A Deep Reinforcement Learning Approach for Collaborative Mobile Edge Computing

Jiaqi Wu, Huang Lin, Huaize Liu, and Lin Gao

ICC 2022 - IEEE International Conference on Communications | 978-1-5386-8347-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICC45855.2022.9839202

***Abstract*—Mobile edge computing (MEC) is a promising approach to reduce the network traffic load and alleviate the backhaul congestion by pushing computation down to the network edge (e.g., base stations) that are close to the origin of data. However, when many mobile devices (MDs) offload tasks to a base station (BS) in a dynamic and stochastic environment (e.g., with time-varying wireless channels and uncertain task models), it is often challenging for MDs to make offloading decisions in decentralized manner. In this work, we consider a *collaborative MEC* scenario, where an MD can offload its task to the associated BS or to other BSs through the associated BS. In such a scenario, we study the *joint computation offloading and resource allocation* problem, aiming at minimizing the expected long-term delay, taking the energy consumption constraint into consideration. The problem is challenging due to time-varying system and distributed decisions. To solve the problem in an online and decentralized manner, we propose a *deep reinforcement learning* (DRL) based distributed online algorithm. By incorporating the double deep Q network and dueling deep Q network technique, the proposed algorithm can improve the performance of the whole system significantly. Simulation results show that the proposed DRL-based algorithm outperforms baseline methods and can reduce the average delay of tasks by 76.4%-91.2%.**

I. Introduction

A. Background and Motivations

With the advancement of wireless technology and service, mobile devices (MDs) are generating more computation-intensive and delay-sensitive tasks, such as AR/VR, online games, and AI applications. However, the traditional cloud computing scheme may not be able to guarantee the quality-of-service (QoS) of such tasks, due to the large communication latency between MDs and cloud servers. Mobile Edge Computing (MEC) [1] has recently been introduced as a promising technology to address this challenging, by pushing computation down to the network edge (e.g., base stations, BSs) close to the origin of data and tasks [2]–[4]. In a typical MEC scenario, computing servers (called MEC servers) are often deployed on BSs, and MDs can offload tasks to different MEC servers to reduce latency [5].

Jiaqi Wu, Huang Lin, Huaize Liu are with the School of Electronics and Information Engineering, Harbin Institute of Technology, Shenzhen, China. Lin Gao is with the School of Electronics and Information Engineering, Harbin Institute of Technology, Shenzhen, China, and the Shenzhen Institute of Artificial Intelligence and Robotics for Society, Shenzhen, China. Email: gaol@hit.edu.cn. (*Corresponding Author: Lin Gao*)

This work is supported in part by the National Natural Science Foundation of China (Grant No. 61972113), the Basic Research Project of Shenzhen Science and Technology Program (Grant No. JCYJ20190806112215116 and KQTD20190929172545139), and Guangdong Science and Technology Planning Project under Grant 2018B030322004. This work is also supported in part by the funding from Shenzhen Institute of Artificial Intelligence and Robotics for Society.

In an MEC system, an important classic problem is how to efficiently offload the MDs' computation tasks to MEC servers, taking the dynamic changing of MEC servers' computation loads and MDs' wireless environments into consideration [6]. This essentially consists of the following subproblems. The first one is whether a task of an MD should be offloaded or processed locally. The second one is which MEC server should the task be offloaded to, if the MD decides to offload a task. The third one is how much computation and communication resources should the MD allocate for a task. Many recent works (e.g., [7], [8]) have investigated these problems comprehensively. In [7], Wang *et al.* proposed an algorithm to determine MDs' offloading decisions to optimize the total revenue of network for MEC-enabled wireless cellular network. In [8], Liu *et al.* proposed a decomposition and iteration-based algorithm to jointly optimize the CPU frequencies, offloading amount, transmit power, and UAV's trajectory for an UAV-enabled wireless power cooperative MEC system.

The key challenges in the above problems are the MDs' limited resources and the MEC servers' unbalanced loads. To deal with these issues, many researches focused on the fine-grained task assignment scheme [9], [10]. In [9], Lyu *et al.* considered divisible tasks and presented a distributed online optimization algorithm to maintain the stability of each task queue at MDs. In [10], Yi *et al.* characterized the fine-grained task, the dynamic nature of the system, and computation workload distribution among different servers. However, due to the dependency among different components in a task, the fine-grained task scheme may *not* be realistic. Thus, some recent works studied the computation offloading for non-divisible tasks [11], [12]. In [11], Tang *et al.* designed a distributed offloading algorithm that can deal with dynamic loads at the edge nodes and non-divisible tasks. In [12], Xia *et al.* considered non-divisible tasks and task offloading problem in an ultra-dense network.

Inspired by [11], we consider the non-divisible task model and the *queue-based* offloading framework, where each BS has three queues for multilateral cooperations among BSs. To optimize the computation offloading in the dynamic environment with time-varying channel conditions and network loads, we introduce the Deep Reinforcement Learning (DRL), which has been widely used in stochastic network systems [13]–[16]. In [13], Chen *et al.* developed an online double deep Q-network (DDQN) algorithm for stochastic computation offloading in an ultra-dense sliced radio access network, indicating that DDQN can handle the high-dimensional state space. In [14], Huang *et al.* proposed a DRL-based online offloading algorithm (DROO) to maximize the sum of computation rate

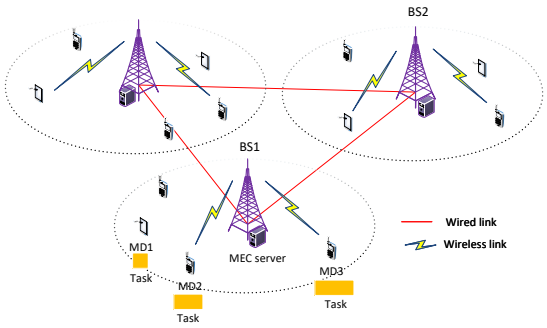


Fig. 1. A collaborative MEC system network where MD1 processes its task locally; MD2 offloads its task to BS1 for processing; MD3 forwards its task to BS2 via BS1 for processing.

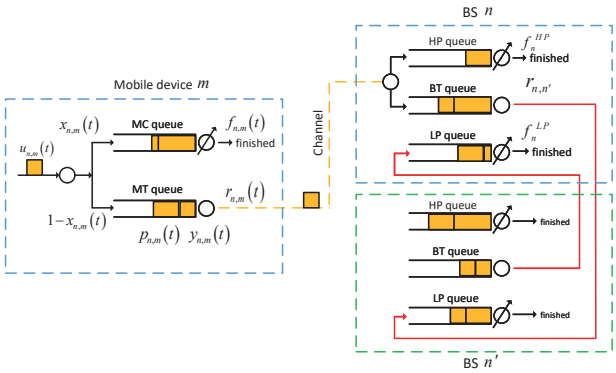


Fig. 2. An illustration of a collaborative MEC system with an MD $m \in \mathcal{M}_n$, $n \in \mathcal{N}$ and the adjacent BS $n' \in \mathcal{N} \setminus \{n\}$ of BS n.

under time-varying wireless channel conditions. In [15], He *et al.* proposed a dueling DDQN algorithm to handle an integrated framework of networking, caching, and computing resources. To support low-latency services, Sun *et al.* proposed centralized Soft Actor Critic (SAC) and decentralized Soft Actor Critic offloading algorithm in [16]. However, the above works didn't jointly consider the load sharing between MEC servers, non-divisible task, resource allocation, and computation offloading. In our work, we will consider a collaborative MEC system scenario for non-divisible task offloading, where a queue system is constructed to tackle the load sharing between MEC servers, and each MD makes offloading and resource decisions in an online and distributed manner.

B. Solution and Contributions

To deal with the unbalanced loads and limited resource issues, we proposed a collaborative MEC system with the load sharing between MEC servers. In such a scenario, we formulate a joint computation offloading and resource allocation problem, aiming at minimizing the delay of tasks. To solve the problem, we propose a model-free DRL-based online and distributed algorithm integrated with the dueling DQN and double DQN technique. The proposed algorithm is simpler than SAC approach, and meanwhile can achieve the near-optimal performance.

Overall, in this work, we consider a multilateral cooperation MEC system and study a computation offloading and resource allocation problem, which is solved by a DRL-based online and distributed algorithm. The key contributions are summarized as follows.

- *Joint Computation Offloading and Resource Allocation:* We consider a collaborative MEC system and study the joint optimization of computation offloading and resource allocation under the energy constraint.
- *DRL-based Online and Distributed Algorithm:* To minimize the expected long-term delay of tasks, we propose a model-free DRL-based online and distributed algorithm integrated with the dueling DQN and double DQN technique. It can handle the high-dimensional state space and dynamic system to achieve near-optimal solution.
- *Performance Evaluation:* Simulation results show that the proposed algorithm enables each MD to make a better strategy in comparison with the existing baseline schemes. More specifically, the proposed DRL-based algorithm outperforms baseline methods and can reduce the average delay of tasks by 76.4%-91.2%.

The rest of this paper is organized as follows. The system model is discussed in Section II. In Section III, we formulate a computation offloading and resource allocation problem. The DRL-based online and distributed algorithm is proposed in IV.

II. System Model

A. Network Model

We consider a collaborative MEC system (see Fig.1) consisting of a set $\mathcal{M} = \{1, 2, \cdots, M\}$ MDs who generate computation task dynamically and randomly, and a set $\mathcal{N} = \{1, 2, \cdots, \mathcal{N}\}$ of BSs, each equipped with an MEC server with computing capability. Specifically, the coverage areas of BSs are nonoverlapping, and each MD is associated with a particular BS (i.e. the BS which covers the MD). For each BS n, we denote $\mathcal{M}_n = \{1, 2, \cdots, M_n\}$ as the MDs set in the coverage of the BS and $\sum_{n=1}^{N} M_n = M$. BSs are connected via fiber optic together, hence they can collaboratively share the computing capability of their equipped MEC servers. In this scenario, each MD can offload its tasks to the associated BS or other adjacent BSs through its associated BS. For the convenience of analysis, we consider a time-slotted structure, where each time period (called an episode) is divided into a set $\mathcal{T} = \{1, , 2, \cdots, T\}$ of time slots with equal length τ (e.g., 0.1 second). As in many literatures, we consider a quasi-static system, where the environment (e.g., wireless channel fading) keeps unchanged within one time slot, but change stochastically across different time slots.

B. Mobile Device Model

At the beginning of each time slot, each MD generates new task randomly according to a certain task arrival rate.[1] Each MD has a task scheduler, which decides whether to offload the task and (if so) which MEC server it is going to offload, whenever a new task is generated. As a task may be processed or transmitted (when being offloaded) over multiple time slots, we introduce a computation queue (called MC queue) and a

[1]Note that our work can be directly extended to the case where each MD may generate multiple new tasks in each time slot.

transmission queue (called MT queue) for each MD to record the set of tasks to be processed locally and the set of tasks to be transmitted (offloaded). That is, if a task is scheduled to be processed locally, it will be pushed into the MC queue, and if a task is scheduled to be offloaded, it will be pushed into the MT queue. Both queues follow the first-in-first-out (FIFO) principle, and hence the tasks in both queues will be processed or transmitted sequentially.

1) Task Model: Let $u_{n,m}(t)$ denote the index of a new task generated by MD m of BS n in time slot t. Let $\lambda_{n,m}(t)$ (in bits) denote the data size of the task to be scheduled into a queue in time slot t and the data size set is $\Omega = \{0, \lambda_1, \lambda_2, \cdots, \lambda_{|\Omega|}\}$, i.e., $\lambda_{n,m}(t) \in \Omega$. Noth that if there does not a newly arrival task, we set $\lambda_{n,m}(t) = 0$. Let $\rho_{n,m}$ (in CPU cycles per bit) denote the computation density of the task, that is, the number of CPU cycles required to process a unit of data.

2) Task Offloading Decision: When an MD m generates a new task $u_{n,m}(t)$ at the beginning of time slot t, it needs to decide whether to offload the task $u_{n,m}(t)$ and which MEC server it is going to offload. Let $x_{n,m}(t) \in \{0, 1\}$ denote whether task $u_{n,m}(t)$ is offloaded or processed locally. $x_{n,m}(t) = 1$ implies that the task is allocated into the MC queue and waiting for processing locally and $x_{n,m}(t) = 0$ implies that the task is allocated into the MT queue and waiting for offloading to the MEC server.

Let $y_{n,m}(t) \in \{0, 1, \cdots, N\}$ denote the target MEC server that the task $u_{n,m}(t)$ is offloaded to. Note that for notational convenience, we denote $y_{n,m}(t) = 0$ if the task is processed locally (i.e., $x_{n,m}(t) = 1$). It is worth noting that no matter which MEC server the task $u_{n,m}(t)$ is offloaded to, the data of task must be transmitted through the associated BS n.

3) MC Queue: The MC queue follows the FIFO principle. That is, when a task is allocated into the MC queue, it will wait until all previous tasks in the queue are completed. Thus, the completion time of a task (when being processed locally) consists of the waiting time and the processing time. Let $c^{\mathrm{MC}}_{n,m}(t)$ denote the completion time of task $u_{n,m}(t)$ in the MC queue. For convenience, we also define a virtual completion time for those tasks not in the MC queue: $c^{\mathrm{MC}}_{n,m}(t) = c^{\mathrm{MC}}_{n,m}(t-1)$ if the task is not allocated into the MC queue.

For the waiting time, it depends on the completion time of the last task in the MC queue. Note that if the MC queue is empty, the waiting time is zero. Formally, the waiting time of task $u_{n,m}(t)$ is

$$w^{\mathrm{MC}}_{n,m}(t) = \left[c^{\mathrm{MC}}_{n,m}(t-1) - t\right]^+, \tag{1}$$

where $[z]^+ = \max\{z, 0\}$ and $c^{\mathrm{MC}}_{n,m}(0) = 0$.

Let $\hat{c}^{\mathrm{MC}}_{n,m}(t)$ denote the time slot when task $u_{n,m}(t)$ begins processing in the MC queue. Then we have $\hat{c}^{\mathrm{MC}}_{n,m}(t) = \max\{c^{\mathrm{MC}}_{n,m}(t-1), t\}$. For the processing time, it depends on the data size of the task and computation capacity of the device. Let $f_{n,m}(t) \in [0, f^{\max}_{n,m}]$ (in CPU cycles) denote the computation capacity of MD m under BS n within a time slot t and $f^{\max}_{n,m}$ is the total computation capacity of the MD. Then the processing time of task $u_{n,m}(t)$ is

$$e^{\mathrm{MC}}_{n,m}(t) = \left\{ \kappa \in \mathbb{N}^* \middle| \delta_1(\kappa - 1) < \lambda_{n,m}(t) \le \delta_1(\kappa) \right\}, \tag{2}$$

where $\delta_1(o) = \sum_{t'=\hat{c}^{\mathrm{MC}}_{n,m}(t)}^{\hat{c}^{\mathrm{MC}}_{n,m}(t)+o} f_{n,m}(t')/\rho_{n,m}$. From the above analysis, we define the completion time of task $u_{n,m}(t)$:

$$c^{\mathrm{MC}}_{n,m}(t) = t + w^{\mathrm{MC}}_{n,m}(t) + e^{\mathrm{MC}}_{n,m}(t). \tag{3}$$

4) MT Queue: Similar as the MC queue, let $c^{\mathrm{MT}}_{n,m}(t)$ denote the time slot in which task $u_{n,m}(t)$ in the MT queue has been sent. Formally, the waiting time of task $u_{n,m}(t)$ is

$$w^{\mathrm{MT}}_{n,m}(t) = \left[c^{\mathrm{MT}}_{n,m}(t-1) - t\right]^+, \tag{4}$$

where $c^{\mathrm{MT}}_{n,m}(0) = 0$.

Let $\hat{c}^{\mathrm{MT}}_{n,m}(t)$ denote the time slot when task $u_{n,m}(t)$ begins transmitting. Then we have $\hat{c}^{\mathrm{MT}}_{n,m}(t) = \max\{c^{\mathrm{MT}}_{n,m}(t-1), t\}$. For the transmission time, it depends on the transmission rate $r_{n,m}(t)$ between MD m and BS n. By the Shannon capacity formula, $r_{n,m}(t)$ can defined as

$$r_{n,m}(t) = \frac{B}{|\mathcal{M}_n|} \log_2 \left(1 + \frac{|h_{n,m}|^2 p_{n,m}(t)}{\sigma^2}\right), \tag{5}$$

where B and σ is the channel bandwidth and the noise; $|\mathcal{M}_n|$ is the cardinality of set $\mathcal{M}_n$; $p_{n,m}(t) \in [0, p^{\max}_{n,m}]$ denotes the transmit power and its maximum value is $p^{\max}_{n,m}$; $h_{n,m}$ is the channel gain between MD m and BS n. Hence, the transmission time of task $u_{n,m}(t)$ is evaluated:

$$e^{\mathrm{MT}}_{n,m}(t) = \left\{ \kappa \in \mathbb{N}^* \middle| \delta_2(\kappa - 1) < \lambda_{n,m}(t) \le \delta_2(\kappa) \right\}, \tag{6}$$

where $\delta_2(o) = \sum_{t'=\hat{c}^{\mathrm{MT}}_{n,m}(t)}^{\hat{c}^{\mathrm{MT}}_{n,m}(t)+o} r_{n,m}(t')$. Hence, the time slot $c^{\mathrm{MT}}_{n,m}(t)$ is

$$c^{\mathrm{MT}}_{n,m}(t) = t + w^{\mathrm{MT}}_{n,m}(t) + e^{\mathrm{MT}}_{n,m}(t). \tag{7}$$

C. Base Station Model

When a BS receives a task offloaded from an associated MD, it first checks whether the task is offloaded to its own server, or to other MEC servers. In the former case, the task will be allocated into a high-priority computation queue (called HP queue) and processed on the MEC server. In the latter case, the task will be allocated into a transmission queue (called BT queue) and forwarded to the corresponding BS. When a BS receives a task forwarded from other BSs, the task will be allocated into a low-priority computation queue (called LP queue) and processed on the MEC server. Fig.2 shows the three queues in MEC servers.

All three queue follows the FIFO principle. Besides, let F_n denote the total computation capacity of BS n, and we denote the computation capacity of HP and LP queues as f^{HP}_n and f^{LP}_n respectively. Specifically, the task in the LP queue will be processed in low computation capacity while the task in the HP queue will be processed in highly computation capacity (i.e., $f^{\mathrm{LP}}_n < f^{\mathrm{HP}}_n$), which implies that the tasks in the HP queue are more important than those in the LP queue.

Let $u_{n,m}^{\mathrm{ed}}(t)$ denote the index of a task that BS n receives from MD m, and $\lambda_{n,m}^{\mathrm{ed}}(t) \in \Omega$ (in bits) denote the data size arrived in the corresponding queue of BS n at time slot t. In the following, we illustrate the operation of three queues.

1) HP Queue: The HP queue mainly processes the tasks from the associated MDs with high-priority. The completion time of a task (when being offloaded to the corresponding MEC server) includes the waiting and processing time. Let $c_{n,m}^{\mathrm{HP}}(t)$ denote the completion time of task $u_{n,m}^{\mathrm{ed}}(t)$ in the HP queue. Specifically, if the task is not place in the HP queue, we set $c_{n,m}^{\mathrm{HP}}(t) = c_{n,m}^{\mathrm{HP}}(t-1)$.

For the processing time, it is determined by the data size of the newly arrival task and the computation capacity allocated by MEC server. Then the processing time of task $u_{n,m}^{\mathrm{ed}}(t)$ is

$$e_{n,m}^{\mathrm{HP}}(t) = \left\lceil \lambda_{n,m}^{\mathrm{ed}}(t)\rho_{n,m}/f_n^{\mathrm{HP}} \right\rceil, \tag{8}$$

where $\lceil \cdot \rceil$ is the ceiling function. According to the definition of $c_{n,m}^{\mathrm{HP}}(t)$, the waiting time of task $u_{n,m}^{\mathrm{ed}}(t)$ and the completion time of task $u_{n,m}^{\mathrm{ed}}(t)$ can be update as follows:

$$\begin{aligned} w_{n,m}^{\mathrm{HP}}(t) &= \left[c_{n,m}^{\mathrm{HP}}(t-1) - t\right]^+, \\ c_{n,m}^{\mathrm{HP}}(t) &= t + w_{n,m}^{\mathrm{HP}}(t) + e_{n,m}^{\mathrm{HP}}(t). \end{aligned} \tag{9}$$

2) BT Queue: The BT queue mainly forwards the tasks from the associated MDs under BS n to the adjacent BS $n' \in \mathcal{N} \setminus \{n\}$ for processing. Let $r_{n,n'}$ denote the total transmission capacity from BS n to BS n' in each time slot. When a task $u_{n,m}^{\mathrm{ed}}(t)$ is placed in the BT queue, it will wait for transmitting until all previous tasks in the queue are sent. Let $c_{n,m}^{\mathrm{BT}}(t)$ denote the time slot in which the transmission process of task $u_{n,m}^{\mathrm{ed}}(t)$ in the BT queue has been finished.

Similar as the HP queue, the transmission time $e_{n,m}^{\mathrm{BT}}(t)$, waiting time $w_{n,m}^{\mathrm{BT}}(t)$ and the time slot $c_{n,m}^{\mathrm{BT}}(t)$ are respectively evaluated as follows:

$$\begin{aligned} e_{n,m}^{\mathrm{BT}}(t) &= \left\lceil \lambda_{n,m}^{\mathrm{ed}}(t)/r_{n,n'} \right\rceil, \\ w_{n,m}^{\mathrm{BT}}(t) &= \left[c_{n,m}^{\mathrm{BT}}(t-1) - t\right]^+, \\ c_{n,m}^{\mathrm{BT}}(t) &= t + w_{n,m}^{\mathrm{BT}}(t) + e_{n,m}^{\mathrm{BT}}(t). \end{aligned} \tag{10}$$

3) LP Queue: The LP queue processes the tasks from other BSs with lower-priority. Let $u_{n,n'}^{\mathrm{ne}}(t)$ denote the index of a task from BS n' to BS n. Correspondingly, let $\lambda_{n,n'}^{\mathrm{ne}}(t) \in \Omega$ (in bits) denote the data size of the task arrived in the LP queue of BS n. Specifically, If there is no task $u_{n,n'}^{\mathrm{ne}}(t)$ in the LP queue, $\lambda_{n,n'}^{\mathrm{ne}}(t)$ is set to zero. Similar as the HP queue, let $c_{n,n'}^{\mathrm{LP}}(t)$ denote the completion time of task $u_{n,n'}^{\mathrm{ne}}(t)$ in the LP queue. Then the processing, waiting and completion time are

$$\begin{aligned} e_{n,n'}^{\mathrm{LP}}(t) &= \left\lceil \lambda_{n,n'}^{\mathrm{ne}}(t)\rho_{n,n'}/f_n^{\mathrm{LP}} \right\rceil, \\ w_{n,n'}^{\mathrm{LP}}(t) &= \left[c_{n,n'}^{\mathrm{LP}}(t-1) - t\right]^+, \\ c_{n,n'}^{\mathrm{LP}}(t) &= t + w_{n,n'}^{\mathrm{LP}}(t) + e_{n,n'}^{\mathrm{LP}}(t). \end{aligned} \tag{11}$$

Besides, we define the queue length of three queues as $Q_n^i(t)$, $i \in \Phi = \{\mathrm{HP}, \mathrm{BT}, \mathrm{LP}\}$ at the end of time slot t. Given the tuple $(i,j) \in \{(\mathrm{HP},\mathrm{T1}), (\mathrm{BT},\mathrm{T2}), (\mathrm{LP},\mathrm{T3})\}$, we have

$$Q_n^i(t) = \left[Q_n^i(t-1) + j\right]^+, \tag{12}$$

where $Q_n^i(0) = 0$, $i \in \Phi$, $\mathrm{T1} = \sum_{m \in \mathcal{M}_n} \lambda_{n,m}^{\mathrm{ed}}(t) - f_n^{\mathrm{HP}}/\rho_{n,m}$, $\mathrm{T2} = \sum_{m \in \mathcal{M}_n} \lambda_{n,m}^{\mathrm{ed}}(t) - r_{n,n'}$ and $\mathrm{T3} = \sum_{n' \in \mathcal{N} \setminus \{n\}} \lambda_{n,n'}^{\mathrm{ne}}(t) - f_n^{\mathrm{LP}}/\rho_{n,n'}$.

D. Problem Description

In this work, we will study the joint computation offloading and resource allocation problem for MDs, aiming at minimizing the total latency subject to the energy consumption constraint. The problem is very challenging due to the time coupling of different tasks and the varying of environment. DRL dose not require the knowledge of the underlying mathematical model for providing online solutions to complicated sequential decision-making problems. Thus, we will introduce DRL to solve the joint problem.

III. Task Offloading and Resource Allocation Problem in MEC System

In this section, we formulate the task offloading and resource allocation problem from the perspective of DRL. The DRL framework consists of an agent, an environment and three key elements: state, action and reward. In this MEC system (called environment), each MD (called agent) can observe its state (e.g., task size, queue status) at each time slot. Once a task is generated by an MD, the MD chooses an action (i.e., computation offloading and resource allocation). Then, the MEC system will give a reward (e.g., the delay) to judge whether the action performs well or not.

A. State

Each MD $m \in \mathcal{M}_n$ of BS n observes the newly arrival task size, waiting time, and transmission time while receives the queue lengths of BSs. Let $\mathcal{S}$ denote the set of system states, i.e., $\mathcal{S} = \Omega \times \{1, 2, \cdots, T\}^2 \times \mathcal{Q}^{3N}$, where $\mathcal{Q}$ is a set of the possible queue length of a BS within the total time slots. Then we define the state of MD m as $\boldsymbol{s}_{n,m}(t) = \left(\lambda_{n,m}(t), w_{n,m}^{\mathrm{MC}}(t), w_{n,m}^{\mathrm{MT}}(t), \boldsymbol{Q}_n(t)\right)$, where $\boldsymbol{Q}_n(t) = \left(Q_n^i(t-1), i \in \Phi, n \in \mathcal{N}\right)$.

B. Action

When MD generates a new task $u_{n,m}(t)$, it will make two decisions: (i) an offloading action $(x_{n,m}(t), y_{n,m}(t))$; (ii) a resource allocation action $(f_{n,m}(t), p_{n,m}(t))$. Let $\mathcal{A} = \{0,1\} \times \{0, 1, \cdots, N\} \times [0, f_{n,m}^{\max}] \times [0, p_{n,m}^{\max}]$ denote the action space of each MD. Then, we define the action of MD m as $\boldsymbol{a}_{n,m}(t) = \left(x_{n,m}(t), y_{n,m}(t), f_{n,m}(t), p_{n,m}(t)\right)$.

C. Reward

Given the observed state $\boldsymbol{s}_{n,m}(t)$ and the chosen action $\boldsymbol{a}_{n,m}(t)$, the delay $U(\boldsymbol{s}_{n,m}(t), \boldsymbol{a}_{n,m}(t))$ can be computed by: (i) $c_{n,m}^{\mathrm{MC}}(t) - t + 1$ if $x_{n,m}(t) = 1$; (ii) $\sum_{t'=t}^{T} \upsilon(t') c_{n,m}^{\mathrm{HP}}(t') - t + 1$ if $x_{n,m}(t) = 0$ and $y_{n,m}(t) = n$, where $\upsilon(t') = \mathbb{I}\left(u_{n,m}^{\mathrm{ed}}(t') = u_{n,m}(t)\right)$ and $\mathbb{I}[\cdot]$ is the indicator function; (iii) $\sum_{t'=t}^{T} \upsilon(t') \sum_{t''=t'}^{T} \hat{\upsilon}(t'') c_{n',n}^{\mathrm{LP}}(t'') - t + 1$ if $x_{n,m}(t) = 0$ and $y_{n,m}(t) = n' \in \mathcal{N} \setminus \{n\}$, where $\hat{\upsilon}(t'') = \mathbb{I}\left(u_{n',n}^{\mathrm{ne}}(t'') = u_{n,m}(t')\right)$. Specifically, we assume that $U(\boldsymbol{s}_{n,m}(t), \boldsymbol{a}_{n,m}(t)) = 0$ if there is not a task.

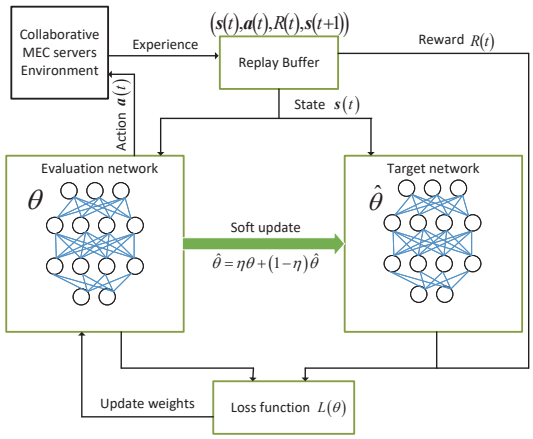


Fig. 3. DRL framework in MEC system

For simplicity, we define reward function as $R_{n,m}(t) = -U(\boldsymbol{s}_{n,m}, \boldsymbol{a}_{n,m}(t))$.

D. Problem Formulation

As stated in RL, a policy of MD m of BS n is a mapping $\pi_{n,m} : \mathcal{S} \to \mathcal{A}$. The purpose of MD m is to learn a policy that maximizes the expected long-term rewards under the energy consumption constraint by interacting with the collaborative MEC system, which can be formulated as follows:

$$\begin{aligned} \max_{\pi_{n,m}} \quad & \mathbb{E}\left[\sum_{t\in\mathcal{T}} \gamma^{t-1} R_{n,m}(t) | \pi_{n,m}\right] \\ s.t \quad & (1)-(12), \\ & \lim_{T\to\infty} \textstyle\sum_{t=1}^{T} p_{n,m}(t)/T \le P, \end{aligned} \tag{13}$$

where γ is the discount factor and P is a constant. Clearly, it is a challenging problem to solve (13). The reason for this is that an MD is unaware of the dynamically future loads at BSs, which may be caused by the action and task models of other MD. Another critical reason is the unknown future task model of the MD. Hence, we present a DRL-based online and distributed algorithm to deal with this challenging.

IV. DRL-Based Computation Offloading and Resource Allocation Algorithm

In this section, we mainly discuss computation offloading and resource allocation algorithm based on DRL, where each MD makes its own decision to learn an optimal policy distributedly. We integrate with the dueling DQN [17] and double DQN approach to solve the problem (13). As the DQN outputs discrete actions, we discretize the MD's transmit power and computation frequency decision, i.e., $p_{n,m}(t) \in \left\{0, p_{n,m}^{\max}/L, 2p_{n,m}^{\max}/L, \cdots, p_{n,m}^{\max}\right\}$ and $f_{n,m}(t) \in \left\{0, f_{n,m}^{\max}/L, 2f_{n,m}^{\max}/L, \cdots, f_{n,m}^{\max}\right\}$.

In DRL-based algorithm integrating with double and dueling DQN technique, MD m is associated with two neural networks (evaluation network and target network) and a replay buffer $\mathcal{B}_{n,m}$ (as shown in Fig.3). Two neural networks share the same architecture but different network weights, denote by $\theta_{n,m}$ and $\hat{\theta}_{n,m}$, respectively. For simplicity, we omit the subscript in the following. Given an observed state $\boldsymbol{s}(t) \in \mathcal{S}$ and a chosen action $\boldsymbol{a}(t) \in \mathcal{A}$, the evaluation network and target network of the Q-values are represented by $Q(\boldsymbol{s}(t), \boldsymbol{a}(t); \theta)$ and $\hat{Q}(\boldsymbol{s}(t), \boldsymbol{a}(t); \hat{\theta})$. The replay buffer $\mathcal{B}$ stores the experience $(\boldsymbol{s}(t), \boldsymbol{a}(t), R(t), \boldsymbol{s}(t+1))$. The experience in replay buffer is used to train the evaluation network. Here, we use ϵ-greedy approach to choose an action. That is, the MD will select a random action from $\mathcal{A}$ with probability ϵ or $\arg\max_{\boldsymbol{a}\in\mathcal{A}} Q(\boldsymbol{s}(t), \boldsymbol{a}; \theta)$ with probability $1-\epsilon$.

Algorithm 1: DRL-based algorithm at each MD

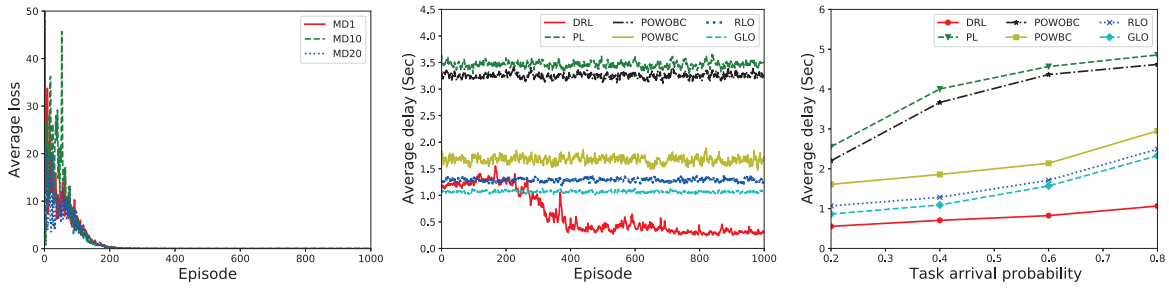
```
Initialize replay buffer B;
Initialize the evaluation network with random weights θ and target network with random weights θ̂;
for episode= 1, ..., E do
    Initialize state s(1);
    for t = 1, ..., T do
        Select action a(t) according to ε-greedy approach;
        Observe the next state s(t+1) and obtain the reward R(t) by interacting with the MEC system;
        Store the tuple (s(t), a(t), R(t), s(t+1)) in replay buffer B;
        Sample randomly a set of experiences D from B;
        Update the weights θ to minimize L(θ) by D;
        Every C steps reset θ̂ = ηθ + (1−η)θ̂;
    end for
end for
```

The MD will randomly sample a set of experience form the replay buffer memory $\mathcal{B}$, denoted by D. Based on the sampled experiences, it updates the weights θ of evaluation network by minimizing the loss function: $L(\theta) = \frac{1}{|D|}\sum_{i\in D}\left(Q(\boldsymbol{s}(i), \boldsymbol{a}(i); \theta) - y_i\right)^2$, where $y_i = R(i) + \gamma\hat{Q}(\boldsymbol{s}(i+1), \boldsymbol{a}_i; \hat{\theta})$, $\boldsymbol{a}_i = \arg\max_{\boldsymbol{a}\in\mathcal{A}} Q(\boldsymbol{s}(i+1), \boldsymbol{a}; \theta)$ and $|D|$ is the cardinality of set D. The DRL-based algorithm to be performed at the MD is given in Algorithm 1 (denoted by DRL) in details.

V. Simulation Results

In this section, we show some simulation results. The basic parameters are shown in Table I. For comparisons, we introduce some baseline schemes including pure local (denoted by PL), pure offloading without BS collaboration (denoted by POWOBC), pure offloading with BS collaboration (denoted by POWBC), random local and offloading (denoted by RLO), and greedy local and offloading (denoted by GLO). To evaluate the performance of the offloading schemes, the criteria is the average delay of all the tasks that have been finished.

Fig.4(a) depicts the convergence of the proposed algorithm at MD1, MD10 and MD20 across episodes based on the basic parameters. MD1, MD10 and MD20 converge after approximately 200 episodes, which implies the neural network is well approximate to Q-values. Fig.4(b) shows the average delay of DRL and other baseline schemes across episodes. We can see that: (i) DRL achieves an around 0.3 second average



(a) (b) (c)

Fig. 4. (a) Convergence performance across episodes; (b) Average delay across episodes; (c) Average delay under different task arrival probabilities.

TABLE I
PARAMETER SETTINGS

Parameter	Value
M, N, τ, L	30, 3, 0.1, 9
$f_{n,m}$, $p_{n,m}^{\max}$ $m \in \mathcal{U}_n$, $n \in \mathcal{N}$	2.5 GHz, 1.8W
F_n, $n \in \mathcal{N}$	41.8 GHz
$\lambda_{n,m}(t)$, $m \in \mathcal{U}_n$, $n \in \mathcal{N}$, $t \in \mathcal{T}$	$\{2.0, 2.1, 2.2, \cdots, 10.0\}$ Mbits
$\rho_{n,m}$, $m \in \mathcal{U}_n$, $n \in \mathcal{N}$	0.297 gigacycles per Mbits
Task arrival probability	0.3
Learning rate, γ	0.01, 0.9
batch size, replay buffer size	128, 1000
Hidden layers, Optimizer	[32, 128, 128, 64], Adam

delay after around 800 episodes, which outperforms the other baseline schemes; (ii) POWOBC scheme performs well than PL scheme because it makes fully use of the MEC server's resource, while DRL, RLO, POWBC and GLO schemes are better than the other two because they utilizes the adjacent MEC servers' resource; (iii) DRL can reduce the average delay by 76.4%-91.2% in terms of the baseline schemes.

A larger task arrival probability indicates the MEC servers' load level is higher. Fig.4(c) illustrates the performance of DRL and the baseline methods with the gradually increasing of the task arrival probability. With the increment of the task arrival probability from 0.2 to 0.8, the average delay of DRL increases by 0.5 second while those of the baseline schemes increase by at least 1.3 seconds. Hence, when MEC servers have the heavy loads, DRL maintains the lower average delay than the baseline schemes.

VI. CONCLUSION

In this work, we study the time-varying and collaborative system scenario, where a computation offloading and resource allocation problem is formulated to minimize the long-term delay. Due to the complex problem and dynamic environment, we devise a DRL-based online and distributed algorithm for solving it. The proposed algorithm integrates with double DQN and dueling DQN techniques and can handle the dynamically future loads of BSs and the time-varying system environment. Simulation results demonstrate the proposed algorithm outperforms the baseline schemes while it can maintain the lower average delay. In the future work, we will consider the energy consumption and energy harvesting of MDs. Furthermore, we will design a novel algorithm integrated with Actor-Critic method to address the issue where action includes the integer and continuous variables without using the discretized technique.

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*, 19(4):2322–2358, 2017.

[2] M. Tang, L. Gao, and J. Huang, "Communication, Computation, and Caching Resource Sharing for Internet-of-Things," *IEEE Communications Magazine*, 58(4):75–80, 2020.

[3] C. Jiang, L. Gao, T. Wang, Y. Jiang, and J. Li, "Crowd-MECS: A Novel Crowdsourcing Framework for Mobile Edge Caching and Sharing," *IEEE Internet of Things Journal*, 7(10):9426–9440, 2020.

[4] M. Tang, L. Gao, and J. Huang, "Enabling Edge Cooperation in Tactile Internet via 3C Resource Sharing," *IEEE Journal on Selected Areas in Communications*, 36(11):2444-2454, 2018.

[5] Z. Wang, L. Gao, T. Wang, and J. Luo, "Monetizing Edge Service in Mobile Internet Ecosystem," *IEEE Transactions on Mobile Computing*, Early Access, 2020.

[6] W. He, L. Gao, and J. Luo, "A Multi-Layer Offloading Framework for Dependency-Aware Tasks in MEC," *Proc. IEEE ICC*, 2021.

[7] C. Wang, C. Liang, *et al.*, "Computation Offloading and Resource Allocation in Wireless Cellular Networks with Mobile Edge Computing," *IEEE Transactions on Wireless Communications*, 16(8):4924–4938, 2017.

[8] Y. Liu, K. Xiong, *et al.*, "UAV-assisted Wireless Powered Cooperative Mobile Edge Computing: Joint Offloading, CPU Control, and Trajectory Optimization," *IEEE Internet of Things Journal*, 7(4):2777–2790, 2019.

[9] X. Lyu, W. Ni, *et al.*, "Distributed Online Optimization of Fog Computing for Selfish Devices with Out-of-Date Information," *IEEE Transactions on Wireless Communications*, 17(11):7704–7717, 2018.

[10] C. Yi, J. Cai, K. Zhu, and R. Wang, "A Queueing Game based Management Framework for Fog Computing with Strategic Computing Speed Control," *IEEE Transactions on Mobile Computing*, Early Access, 2020.

[11] M. Tang and V. W. Wong, "Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems," *IEEE Transactions on Mobile Computing*, Early Access 2020.

[12] S. Xia, Z. Yao, Y. Li, and S. Mao, "Online Distributed Offloading and Computing Resource Management with Energy Harvesting for Heterogeneous MEC-enabled IoT," *IEEE Transactions on Wireless Communications*, 20(10):6743–6757, 2021.

[13] X. Chen, H. Zhang, *et al.*, "Optimized Computation Offloading Performance in Virtual Edge Computing Systems via Deep Reinforcement Learning," *IEEE Internet of Things Journal*, 6(3):4005–4018, 2018.

[14] 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*, 19(11):2581–2593, 2019.

[15] Y. He, N. Zhao, and H. Yin, "Integrated Networking, Caching, and Computing for Connected Vehicles: A Deep Reinforcement Learning Approach," *IEEE Transactions on Vehicular Technology*, 67(1):44–55, 2017.

[16] C. Sun, X. Wu, *et al.*, "Cooperative Computation Offloading for Multi-Access Edge Computing in 6G Mobile Networks via Soft Actor Critic," *IEEE Transactions on Network Science and Engineering*, Early Access, 2021.

[17] Z. Wang, T. Schaul, *et al.*, "Dueling Network Architectures for Deep Reinforcement Learning," *Proc. PMLR*, 2016.