol{\epsilon}$ selects the action associated with the highest Q-value under Net_i^{E} in the observed state $\boldsymbol{s}_i(t)$.

In the next time slot $t+1$, MD i observes the state $\mathcal{S}_i(t+1)$. However, due to the potential for tasks to extend across multiple time slots, QoE $\boldsymbol{q}_i(t)$ associated with task $z_i(t)$ may not be observable in time slot $t+1$. On the other hand, MD i may observe a group of QoEs associated with some tasks $z_i(t')$ in time slots $t' \leq t$. For each MD i, we define the set $\mathcal{F}_i^t \subset \mathcal{T}$ to denote the time slots during which each arriving task $z_i(t')$ is either processed or dropped in time slot t, as given by:

$$\mathcal{F}_i^t = \Bigg\{ t' \Bigg| \; t' \leq t,\ \lambda_i(t') > 0,\ (1 - x_i(t'))\, l_i^{\mathrm{C}}(t') + x_i(t') \sum_{j \in \mathcal{J}} \sum_{n=t'}^{t} \mathbb{1}(z_{i,j}^{\mathrm{E}}(n) = z_i(t')) l_{i,j}^{\mathrm{E}}(n) = t \Bigg\}.$$

Therefore, MD i observes a set of QoEs $\{\boldsymbol{q}_i(t') \mid t' \in \mathcal{F}_i^t\}$ at the beginning of time slot $t+1$, where the set $\mathcal{F}_i^t$ for some $i \in \mathcal{I}$ can be empty. Subsequently, MD i sends its experience $(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t), \boldsymbol{q}_i(t), \boldsymbol{s}_i(t+1))$ to EN j_i for each task $z_i(t')$ in $t' \in \mathcal{F}_i^t$.

2) Training Process Algorithm at EN $j \in \mathcal{J}$: Upon initializing the replay buffer $\mathcal{M}_i$ with the neural networks Net_i^{E} and Net_i^{T} for each MD $i \in \mathcal{I}_j$, EN $j \in \mathcal{J}$ waits for messages from the MDs in the set $\mathcal{I}_j$. When EN j receives an *UpdateRequest* signal from an MD $i \in \mathcal{I}_j$, it responds by transmitting the updated parameter vector θ_i^{E}, obtained from Net_i^{E}, back to MD i. On the other side, if EN j receives an experience $(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t), \boldsymbol{q}_i(t), \boldsymbol{s}_i(t+1))$ from MD $i \in \mathcal{I}_j$, the

Algorithm 2 Training Process Algorithm at EN $j \in \mathcal{J}$

Input: experience $(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t), \boldsymbol{q}_i(t), \boldsymbol{s}_i(t+1))$ from MD $i \in \mathcal{I}$
Output: parameters vector θ_i^{E}

1: Initialize replay buffer $\mathcal{M}_i$ for each MD $i \in \mathcal{I}_j$;
2: Initialize Net_i^{E} and Net_i^{T} with random parameters θ_i^{E} and θ_i^{T} respectively, for each MD $i \in \mathcal{I}_j$;
3: Set Count := 0
4: **while** True **do** ▷ *infinite loop*
5: **if** receive an *UpdateRequest* from MD $i \in \mathcal{I}_j$ **then**
6: Send θ_i^{E} to MD $i \in \mathcal{I}$;
7: **end if**
8: **if** an experience $(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t), \boldsymbol{q}_i(t), \boldsymbol{s}_i(t+1))$ is received
9: from MD $i \in \mathcal{I}_j$ **then**
10: Store $(\boldsymbol{s}_i(t'), \boldsymbol{a}_i(t'), \boldsymbol{q}_i(t'), \boldsymbol{s}_i(t'+1))$ in $\mathcal{M}_i$;
11: Get a collection of experiences $\mathcal{I}$ from $\mathcal{M}_i$;
12: **for** each experience $i \in \mathcal{I}$ **do**
13: Get experience $(\boldsymbol{s}_i(n), \boldsymbol{a}_i(n), \boldsymbol{q}_i(n), \boldsymbol{s}_i(n+1))$;
14: Generate $\hat{Q}_{i,n}^{\mathrm{T}}$ according to (27);
15: **end for**
16: Set vector $\hat{\mathbf{Q}}_i^{\mathrm{T}} := (\hat{Q}_{i,n}^{\mathrm{T}})_{n \in \mathcal{N}}$;
17: Update θ_i^{E} to minimize $L(\theta_i^{\mathrm{E}}, \hat{\mathbf{Q}}_i^{\mathrm{T}})$ in (28);
18: Count := Count + 1;
19: **if** mod(Count, *ReplaceThreshold*) = 0 **then**
20: $\theta_i^{\mathrm{T}} := \theta_i^{\mathrm{E}}$;
21: **end if**
22: **end if**
23: **end while**

EN stores this experience in the replay buffer $\mathcal{M}_i$ associated with that MD.

The EN randomly selects a sample collection of experiences from the replay buffer, denoted as $\mathcal{N}$. For each experience $n \in \mathcal{N}$, it calculates the value of $\hat{Q}_{i,n}^{\mathrm{T}}$. This value represents the QoE in experience n and includes a discounted Q-value of the action anticipated to be taken in the subsequent state of experience n, according to the network Net_i^{T}, given by

$$\hat{Q}_{i,n}^{\mathrm{T}} = \boldsymbol{q}_i(n) + \gamma Q_i^{\mathrm{T}}(\boldsymbol{s}_i(n+1)), \tilde{\boldsymbol{a}}_n; \theta_i^{\mathrm{T}}), \tag{27}$$

where $\tilde{\boldsymbol{a}}_n$ denotes the optimal action for the state $\boldsymbol{s}_i(n+1)$ based on its highest Q-value under Net_i^{E}, as given by:

$$\tilde{\boldsymbol{a}}_n = \arg \max_{\boldsymbol{a} \in \mathcal{A}} Q_i^{\mathrm{E}}(\boldsymbol{s}_i(n+1), \boldsymbol{a}; \theta_i^{\mathrm{E}}). \tag{28}$$

In particular, regarding experience n, the target-Q value $\hat{Q}_{i,n}^{\mathrm{T}}$ represents the long-term QoE for action $\boldsymbol{a}_i(n)$ under state $\boldsymbol{s}_i(n)$. This value corresponds to the QoE observed in experience n, as well as the approximate expected upcoming QoE. Based on the set $\mathcal{N}$, the EN computes the vector $\hat{\mathbf{Q}}_i^{\mathrm{T}} = (\hat{Q}_{i,n}^{\mathrm{T}})_{n \in \mathcal{N}}$ and trains the MD's neural network (Lines 11-21 of Algorithm 2) to update the parameter vector θ_i^{E} in Net_i^{E} for the next MD's *UpdateRequest*. The key idea of updating Net_i^{E} is to minimize the disparity in Q-values between Net_i^{E} and Net_i^{T}, as indicated by the following loss function:

$$L(\theta_i^{\mathrm{E}}, \hat{\mathbf{Q}}_i^{\mathrm{T}}) = \frac{1}{|\mathcal{N}|} \sum_{n \in \mathcal{N}} \left(Q_i^{\mathrm{E}}(\boldsymbol{s}_i(n), \boldsymbol{a}_i(n); \theta_i^{\mathrm{E}}) - \hat{Q}_{i,n}^{\mathrm{T}} \right)^2. \tag{29}$$

TABLE II
SIMULATION PARAMETERS

Parameter	Value
Computation capacity of MD f_i	2.6 GHz
Computation capacity of EN f_j^{E}	42.8 GHz
Transmission capacity of MD $r_{i,j}(t)$	14 Mbps
Task arrival rate	150 Task/sec
Size of task $\lambda_i(t)$	$\{1.0, 1.1, \ldots, 7.0\}$ Mbits
Required CPU cycles of task $\rho_i(t)$	$\{0.197, 0.297, 0.397\} \times 10^3$
Deadline of task Δ_i	10 time slots (1 Sec)
Percentage of MD i's battery level $\phi_i(t)$	$\{25, 50, 75\}$
Computation power of EN p_j^{E}	5 W
Transmission power of MD p_i^{T}	2.3 W
Standby power of MD p_i^{I}	0.1 W

In every *ReplaceThreshold* iterations, the update of Net_i^{T} will involve duplicating the parameters from Net_i^{E} ($\theta_i^T = \theta_i^E$). The objective is to consistently update the network parameter θ_i^T in Net_i^{T}, which enhances the approximation of the long-term QoE when computing the target Q-values in (27).

3) Computational Complexity: The computational complexity of the QECO algorithm is determined by the number of experiences required to discover the optimal offloading policy. Each experience involves backpropagation for training, which has a computational complexity of $\mathcal{O}(C)$, where C represents the number of multiplication operations in the neural network. During each training round triggered by the arrival of a new task, a sample collection of experiences of size $|\mathcal{N}|$ is utilized from the replay buffer. Since the training process encompasses N^{ep} episodes and there are K expected tasks in each episode, the computational complexity of the proposed algorithm is $\mathcal{O}(N^{\mathrm{ep}}K|\mathcal{N}|C)$, which is polynomial. Given the integration of neural networks for function approximation, the convergence guarantee of the DRL algorithm remains an open problem. In this work, we will empirically evaluate the convergence of the proposed algorithm in Section VI-B.

VI. PERFORMANCE EVALUATION

In this section, we first present the simulation setup and training configuration. We then evaluate the performance of QECO in comparison to three baseline schemes in addition to the existing state-of-the-art works [12], [13]. We further analyze the convergence of our proposed algorithm.

A. Simulation Setup

We consider a MEC environment with 50 MDs and 5 ENs, similar to [44]. We also follow the model presented in [21] to determine the energy consumption. All the parameters are given in Table II. Since there is no real dataset due to the challenges of capturing representative samples over extended periods, we use a simulated MEC system that enables DRL agents to continuously gather experience and improve their performance based on system feedback (e.g., QoE). To train the MDs' neural networks, we adopt a scenario comprising 1000 episodes. Each episode contains 100 time slots, each of length 0.1 second. The QECO algorithm incorporates real-time experience into its training process to continuously enhance the offloading strategy. Specifically, we employ a batch size of 16, maintain a fixed learning rate of 0.001, and set the discount factor γ to 0.9. The probability of random exploration gradually decreases from an initial value 1, progressively approaching 0.01, all of which is facilitated by an RMSProp optimizer. The algorithm's source code is available at [45].

We use the following methods as benchmarks.

1) **Local Computing (LC):** The MDs execute all of their computation tasks using their own computing capacity.
2) **Full Offloading (FO):** Each MD dispatches all of its computation tasks while choosing the offloading target randomly.
3) **Random Decision (RD):** In this approach, when an MD receives a new task, it randomly makes the offloading decisions and selects the offloading target if it decides to dispatch the task.
4) **PGOA** [12]: This existing method is a distributed optimization algorithm designed for delay-sensitive tasks in an environment where MDs interact strategically with multiple ENs.
5) **DCDRL** [13]: This method is based on the actor-critic framework [46], which underpins many state-of-the-art DRL algorithms, such as DDPG [47], and PPO [48]. DCDRL is designed for distributed computation offloading in a queuing-based MEC environment.

B. Performance Comparison and Convergence

We first evaluate the number of completed tasks when comparing our proposed QECO algorithm with the other four schemes. As illustrated in Fig. 3 (a), the QECO algorithm consistently outperforms the benchmark methods when we vary the task arrival rate. At a lower task arrival rate (i.e., 50), most of the methods demonstrate similar proficiency in completing tasks. However, as the task arrival rate increases, the efficiency of QECO becomes more evident. Specifically, when the task arrival rate increases to 250, our algorithm can increase the number of completed tasks by 72.1%, 46.8%, and 28.6% compared to RD, PGOA, and DCDRL, respectively. Similarly, in Fig. 3 (b), as the number of MDs increases, QECO shows significant improvements in the number of completed tasks compared to other methods, especially when faced with a large number of MDs. When there are 110 MDs, our proposed algorithm can effectively increase the number of completed tasks by at least 22.8% comparing with other methods. This achievement is attributed to the QECO's ability to effectively handle unknown workloads and prevent congestion at the ENs.

Figs. 4 (a) and 4 (b) illustrate the overall energy consumption for different values of task arrival rate and the number of MDs, respectively. At the lower task arrival rate, the total energy consumption of all methods is close to each other. The total energy consumption increases when we have a higher task arrival rate. As can be observed from Fig. 4 (a), at task arrival rate 450, QECO effectively reduces overall energy consumption by 18.6%, 15.5%, and 13.9% compared to RD, PGOA, and DCDRL, respectively, as it takes into account the energy

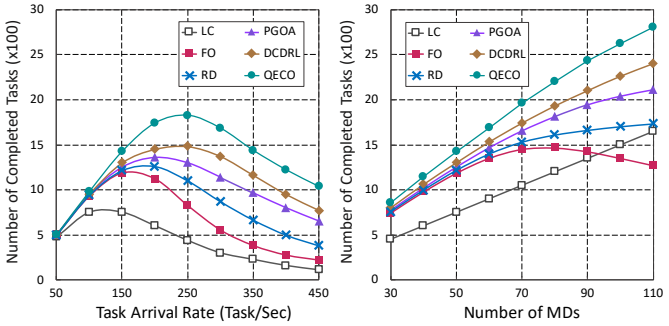


Fig. 3. The number of completed tasks under different computation workloads: (a) task arrival rate; (b) the number of MDs.

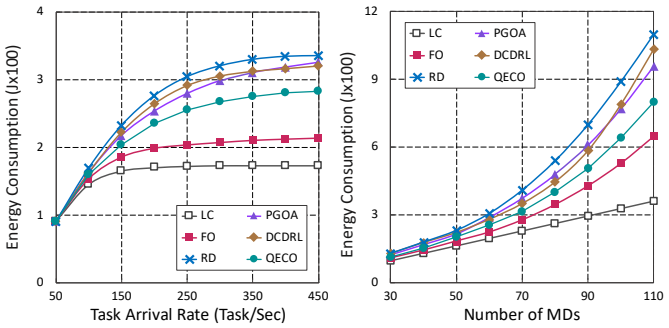


Fig. 4. The overall energy consumption under different computation workloads: (a) task arrival rate; (b) the number of MDs.

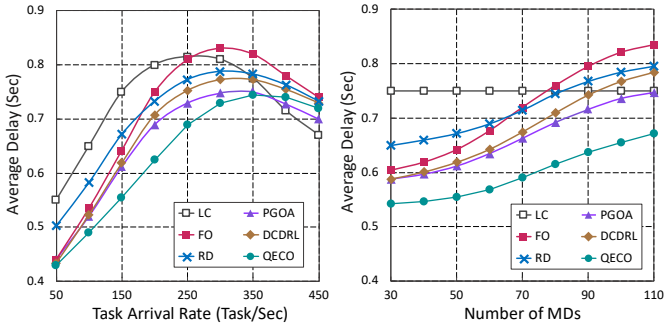


Fig. 5. The average delay under different computation workloads: (a) task arrival rate; (b) the number of MDs.

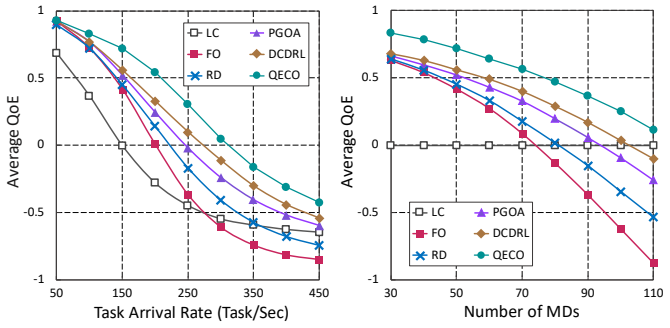


Fig. 6. The average QoE under different computation workloads: (a) task arrival rate; (b) the number of MDs.

level of the MD in its decision-making process. However, it consumes more energy compared to LC and FO because they do not utilize all computing resources. In particular, LC only uses the MD's computational resources, while FO utilizes the allocated EN computing resources. In Fig. 4 (b), an increasing trend in overall energy consumption is observed as the number of MDs increases since the number of resources available in the system increases, which leads to higher energy consumption. The QECO algorithm consistently outperforms other methods in overall energy consumption, especially when there are a large number of MDs. Specifically, QECO demonstrates a 27.4%, 16.7%, and 23.5% reduction in overall energy consumption compared to RD, PGOA, and DCDRL, respectively, when the number of MDs increases to 110.

As shown in Fig. 5 (a), the QECO algorithm maintains a lower average delay compared to other methods as the task arrival rate increases from 50 to 350. Specifically, when the task arrival rate is 200, it reduces the average delay by at least 12.4% compared to other methods. However, for task arrival rates exceeding 350, QECO may experience a higher average delay compared to some of the other methods. This can be attributed to the fact that the other algorithms drop more tasks while our proposed algorithm is capable of completing a higher number of tasks, potentially leading to an increase in average delay. In Fig. 5 (b), as the number of MDs increases, we observe a rising trend in the average delay. It can be inferred that an increase in computational load in the system can lead to higher queuing delays and computations at ENs. Considering the QECO's ability to schedule workloads, when the number of MDs increases from 30 to 110, it consistently maintains a lower average delay which is at least 8.3% less than the other methods.

We further investigate the overall improvement achieved by the QECO algorithm in comparison to other methods in terms of the average QoE. This metric signifies the advantages MDs obtain by utilizing different algorithms. Fig. 6 (a) shows the average QoE for different values of the task arrival rate. This figure highlights the superiority of the QECO algorithm in providing MDs with an enhanced experience. At lower task arrival rates (i.e., 50-150), QECO maintains an average QoE of at least 0.72, while the other methods experience a steeper decline, with average QoE dropping to 0–0.56. Specifically, when the task arrival rate is 200, QECO improves the average QoE by at least 65.7% compared to other methods. As task arrival rates increase to 300, the average QoE significantly decreases for all methods due to increased competition for resources in the MEC system. However, QECO still maintains a positive average QoE, while other methods fall to negative values. At higher task arrival rates (i.e., 350–450), QECO achieves 42.6%, 28.5%, and 22.6% improvement in QoE compared to RD, PGOA, and DCDRL, respectively.

Fig. 6 (b) illustrates the average QoE when we increase the number of MDs. The EN's workload grows when there are a larger number of MDs, leading to a reduction in the average QoE of all methods except LC. However, QECO effectively manages the uncertain load at the ENs. When the number of MDs increases from 30 to 110, QECO consistently maintains at least a 24.8% higher QoE compared to the other

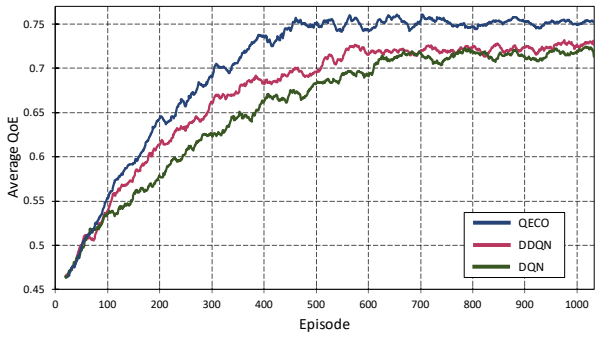
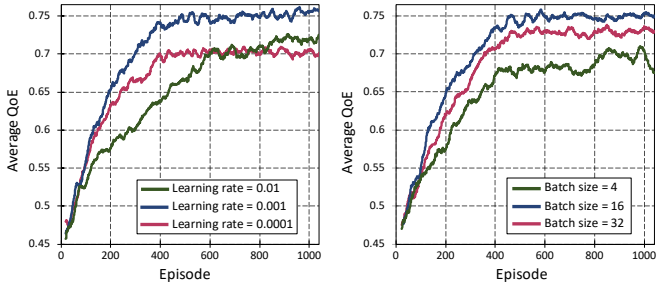


Fig. 7. The convergence of the average QoE across episodes under different DQN-based methods.



(a) (b)

Fig. 8. The convergence of the average QoE across episodes under different hyper-parameters: (a) Learning rate; (b) Batch size.

methods. Specifically, at a moderate number of 70 MDs, QECO achieves an average QoE of 0.57, showing a 70.3% and 44.7% improvement compared to PGOA and DCDRL, respectively. It is worth noting that although improvements in each of the QoE factors can contribute to enhancing system performance, it is essential to consider the user's demands in each time slot. Therefore, the key difference between QECO and other methods is that it prioritizes users' demands, enabling it to strike an appropriate balance among them, ultimately leading to a higher QoE for MDs.

We finally delve into the investigation of the convergence performance of the QECO algorithm in Fig. 7 and Fig. 8. To validate the effectiveness of the QECO algorithm, we assess its convergence rate compared to the vanilla DQN and DDQN configurations [9], measured by the average QoE across episodes. As shown in Fig. 7, the MD's network progressively learns efficient policies over the episodes, ultimately stabilizing as it approaches convergence. Specifically, DQN, DDQN, and our proposed QECO algorithm converge after approximately 650, 550, and 450 iterations, respectively. The QECO algorithm demonstrates the faster convergence while achieving a higher average QoE than the other methods. This underscores the beneficial impact of workload prediction by the LSTM network and highlights its effectiveness in efficiently utilizing the computing resources of MDs and ENs.

Furthermore, we explore the impact of two main hyper-parameters on the convergence speed and the converged result of the proposed algorithm. Fig. 8 (a) illustrates the convergence of the proposed algorithm under different learning rates, where the learning rate regulates the step size per iteration towards minimizing the loss function. The QECO algorithm achieves an average QoE of 0.75 when the learning rate is 0.001, indicating relatively rapid convergence. However, with smaller learning rates (e.g., 0.0001) or larger values (e.g., 0.01), a slower convergence is observed. Fig. 8 (b) shows the convergence of the proposed algorithm under different batch sizes, which refer to the number of sampled experiences in each training round. An improvement in convergence performance is observed as the batch size increases from 4 to 16. However, further increasing the batch size from 16 to 32 does not notably enhance the converged QoE or convergence speed. Hence, a batch size of 16 may be more appropriate for training processes.

VII. Conclusion

In this paper, we focused on addressing the challenge of offloading in MEC systems, where strict task processing deadlines and energy constraints adversely impact system performance. We formulated an optimization problem that aims to maximize the QoE of each MD individually, while QoE reflects the energy consumption and task completion delay. To address the dynamic and uncertain mobile environment, we proposed a QoE-oriented DRL-based computation offloading algorithm called QECO. Our proposed algorithm empowers MDs to make offloading decisions without relying on knowledge about task models or other MDs' offloading decisions. The QECO algorithm not only adapts to the uncertain dynamics of load levels at ENs, but also effectively manages the ever-changing system environment. Through extensive simulations, we showed that QECO outperforms several established benchmark techniques, while demonstrating a rapid training convergence. Specifically, QECO increases the average user's QoE by 29.8% – 37.1% compared to several existing algorithms. This advantage can lead to improvements in key performance metrics, including task completion rate, task delay, and energy consumption, under different system conditions and varying user demands.

There are multiple directions for future work. A complementary approach involves extending the task model by considering interdependencies among tasks. This can be achieved by incorporating a task call graph representation. Furthermore, in order to accelerate the learning of optimal offloading policies, it will be beneficial to take advantages of federated learning techniques in the training process. This will allow MDs to collectively contribute to improving the offloading model and enable continuous learning when new MDs join the network.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tutor.*, vol. 19, no. 4, pp. 2322–2358, Aug 2017.

[2] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug 2019.

[3] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *J. Syst. Archit.*, vol. 98, pp. 289–330, Sep 2019.

[4] A. Kaur and A. Godara, "Machine learning empowered green task offloading for mobile edge computing in 5G networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 21, no. 1, pp. 810–820, Feb 2024.

[5] H. Shah-Mansouri and V. W. Wong, "Hierarchical fog-cloud computing for IoT systems: A computation offloading game," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 3246–3257, Aug 2018.

[6] C. Jiang, X. Cheng, H. Gao, X. Zhou, and J. Wan, "Toward computation offloading in edge computing: A survey," *IEEE Access*, vol. 7, pp. 131 543–131 558, Aug 2019.

[7] L. Wu, P. Sun, H. Chen, Y. Zuo, Y. Zhou, and Y. Yang, "NOMA-enabled multiuser offloading in multicell edge computing networks: A coalition game based approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 11, no. 2, pp. 2170–2181, Mar 2024.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb 2015.

[9] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[10] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. of International Conference on Machine Learning*. New York, NY, Jun 2016.

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

[12] L. Yang, H. Zhang, X. Li, H. Ji, and V. C. Leung, "A distributed computation offloading strategy in small-cell networks integrated with mobile edge computing," *IEEE ACM Trans. Netw.*, vol. 26, no. 6, pp. 2762–2773, Dec 2018.

[13] 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.

[14] B. Zhang, F. Xiao, and L. Wu, "Offline reinforcement learning for asynchronous task offloading in mobile edge computing," *IEEE Trans. Netw. Serv. Manag.*, vol. 21, no. 1, pp. 939–952, Feb 2024.

[15] C. Sun, X. Li, C. Wang, Q. He, X. Wang, and V. C. Leung, "Hierarchical deep reinforcement learning for joint service caching and computation offloading in mobile edge-cloud computing," *accepted for publication in IEEE Trans. Services Computing*, Aug 2024.

[16] Y. Liu, J. Yan, and X. Zhao, "Deep reinforcement learning based latency minimization for mobile edge computing with virtualization in maritime uav communication network," *IEEE Trans. Veh. Technol.*, vol. 71, no. 4, pp. 4225–4236, Apr 2022.

[17] H. Tang, H. Wu, G. Qu, and R. Li, "Double deep q-network based dynamic framing offloading in vehicular edge computing," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 3, pp. 1297–1310, Jun 2023.

[18] X. Li, L. Huang, H. Wang, S. Bi, and Y.-J. A. Zhang, "An integrated optimization-learning framework for online combinatorial computation offloading in mec networks," *IEEE Wirel. Commun.*, vol. 29, no. 1, pp. 170–177, Feb 2022.

[19] Z. Wei, B. Li, R. Zhang, X. Cheng, and L. Yang, "Many-to-many task offloading in vehicular fog computing: A multi-agent deep reinforcement learning approach," *IEEE Trans. Mob. Comput.*, vol. 23, no. 3, pp. 2107–2122, Mar 2024.

[20] M. S. Munir, N. H. Tran, W. Saad, and C. S. Hong, "Multi-agent meta-reinforcement learning for self-powered and sustainable edge computing systems," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 3, pp. 3353–3374, Sep 2021.

[21] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1517–1530, Jun 2021.

[22] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang, "Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 12 175–12 186, Oct 2020.

[23] S. Chouikhi, M. Esseghir, and L. Merghem-Boulahia, "Energy-efficient computation offloading based on multi-agent deep reinforcement learning for industrial internet of things systems," *IEEE Internet Things J.*, vol. 11, no. 7, pp. 12 228–12 239, Apr 2024.

[24] G. Wu, X. Chen, Y. Shen, Z. Xu, H. Zhang, S. Shen, and S. Yu, "Combining lyapunov optimization with actor-critic networks for privacy-aware iiot computation offloading," *IEEE Internet Things J.*, May 2024.

[25] H. Huang, Q. Ye, and Y. Zhou, "Deadline-aware task offloading with partially-observable deep reinforcement learning for multi-access edge computing," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 6, pp. 3870–3885, Nov 2022.

[26] L. Liao, Y. Lai, F. Yang, and W. Zeng, "Online computation offloading with double reinforcement learning algorithm in mobile edge computing," *J. Parallel Distrib. Comput.*, vol. 171, pp. 28–39, Jan 2023.

[27] G. Wu, Z. Xu, H. Zhang, S. Shen, and S. Yu, "Multi-agent drl for joint completion delay and energy consumption with queuing theory in mec-based iiot," *J. Parallel Distrib. Comput.*, vol. 176, pp. 80–94, Jun 2023.

[28] Y. Gong, H. Yao, J. Wang, M. Li, and S. Guo, "Edge intelligence-driven joint offloading and resource allocation for future 6G Industrial Internet of Things," *accepted for publication in IEEE Trans. Netw. Sci. Eng.*, Dec 2024.

[29] Z. Liu, Y. Zhao, J. Song, C. Qiu, X. Chen, and X. Wang, "Learn to coordinate for computation offloading and resource allocation in edge computing: A rational-based distributed approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 5, pp. 3136–3151, Sep 2022.

[30] H. She, L. Yan, and Y. Guo, "Efficient end-edge-cloud task offloading in 6g networks based on multi-agent deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 11, no. 11, pp. 20 260–20 270, Jun 2024.

[31] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Trans. Mob. Comput.*, vol. 22, no. 6, pp. 3425–3443, Jun 2023.

[32] G. Wu, X. Chen, Z. Gao, H. Zhang, S. Yu, and S. Shen, "Privacy-preserving offloading scheme in multi-access mobile edge computing based on madrl," *J. Parallel Distrib. Comput.*, vol. 183, p. 104775, Jan 2024.

[33] M. Bolourian and H. Shah-Mansouri, "Deep Q-learning for minimum task drop in SWIPT-enabled mobile-edge computing," *IEEE Wireless Commun. Letters*, vol. 13, no. 3, pp. 894–898, Mar 2024.

[34] J. Chen, H. Xing, Z. Xiao, L. Xu, and T. Tao, "A drl agent for jointly optimizing computation offloading and resource allocation in mec," *IEEE Internet Things J*, vol. 8, no. 24, pp. 17 508–17 524, Dec 2021.

[35] G. Wu, H. Wang, H. Zhang, Y. Zhao, S. Yu, and S. Shen, "Computation offloading method using stochastic games for software-defined-network-based multiagent mobile edge computing," *IEEE Internet Things J.*, vol. 10, no. 20, pp. 17 620–17 634, Oct 2023.

[36] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 11, pp. 5141–5152, Nov 2019.

[37] F. Wang, F. Wang, J. Liu, R. Shea, and L. Sun, "Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, Aug 2020, pp. 2499–2508.

[38] C. Zhang, H. Pang, J. Liu, S. Tang, R. Zhang, D. Wang, and L. Sun, "Toward edge-assisted video content intelligent caching with long short-term memory learning," *IEEE access*, vol. 7, pp. 152 832–152 846, Oct 2019.

[39] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec 2016.

[40] A. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Trans. Netw*, vol. 1, no. 3, pp. 344–357, Jun 1993.

[41] Y. Wang, W. Zhou, and P. Zhang, "Qoe management in wireless networks," *Springer*, pp. XII, 60, Aug 2016.

[42] 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.

[43] F. Wang, C. Zhang, J. Liu, Y. Zhu, H. Pang, and L. Sun, "Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized qoe," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, May 2019, pp. 910–918.

[44] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mob. Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun 2022.

[45] I. Rahmati, "QECO source code GitHub repository," https://github.com/ImanRHT/QECO, 2024.

[46] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Adv Neural Inf Process Syst*, vol. 12, pp. 1008–1014, Apr 1999.

[47] T. P. Lillicrap, "Continuous control with deep reinforcement learning," *4th International Conference on Learning Representations, ICLR*, vol. abs/1509.02971, May 2016.

[48] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, Jul 2017.