Article

Not peer-reviewed version

Multi-Agent DRL-Based Dynamic Task Offloading in D2D-MEC Network to Minimize Average Task Delay with Deadline Constraints

Huaiwen He , Xiangdong Yang , Xin Mi , Hong Shen , Xuefeng Liao [*]

Posted Date: 7 June 2024

doi: 10.20944/preprints202406.0480.v1

Keywords: Mobile Edge Computing, Dynamic Matching, D2D, Delay Constraint, Multi-agent Reinforcement Learning

Article

Multi-Agent DRL-Based Dynamic Task Offloading in D2D-MEC Network to Minimize Average Task Delay with Deadline Constraints

Huaiwen He [1], Xiangdong Yang [1,2], Xin Mi [1], Hong Shen [3] and Xuefeng Liao [4,*]

[1] School of Computer, Zhongshan Institute, University of Electronic Science and Technology of China, Zhognshan 528400, China
[2] Computer Science and Engineering School, University of Electronic Science and Technology of China, China; yangxiangdong.cs@aliyun.com
[3] Engineering and Technology, Central Queensland University, Australia; hongsh01@gmail.com
[4] School of Data Science and Artificial Intelligence, Wenzhou University of Technology, Wenzhou, China; Liaoxuefeng@wzut.edu.cn
* Correspondence: Liaoxuefeng@wzut.edu.cn

Abstract: Device to Device (D2D) is a pivotal technology in the next generation of communication, allowing for direct task offloading between mobile devices (MDs) to improve the efficient utilization of idle resources. This paper proposes a novel algorithm for dynamic task offloading between active MDs and idle MDs in the D2D-MEC (Mobile Edge Computing) system by deploying multi-agent deep reinforcement learning (DRL) to minimize the long-term average delay of delay-sensitive tasks under deadline constraints. Our core innovation is a dynamic partitioning scheme for idle and active devices in the D2D-MEC system, accounting for stochastic task arrivals and multi-time-slot task execution, which has been insufficiently addressed in existing literature. We adopt a queue-based system to formulate a dynamic task offloading optimization problem. To address the challenges of large action space and coupling of actions across time slots, we model the problem as a Markov Decision Process (MDP) and perform the multi-agent DRL through Multi-Agent Proximal Policy Optimization (MAPPO). We employ a centralized training with decentralized execution (CTDE) framework to enable each MD to make offloading decisions solely based on its local system state. Extensive simulations demonstrate the efficiency and fast convergence of our algorithm. In comparison with the existing sub-optimal results deploying single-agent DRL, our algorithm reduces the average task completion delay by 11.0% and the ratio of dropped tasks by 17.0%.

Keywords: mobile edge computing; dynamic matching; D2D; delay constraint; multi-agent reinforcement learning

1. Introduction

Recently , Mobile Edge Computing (MEC) has emerged as a promising computing paradigm that aims to reduce response time for computation tasks and enhance the Quality of Experience (QoE) of users by offloading tasks to edge servers [1]. However, the dynamic and random nature of task arrivals can lead to a significant increase in workload during certain periods. This surge in workload poses challenges for edge servers, making it difficult to meet the latency requirements of tasks like face recognition, virtual reality, augmented reality, online games, and more. D2D technology addresses this issue by enabling MDs to offload tasks directly to idle MDs, facilitating resource utilization and collaboration across the network. [2].

In D2D-MEC networks, MDs typically operate in two modes: requester MDs (active devices) and server MDs (idle devices) [2]. Requester MDs can offload tasks to server MDs or edge server, while server MDs not only handle their own tasks but also accept tasks from other requester MDs. This approach can effectively utilize idle resources within the network by enabling task offloading and collaboration among mobile devices. However, most existing research in this field primarily emphasizes static partitioning of MDs, where active devices and idle devices are pre-determined [3]. This restricts the flexibility of D2D communication in real-world scenarios. Additionally, the dynamic nature of task arrivals and the need for collaboration among active devices present considerable challenges for task offloading in D2D-MEC.

In order to ensure a high Quality of Experience (QoE) for users in Mobile Edge Computing (MEC) systems, it is crucial to process tasks within their deadlines, especially for delay-sensitive tasks. Zuo et al. [4] propose an alternating iterative algorithm, based on continuous relaxation and greedy rounding (CRGR), to achieve the Nash equilibrium. Abbas et al. [5] introduce an algorithm that maximizes the number of completed tasks through hierarchical and iterative allocation while minimizing energy consumption and monetary costs. Hamdi et al. [6] put forth a layered optimization method to maximize energy efficiency (EE) in D2D-MEC systems under delay constraints. However, none of the aforementioned studies considered stochastic task arrivals or dynamic partitioning of D2D devices.

The DRL algorithms have been successfully applied to the task offloading problem in MEC in various existing works [7–11]. [7,8] propose learning algorithms based on Q-learning or deep Q-learning network (DQN) for task offloading in MEC systems. Luo et al. [9] introduce a distributed learning algorithm based on the asynchronous advantage actor-critic (A3C) technique to optimize energy consumption and quality of experience in software-defined mobile edge networks. Li et al. [11], Qiao et al. [12] design a reinforcement learning framework for D2D edge computing and networks to address challenges related to the dynamic nature and uncertainty of the environment. However, most of the aforementioned studies predominantly utilize single-agent reinforcement learning algorithms or overlook the unknown load dynamics at each mobile device. Due to the dynamical and random nature of MEC systems, the decision space size increases exponentially with the number of mobile devices, which may bring the curse of dimensionality and struggle to converge for single-agent DRL algorithms.

In this paper, we investigate the dynamic task offloading problem in D2D-MEC system for delay-sensitive tasks under delay constraints. Our objective is to minimize the long-term average task delay under deadline constraints. To achieve this, we propose a dynamic D2D partitioning approach for MDs set based on queuing-based system, considering the dynamic load level of MDs and the presence of multiple time-slot tasks. To tackle the challenges posed by a huge decision space and coupling of actions across time slots, we formulate the problem as a cooperative Markov game and propose a multi-agent(MA) DRL-based algorithm based on MAPPO technique and implementation in CDTE manner. Our main contributions can be summarized as follows:

- We introduce a novel dynamic D2D partitioning method based on queuing system for handling delay-sensitive tasks in D2D-MEC networks. Furthermore, we formulate the problem of minimizing the long-term average task delay under deadline constraints as a dynamic assignment problem, considering the random load level at MDs and multi-slot spanned tasks. Our proposed model surpasses existing approaches by providing a more precise characterization of task latency and improving the utilization of network computing resources. Additionally, it exhibits superior scalability and practicality.
- We formulate the dynamic offloading problem as a cooperative Markov game and propose a multi-agent DRL-based algorithm utilizing the MAPPO technique to address the exponential growth of the decision space. Our proposed algorithm, based on the CDTE framework, enables online task decision-making in the dynamic and volatile network environment, relying solely on its local observations.
- We conduct comprehensive experiments and the numerical results demonstrate the effectiveness and fast convergence of our proposed algorithm in a time-varying system environment. Compared to the sub-optimal outcomes obtained by deploying single-agent DRL, our algorithm, which enables distributed decision-making, achieves a significant reduction of 11.0% in average task completion delay and a 17.0% decrease in ratio of dropped tasks.

The rest of this paper is organized as follows. Section 2 presents a review of related works. In Section 3, we provide details of system model. In Section 4, we formulate the task offloading problem with delay constraint as a dynamic assignment problem. In Section 4, we propose a multi-agent

DRL-based algorithm based on MAPPO technique. The experimental setup and numerical results are presented in Section 6.1. Finally, Section 7 concludes this paper.

2. Related Works

Studies on optimization to reduce the latency of delay-sensitive task in D2D-MEC network have been attract plenty of attentions. Yang et al. [13] propose a novel offloading framework for the multi-server MEC network to jointly optimize jobs offloading decision and computing resource allocation by using multi-task learning. Chai et al. [14] propose a heuristic algorithms based on Kuhn–Munkres algorithm and Lagrangian dual method for jointly computation offloading and resource allocation in a D2D-MEC systems. However, the above studies did not consider the cooperation among mobile devices. He et al. [15,16] classify MDs into active and idle devices based on their ability to complete tasks on time through local computing or according to devices' running states (including computing and transmitting states). Peng et al. [1] take into account the dynamic partitioning of devices, and proposed an online resource coordinating and allocating scheme based on Lyapunov optimization framework. But they assume that data computing and transmitting are implement in parallel without considering the waiting time in the queue.

Due to the ability to extract valuable knowledge from the environment and make adaptive decisions, DRL technology has received significant attention recently for edge computing task offloading [7,8,17,18]. Chen et al. [8] first propose a DQN-based algorithm to handle the huge state spaces and learn the optimal computation offloading policy. Wang et al. [19] propose a task offloading algorithm based on meta-reinforcement learning to enable the model to quickly adapt to different environments. However, the above research works are based on single-agent and make centralized decision, which maybe impractical in the real world as the number of MDs increasing due to exploration of the decision space in MEC system. Some works based on multi-agent RDL scheme are proposed for task offloading problems [20–22]. However, the majority of these approaches relied on the Multi-Agent Deep Deterministic Policy Gradient(MDDPG) framework, which may encounter training instability and necessitates training distinct policy networks and value networks for each individual agent. Consequently, this results in heightened computational complexity within large-scale multi-agent systems.

3. System Model

In this paper, we consider a heterogeneous MEC network system consisting of D mobile devices and an edge server with D2D communication support, which is denoted as $\mathcal{D} = \{0, 1, 2, \ldots, D\}$, where 0 represents the edge server. The system architecture is illustrated in Figure 1. We assume that the D2D-MEC system operates in discrete time slots represented by $\mathcal{T} = \{1, 2, \ldots, T\}$, where each time slot last for Δt seconds. In each time slot, tasks are generated in each MD in a certain probability.

In the following subsections, we present the details of the device model, computation model, transmission model, energy model, and delay model employed in the system. The key notation used in this paper is summarized in Table 1.

To accurately mode the service time of computation task, we adopt queuing system to represent the task processing procedure. Each MD $d \in [1, D]$ has two types of queues: computation queue Q_d^{comp} used for task execution and transmission queue Q_d^{tran} used to offload task to D2D devices or edge server. For the edge server, there only exists one computation queue, denoted as Q_0^{comp}.

Since the load of MDs are varying stochastic, we dynamically divide MDs set into idle devices (requester) and active devices (server) based on the backlog of their computation queues, which are defined as follows:

1) Idle device: In time slot t, $Q_d^{comp}(t) = \emptyset$. Idle devices can provide computing service for other MDs by D2D link.

2) Active device: In time slot t, $Q_d^{comp}(t) \neq \emptyset$. Active device only process task locally, or offload tasks to edge server or idle MDs, but cannot accept tasks from other MDs.

Table 1. KEY NOTATIONS.

Symbol	Definition
$\mathcal{D}$	The set of mobile deivces
$\mathcal{T}$	The whole time slots
Δt	The duration of time slot t
$w_m(t)$	The computation task generated on mobile device m at time slot t
s_m^t	The size of task $w_m(t)$
c_m^t	The computation complexity of task $w_m(t)$
τ_m^t	The deadline of task $w_m(t)$
$\boldsymbol{A(t)}$	The task offloading decision for all MDs at time slot t
$a_m^t(t)$	The offloading decision of the mth active device at time slot t
Q_d^{comp}	The computing queue of mobile device d
Q_d^{tran}	The transmission queue of mobile device d
$l_{m,d}^{comp}(t)$	The time slot when task $w_m(t)$ is fully processed at mobile device d
$l_m^{tran}(t)$	The time slot when task $w(m,t)$ transmission is completed at device m
$\phi_{m,d}^{comp}(t)$	The waiting time slots of task $w_m(t)$ in computation queue at mobile device d
$\phi_m^{tran}(t)$	The waiting time slots of task $w_m(t)$ in transmission queue at mobile device m
$r_m(t)$	The transmission rate of mobile device m at time slot t
p_m	The transmission power at device m
$h_m^d(t)$	The channel gain between active device m and idle device d at slot t
N_0	The white noise
$B_m(t)$	The bandwidth of device m at time slot t
$\mathcal{L}(m,t)$	The total duration of task $w(m,t)$ from generation to execution completion
$S_m(t)$	Local observation information of device m on time slot t
$S(t)$	Global state information on time slot t
$r(t)$	Reward in time slot t

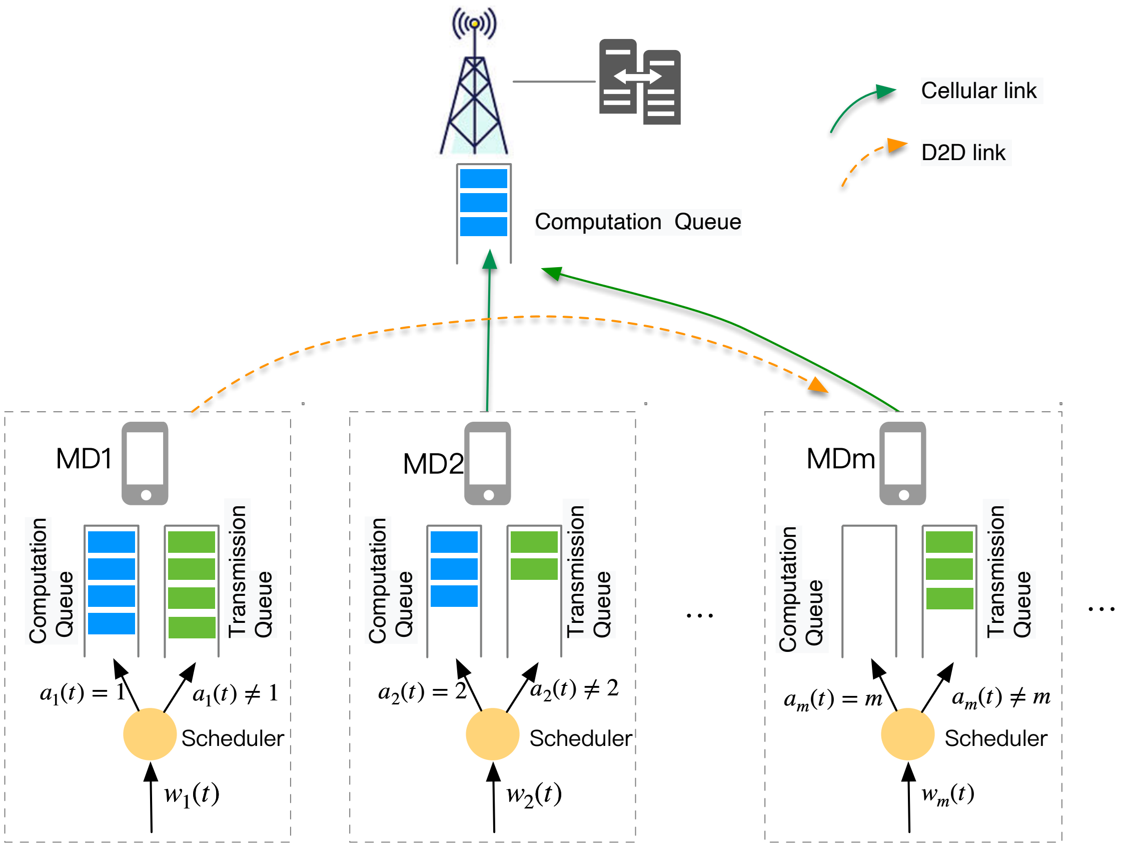


Figure 1. Architecture of D2D-MEC network.

3.1. Task Model

We consider time-sensitive tasks, such as virtual reality (VR) or animation rendering, that can span over multiple time slots. At time t, each MD m stochastically generate a computation task, denoted as $w_m(t)$, specified by a three-tuple $< s_m^t, c_m^t, \tau_m^t >$, where s_m^t represents the size of task w in 1-bit unit, c_m^t denotes the computation complexity of task w measured in the number of CPU cycles required for a 1-bit operation, and τ_m^t indicates the task deadline for completion (an integer multiple of the time slot). Each job should be processed within τ_m^t time slot to avoid incurring an expiration penalty.

We assume that each task is indivisible such that the system adopt a binary offloading mode, which allows tasks to be executed either locally or offloaded to edge server through cellular links, or transferred to idle devices via D2D links. Each MD employs with a scheduler that determines the target device for task execution. The active devices set at slot t is denoted as $\mathcal{AD}(t) = \{ad_1, ad_2, \ldots, ad_{|\mathcal{AD}(t)|}\}$, and the idle devices at slot t is denoted as $\mathcal{ID}(t) = \{id_0, id_1, id_2, \ldots, id_{|\mathcal{ID}(t)|}\}$, where id_0 represents the edge server which is always regarded as idle device due to its sufficient computing resource. Hence, we have $|\mathcal{AD}(t)| + |\mathcal{ID}(t)| = D$.

3.2. Computation Model

The computation queue follows the First-In-First-Out (FIFO) principle, where tasks in the queue buffer must wait for the completion of the preceding task before being scheduled. The computing delay of a task is composed of two parts: waiting time and execution time.

Let $\widetilde{t}$ denote the time slot of task $w_m(t)$ placed in the queue Q_d^{comp} at device $d \in \mathcal{D}$, and $l_{m,d}^{comp}(t)$ denote the time slot of task $w_m(t)$ to be completely processed or dropped in device d. Note that if task is computed locally, we have $\widetilde{t} = t$ and $l_{m,d}^{comp}(t)$ can be written as $l_m^{comp}(t)$. Hence, the amount of time slots that task $w_m(t)$ will wait for scheduling to execute in computation queue Q_d^{comp} can be expressed as follows :

$$\phi_{m,d}^{comp}(t) = \left[\max_{t' \in \{0,1,\cdots,t-1\}, d' \in [1,D]} l_{d',d}^{comp}(t') - \widetilde{t} + 1 \right]^+ \tag{1}$$

where the operator $[x]^+ = max\{0, x\}$, and we set $l_{d',d}^{comp}(0) = 0$ for presentation simplicity. The term $\max_{t' \in \{0,1,\cdots,t-1\}, d' \in [1,D]} l_{d',d}^{comp}(t')$ determines the time slot when all the task placed in the computation queue Q_d^{comp} before time slot $\widetilde{t}$ has either been processing or dropped. Hence, $\phi_{m,d}^{comp}(t)$ determines the number of waiting time slots in the computation queue.

For each task $w_m(t)$ that arrives at the computation queue Q_d^{comp} at the beginning of time slot $\widetilde{t}$, it will either be processed completely or dropped in slot $l_{m,d}^{comp}(t)$:

$$l_{m,d}^{comp}(t) = \min\left\{ \widetilde{t} + \phi_{m,d}^{comp}(t) + \lceil \frac{s_m^t c_m^t}{f_d \Delta t} \rceil - 1, t + \tau_m^t - 1 \right\} \tag{2}$$

where f_d represents the process rate of device d. Specifically, the task $w_m(t)$ will be scheduled to execute at the beginning of slot $\widetilde{t} + \phi_{m,d}^{comp}(t)$. $\lceil \cdot \rceil$ is the ceiling function, which means devices only switch to execute next task at the beginning of a new time slot. The term $\left\lceil \frac{s_m^t c_m^t}{f_d \Delta t} \right\rceil$ represents the total time slots required to process task completely. Considering the deadline of task, here use min operator to determine the minor value of process completely and the drop time slot.

In the case when multiple offloaded tasks or a newly generated task arrive at device d simultaneously, they will enter queue Q_d^{comp} according to the following rules: (1) offloaded tasks have higher priority than locally generated tasks. (2) among the offloaded tasks, tasks with shorter deadlines will be placed ahead in the queue.

3.3. Task Offloading Model

We assumed that cellular links and D2D links operate on different frequency bands and adopt orthogonal frequency division multiple access (OFDMA) for access. Therefore, communication between any two devices does not interfere with the communication between other devices [23]. The transmission queue $Q_d^{tran}, d \in [1, D]$ also follows FIFO principle. Similar to computation queue, let $l_m^{tran}(t)$ denote the slot when $w_m(t)$ is successfully transmitted to the target device or dropped since from it enters queue Q_d^{tran}, that we have the amount of time slots that $w_m(t)$ will wait for transfer in Q_d^{tran} as follows:

$$\phi_m^{tran}(t) = \left[\max_{t' \in \{0,1,\cdots,t-1\}} l_m^{tran}(t') - t + 1 \right]^+ \tag{3}$$

where $l_m^{tran}(t)$ represents the time slot that task $w_m(t)$ leaves the queue Q_d^{tran}, and we set $l_m^{tran}(0) = 0$ for presentation simplicity. The term $\max_{t' \in \{0,1,\cdots,t-1\}} l_m^{tran}(t')$ determines the time slot when all the task placed in the transmission queue before time slot t has either been processing or dropped.

In mobile device $m \in [1, D]$, task $w_m(t)$ placed in the transmission queue at the beginning of time slot t, then task $w_m(t)$ will either be completely sent or dropped in time slot $l_m^{tran}(t)$:

$$l_m^{tran}(t) = \min\left\{ t + \phi_m^{tran} + \operatorname{argmin}\left\{ s_m^t \leq \sum_{i=t}^{t+\theta} r_m(i)\Delta t \right\}, t + \tau_m^t - 1 \right\} \tag{4}$$

where $r_m(t)$ represents the transmission rate of MD m in t. Specifically, the task $w_m(t)$ will start to transmit at the beginning of time slot $t + \phi_m^{tran}(t)$. The term $\operatorname{argmin}\left\{ s_m^t \leq \sum_{i=t}^{t+\theta} r_m(i)\Delta t \right\}$ represents the total time slots required to transfer data successfully, where $r_m(i)\Delta t$ represents transfer data size in slot i. Hence, $l_m^{tran}(t)$ decides the time slot when task will either be sent successfully or dropped.

The transmission rate $r_m(t)$ of MD m at t can be calculated based on Shannon's theorem:

$$r_m(t) = B_m(t) \log_2(1 + \frac{p_m h_m^d(t)}{N_0}) \tag{5}$$

where $B_m(t)$ is the bandwidth allocated to MD m and p_m represents the transmission power of device m, which is a constant value. $h_m^d(t)$ represents the channel gain between device m and target device d at time slot t, which keeps fix within a time slot and follows a Rayleigh distribution that varies over time. N_0 represents the white noise during transmission. Here we adopt the average bandwidth allocation scheme, that the total bandwidth is allocated equally among each pair of communicating nodes.

Due to the negligible size of task results compared to the original task data, the transmission time for task result return is extremely short. Therefore, similar to [17], the transmission time for task result return is ignored.

3.4. Task Delay Model

Let $a_m(t) = m$ denote the target device to execute task $w_m(t)$, we have $a_m(t) \in \mathcal{ID}(t) \bigcup \{m\}$.

If $a_m(t) = m$, which means task $w_m(t)$ will be processed locally, base on eq.(2) we have the total delay $\mathcal{L}_m(t) = l_m^{comp}(t) - t + 1$, which can be written as

$$\mathcal{L}_m^{loc}(t) = \min\left\{ \phi_m^{comp}(t) + \left\lceil \frac{s_m^t c_m^t}{f_m \Delta t} \right\rceil, \tau_m^t \right\} \tag{6}$$

If $a_m(t) \neq m$, then task $w_m(t)$ will be offloaded to remote device d to process. Such that transmission delay is $\mathcal{L}_m^{tran}(t) = l_m^{tran}(t) - t + 1$, processing delay is $\mathcal{L}_m^{comp}(t) = \phi_{m,d}^{comp}(t) + \lceil \frac{s_m^t c_m^t}{f_m \Delta T} \rceil$. Hence we have the total delay of remote execution task as :

$$\mathcal{L}_m^{rem}(t) = \min\left\{ \mathcal{L}_m^{tran}(t) + \mathcal{L}_m^{comp}(t), \tau_m^t \right\} \tag{7}$$

Therefore, the total delay of task $w_m(t)$ can be derived as follows:

$$\mathcal{L}(m,t) = \mathbb{1}_{(a_m(t)=m)} \mathcal{L}_m^{local}(t) + \mathbb{1}_{(a_m(t)\neq m)} \mathcal{L}_m^{rem}(t) \tag{8}$$

where $\mathbb{1}(\cdot)$ is an indicator function that outputs 1 when $\cdot$ is true and 0 otherwise.

4. Problem Formulation

In this paper, we aim to minimize the long-term average task delay under deadline constraints by making task offloading decisions $\mathcal{A}(t) = \{a_1(t), a_2(t), \ldots, a_d(t)\}$ at each MD in each time slot t. The objective function at slot t is written as $\frac{1}{D}\sum_{d=1}^{D} \mathcal{L}(d,t)$. Therefore, the task offloading optimization can be formulated as:

***P*1 :**

$$\text{(P1)} \quad \min_{\mathcal{A}(t)} \frac{1}{T}\sum_{t=1}^{T} \frac{1}{D}\sum_{d=1}^{D} \mathcal{L}(d,t)$$

s.t.

$$\mathcal{L}(d,t) \leq \tau_d^t, \forall d \in [1,D] \tag{9a}$$

$$a_d(t) \in \mathcal{ID}(t) \bigcup \{d\}, \forall d \in [1,D] \tag{9b}$$

In problem ***P*1**, constraint (9a) ensures tasks are completed or dropped upon reaching their deadlines, minimizing average task delay. Constraint (9b) specifies the target device for task execution, including both idle devices and the local device where the task is generated.

Problem ***P*1** can be classified as a dynamic assignment problem, characterized by its complexity in high-dimensional space and being NP-hard. The time complexity of this problem increases exponentially with the cardinality of the sets of available devices $|\mathcal{AD}(t)|$ and idle devices $|\mathcal{ID}(t)|$ at each time slot t. Additionally, the strong coupling of action decisions across multiple time slots exacerbates the computational challenge.

Even the offline information of system is known advance, it is challenging to be solve by traditional technique due to the curse of dimensionality. Hence, we propose a novel algorithm with low complexity that operates online using a multi-agent DRL framework.

5. Algorithm Design

Proximal Policy Optimization (PPO) is a robust policy gradient algorithm within the actor-critic framework, demonstrating exceptional performance in diverse reinforcement learning tasks [24]. Its simplicity and effectiveness have made it a popular choice for numerous RL applications, addressing the stability and sample efficiency concerns of traditional policy gradient methods through careful policy network update clipping and the use of a surrogate objective function.

Moving beyond single-agent DRL methods, MAPPO extends the capabilities of PPO, particularly excelling in scenarios where multiple agents need to interact and make decisions independently. This extension offers a scalable and efficient solution for cooperative decision-making in complex environments, showcasing its significance in addressing challenges that go beyond the scope of single-agent approaches.

In this section, we present a novel framework for task offloading based on multi-agent DRL technique. Leveraging the MAPPO technique, our proposed algorithm, builds upon the cooperative decision-making abilities of MAPPO to effectively address the dynamic offloading challenges in MEC systems. By formulating the problem as a cooperative Markov game and employing the CTDE architecture, our algorithm provides a robust and efficient solution for dynamic offloading in MEC systems, effectively managing the heavy communication burden and offering scalability in decision-making processes.

Firstly, we formulate the problem of minimizing long-term average task delay as a cooperative Markov game.

5.1. MDP of P1

5.1.1. State

Our algorithm incorporates both local and global states. The local state includes the system information of each MD and is used to train the Actor network for individual decision-making. On the other hand, the global state comprises the entire system information at the current time slot and is utilized to train the Critic network.

At the beginning of slot t, each device $m \in [1, D]$ observes its state information, which includes task properties such as task size, task complexity, and expiration time, as well as information about the computation queue, transmission queue, and channel state. Let $B_m^{comp}(t) = \max_{t' \in 0,1,\cdots,t-1, m' \in [1,D]} l_{m',m}^{comp}(t') - t + 1$ denote the backlog of computation queue at slot t, and $B_m^{tran}(t) = \max_{t' \in 0,1,\cdots,t-1} l_m^{tran}(t') - t + 1$ denote the backlog of transmission queue MD, we obtain the local state vector as follows:

$$\mathcal{S}_m(t) = \left(s_m^t, c_m^t, \tau_m^t, B_m^{comp}(t), B_m^{tran}(t), h_m(t), \overrightarrow{id(t)} \right) \tag{10}$$

where $\overrightarrow{id(t)}$ is a two dimension vector, consists of the index and the backlog of computation queue of all idle devices. We assume that each idle MD will broadcast its state of computation queue at the end of each time slot. For device m at time slot t, if no task is generated, s_m^t, c_m^t, τ_m^t are both set to be 0.

The global state integrates the local state information from each MD and the queue information from the edge server, denoted as:

$$\mathcal{S}(t) = \left(\mathcal{S}_1(t), \mathcal{S}_2(t), \ldots, \mathcal{S}_D(t) \right) \tag{11}$$

5.1.2. Action

At the beginning of slot t, when MD m generates a task $w_m(t)$, the scheduler decides whether to execute the task locally or offload it to an idle device. Therefore, the action vector of each MD can be present as:

$$\mathcal{A}(t) = \left(id_0, id_1, id_2, \ldots, id_{\mathcal{ID}} \right) \cup m \tag{12}$$

5.1.3. Reward

According to problem ***P1***, the optimization objective is the ratio of the total task delay to the task deadline. Therefore, the reward signal can be directly defined as:

$$r(t) = \frac{\sum_{m \in [1,D]} \mathbb{1}_{(a_m^t = m)} \mathcal{L}_m^{local}(t) + \mathbb{1}_{(a_m^t \neq m)} \mathcal{L}_m^{rem}(t)}{\sum_{m \in [1,D]} \mathbb{1}(s_m^t) \neq 0)} \tag{13}$$

where the item $\sum_{m \in [1,D]} \mathbb{1}(s_m^t) \neq 0)$ represents all tasks generated in time slot t.

5.2. Mutil-Agent DRL-Based Algorithm

To reduce the exponential decision space of P1, we propose a multi-agent DRL-based algorithm that leverages the state-of-the-art multi-agent RL named MAPPO. MAPPO exhibit significantly higher algorithmic runtime efficiency and comparable data sample efficiency compared to off-policy algorithms under limited computing resources, which make it well-suited for MEC system. Here we adopt the popular CTDE architecture, which consists of two crucial components: one central Critic network and multi Actor networks. The architecture of the Multi-agent DRL algorithm is depicted as Figure 2. The process involves two phases, described below.

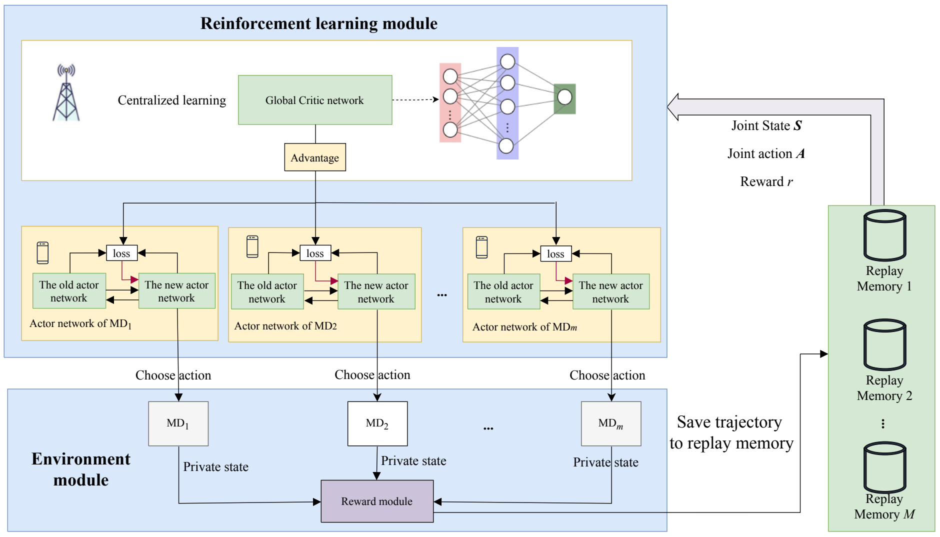


Figure 2. Architecture of Multi-agent DRL algorithm

1) Centralized training

We utilize a global Critic network to obtain the accurate evaluations of the global system states to guide the training of Actor networks on each MD. Similar to PPO algorithm, these networks are trained as follows:

a) Firstly, each agent m interacts with environment by randomly sampling actions based on its observed system state $\mathcal{S}_m(t)$ and executing the selected action. Then agents observe the local state in the next time slot, obtain the reward $r_m(t+1)$. The trajectory data $(\mathcal{S}_m(t), a_m(t), r_m(t+1), \mathcal{S}_m(t+t))$ is stored in the local experience replay pool.

Afterward, agent m sent its local state $\mathcal{S}_m(t)$, the next time step state $\mathcal{S}_m(t+1)$, and local reward $r_m(t+1)$ to the central controller for further processing.

b) By aggregating the information from each agent, we acquire the global state $\mathcal{S}(t)$, $\mathcal{S}(t+1)$ and global reward $r(t)$. These values serve as inputs to the Critic network to for computing the state values $V(S)$ and $V(S^{'})$. Here we employ a target network to compute the Temporal Difference (TD) target and TD error $\delta_t = r(t) + \gamma V(\mathcal{S}(t+1)) - V(\mathcal{S}(t)$, where γ is the discount factor.

In the training phase, the system samples a batch of trajectories from the replay memory in the central controller and performs updates on the Critic network. The update equation for the Critic network can be be expressed as follows:

$$\mathcal{L}(\phi_{cri}) = \mathbb{E}_{S(t),\mathcal{A}(t)}[r(t) + \gamma V_{\phi_{cri}}(S(t+1)) - V_{\phi_{cri}}(S(t))]^2 \tag{14}$$

where $V_{\phi_{cri}}(.)$ is the value function of Critic network under parameter ϕ_{cri}.

We employ Generalized Advantage Estimation (GAE) trick to compute the advantage function as follows,

$$\hat{A}_t = \sum_{l=0}^{\infty}(\gamma\lambda)^l \delta_{t+l} \tag{15}$$

where γ is used to determines the importance given to future rewards, while λ is a parameter similar to $TD(\lambda)$, with a trade-off between variance and bias. δ^V_{t+l} refers to the $l-step$ TD error, which is computed as $\delta_{t+l} = r(t+l) + \gamma V(t+l+1) - V(t+l)$.

The value of GAE will be broadcasted to each agent for training the actor network.

c) Upon receiving the advantage function $\hat{A}_t$, each agent conducts batch sampling from its own experience replay pool and calculates the surrogate value by considering the probability distribution of old and new actions along with the advantage function $\hat{A}_t$. This surrogate value is used to update the parameters of the actor network. The update equation for the Actor network at agent m is as follows:

$$\mathcal{L}(\theta_m^{act}) = \mathbb{E}_{\pi_{\theta_m}}[\min(\psi_t(\theta_m^{act})\hat{A}_t, \text{clip}(\psi_t(\theta_m^{act}), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{16}$$

where $\psi_t(\theta_m^{act}) = \frac{\pi_{\theta_m}(a_m(t)|S_m(t))}{\pi^{old}_{\theta_m}(a_d(t)|S(t))}$ represents the probability ratio, ϵ is a hyperparameter in PPO that limits the deviation between the new and old networks and $\hat{A}_t$ is used to assess the quality of action $\mathcal{A}(t)$ in state $S(t)$.

2) Decentralized Execution

After completing centralized training, the Critic network becomes unnecessary. The Actor network is deployed to each individual MD and utilizes locally observed system states for decision-making. Communication is not required during the decision-making process. Decentralized decision-making is fast and can be performed in real-time.

The details of our MARL dynamic offloading algorithm are summarized in Algorithm 1.

Algorithm 1: Multi-agent DRL-Based Dynamic Offloading Algorithm

```
Input: Equipment Set $E$, Total Episode number $N$, Time domain $\mathcal{T}$
Output: Offloading policy $\mathcal{A}$
1  for episode $n \leftarrow 1$ to $N$ do
2      for time slot $t \in \mathcal{T}$ do
3          for equipment $e \in E$ do
4              for task $w \in Q_e^{comp} \bigcup Q_e^{tran}$ do
5                  if task $w$ expires then
6                      drop $w$;
7                  end
8              end
9          end
10         for device $d \in \mathcal{D}$ do
11             Use actor network to output action probability distribution;
12             Sample the action $a_d(t)$ of current time slot;
13         end
14         Allocate equal bandwidth to each communication pair based on $\mathcal{A}(t)$;
15         Interact with the environment using joint actions $\mathcal{A}(t)$;
16         Get the next state $s(t+1)$ and cost $C(S(t), \mathcal{A}(t))$;
17         for $d \in \mathcal{D}$ do
18             storing the trajectories $(s_d(t), a_d(t), C(S(t), \mathcal{A}(t)), s_d(t+1))$ into the replay buffer
                $RB_d$;
19         end
20     end
21     Sample the same batch of trajectories from the replay buffers of different devices;
22     Update the Global Critic network using (14);
23     Update the Actor network using (16);
24 end
```

6. Simulation Result and Analyses

6.1. System Parameter Settings

To verify our algorithm, we conducted extensive simulations, demonstrating its convergence and performance superiority compared to baseline algorithms. Table 2 lists the basic system parameters.

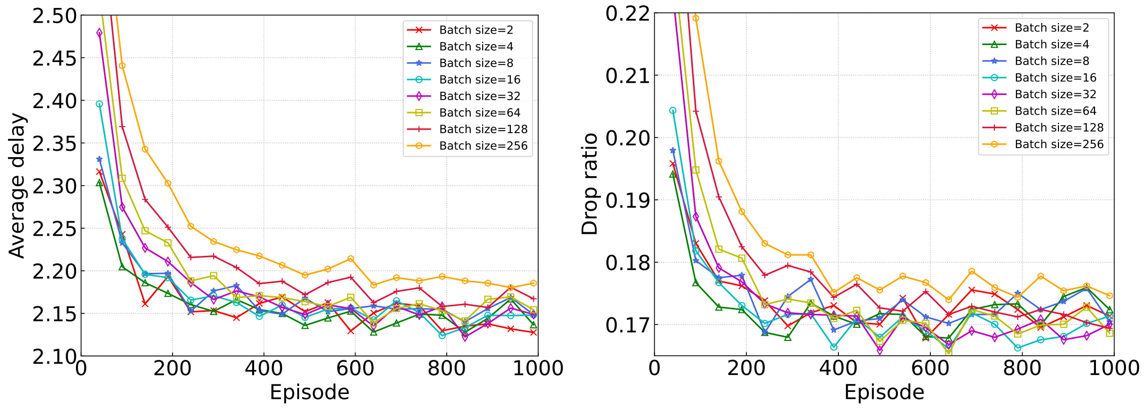
Table 2. Simulation system parameters.

parameters	values
Mobile device number D	20
The CPU frequency of mobile device f_d	2 GHz
The CPU frequency of edge server f_0	3 GHz
Minimum task size s_{min}	3 Mbits
Maximum task size s_{max}	10 Mbits
Minimum task complexity c_{min}	0.5 gigacycles per Mbits
Maximum task complexity c_{max}	2 gigacycles per Mbits
Minimum task generation probability	0.1
Maximum task generation probability	0.5
Total bandwidth B	3 MHz
Device transmission power p_d	3 W
White noise N_0	-14 dbm\Hz

The centralized Critic networks and the Actor network in each MD utilize a three-layer network structure with 64 nodes in the intermediate layer. During the neural network training process, a batch size of 64 is employed, the learning rate is set to 1e-3, and the number of MDs is fixed at 20. The task generation probability gradually increases from 0.1 to 0.5. Network parameters are updated using the Adam optimizer. To ensure high randomness and unpredictability in the experimental data generation, the relationship between random numbers and the task generation probability determines whether a task is generated. The channel gains between devices follow the Rayleigh distribution.

6.2. Algorithm Convergence Performance

In this part, we mainly study the convergence performance of the proposed algorithm under different values of different hyper-parameters. We consider 1000 episodes and each episode has 100 time frames. The experimental results are shown in Figure 3, where the X-axis represents the episodes and the Y-axis represents the average completion delay of the task (the average time required by the task from creation to completion).

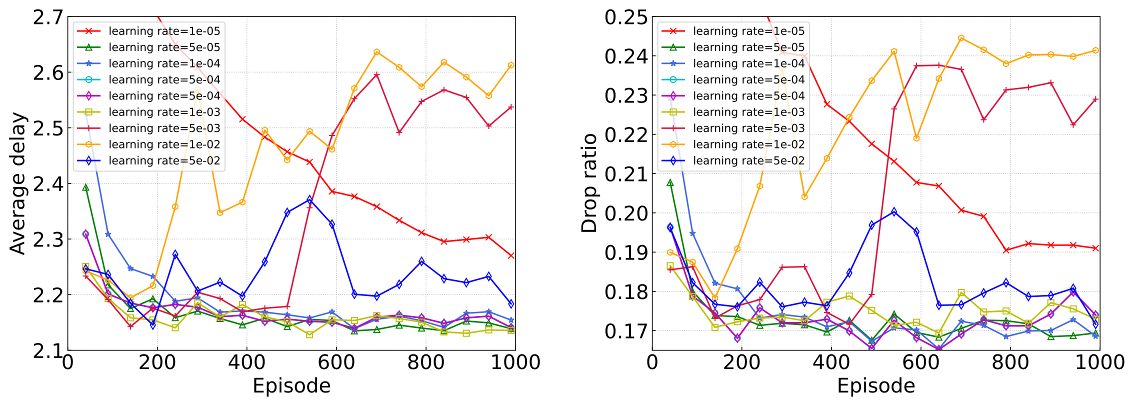


(a) average delay **(b)** ratio of dropped tasks

Figure 3. Performance evaluation under different batch sizes.

Figure 3 shows the convergence of the proposed algorithm under different batch sizes which means the number of training examples utilized in one iteration. Generally, as the batch size increases, the gradient estimation during each training step becomes more accurate, thereby reducing the variance of parameter updates. However, larger batch sizes may make the model more prone to getting stuck in local minima and lose some ability to escape from them. On the other hand, smaller batch sizes can provide more randomness, helping the model to jump out of local minima. As we can see from Figure 3, changing the parameter of batch size has only about a 1% impact on performance. Thus we choose appropriate batch size(e.g., 64) to speed up the convergence speed without reducing the performance significantly.

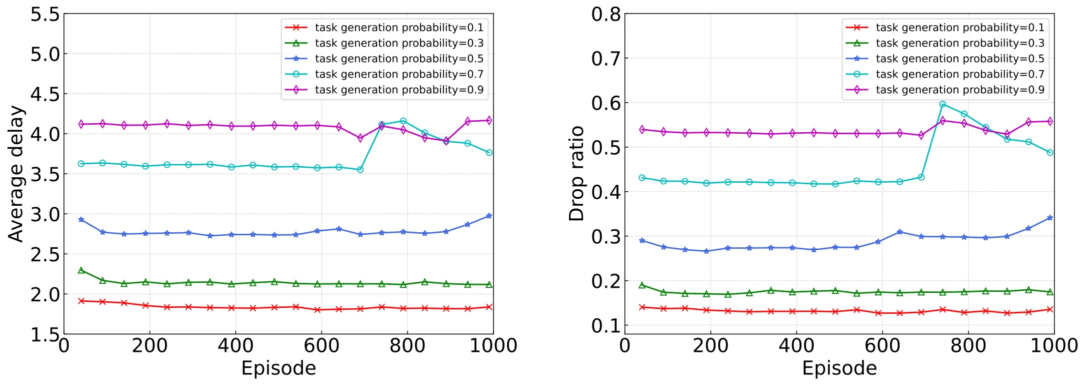
Figure 4 shows the convergence of the proposed algorithm under different learning rates which is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward the minimum of a loss function. In Figure 4, when learning rate is too small (e.g. 1e-5) ,it will lead to a relatively slow convergence and therefore require more computational resources. But if the learning rate is too large(e.g. 1e-2, 5e-2) , the neural network can't converge to a good performance because of the great vibration of loss function.



(a) average delay **(b)** ratio of dropped tasks

Figure 4. Performance evaluation under different batch sizes.

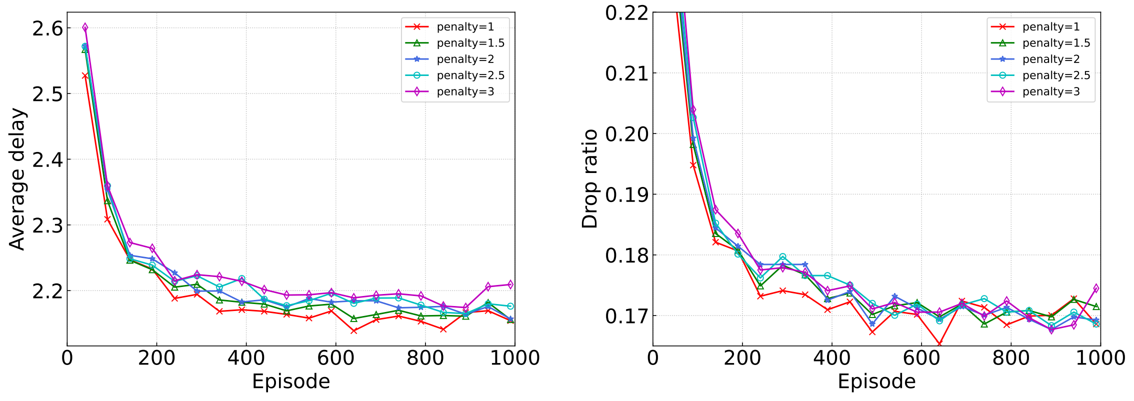
Figure 5 shows the convergence of the proposed algorithm under different task generation probabilities, where the task generation probability is the probability of each device producing a task in each time frames. As shown in Figure 5, there is an obvious correlation between task generation probability and algorithm performance: the larger the task generation probability, the larger the average delay. This is because with limited computing and communication resources, the higher the probability of task generation, the less resources are allocated on average, and the higher the delay of task completion.



(a) average delay **(b)** ratio of dropped tasks

Figure 5. Performance evaluation under different task generation probabilities.

Figure 6 illustrates the performance of proposed algorithm under different drop penalty values, where the penalty value are represented as:



(a) average delay **(b)** ratio of dropped tasks

Figure 6. Performance evaluation under different penalties.

$$C(S(t), \mathcal{A}(t)) = \begin{cases} Penalty\ Value, & \text{if task is dropped} \\ \frac{\mathbb{1}_{(a_m(t)=m)}\mathcal{L}_m^{local}(t)+\mathbb{1}_{(a_m(t)\neq m)}\mathcal{L}_m^{rem}(t)}{\tau_m^t}, & \text{if task is not dropped} \end{cases} \tag{17}$$

Generally, as the penalty value increases, the model should prone to dropout and thus result in a decrease in the drop ratio. Interestingly, the experimental results show the opposite behavior. This may be due to the change from a continuous range of penalty values with an upper limit of 1 to discrete values. This change may make it difficult for the Critic network to converge, resulting in the observed results.

6.3. Performance Comparison Evaluation

To valuate the performance of our algorithm, we consider the following benchmark algorithms:

1) IPPO: Each agent employs an independent PPO algorithm to select actions based on global state information;

2) A2C: A2C algorithm is used to sample from $(1 + N)^N$ types of actions;

3) TD3:The multidimensional discrete action space is transformed into multiple continuous actions, and the selection of an action is based on the shortest Euclidean distance [2];

4) Heuristic: If a task can be completed within two time slots, it is computed locally; otherwise, it is offloaded to the edge server;

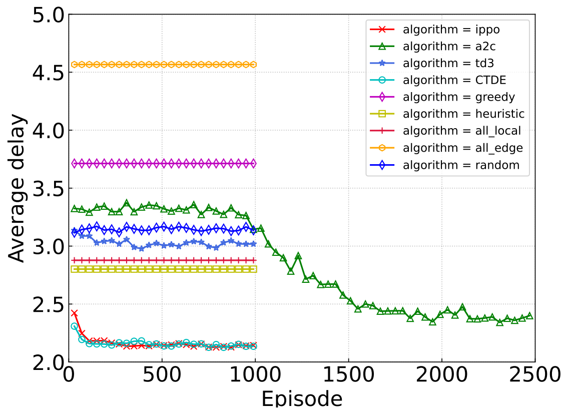
5) All_local: All tasks are executed locally;

6) All_edge: All tasks are offloaded to edge server;

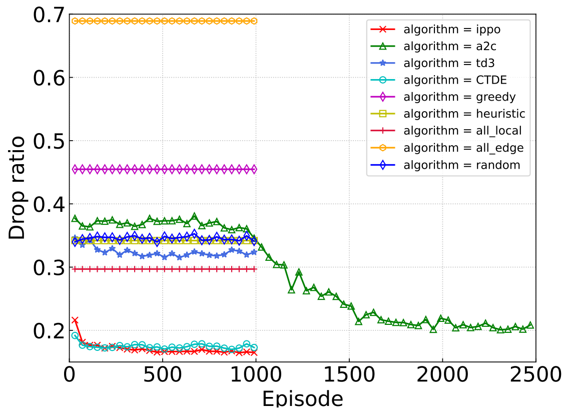
7) Greedy: The action with the shortest expected time, considering the current queue situation and channel status, is selected;

8) Random: Actions for each task are randomly selected. Here we use two performance metrics: the average delay and the drop ratio(ratio of the number of timeout tasks to the total number of tasks).

Figure 7 illustrates the convergence of our proposed algorithm and baseline algorithms. It can be observed that our algorithm presents similar performance to the IPPO algorithm in terms of the delay metric. However, our algorithm only relies on local state information for decision-making, significantly reducing communication overhead. In comparison to the sub-optimal A2C algorithm, our approach achieves an 11.0% reduction for the delay metric and a 17.0% reduction for the drop ratio metric.



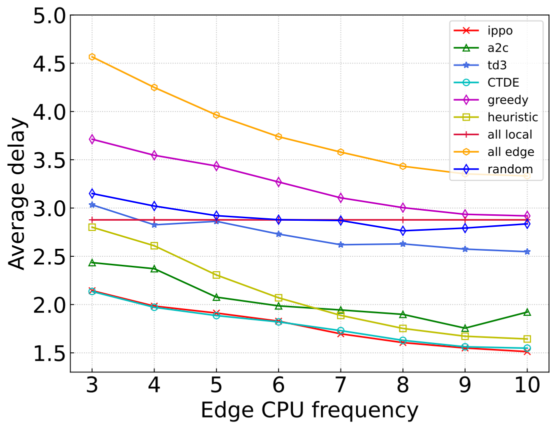
(a) average delay



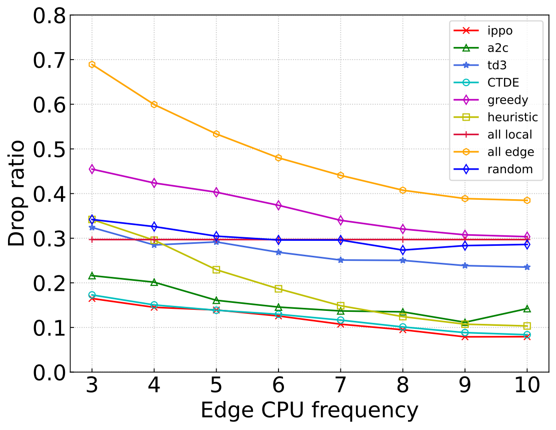
(b) ratio of dropped tasks

Figure 7. Performance of different algorithms.

Figure 8 illustrates the comparison of the average delay and drop ratio between our proposed algorithm and the baseline algorithms under different edge server frequencies. As the computing power of the edge server increases, the overall system computing resources increase, resulting in decreased average delay and drop ratio. It is important to note that the all_local algorithm, which does not rely on edge servers, maintains a stable performance curve.



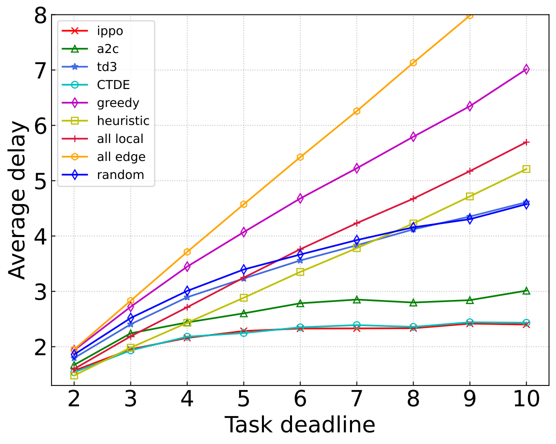
(a) average delay



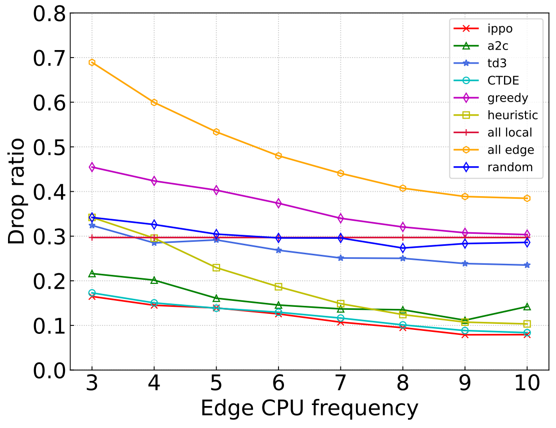
(b) ratio of dropped tasks

Figure 8. Performance of different algorithms across different edge cpu frequency.

Figure 9 shows the comparison of the average delay and drop ratio between our proposed algorithm and the baseline algorithms under different task deadlines. From Fig. 9, it is evident that several benchmark algorithms, such as all_local, all_edge, or greedy, face a limitation imposed by the deadline, causing the average delay to approach the upper limit, i.e., the deadline itself. Consequently, as the deadline increases, the average delay also grows, while the drop ratio decreases due to the slack in the deadline. In contrast, the CTDE, IPPO, and A2C algorithms exhibit superior performance. When the deadline is less than 5, their curves resemble those of other algorithms, although relatively flat. However, as the deadline continues to increase, these algorithms reach a stable curve, suggesting that the tasks have been fully processed.



(a) average delay



(b) ratio of dropped tasks

Figure 9. Performance of different algorithms across different task deadline.

7. Conclusion

In this paper, we investigate the dynamic task offloading problem in D2D-MEC systems for delay-sensitive tasks under delay constraints. We propose a dynamic partitioning approach for D2D MDs set based on queuing-based system. We formulate the minimization long-term average task delay with deadline constraints problem as a cooperative Markov game and propose a multi-agent DRL-based algorithm. Our proposed algorithm is implemented in a CTDE manner, which enables each MD to make offloading decisions solely based on its local system state. Extensive simulations show the efficiency of our algorithm. For the future work, we will consider bandwidth resource allocation strategy in the D2D-MEC network.

Author Contributions: Conceptualization, H.H. and X.M. ; methodology, X.Y. and H.H.; software, X.M.; validation, H.H. and X.L.; investigation, X.M.; resources, H.H.; writing—original draft preparation, X.M.; writing—review, H.H., X.Y., X. L. and H.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported in part by the Science and Technology Foundation of Guangdong Province, China, No. 2021A0101180005. The corresponding author is Xuefeng Liao.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Available on request.

Acknowledgments: The authors would like to thank the editor and all reviewers for their valuable comments and efforts on this article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Jie Peng, Hongbing Qiu, Jun Cai, Wenjun Xu, and Junyi Wang. D2d-assisted multi-user cooperative partial offloading, transmission scheduling and computation allocating for mec. *IEEE Transactions on Wireless Communications*, 20(8):4858–4873, 2021.
2. Tao Zhang, Kun Zhu, and Junhua Wang. Energy-efficient mode selection and resource allocation for d2d-enabled heterogeneous networks: A deep reinforcement learning approach. *IEEE Transactions on Wireless Communications*, 20(2):1175–1187, 2020.
3. Tao Fang, Feng Yuan, Liang Ao, and Jiaxin Chen. Joint task offloading, d2d pairing, and resource allocation in device-enhanced mec: A potential game approach. *IEEE Internet of Things Journal*, 9(5):3226–3237, 2021.
4. Yiping Zuo, Shi Jin, Shengli Zhang, Yu Han, and Kai-Kit Wong. Delay-limited computation offloading for mec-assisted mobile blockchain networks. *IEEE Transactions on Communications*, 69(12):8569–8584, 2021.
5. Nadine Abbas, Sanaa Sharafeddine, Azzam Mourad, Chadi Abou-Rjeily, and Wissam Fawaz. Joint computing, communication and cost-aware task offloading in d2d-enabled het-mec. *Computer Networks*, 209:108900, 2022.

6. Monia Hamdi, Aws Ben Hamed, Di Yuan, and Mourad Zaied. Energy-efficient joint task assignment and power control in energy-harvesting d2d offloading communications. *IEEE Internet of Things Journal*, 9(8): 6018–6031, 2021.
7. Liang Huang, Suzhi Bi, and Ying-Jun Angela 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.
8. Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Medhi Bennis. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet of Things Journal*, 6(3):4005–4018, 2018.
9. Jia Luo, F Richard Yu, Qianbin Chen, and Lun Tang. Adaptive video streaming with edge caching and video transcoding over software-defined mobile networks: A deep reinforcement learning approach. *IEEE Transactions on Wireless Communications*, 19(3):1577–1592, 2019.
10. Nan Zhao, Ying-Chang Liang, Dusit Niyato, Yiyang Pei, Minghu Wu, and Yunhao Jiang. Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks. *IEEE Transactions on Wireless Communications*, 18(11):5141–5152, 2019.
11. Gaibin Li, Mingkai Chen, Xin Wei, Ting Qi, and Wenqin Zhuang. Computation offloading with reinforcement learning in d2d-mec network. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 69–74. IEEE, 2020.
12. Guanhua Qiao, Supeng Leng, and Yan Zhang. Online learning and optimization for computation offloading in d2d edge computing and networks. *Mobile networks and applications*, pages 1–12, 2019.
13. Bo Yang, Xuelin Cao, Joshua Bassey, Xiangfang Li, and Lijun Qian. Computation offloading in multi-access edge computing: A multi-task learning approach. *IEEE transactions on mobile computing*, 20(9):2745–2762, 2020.
14. Rong Chai, Junliang Lin, Minglong Chen, and Qianbin Chen. Task execution cost minimization-based joint computation offloading and resource allocation for cellular d2d mec systems. *IEEE Systems Journal*, 13(4): 4110–4121, 2019.
15. Yinghui He, Jinke Ren, Guanding Yu, and Yunlong Cai. D2d communications meet mobile edge computing for enhanced computation capacity in cellular networks. *IEEE Transactions on Wireless Communications*, 18(3): 1750–1763, 2019.
16. Yinghui He, Jinke Ren, Guanding Yu, and Yunlong Cai. Joint computation offloading and resource allocation in d2d enabled mec networks. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.
17. Ming Tang and Vincent WS Wong. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing*, 2020.
18. Suzhi Bi, Liang Huang, Hui Wang, and Ying-Jun Angela Zhang. Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks. *IEEE Transactions on Wireless Communications*, 20(11):7519–7537, 2021. doi: 10.1109/TWC.2021.3085319.
19. Jin Wang, Jia Hu, Geyong Min, Albert Y Zomaya, and Nektarios Georgalas. Fast adaptive task offloading in edge computing based on meta reinforcement learning. *IEEE Transactions on Parallel and Distributed Systems*, 32(1):242–253, 2020.
20. Xiaoyan Huang, Supeng Leng, Sabita Maharjan, and Yan Zhang. Multi-agent deep reinforcement learning for computation offloading and interference coordination in small cell networks. *IEEE Transactions on Vehicular Technology*, 70(9):9282–9293, 2021.
21. Alessio Sacco, Flavio Esposito, Guido Marchetto, and Paolo Montuschi. Sustainable task offloading in uav networks via multi-agent reinforcement learning. *IEEE Transactions on Vehicular Technology*, 70(5):5003–5015, 2021.
22. Honghao Gao, Xuejie Wang, Xiaojin Ma, Wei Wei, and Shahid Mumtaz. Com-ddpg: A multiagent reinforcement learning-based offloading strategy for mobile edge computing. *arXiv preprint arXiv:2012.05105*, 2020.

23. Haipeng Wang, Zhipeng Lin, and Tiejun Lv. Energy and delay minimization of partial computing offloading for d2d-assisted mec systems. In *2021 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2021.
24. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.