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

Iman Rahmati†, Hamed Shah-Mansouri‡, and Ali Movaghar†
†Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
‡Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran
email: {iman.rahmati, hamedsh, movaghar}@sharif.edu

***Abstract*—In the realm of mobile edge computing (MEC), the focal point is the efficient computation task offloading problem, which 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 responsive and reliable services. This challenge stands as one of the most primary key factors contributing to handling dynamic load uncertainties. The decentralized determination of offloading decisions poses a challenging problem for each mobile device. 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 user's QoE individually. We propose a QoE-oriented computation offloading (QOCO) algorithm based on deep reinforcement learning (DRL) that empowers mobile devices to make their offloading decisions without requiring knowledge of decisions made by other mobile devices. Through numerical studies, we evaluate mobile devices' QoE as well as multiple key performance metrics. When evaluating the simulation results for 50 mobile devices and 5 edge nodes, it becomes evident that the QOCO algorithm efficiently exploits the computational capacities of edge nodes. Consequently, it can bring about a significant improvement in average QoE by at least 37%. This improvement is achieved by increasing the number of completed tasks by 14% and reducing task delay and energy consumption by 9% and 6%, respectively, when compared to an existing algorithm.**

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

I. Introduction

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 technological advancements in MDs, their limitations in computing capacity and battery life, 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' QoE, especially for time-sensitive computation tasks [4]. 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 [5]. In particular, determining the optimal offloading strategy, scheduling the tasks, and selecting the most suitable EN are main challenges that demand careful consideration.

Researchers have recently proposed several task offloading algorithms to tackle the aforementioned issues. Mao *et al.* in [6] introduced a computation offloading algorithm for MEC systems.This scheme aims to reduce the MD's energy consumption while still meeting its computation delay constraints. In [7], Zhang *et al.* proposed an offloading scheme for MEC in heterogeneous cellular networks, taking into account the energy-constrained MEC scenario and the heterogeneity of MDs when optimizing the offloading decisions. Bi *et al.* in [8] proposed an algorithm to optimize decision-making about computation offloading and power transferring in a wireless-powered MEC. Ning *et al.*, in [9], introduced a heuristic algorithm designed to enhance the efficiency of the partial computation offloading model. This algorithm effectively adapts offloading decisions dynamically. The works in [6]-[9] primarily focus on quasi-static systems and seldom account for time-varying conditions and are not well-suited for dynamic systems. Furthermore, the uncertain requirements and sensitive latency properties of computation tasks in MEC systems with limited resources pose nontrivial challenges that can significantly impact computation offloading performance.

To cope with the dynamic nature of the network, recent research has proposed several DRL-based algorithms. DRL combines reinforcement learning principles with deep neural networks to determine optimal decision-making policies [10]. By capturing the dynamics of environments and learning strategies for accomplishing long-term objectives through repeated interactions, DRL can effectively tackle the challenges arising from the ever-changing nature of networks, MDs, and servers' heterogeneity. This ultimately improves the MD users' QoE. In [11], Huang *et al.* focus on a wireless-powered MEC network. They propose a DRL-based online approach, capable of attaining near-optimal decisions. This is achieved by selectively considering a compact subset of candidate actions in each iteration. Liao *et al.* in [12] introduced a double reinforcement learning algorithm for performing online computation offloading in MEC. This algorithm optimizes

transmission power and scheduling of CPU frequency to minimize both task computation delay and energy consumption. In [13], Zhao *et al.* proposed a computation offloading algorithm based on DRL, which addresses the competition for wireless channels to optimize long-term downlink utility. In this approach, each MD requires quality-of-service information from other MDs. Tang *et al.*, in [14], investigated the task offloading problem for indivisible and deadline-constrained computational tasks in MEC systems. The authors proposed a distributed DRL-based offloading algorithm designed to handle uncertain workload dynamics at the ENs. In [15], Chen *et al.* introduced an online algorithm with polynomial time complexity for offloading in MEC. This algorithm achieves a close-to-optimal energy consumption while ensuring that the delay is bounded. In [16], Dai *et al.* introduced the integration of action refinement into DRL and designed an algorithm to concurrently optimize resource allocation and computation offloading. Zhou *et al.*, in [17], used an MDP to study MEC and model the interactions between the policy-system environment. They proposed a Q-learning approach to achieve optimal resource allocation strategies and computation offloading. In their work [18], Gao *et al.* introduced an attention-based multi-agent algorithm designed for decentralized computation offloading. This algorithm effectively tackles the challenges of dynamic resource allocation in large-scale heterogeneous networks.

Although DRL-based methods have demonstrated their effectiveness in handling network dynamics, task offloading still encounters several challenges that require further attention. QoE is a time-varying performance measure that reflects user satisfaction and is not affectedonly by delay as assumed in [13]-[14], but also by energy consumption. A more comprehensive approach is required to address the dynamic requirements of individual users in real-time scenarios with multiple MDs and ENs. Nevertheless, prior research in [11]-[18] has failed to tackle some or all of these challenges.

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 the distributed QOCO algorithm that leverages DRL to efficiently handle task offloading in 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. By adopting the appropriate policy based on each MD's specific requirements at any given time, the QOCO algorithm significantly improves the QoE for individual users.

Our main contributions are summarized as follows:

- *Task Offloading problem in the MEC System:* We formulate the task offloading problem as an MDP for time-sensitive tasks. This approach takes into account the dynamic nature of workloads at the ENs and concentrates on providing high performance in the MEC system while maximizing the long-term QoE.
- *DRL-based offloading Algorithm:* To address the problem of long-term QoE maximization, we focus on task completion, task delay, and energy consumption as QoE factors, to quantify the MDs' QoE. We propose QOCO algorithm based on DRL that empowers each MD to make offloading decisions independently, without prior knowledge of the other MDs' tasks and offloading models. With a focus on the MD's battery level, our approach leverages DQN [19] and LSTM [20] to prioritize and strike an appropriate balance between QoE factors.
- *Performance Evaluation:* We conducted comprehensive experiments to evaluate QOCO's performance. The simulation results confirm that when compared with the potential game-based offloading algorithm (PGOA) in [21] and several benchmark methods, our algorithm effectively utilizes the processing capabilities of MDs and ENs, resulting in a substantial improvement in the average QoE of the system.

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

II. System Model

We investigate an 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 represents the number of MDs and J represents the number of ENs. We regard time as a specific episode containing a series of time slots denoted by $\mathcal{T} = \{1, 2, \ldots, T\}$, with T time slots, which represents 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 the ENs, it is enqueued in the corresponding MD's queue.

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 from a discrete set $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_\theta\}$, where θ represents the number of available values. Hence, $\lambda_i(t) \in \Lambda \cup \{0\}$. We also denote the task's processing density $z_i(t)$ as $\rho_i(t)$ indicates the number of CPU frequency 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 not be dropped.

We define two binary variables, $x_i(t)$ and $y_{i,j}(t)$, to determine the offloading decision and offloading target, respectively. Specifically, $x_i(t)$ indicates whether task $z_i(t)$ is assigned to the execution queue ($x_i(t) = 0$) or to the dispatch 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, $y_{i,j}(t) = 0$.

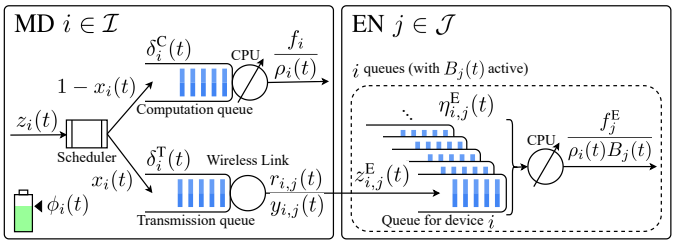

Fig. 1. An illustration MD $i \in \mathcal{I}$ and EN $j \in \mathcal{J}$ in the MEC system.

A. Communication Model

We consider that the tasks in the dispatch queues 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 $j \in \mathcal{J}$ at time t as $r_{i,j}(t)$. At time slot $t \in \mathcal{T}$, if task $z_i(t)$ is assigned to the dispatch 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 $i \in \mathcal{I}$ assigns task $z_i(t)$ for dispatching at time slot $t \in \mathcal{T}$, then it will either be dispatched or dropped at 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

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

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 execution 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 execution 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 execution queue is empty, $l_i^{\mathrm{C}}(t) = 0$. We also define $\delta_i^{\mathrm{C}}(t)$ denote the number of remaining time slots before processing task $z_i(t)$ in the execution queue. The value of $\delta_i^{\mathrm{C}}(t)$ is calculated as follows:

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

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 execution 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 has been assigned to the execution 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 [6].

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 where $t' < t$, we let $z_{i,j}^{\mathrm{E}}(t)$ and $\lambda_{i,j}^{\mathrm{E}}(t)$ (in bits) denote the unique index of the task and the size of the task in the m^{th} queue at EN j. 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 active queue set at EN j in time slot t, given by:

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

We introduce $B_j(t) \triangleq |\mathcal{B}_j(t)|$ that represents the number of active queues in EN $j \in \mathcal{J}$ at 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 [22]. Let variable f_j^{E} (in cycle 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 execution 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 $\eta^{\mathrm{E}}_{i,j}(t)$ is given by:

$$\eta^{\mathrm{E}}_{i,j}(t)=\left[\eta^{\mathrm{E}}_{i,j}(t-1)+\lambda^{\mathrm{E}}_{i,j}(t)-\frac{f^{\mathrm{E}}_{j}}{\rho_i(t)B_j(t)}-\omega_{i,j}(t)\right]^{+}. \tag{10}$$

We also define $l^{\mathrm{E}}_{i,j}(t) \in \mathcal{T}$ as the time slot during which the offloaded task $z^{\mathrm{E}}_{i,j}(t)$ 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 $l^{\mathrm{E}}_{i,j}(t)$ until the corresponding task $z^{\mathrm{E}}_{i,j}(t)$ is either processed or dropped. Let $\hat{l}^{\mathrm{E}}_{i,j}(t)$ represent the time slot at which the execution of task $z^{\mathrm{E}}_{i,j}(t)$ starts. In mathematical terms, for $i \in \mathcal{I}$, $j \in \mathcal{J}$, and $t \in \mathcal{T}$, we have:

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

where $l^{\mathrm{E}}_{i,j}(0)=0$. Indeed, the initial processing time slot of task $z^{\mathrm{E}}_{i,j}(t)$ 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.

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

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

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:

$$D^{\mathrm{E}}_{i,j}(t)=\frac{\lambda^{\mathrm{E}}_{i,j}(t)\rho_i(t)}{f^{\mathrm{E}}_{j}\tau/B_j(t)}. \tag{14}$$

We 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:

$$E^{\mathrm{E}}_{i,j}(t)=\frac{D^{\mathrm{E}}_{i,j}(t)p^{\mathrm{E}}_{j}\tau}{B_j(t)}, \tag{15}$$

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 the 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 $E^{\mathrm{I}}_{i,j}(t)$ as the energy consumption associated with the user interface of MD $i \in \mathcal{I}$, which is given by

$$E^{\mathrm{I}}_{i}(t)=D^{\mathrm{E}}_{i,j}(t)p^{\mathrm{I}}_{i}\tau, \tag{16}$$

where p^{I}_{i} is the energy consumption of MD $i \in \mathcal{I}$ in the standby state. Accordingly, by combining (4), (15) and (16), the overall energy consumption associated with offloading task $z_i(t)$ is formulated as

$$E^{\mathrm{O}}_{i}(t)=E^{\mathrm{T}}_{i}(t)+\sum_{\mathcal{J}} E^{\mathrm{E}}_{i,j}(t)+E^{\mathrm{I}}_{i}(t). \tag{17}$$

III. 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 battery level, which plays a crucial role in decision-making. Specifically, when an MD observes its state (e.g. task size, queue details, and battery level) and encounters a newly arrived task, it is required to select an appropriate action for that task. The selected action, based on the observed state, will result in enhanced QoE for the MD. Each MD strives to maximize its long-term QoE by optimizing the policy mapping from states to actions. In what follows, we will initially present the state space, action space, and QoE function, respectively. Then, we will 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 at time slot t as vector $\boldsymbol{s}_i(t)$ that includes the newly arrived task size, the queues information, the MD's battery level, 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:

$$\boldsymbol{s}_i(t)=\Big(\lambda_i(t), \delta^{\mathrm{C}}_{i}(t), \delta^{\mathrm{T}}_{i}(t), \boldsymbol{\eta}^{\mathrm{E}}_{i}(t-1), \phi_i(t), \mathcal{H}(t)\Big), \tag{18}$$

where the vector $\boldsymbol{\eta}^{\mathrm{E}}_{i}(t-1)=(\eta^{\mathrm{E}}_{i,j}(t-1), j \in \mathcal{J})$ represents the queue length of MD i in ENs at the previous time slot, which the MD can compute this vector according to (10). Let $\phi_i(t)$ denote the battery level of MD i at time slot t. Considering the power modes of Android MDs, $\phi_i(t)$ is derived 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 predicting future EN workloads, we define the matrix $\mathcal{H}(t)$ as historical data, indicating the number of active queues for each EN. This data is recorded over T^{s} time slots, ranging from $t-T^{\mathrm{s}}$ to $t-1$, in a matrix with a size of $T^{\mathrm{s}} \times J$. For EN j workload history at i^{th} time slot from $T^{\mathrm{s}}-t$, define $h_{i,j}(t)$ as:

$$h_{i,j}(t)=B_j(t-T^{\mathrm{s}}+i-1), \tag{19}$$

Where EN $j \in \mathcal{J}$ broadcasted $B_j(t)$ at the end of each time slot. Given $B_j(t) \leq I$, can be represented using only a few bits, the broadcast incurs minimal signaling overhead.

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 made by the agent. In this context, we define $\boldsymbol{a}_i(t)$ to denote the action taken by MD $i \in \mathcal{I}$ at time slot $t \in \mathcal{T}$. These actions involve two decision, (a) Offloading decision to determine whether 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:

$$\boldsymbol{a}_i(t) = (x_i(t), \boldsymbol{y}_i(t)), \tag{20}$$

where $\boldsymbol{y}_i(t) = (y_{i,j}(t), j \in \mathcal{J})$ is the selected EN for offloading this task.

C. QoE Function

The QoE function evaluates the influence of an agent's actions by taking into account several key performance factors. 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:

$$\mathcal{D}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = (1 - x_i(t))\Big(l_i^{\mathrm{C}}(t) - t + 1\Big) + x_i(t)\bigg(\sum_{\mathcal{J}} \sum_{t'=t}^{T} \mathbb{1}\big(z_{i,j}^{\mathrm{E}}(t') = z_i(t)\big) l_{i,j}^{\mathrm{E}}(t') - t + 1\bigg), \tag{21}$$

where $\mathcal{D}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = 0$ when task $z_i(t)$ is dropped. Correspondingly, we denote the energy consumption of task $z_i(t)$ with concerning the selected 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:

$$\mathcal{E}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = (1 - x_i(t)) E_i^{\mathrm{L}}(t) + x_i(t)\bigg(\sum_{\mathcal{J}} \sum_{t'=t}^{T} \mathbb{1}\big(z_{i,j}^{\mathrm{E}}(t') = z_i(t)\big) E_i^{\mathrm{O}}(t)\bigg). \tag{22}$$

Given the delay and energy consumtion of task $z_i(t)$, we also define $\mathcal{C}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t))$ denote the assosiate cost of task $z_i(t)$ given the action $\boldsymbol{a}_i(t)$ in the state $\boldsymbol{s}_i(t)$ as follows:

$$\mathcal{C}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = \phi_i(t)\, \mathcal{D}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) + (1 - \phi_i(t))\, \mathcal{E}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)), \tag{23}$$

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

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:

$$\boldsymbol{q}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) = \begin{cases} \mathcal{R} - \mathcal{C}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) & \text{if task } z_i(t) \text{ processed,} \\ -\mathcal{E}_i(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t)) & \text{if task } z_i(t) \text{ dropped,} \end{cases} \tag{24}$$

where $\mathcal{R} > 0$ represents a constant reward for task completion. If task $z_i(t) = 0$, 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

The task offloading policy for MD $i \in \mathcal{I}$ can be defined as a mapping from its state to its corresponding action, i.e., $\pi_i : \mathcal{S} \to \mathcal{A}$. Especially, the 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 i aims to find its optimal policy π_i^* which maximizes the long-term QoE,

$$\pi_i^* = \arg \max_{\pi_i} \mathbb{E}\left[\sum_{t \in \mathcal{T}} \gamma^{t-1} \boldsymbol{q}_i(t) \middle| \pi_i\right], \tag{25}$$

where $\gamma \in (0, 1]$ is a discount factor and determines the balance between instant QoE and long-term QoE considerations. 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 (25) is particularly challenging due to the dynamic nature of the network. To address this challenge, we introduce an DRL-based offloading algorithm to learn the mapping between each state-action pair and their long-term QoE.

IV. DRL-Based Offloading Algorithm

We present QOCO algorithm so as to address the distributed offloading decision-making of MDs in this section. 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 of an MD that characterizes its state-action Q-values mapping, followed by a description of the communication interchange between the MDs and ENs.

A. DQN-based solution

We utilize the DQN technique to address the formulated MDP to find the mapping between each state-action pair to Q-values. 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

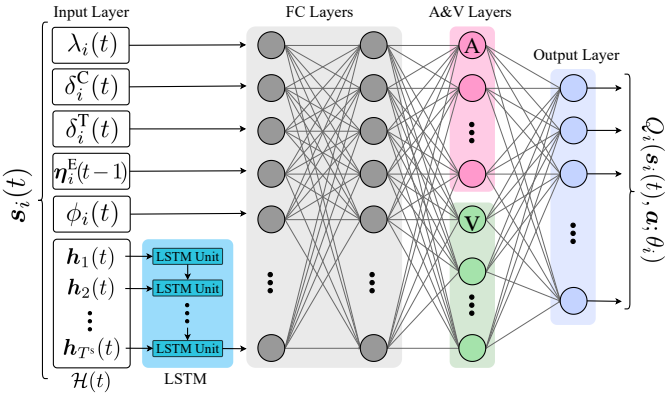

Fig. 2. The neural network of MD $i \in \mathcal{I}$, which characterize the Q-value of each action $\boldsymbol{a} \in \mathcal{A}$ under state $\boldsymbol{s}_i(t) \in \mathcal{S}$.

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 directed 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.

1) Predicting Workloads at ENs: In order to capture the dynamic behavior of workloads at the ENs, we employ the LSTM network [20]. This network maintains a memory state $\mathcal{H}(t)$ that evolves over time, enabling 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, they track the variations in sequences from $\boldsymbol{h}_1(t)$ to $\boldsymbol{h}_{T^{\mathrm{s}}}(t)$, where $\boldsymbol{h}_i(t) = (h_{i,j}(t), j \in \mathcal{J})$, thereby revealing workload fluctuations at the ENs across different time slots. The final LSTM unit produces output that encapsulates the anticipated workload dynamics, and this output 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 workload 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 [23] 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 learning 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 vector θ_i is responsible for determining these values, and it can be adjusted and subjected to training in the QOCO 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:

$$Q_i(\boldsymbol{s}_i(t), \boldsymbol{a}; \theta_i) = V_i(\boldsymbol{s}_i(t); \theta_i) + \left(A_i(\boldsymbol{s}_i(t), \boldsymbol{a}; \theta_i) - \frac{1}{|\mathcal{A}|} \sum_{\boldsymbol{a}' \in \mathcal{A}} (A_i(\boldsymbol{s}_i(t), \boldsymbol{a}'; \theta_i) \right), \quad (26)$$

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

B. QoE-Oriented DRL-Based Algorithm

The QOCO algorithm has been 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 $j_i \in \mathcal{J}$, 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 QOCO algorithm 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 pairs. 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, the EN $j \in \mathcal{J}$ maintains a replay buffer $\mathcal{M}_i$ 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 the MD $i \in \mathcal{I}_j$, associating the action $\boldsymbol{a} \in \mathcal{A}$ under the state $\boldsymbol{s}_i(t) \in \mathcal{S}$, respectively. 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 the MD i for some $t \in \mathcal{T}$. The Net_i^{E} is responsible for action selection, while the 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

Algorithm 1 QOCO Algorithm (Offloading Decision)

Input: state space $\mathcal{S}$, action space $\mathcal{A}$
Output: MD $i \in \mathcal{I}$ experience

for episode from 1 to *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 a *UpdateRequest* to EN j_i;
      Receive network parameter vector θ_i^{E};
      Select action $\boldsymbol{a}_i(t)$ based on the equation (27);
    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 j_i;
    end for
  end for
end for

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 in MD $i \in \mathcal{I}$ and the training process algorithm in EN $j \in \mathcal{J}$, respectively.

1) Offloading Decision Algorithm at MD $i \in \mathcal{I}$: We analyze a series of episodes, where *Ep.* denote the number of them. At the start of each episode, if MD $i \in \mathcal{I}$ receives a new task arrival $z_i(t)$, it initializes the state $\mathcal{S}_i(1)$ and sends a *UpdateRequest* to EN n_i. After receiving the requested parameter vector θ_i^{E} of Net_i^{E} from EN j_i, MD i selects its action for task $z_i(t)$ in the following manner:

$$\boldsymbol{a}_i(t) = \begin{cases} \arg\max_{\boldsymbol{a} \in \mathcal{A}} Q_i^{\mathrm{E}}(\boldsymbol{s}_i(t), \boldsymbol{a}; \theta_i^{\mathrm{E}}), & \text{with p}(1-\boldsymbol{\epsilon}), \\ \text{pick an random action from } \mathcal{A}, & \text{with p}(\boldsymbol{\epsilon}), \end{cases} \tag{27}$$

where $\mathrm{p}(\cdot)$ is 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, the 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 $t' \le t$. For each MD i, we define the set $\mathcal{F}_i^t \subset \mathcal{T}$ to denote the time slots during which each arrival task $z_i(t')$ is either processed or dropped at time slot t, as given by:

$$\mathcal{F}_i^t = \Bigg\{ t' \Bigg| \; t' \le t,\ \lambda_i(t') > 0,\ (1 - x_i(t'))\, l_i^{\mathrm{C}}(t') + x_i(t') \sum_{\mathcal{J}} \sum_{n=t'}^{t} \mathbb{1}(z_{i,j}^{\mathrm{E}}(n) = z_i(t'))\, l_{i,j}^{\mathrm{E}}(n) = t \Bigg\}. \tag{28}$$

Algorithm 2 QOCO Algorithm (Training Process)

Initialize replay buffer $\mathcal{M}_i$ for each MD $i \in \mathcal{I}_j$;
Initialize Net_i^{E} and Net_i^{T} with random parameters θ_i^{E} and θ_i^{T} respectively, for each MD $i \in \mathcal{I}_j$;
Set Count := 0
while True **do** ▷ *infinite loop*
  if receive a *UpdateRequest* from MD $i \in \mathcal{I}_j$ **then**
    Send θ_i^{E} to MD $i \in \mathcal{I}$;
  end if
  if receive an experience $(\boldsymbol{s}_i(t), \boldsymbol{a}_i(t), \boldsymbol{q}_i(t), \boldsymbol{s}_i(t+1))$
  from MD $i \in \mathcal{I}_j$ **then**
    Store $(\boldsymbol{s}_i(t'), \boldsymbol{a}_i(t'), \boldsymbol{q}_i(t'), \boldsymbol{s}_i(t'+1))$ in $\mathcal{M}_i$;
    Get a collection of experiences $\mathcal{I}$ from $\mathcal{M}_i$;
    for each experience $i \in \mathcal{I}$ **do**
      Get experience $(\boldsymbol{s}_i(i), \boldsymbol{a}_i(i), \boldsymbol{q}_i(i), \boldsymbol{s}_i(i+1))$;
      Create $\hat{Q}_{i,i}^{\mathrm{T}}$ according to (29);
    end for
    Set vector $\hat{\mathbf{Q}}_i^{\mathrm{T}} := (\hat{Q}_{i,i}^{\mathrm{T}}, i \in \mathcal{I})$;
    Update θ_i^{E} to minimize $L(\theta_i^{\mathrm{E}}, \hat{\mathbf{Q}}_i^{\mathrm{T}})$ in (31);
    Count := Count + 1;
    if mod(Count, *ReplaceThreshold*) = 0 **then**
      $\theta_i^{\mathrm{T}} := \theta_i^{\mathrm{E}}$;
    end if
  end if
end while

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 $i \in \mathcal{I}_j$, the EN $j \in \mathcal{J}$ waits for messages from the MDs in the set $\mathcal{I}_j$. When EN j receives a *UpdateRequest* signal from a 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 EN stores this experience in the replay buffer $\mathcal{M}_i$ associated with that MD. The replay buffer operates at maximum capacity and FIFO principle.

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

$$\hat{Q}_{i,n}^{\mathrm{T}} = \boldsymbol{q}_i(n) + \gamma Q_i^{\mathrm{T}}(\boldsymbol{s}_i(n+1)), \tilde{\boldsymbol{a}}_n; \theta_i^{\mathrm{T}}), \tag{29}$$

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:

$$\tilde{\boldsymbol{a}}_n = \arg\max_{\boldsymbol{a} \in \mathcal{A}} Q_i^{\mathrm{E}}(\boldsymbol{s}_i(n+1), \boldsymbol{a}; \theta_i^{\mathrm{E}}). \tag{30}$$

TABLE I
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	250 Task/Sec
Size of task $\lambda_i(t)$	{1.0, 1.1, ..., 7.0} Mbits
Required CPU cycles of task $\rho_i(t)$	{0.197,0.297,0.397}G/Mbits
Deadline of task Δ_i	10 time slots (1 Sec)
Battery level of MD $\phi_i(t)$	{25, 50, 75} Percent
Computation power of MD p_i^{C}	$10^{-27}(f_i)^3$
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 particular, regarding the experience n, the target-Q value $\hat{Q}_{i,n}^{\mathrm{T}}$ represents the long-term QoE for the 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 trains the MD's neural network using previous sample experiences. Simultaneously, it updates θ_i^{E} in Net_i^{E} and computes $\hat{\mathbf{Q}}_i^{\mathrm{T}} = (\hat{Q}_{i,n}^{\mathrm{T}}, n \in \mathcal{N})$. The key idea of updating Net_i^{E} is minimizing the disparity in Q-values between Net_i^{E} and Net_i^{T}, as indicated by the following loss function:

$$L(\theta_i^{\mathrm{E}}, \hat{\mathbf{Q}}_i^{\mathrm{T}}) = \frac{1}{|\mathcal{N}|} \sum_{n \in \mathcal{N}} \left(Q_i^{\mathrm{E}}(\boldsymbol{s}_i(n), \boldsymbol{a}_i(n); \theta_i^{\mathrm{E}}) - \hat{Q}_{i,n}^{\mathrm{T}} \right)^2. \quad (31)$$

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 (29).

V. PERFORMANCE EVALUATION

We begin by presenting the simulation setting and training configuration, followed by the evaluation of QOCO algorithm under multiple key performance metrics in comparison to benchmark methods.

A. Simulation Settings

As shown in the basic simulation parameter settings in Table I, we consider a computation task offloading scenario involving 50 MDs and 5 ENs in the MEC environment. To train the MDs' neural networks, we adopt a scenario comprising 1000 episodes. Each episode contains 100 time slots, equivalent to 0.1 seconds. QOCO algorithm performs online training, incorporating real-time experience to continually improve 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 of 1, progressively approaching 0.01, all of which is facilitated by an RMSProp optimizer.

In these simulations, we first evaluate the QOCO algorithm for each of the influencing QoE factors, including the number of completed tasks, overall energy consumption, and average delay. Subsequently, we demonstrate the overall improvement of QOCO in terms of average QoE by comparing it against the following benchmark methods across multiple challenging scenarios.

1) **Local Computing (LC):** In this mode, the MDs execute all of their computation tasks using their own computing capacity to meet their deadlines.
2) **Full Offloading (FO):** In the FO method, each MD dispatches all of their computation tasks to ENs and selects their offloading target randomly. It leverages their complete communication capacity and the computation resources allocated to them by the EN.
3) **Random Decision (RD):** In this approach, when an MD receives a new task, it randomly makes offloading decisions and selects the offloading target (if it decides to dispatch the task). This method leverages the MD's complete computational and communication capabilities, and the computational resources allocated to it by the EN.
4) **PGOA:** This method is a distributed optimization algorithm designed for delay-sensitive tasks in an environment where MDs interact strategically with multiple ENs. We selected PGOA as a benchmark method due to its similarity to our work, as described in [21].

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 QOCO 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.

In the following, we will compare the performance of the QOCO algorithm with benchmark methods under two different computation workloads. This comparison will involve increasing the task arrival rate and the number of MDs.

B. Performance Comparison

In Fig. 3 (a), the QOCO algorithm consistently outperforms the benchmark methods in terms of the number of completed tasks. At a lower task arrival rate (i.e., 50), most of the methods demonstrate equal proficiency in completing tasks. However, as the task arrival rate increases, the efficiency of QOCO becomes evident. Specifically, when the task arrival rate increases to 250, it achieves approximately 1900 completed tasks, whereas the other methods achieve at most about 1300 completed tasks. This demonstrates that our algorithm can increase the number of completed tasks by 73% and 47% compared to RD and PGOA, respectively.

Similarly, in Fig. 3 (b), as the number of MDs increases, QOCO shows significant improvements in the number of completed tasks compared to other methods, especially when faced with a large number of MDs. When the number of MDs reaches 110, our proposed algorithm can effectively increase the number of completed tasks by at least 34% more than

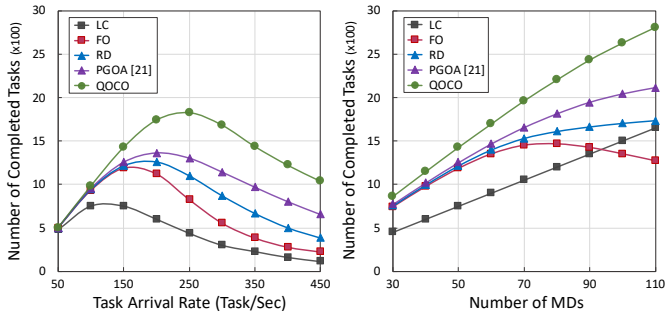

Fig. 3. The number of completed tasks under different computation workloads: (a) task arrival rate; (b) the number of MDs.

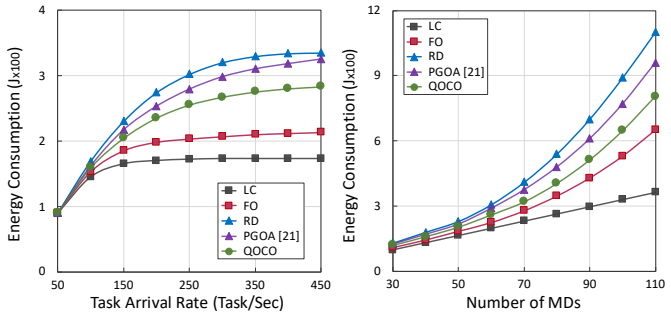

Fig. 4. The overall energy consumption under different computation workloads: (a) task arrival rate; (b) the number of MDs.

other methods. This achievement is attributed to the QOCO's ability to effectively handle unknown workloads and prevent congestion at the ENs.

Fig. 4 (a) illustrates the overall energy consumption when the task arrival rate increases. At the lower task arrival rate, the total energy consumption of all methods is close to each other, and when the task arrival rate increases, the total energy consumption of each method increases differently. It can be found that the production of more computational tasks leads to a continuous increase in energy consumption until the capacity is saturated, and the energy consumption remains constant at its maximum. The QOCO algorithm takes into account the battery level of the MD in its decision-making process. At task arrival rate 450, QOCO effectively reduces overall energy consumption by 18% and 15% compared to RD and PGOA, respectively. However, it consumes more energy compared to LC and FC because they do not utilize the full capacity (LC uses the MD's computational resources, while FC utilizes the transmission link and allocated capacity from EN to MD).

In Fig. 4 (b), an increasing trend in overall energy consumption is observed as the number of MDs increases. With the increase in MDs, the number of resources available in the system increases, which leads to higher energy consumption in the system. The QOCO algorithm consistently outperforms the benchmark methods in overall energy consumption, especially when the number of MDs is large. Specifically, QOCO demonstrates a 27% and 16% reduction in overall energy

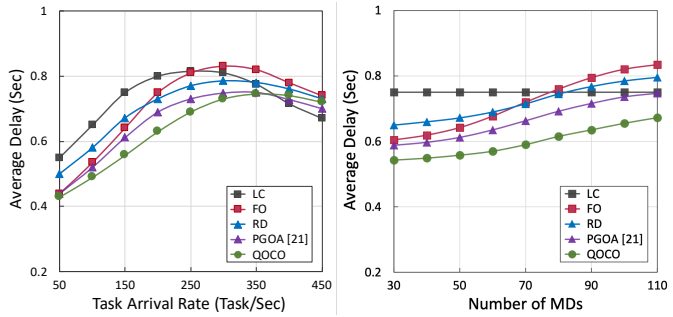

Fig. 5. The average delay under different computation workloads: (a) task arrival rate; (b) the number of MDs.

consumption compared to RD and PGOA, respectively, when the number of MDs increases to 110.

As shown in Fig. 5 (a), the QOCO 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% compared to other methods. However, for task arrival rates exceeding 350, QOCO may experience a higher average delay compared to some of the other methods. This can be attributed to the fact that our algorithm processes a larger number of completed tasks in the same time slot, 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 QOCO's ability to schedule workloads, when the number of MDs increases from 30 to 110, it consistently maintains an average delay at least 8% lower than the other methods.

In the following, we delve into the investigation of the overall improvement achieved by the QOCO algorithm in comparison to other methods in terms of the average QoE as the task arrival rate and the number of MDs increase. This metric signifies the average advantage created by each method for MDs, including a combination of the values of the previously evaluated metrics (task completion, delay, energy consumption) according to the demands of MDs.

The average QoE comparison in Fig. 6 (a) indicates the superiority of the QOCO algorithm in providing MDs with a better experience. By taking into account the specific demands of MDs for each key performance metric, it consistently achieves a higher average QoE across various values of arrival task rates when compared to other methods. Specifically, when the task arrival rate is moderate (i.e., 250), QOCO increases the average QoE by 57% and 33% compared to RD and PGOA, respectively. In Fig. 6 (b), As the number of MDs increases, the potential workload on the ENs also rises, leading to a decrease in the average QoE for each method (excluding LC). It can be observed that an increase in the ENs' workload leads to a reduction in the allocated computing capacity for MDs, resulting in a decrease in the average QoE. However, QOCO

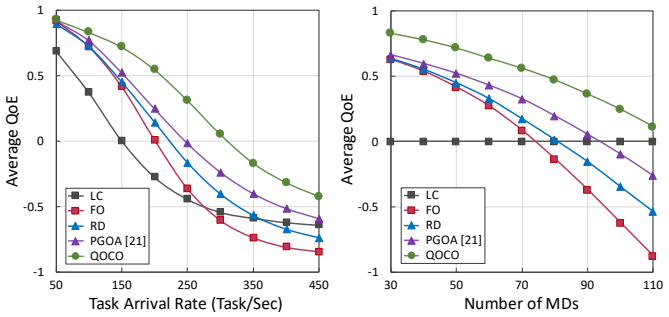

Fig. 6. The average QoE under different computation workloads: (a) task arrival rate; (b) the number of MDs.

effectively manages the uncertain load at the ENs. When the number of MDs increases to 90, it increases the average QoE by at least 29% more than the other methods.

VI. Conclusion

In this study, our focus was directed towards addressing the challenge of offloading in MEC systems, where strict task processing deadlines and energy constraints adversely impact system performance. To address this challenge, we propose the QOCO algorithm specifically designed to empower MDs to make offloading decisions without relying on knowledge about task models or other MDs' offloading decisions. The QOCO 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 demonstrate that QOCO outperforms several established benchmark techniques. Specifically, it can significantly increase the average user's QoE. This advantage can lead to improvements in key performance metrics, including task completion rate, task delay, and energy consumption, according to the user's demands under different system conditions.

There are multiple directions for extend this research. A complementary approach involves extending the simplified task model by considering dependent divisible tasks that correspond to actual computational tasks. This can be achieved by incorporating a call graph [24] representation to develop relationships among task partitions. Furthermore, in order to accelerate the learning of optimal offloading policies, it is beneficial to enable MDs to take advantage of federated learning [25] techniques in the training process. This allows MDs to collectively contribute to improving the offloading model and enables 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 Communications Surveys & Tutorials*, 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," *Proceedings of the 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," *Journal of Systems Architecture*, vol. 98, pp. 289–330, Sep 2019.

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

[5] 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.

[6] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, Sep 2016.

[7] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, Aug 2016.

[8] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 6, pp. 4177–4190, Apr 2018.

[9] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4804–4814, Sep 2018.

[10] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, Nov 2017.

[11] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 11, pp. 2581–2593, Jul 2019.

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

[13] 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 Transactions on Wireless Communications*, vol. 18, no. 11, pp. 5141–5152, Aug 2019.

[14] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Transactions on Mobile Computing*, vol. 21, no. 6, pp. 1985–1997, Nov 2020.

[15] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. Shen, "Energy efficient dynamic offloading in mobile edge computing for internet of things," *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 1050–1060, Feb 2019.

[16] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang, "Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 12 175–12 186, Aug 2020.

[17] 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 of Things Journal*, vol. 9, no. 2, pp. 1517–1530, Jun 2021.

[18] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 6, pp. 3425–3443, Jan 2023.

[19] 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.

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

[21] 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 transactions on networking*, vol. 26, no. 6, pp. 2762–2773, 2018.

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

[23] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, Jun 2016, pp. 1995–2003.

[24] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE transactions on parallel and distributed systems*, vol. 7, no. 5, pp. 506–521, 1996.

[25] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.