sensors

MDPI

Article

A Multi-Agent RL Algorithm for Dynamic Task Offloading in D2D-MEC Network with Energy Harvesting †

Xin Mi [1,‡], Huaiwen He [1,*,‡] and Hong Shen [2]

[1] School of Computer, Zhongshan Institute, University of Electronic Science and Technology of China, Zhognshan 528400, China; 202021080226@std.uestc.edu.cn
[2] Engineering and Technology, Central Queensland University, Brisbane 4000, Australia; hongsh01@gmail.com
* Correspondence: he_huai_wen@aliyun.com
† This paper is an extended version of our paper published in Mi, X.; He, H. Multi-Agent Deep Reinforcement Learning for D2D-assisted MEC system with Energy Harvesting. In Proceedings of the 2023 25th International Conference on Advanced Communication Technology (ICACT), Pyeongchang, Republic of Korea, 19–22 February 2023. https://ieeexplore.ieee.org/document/10079275.
‡ Xin Mi and Huaiwen He are co-first authors and contributed equally to this article.

Abstract: Delay-sensitive task offloading in a device-to-device assisted mobile edge computing (D2D-MEC) system with energy harvesting devices is a critical challenge due to the dynamic load level at edge nodes and the variability in harvested energy. In this paper, we propose a joint dynamic task offloading and CPU frequency control scheme for delay-sensitive tasks in a D2D-MEC system, taking into account the intricacies of multi-slot tasks, characterized by diverse processing speeds and data transmission rates. Our methodology involves meticulous modeling of task arrival and service processes using queuing systems, coupled with the strategic utilization of D2D communication to alleviate edge server load and prevent network congestion effectively. Central to our solution is the formulation of average task delay optimization as a challenging nonlinear integer programming problem, requiring intelligent decision making regarding task offloading for each generated task at active mobile devices and CPU frequency adjustments at discrete time slots. To navigate the intricate landscape of the extensive discrete action space, we design an efficient multi-agent DRL learning algorithm named MAOC, which is based on MAPPO, to minimize the average task delay by dynamically determining task-offloading decisions and CPU frequencies. MAOC operates within a centralized training with decentralized execution (CTDE) framework, empowering individual mobile devices to make decisions autonomously based on their unique system states. Experimental results demonstrate its swift convergence and operational efficiency, and it outperforms other baseline algorithms.

Keywords: MEC; D2D communication; multi-agent reinforcement learning; energy harvesting; dynamic task offloading

Citation: Mi, X.; He, H.; Shen, H. A Multi-Agent RL Algorithm for Dynamic Task Offloading in D2D-MEC Network with Energy Harvesting. *Sensors* **2024**, *24*, 2779. https://doi.org/10.3390/s24092779

Academic Editor: Nick Harris

Received: 19 March 2024
Revised: 18 April 2024
Accepted: 24 April 2024
Published: 26 April 2024

1. Introduction

With the rapid development of mobile software services, more and more application services are becoming computation-intensive and delay-sensitive, such as virtual reality, augmented reality, and online games. Therefore, the mobile edge computing (MEC) scheme, which allows mobile devices to offload tasks to an edge server close to the end user and significantly reduces the service response delay, is considered to be a promising paradigm and has attracted the attention of researchers [1–3].

However, in a large-scale MEC system with numerous mobile devices, the dynamic and sometimes bursty nature of edge device loads can overwhelm the server, potentially hindering the processing of offloaded computation tasks within the required delay constraints. D2D technology emerges as a pivotal solution within the realm of 5G networks [4–7]. By leveraging idle end devices' computing resources through D2D communication links,

mobile devices can offload tasks to these available resources [6]. Therefore, effectively harnessing these idle computing resources via D2D links to minimize task delays and collaborate seamlessly with edge servers becomes paramount for optimizing the performance efficiency of MEC systems.

Numerous studies have focused on reducing computation task delays in MEC systems [8–12]. However, most of these works assume that tasks are divisible and can be completed within a single time slot. Tang and Wong [13] introduced a long-term cost minimization algorithm for an MEC system that accounts for non-divisible, delay-sensitive tasks and dynamic edge loads spanning multiple time slots. Nevertheless, their approach did not consider integrating D2D communication or consider the energy harvesting constraints of mobile devices. Reinforcement learning (RL) has emerged as a promising approach to tackle the computational complexities associated with MEC systems [14]. It has been leveraged to develop algorithms for task offloading in dynamic MEC environments [3,10,15]. Li et al. [16] utilized the deep Q-network (DQN) to jointly manage computation offloading and resource allocation in multiuser MEC setups to minimize the total cost of delay and energy consumption. Chen et al. [10] combined residual blocks, long short-term memory, and rank-based prioritized experience replay (rPER) within the deep deterministic policy gradient (DDPG) algorithm to address the joint optimization of computation offloading and resource allocation. In our prior study [5], we introduced an independent PPO-based task-offloading algorithm for D2D-MEC systems with energy harvesting, focusing on task-specific latency constraints. Each agent operates a separate PPO model independently, training on its own state information, lacking information sharing and collaboration, resulting in slow convergence. However, in a large-scale MEC system, RL algorithms relying on a single agent may encounter challenges such as action-space explosion, leading to slow convergence and suboptimal performance [17]. Distributed multi-agent reinforcement learning techniques, which focus on local information and offer easier deployment, align well with the architectural requirements of MEC systems.

In this paper, we investigate the computation-task-offloading problem in a D2D-MEC network under harvested energy constraints. The problem aims to minimize the average task service delay by determining the CPU frequency for each active mobile devices and making task-offloading decisions for each task during each time slot. We utilize a queuing system to represent non-divisible tasks and account for computation tasks that extend over multiple time slots. To address the stochastic nature of task generation, dynamic channel states, and unpredictable harvested energy, we frame the problem of minimizing the average task service delay as a sequential decision challenge, involving a vast multi-dimensional discrete action space. To tackle this complexity, we introduce a multi-agent RL algorithm based on the MAPPO technique, which enables each agent to autonomously make decisions, thereby reducing the decision space. Our proposed algorithm adopts a centralized training approach coupled with decentralized execution, enhancing its practical applicability in real-world scenarios. Experimental results show that our proposed algorithm can work efficiently in a distributed manner without requiring system information prediction.

Our key contributions are summarized as follows:

- We propose a novel scheme that leverages dynamic voltage and frequency scaling (DVFS) and energy harvesting (EH) to enhance energy efficiency and minimize task delays in MEC networks operating under energy constraints. This scheme accounts for stochastic environmental factors such as dynamic harvested energy, fluctuating communication channel conditions, and random task generation. By employing a queuing system to model computation tasks spanning multiple time slots with varying processing speeds and data transmission rates, we accurately assess the edge load of each mobile device in a D2D-MEC network.
- To address the curse of dimensionality inherent in sequential Markov decision processes for solving the nonlinear integer programming (NLP) problem of task offload-

ing, we propose the MAOC algorithm by leveraging the multi-agent proximal policy optimization (MAPPO) technique. Our algorithm adopts a CTDE framework, enabling each mobile device to autonomously make decisions based on its system state. This approach is practical for real-world deployment.

- Through comprehensive simulations, we validate the efficacy of our proposed algorithm. The numerical results demonstrate the algorithm's rapid convergence and superior performance compared to baseline algorithms.

The remainder of this paper is organized as follows: Section 2 introduces the related work. Section 3 presents our system model. Section 4 provides the mathematical formulation of the problem. Section 5 gives the details of the algorithm design and implementation for the optimization problem. In Section 6, simulations are conducted to verify the performance of our proposed algorithm. The conclusions are finally drawn in Section 7.

2. Related Works

MEC stands out as a promising paradigm for 5G heterogeneous networks, attracting significant attention [10,18,19]. Machine learning algorithms play a pivotal role in enabling intelligent decision making for task offloading in MEC systems, adapting to the stochastic and dynamically changing environment. Cao et al. [4] reviewed intelligent offloading in multi-access edge computing, highlighting various research endeavors employing ML-based methodologies. Chen et al. [10] proposed a Temporal Attentional Deterministic Policy Gradient (TADPG) algorithm to minimize the average long-term cost of computational tasks in MEC systems. However, prior works assumed that tasks could be completed within a single time slot without impacting subsequent tasks, neglecting scenarios where tasks span multiple time slots. In contrast, Tang and Wong [13] considered the dynamic load at edge device, the non-divisible and delay-sensitive computation tasks, they proposed a learning algorithm integrating LSTM, dueling DQN, and double DQN to minimize task delay in MEC networks. But none of them accounted for D2D communication intricacies and energy constraints.

D2D technology, a pivotal 5G communication innovation, empowers users to offload tasks to neighboring idle devices, thereby enhancing the overall computational efficiency of the system. Wang et al. [9] delved into the task-offloading conundrum, aiming to minimize the weighted sum of delay and energy consumption across multiple independent subtasks within a D2D-assisted MEC framework. Chai et al. [20] formulated the task execution cost minimization challenge as a mixed-integer nonlinear problem, proposing a heuristic algorithm grounded in the Kuhn–Munkres algorithm and Lagrangian dual method.

DVFS and energy harvesting are two important technologies for saving energy and prolonging the lifetimes of batteries in end devices, and they have caught the attention of researchers in recent years. Liang et al. [21] proposed a joint re-ordering and frequency scaling (JRFS) algorithm to minimize makespan in an MEC network, considering precedence constraints among tasks and using DVFS technology to scale the frequencies of edge servers. Xia et al. [22] investigated an EH-enabled MEC offloading system and proposed an online distributed optimization algorithm based on game theory and perturbed Lyapunov optimization theory. Guo et al. [18] formulated an EH computation-offloading game to minimize delay in an MDC system with energy harvesting and developed a distributed EHCOG scheme to solve it. However, the unpredictable amount of harvested energy in D2D-MEC networks also poses challenges for making collaborative task execution decisions.

To tackle the stochastic and dynamic environment of MEC systems, deep reinforcement learning approaches are used to design algorithms for task offloading in MEC networks. Chen et al. [23] investigated the long-term utility performance maximization problem for an MEC with an ultra-dense sliced radio access network, and proposed two double-DQN-based online strategic computation-offloading algorithms. Huang et al. [24] proposed a DQN-based algorithm for joint task offloading and bandwidth allocation in a multi-user MEC system. Huang et al. [25] proposed a DQN-learning-based online offloading algo-

rithm, DROO, for binary computation of offloading in wireless-powered MEC networks. Huang et al. [15] proposed a task-offloading scheme, RRLO, based on DVFS energy consumption reduction, by using the double Q-learning algorithm. But all the above studies are based on single-agent learning algorithms, which may incur difficulty in convergence and bad performance [17] when used on large numbers of multiple devices due to the curse of dimensionality. [26] proposed a novel DRL algorithm based on the latent space to optimize the trajectory of multiple unmanned aerial vehicles (UAVs), considering the task priorities and binary offloading mode in a UAV-enabled MEC network.

3. System Model

In this paper, we consider a D2D-MEC system comprising an edge server and a collection of MDs, as illustrated in Figure 1. We assume that the system operates in discrete time slots, with the entire period divided into T time slots, represented as $\mathcal{T} = \{1, 2, \cdots, T\}$. Each time slot has a duration of Δt, which is sufficiently small to ensure that no more than one task is generated by each device within a slot.

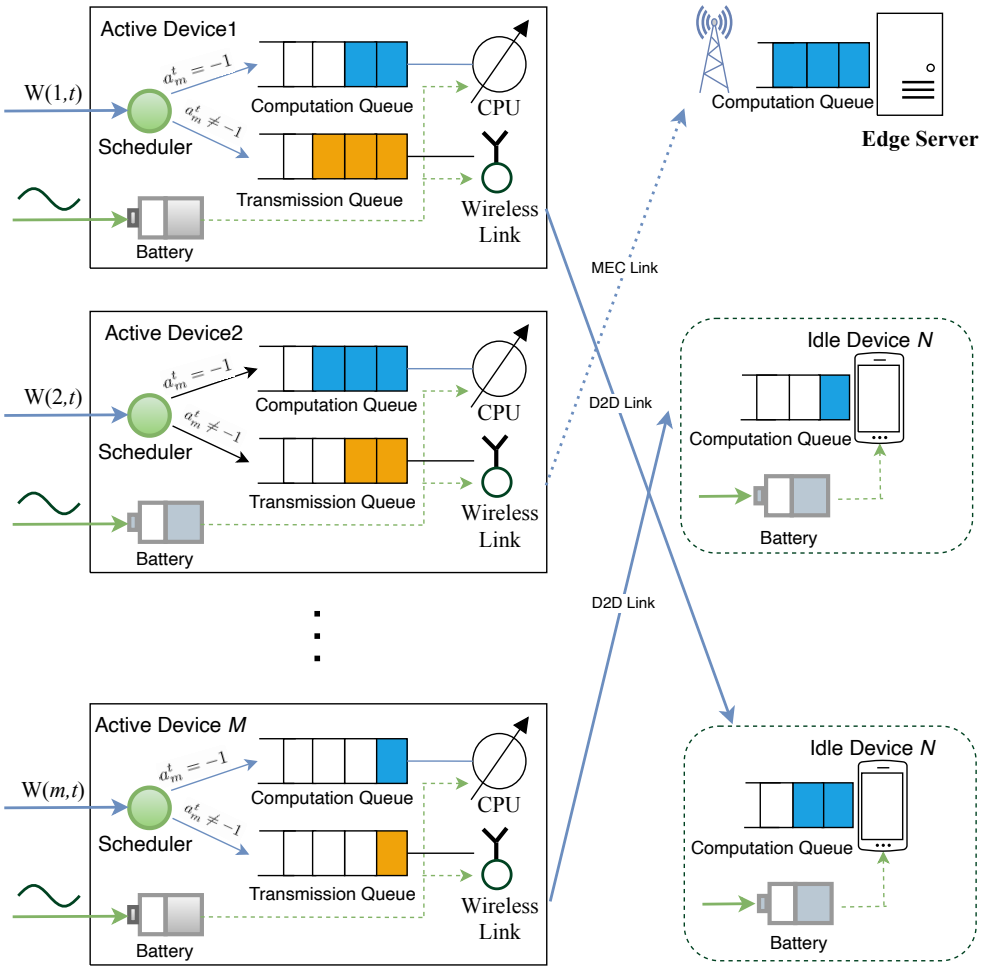

Figure 1. Architecture of D2D-MEC network.

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.

Table 1. Notation definitions.

Symbol	Definition
$\mathcal{D}$	The set of mobile deivces
$\mathcal{M}$	The set of active devices
$\mathcal{N}$	The set of idle devices
$\boldsymbol{A(t)}$	The joint offloading decision at time slot t
$\boldsymbol{F(t)}$	The joint CPU frequency selection at time slot t
$\mathcal{F}$	Optional CPU frequency collection
$a_m^t(t)$	The offloading decision of the mth active device at time slot t
f_m(t)	The CPU frequency selection of the mth active device at time slot t
w_m^t	The task generated by active device m at time slot t
s_m^t	The size of task w_m^t
c_m^t	The CPU cycles required to execute a bit of task w_m^t
Q_d^{comp}	The computing queue of mobile device d
Q_m^{tran}	The transmission queue of active device m
$l_{m,d}^{comp}(t)$	The time slot when task $w(m,t)$ is fully processed at device d
$l_m^{tran}(t)$	The time slot when task $w(m,t)$ transmission is completed at 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
$RB_d(t)$	Battery energy of device d in time slot t
$e_d^c(t)$	Energy consumed by device d in time slot t
$e_d^h(t)$	Energy harvested by device d in time slot t
$S_m(t)$	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

3.1. Device Model

In the D2D-MEC system, the device set $\mathcal{D}$ is divided into two subsets: the active device subset $\mathcal{M} = \{M_1, M_2, \ldots, M_{|\mathcal{M}|}\}$, and the idle device subset $\mathcal{N} = \{N_0, N_1, N_2, \ldots, N_{|\mathcal{N}|}\}$, as depicted in Figure 1. Here, N_0 is the edge server and $\mathcal{D} = \mathcal{M} \cup \mathcal{N}$. We assume that idle devices can handle offloading tasks independently without forwarding them to edge servers, thereby alleviating the load on the latter [2,27]. Each MD $d \in \mathcal{D}$ employs battery RB_d and integrates with the energy-harvesting module to enhance performance. The edge server, on the other hand, is not subject to energy constraints as it is directly connected to base stations via wired connections [27–29].

In each time slot t, a computation task is generated exclusively on each active device. The task has a specific size and computational complexity, a task arriving at m in slot t is denoted as $w_m^t = (s_m^t, c_m^t)$, where s_m^t represents the task size in 1-bit units, and c_m^t represents the computational complexity measured in CPU cycles required for a 1-bit operation. It should be noted that we assume that task w_m^t is indivisible and may span across multiple consecutive time slots.

To accurately assess the edge load of each MD, we employ a queuing system to represent the task execution process. Computation tasks are enqueued and scheduled for processing in a first-come, first-served (FCFS) manner. As illustrated in Figure 1, the D2D-MEC system consists of two types of queues: the computation queue Q_d^{comp}, used for task execution, where $d \in \mathcal{D}$; and the transmission queue Q_m^{tran}, used for task transmission only on the active mobile devices, where $m \in \mathcal{M}$.

When a task is generated on an active MD m, the scheduler determines the location for its execution. Generally, there are three choices: local execution, offloading to an edge server via cellular links, or offloading to idle devices via D2D links. We use

$a_m^t \in [-1, 0, 1, \ldots, |\mathcal{N}|]$ to denote the decision on how to execute task w_m^t. If $a_m^t = -1$, the computation task will enter the local computation queue Q_m^{comp}; if $a_m^t = 0$, task w_m^t will be offloaded to the edge server; otherwise it is offloaded to the a_m^tth device in the idle devices set $\mathcal{N}$.

3.2. Computation Model

Active devices, $m \in \mathcal{M}$, leverage DVFS technology to regulate their CPU frequency, $f_m \in \mathcal{F}$, aiming to conserve power given their limited battery capacity. Nevertheless, it is crucial to acknowledge that lowering the CPU frequency can potentially lead to increased task delays, potentially breaching the user's service-level agreement (SLA). Here, we embrace a discrete CPU frequency model, where $\mathcal{F} = \{f^1, f^2, \cdots, f^K\}$ signifies the available CPU frequency options for each active device. Notably, we assume that the CPUs of idle MDs, indexed by n, where $n \in [1, |\mathcal{N}|]$, maintain a fixed frequency of f_n^{idle}, while the edge server operates at a substantially higher CPU frequency, as f_0^{idle}.

Let $\widetilde{t}$ represent the time slot when task w_m^t is placed in the computation queue. In the case of local computing, we have $\widetilde{t} = t$. Once task w_m^t enters the computation queue of MD d, $d \in \mathcal{D}$, it will be scheduled for processing at the subsequent slot following the completion of preceding tasks [13]. We define $l_{m,d}^{comp}(t)$ as the time slot when task w_m^t is completely processed on the designated device d. The duration that task w_m^t spends in the computation queue can be expressed as the difference between the time of complete processing and the time of entry into the queue:

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

where the operator $[x]^+ = \max\{0, x\}$, and $l_{d',d}^{comp}(0)$ is set as 0 for simplicity of presentation. The term $\max_{t' \in \{0,1,\cdots,\widetilde{t}-1\}, d' \in [1,|\mathcal{M}|]} l_{d',d}^{comp}(t')$ represents the time slot when all the task placed in the computation queue before time slot $\widetilde{t}$ has been processed. Hence, $\phi_{m,d}^{comp}(t)$ determines the number of waiting time slots in the computation queue for task w_m^t.

Let $\widehat{t} = \widetilde{t} + \phi_{m,d}^{comp}(t)$ denote the time slot in which task w_m^t starts to be processed at MD d. Therefore, we can conclude that task w_m^t will have been processed completely at time slot $l_{m,d}^{comp}(t)$, as

$$l_{m,d}^{comp}(t) = \widetilde{t} + \phi_{m,d}^{comp}(t) + \underset{\theta}{\operatorname{argmin}} \left\{ \sum_{i=\widehat{t}}^{\widehat{t}+\theta} f_d(i)\Delta T \geq s_m^t c_m^t \right\} - 1 \tag{2}$$

where $\lceil \cdot \rceil$ is the ceiling function. $f_d(i)$ is the CPU frequency of MD d in time i. In particular, the term $\underset{\theta}{\operatorname{argmin}} \left\{ \sum_{i=\widehat{t}}^{\widehat{t}+\theta} f_d(i)\Delta T \geq s_m^t c_m^t \right\}$ calculates the minimum value of θ such that the total number of time slots required to fully process the task is satisfied.

Here, Equation (2) can be simplified under the following conditions:

(1) For local execution. We have $t = \widetilde{t}$, so the complete time slot of task w_m^t can be written as

$$l_{m,m}^{comp}(t) = t + \phi_{m,m}^{comp}(t) + \underset{\theta}{\operatorname{argmin}} \left\{ \sum_{i=t}^{i+\theta} f_d(i)\Delta T \geq s_m^t c_m^t \right\} - 1 \tag{3}$$

(2) For remote execution, considering that the CPU frequency is fixed for idle MDs throughout the entire time period, we have

$$l_{m,d}^{comp}(t) = \widetilde{t} + \phi_{m,d}^{comp}(t) + \lceil \frac{s_m^t c_m^t}{f_d \Delta T} \rceil - 1, \forall d \in \mathcal{N} \tag{4}$$

In scenarios where multiple offloaded task from different sources arrive at the target device simultaneously, they will be enqueued in the computation queue based on the following rule: among the offloaded tasks, tasks with lower computing demand, such as $s_m^t c_m^t$, will be prioritized and placed ahead in the queue.

3.3. Transmission Model

Each active MD $m \in \mathcal{M}$ maintains a transmission queue Q_m^{tran} to handle task offloading to remote devices. When a task w_m^t is selected for remote execution, it will be enqueued in the transmission queue and will start to transfer at the next time slot when the previous task has been successfully transferred.

Considering a rapidly varying communication environment, the data transmission rate of mobile device m can be determined based on Shannon's theorem as follows:

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

where B_m represents the bandwidth allocated to MD m and p_m denotes transmission power of MD m, both of which are constants, as in [22,30]. Here, we assume 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 [20]. In practical settings, the bandwidth available on edge servers significantly exceeds that allocated for D2D communication with mobile devices. While we have not rigorously distinguished between the bandwidth assigned for cellular network communication and D2D communication for simplicity, it is important to note this distinction. However, our algorithm is easily adaptable to incorporate and optimize for this differentiation. $h_m^d(t)$ represents the channel gain between MD m and target device d at time t, which remains fixed within a time slot and follows a Rayleigh distribution that varies over time. Additionally, N_0 represents the white noise of the channel. Therefore, similar to the computation model, we can obtain the number of waiting slots for task w_m^t in the transmission queue as follows:

$$\phi_m^{tran}(t) = \left[\max_{t' \in \{0,1,\cdots,t-1\}} l_m^{tran}(t') - t + 1 \right]^+ \tag{6}$$

where $\phi_m^{tran}(0)$ is set to be 0 for presentation simplicity. The term $\max_{t' \in \{0,1,\cdots,t-1\}} l_m^{tran}(t')$ represents the time slot when all tasks in the transmission queue before t have been scheduled.

Let $\check{t} = t + \phi_m^{tran}(t)$ represent the time slot for starting to transmit task w_m^t. Therefore, for task w_m^t generated in MD $m \in \mathcal{M}$, it will be completely transmitted by time slot $l_m^{tran}(t)$, which can be expressed as

$$l_m^{tran}(t) = t + \phi_m^{tran}(t) + \underset{\theta}{\operatorname{argmin}} \left\{ \sum_{i=\check{t}}^{\check{t}+\theta} r_m(i)\Delta T \geq s_m^t \right\} - 1 \tag{7}$$

where $r_m(i)\Delta t$ denotes the total transfer data size in time slot i, and the term $\underset{\theta}{\operatorname{argmin}} \left\{ \sum_{i=\check{t}}^{\check{t}+\theta} r_m(i)\Delta T \geq s_m^t \right\}$ represents the total time slots required to transfer the data of w_m^t successfully.

3.4. Energy Model

Each mobile device employs an EH module to obtain energy such as wind, solar, and ambient RF. The EH evolves based on i.i.d. and the process is modeled as a random process across the whole time period.

Let $e_d^h(t)$ denote the amount of harvested energy at the MD d at time t, which is intermittent and hard to predict. The harvested energy is stored in the battery to sup-

port the running of device. Here, we ignore the energy loss caused by battery charging and discharging for simplicity and focus on energy consumption of local computation and data transmission [31]. We denote the energy level of the battery in MD d at t as $RB_d(t) \in [RB_d^{min}, RB_d^{max}]$, where RB_d^{min} is the minimum energy required to support the basic functioning of the mobile device system, and RB_d^{max} is the maximum capacity of the battery. Therefore, the battery status evolves according to the following equation [28]:

$$RB_d(t+1) = \min\left\{ \max\left\{ RB_d(t) - e_d^c(t), RB_d^{min} \right\} + e_d^h(t), RB_d^{max} \right\} \tag{8}$$

where $e_d^c(t)$ denotes the energy consumption in time slot t, which is computed as

$$e_d^c(t) = \begin{cases} \mathbf{1}_{(Q_d^{comp} \neq \emptyset)} \delta f_d(t)^3 \Delta T + \mathbf{1}_{(Q_d^{tran} \neq \emptyset)} p_d \Delta T, \forall d \in \mathcal{M} \\ \mathbf{1}_{(Q_d^{comp} \neq \emptyset)} \delta f_d^3 \Delta T, \forall d \in \mathcal{N} \end{cases} \tag{9}$$

where $\mathbf{1}(\cdot)$ is an indicator function that outputs 1 when $\cdot$ is true and 0 otherwise, and $\delta > 0$ is a coefficient related to the CPU chip architecture. If the computation queue Q_d^{comp} is empty, there is no energy consumption for computation. The same is true for transmission queue Q_d^{tran}. Similar to [22], we assume that if there is not enough energy to support task computation or transmission, the MD system is switched to sleep and blocks the task in queues until there is enough energy in the battery.

3.5. Delay Model

For task w_m^t generated in MD m, if $a_m^t = -1$, which means task w_m^t will be processed locally, then based on Equation (3) we obtain that the delay in task w_m^t is $\left(l_{m,m}^{comp}(t) - t + 1\right)$, which can be rewritten as

$$\mathcal{L}_m^{loc}(t) = \phi_{m,m}^{comp}(t) + \underset{\theta}{\operatorname{argmin}} \left\{ \sum_{i=t}^{i+\theta} f_m(i) \Delta T \geq s_m^t c_m^t \right\} \tag{10}$$

If $a_m^t \neq -1$, the task will be processed remotely, according to Equation (7), we have the transmission delays $\mathcal{L}_m^{tran}(t) = l_m^{tran}(t) - t + 1$, that is, $\widetilde{t} = \mathcal{L}_m^{tran}(t) + t$. According to Equation (4), we have the processing delay $\mathcal{L}_{m,d}^{comp}(t) = \phi_{m,d}^{comp}(t) + \lceil \frac{s_m^t c_m^t}{f_d \Delta T} \rceil$. Hence, we have the total delay of remote execution task as

$$\mathcal{L}_m^{rem}(t) = \mathcal{L}_m^{tran}(t) + \mathcal{L}_m^{comp}(t) \tag{11}$$

Therefore, the total delay of task w_m^t can be derived as follows:

$$\mathcal{L}(m,t) = \mathbf{1}_{(a_m^t = -1)} \mathcal{L}_m^{local}(t) + \mathbf{1}_{(a_m^t \neq -1)} \mathcal{L}_m^{rem}(t) \tag{12}$$

4. Problem Formulation

In this paper, we aim to minimize the time-averaged task delay under energy constraints of mobile devices in a D2D-MEC system. At each time slot t, the MEC system makes task-offloading decisions $\boldsymbol{A(t)} = \{a_1(t), a_2(t), \ldots, a_{|\mathcal{M}|}(t)\}$ for each generated task and CPU frequency decisions $\boldsymbol{F(t)} = \{f_1(t), f_2(t), \ldots, f_{|\mathcal{M}|}(t)\}$ at each active device to optimize the long-term average task delay without knowing the future information. Decisions $a_m(t)$ and $f_m(t)$ are both discrete variable. Here, we adopt discrete CPU frequencies to represent the set of CPU speeds of the active devices, which are denoted as $\mathcal{F} = \{f^1, f^2, \cdots, f^K\}$, where $f_m(t) = k$ means the frequency of device m is set to f^k.

For simplicity, we use $\boldsymbol{A}$, $\boldsymbol{F}$ to represent $\boldsymbol{A(t)}$ and $\boldsymbol{F(t)}$ individually in the following sections. Thus, the time-averaged task delay minimization problem can be formulated as problem P1.

$$\mathbf{P1}: \quad \underset{\boldsymbol{A},\boldsymbol{F}}{\text{minimize}} \quad \frac{1}{T}\sum_{t=1}^{T}\frac{1}{M}\sum_{d=1}^{M}\mathcal{L}(m,t) \tag{13a}$$

s.t.

$$(2),(4),(5),(6),(7),(8) \tag{13b}$$

$$a_m(t) \in \{-1,0,1,2,\ldots,|\mathcal{N}|\}, \forall m \in \boldsymbol{M} \tag{13c}$$

$$f_m(t) \in \{1,2,\ldots,K\}, \forall m \in \boldsymbol{M} \tag{13d}$$

$$RB_d^{min} \leq RB_d(t) - e_d^c + e_d^h \leq RB_d^{max}, \forall d \in \mathcal{D} \tag{13e}$$

Equations (13c) and (13d) define the domain of decisions $a_m(t)$ and $f_m(t)$, respectively. Equation (13e) is battery energy constraint for all mobile devices.

Problem P1 is a nonlinear integer programming (NLP) problem, despite having prior knowledge of all the offline environment information (generated task information, harvested energy, and channel state). Generally, P1 is non-convex and NP-hard, which makes it extremely challenging to solve due to the following reasons: (1) The task generation, the amount of energy harvested, and the communication channel state are fast-varying across time slots, making it difficult to predict accurately, which poses significant challenges in designing an algorithm that can operate online; (2) the varying computing speed and transfer speed in constraints (1) and (4) create a tight coupling, requiring careful decision making for task-offloading and frequency decisions across time slots; (3) the combination decisions $\boldsymbol{A}, \boldsymbol{F}$ grow exponentially with increasing network size, specifically, the whole action space at slot t is $((N+2)K)^M$. Therefore, our objective is seek to design an algorithm based on a DRL technique that can learn from historical environmental data and make rapid decisions.

5. Solution with Multi-Agent DRL-Based Algorithm

Proximal policy optimization (PPO) [32] is a policy gradient algorithm based on the actor–critic (AC) framework [33] that has been shown to achieve state-of-the-art performance in a wide range of tasks, and has become a popular choice for many RL applications due to its simplicity and effectiveness. To improve the stability and sample efficiency of traditional policy gradient methods, PPO clips the update step of the policy network to prevent it from changing too much at once. Specifically, PPO uses a surrogate objective function to measure the difference between the updated policy and the previous policy. Thus, here we seek to derive an RL algorithm based on the PPO method and an AC framework.

However, single-agent DRL methods like PPO, which rely on a trial-and-error process to interact with the environment, cannot be directly applied to solve P1. This is because the action space $((N+2)K)^M$ of P1 grows exponentially, resulting in the curse of dimensionality. Additionally, using single-agent DRL means that the MEC system makes decisions centrally, which introduces a heavy communication burden. To address the challenges, we propose a multi-agent DRL-based dynamic offloading algorithm (MADOA) that builds upon MAPPO, a state-of-the-art multi-agent extension of the PPO method. Our algorithm employs the CTDE architecture, composed of a central controller in the edge server and an intelligence agent in each active MD. Each agent interacts independently with the environment and makes decision individually. The central controller collects the global system state to train the shared critic network, which it uses to evaluate the action decision of each agent. Firstly, we formulate P1 as a cooperative Markov game.

5.1. Markov Decision Process of P1

(1) State space: In each time slot t, agent $m \in \mathcal{M}$ observes its local system state. This state includes the generated task information (s_m^t, c_m^t) , the amount of harvested energy $e_m^h(t)$

and the current battery energy level $RB_m(t)$, the backlog of computation queue $Q_m^{comp}(t)$, the backlog of transmission queue $Q_m^{tran}(t)$, and the network channel state $h_m(t)$. Hence, the local state of agent m can be denoted as

$$\mathcal{S}_m(t) = (s_m^t, c_m^t, h_m(t), RB_m(t), e_m^h(t), Q_m^{comp}(t), Q_m^{tran}(t), \mathbf{ID(t)}) \tag{14}$$

where $h_m(t) = \{h_m^1(t), ..., h_m^{|\mathcal{N}|}(t)\}$, $Q_m^{comp}(t) = \max_{t' \in 0,1,\cdots,t-1} l_m^{comp}(t')$ represents the maximum waiting time in the computation queue Q_m^{comp} (e.g., the backlog of queue at time t), and $Q_m^{tran}(t) = \max_{t' \in 0,1,\cdots,t-1} l_m^{tran}(t')$ represents the backlog of the transmission queue at t. Specially, the matrix $\mathbf{ID(t)} = 3 \times |\mathcal{N}|$ represents the energy level, the harvested energy, and the backlog of the computation queue of idle devices set $\mathcal{N}$. The state information of the idle device n in $\mathbf{ID(t)}$ is denoted as $\{RB_n(t), e_n^h(t), Q_n^{comp}(t), n \in \mathcal{N}\}$, where $Q_n^{comp}(t) = \max_{t' \in \{0,1,\cdots,t-1\}, d' \in [1,|\mathcal{M}|]} l_{d',d}^{tran}(t')$ is the backlog of the computation queue in device n. Note that there is no limit to the energy level and energy harvested in the edge server, so we set them to be -1. We assume that the edge server will collect the state information of idle MDs and broadcast them to active MDs at the end of each time slot.

The global state space $\mathcal{S}$ is the Cartesian product of all local states of each active MD, denoted as $\prod_m \mathcal{S}_m$. Since the edge server collects all idle mobile device data at the end of each time slot, the global system state at t can be expressed as follows:

$$\mathcal{S}(t) = [\mathcal{S}_1(t), \mathcal{S}_2(t), \ldots, \mathcal{S}_{|\mathcal{M}|}(t)] = (\mathbf{AD(t)}, \mathbf{ID(t)}) \tag{15}$$

where $\mathbf{AD(t)} = 7 \times |\mathcal{M}|$ is the state information of the active MDs set, where row d is $(s_m^t, c_m^t, h_m(t), RB_m(t), e_m^h(t), Q_m^{comp}(t), Q_m^{tran}(t))$. Without loss of generality, if no task is generated at active device m in time t, the task size s_m^t and task complexity c_m^t are all set to be 0.

(2) Action space: The local action decisions of agent m at time t are denoted as $(a_m(t), f_m(t))$. These decisions include the task-offloading decision for the current generated task and the CPU frequency selection for the active MD. Therefore, the global action decisions at slot t can be represented as $(\boldsymbol{A}(t), \boldsymbol{F}(t))$.

(3) Reward: As a fully cooperative multi-agent learning model, each agent shares a common goal. We use the average task completion delay at time slot t as the reward for all agents, which is calculated as follows:

$$r(t) = \frac{1}{M} \sum_{d=1}^{M} \mathcal{L}(m, t) \tag{16}$$

Since we are considering joint actions among devices, this reward value is used to update the actor network of all devices and the global critic network.

5.2. MAOC Algorithm

To tackle the huge action space of P1, a multi-agent DRL technique is utilized to decompose the joint actions of all active devices into independent actions of each active MD. We formulate the problem P1 as a fully cooperative multi-agent RL model, where agents learn to interact with each other and their environment to maximize a shared reward signal.

Utilizing the MAPPO technique, we assign an independent actor network to each device for making action decisions based on local state information, as depicted in Figure 2. Additionally, a global critic network evaluates joint actions using global state information, guiding the actor network updates. Moreover, we implement our MARL algorithm using the CTDE framework, which incorporates a centralized critic for training and multiple decentralized actors that leverage shared experience. This setup allows each agent to make decisions based on its local observations. Our algorithm, named the Multi-

Agent Online Control (MAOC) algorithm, consists of two phases: centralized training and decentralized execution.

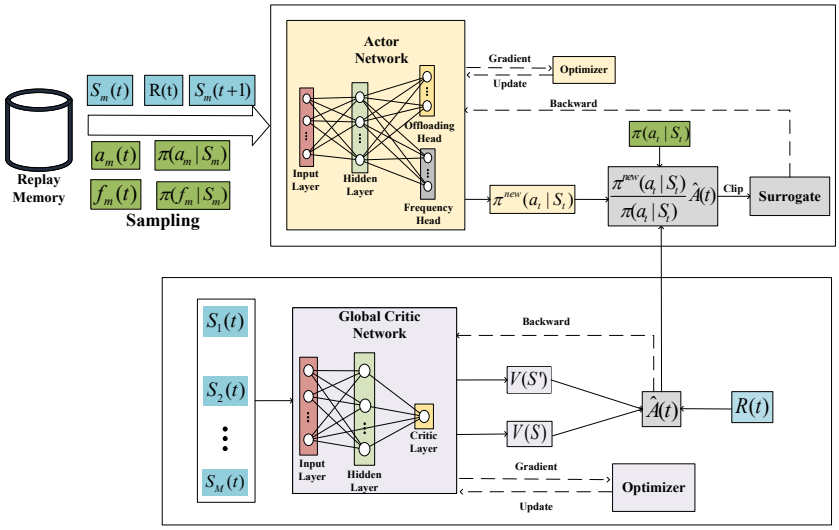

Figure 2. Architecture of the MAOC algorithm.

5.2.1. Centralized Training

The network update mechanism of our proposed algorithm resembles that of the PPO algorithm within the actor–critic framework. To enhance sample efficiency, we incorporate importance sampling to enable the reuse of samples. In order to prevent network instability or crashes caused by aggressive network updates, we employ a policy clip operation to constrain updates within a limited range, ensuring the stability and robustness of the network.

The training phase is illustrated in Figure 2, with the following specific detail: All devices randomly select the same batch of trajectories $(S_m(t), a_m(t), \pi(a_m|S_m), f_m(t), \pi(f_m|S_m), r(t), S_m(t+1))$ from the experience replay memory.

The communication process among devices during the training phase is as follows: (1) At the end of each time slot, idle MDs send their system state information to the edge server. Upon receiving the information from idle devices, the edge server integrates this data with its own state information and broadcasts it to all active MDs. (2) Each active MD, upon receiving this information, combines it with its own state data to form its system state. At slot t, each active MD interacts with the environment, receives current observations o_t, and makes stochastic decisions based on its policy. (3) In the next time step, each intelligent agent can observe the state o_{t+1} and the reward $r_m(t)$, and then, send o_t, o_{t+1}, and $r_m(t)$ to the central controller located at the edge server. (4) Utilizing the acquired data, the edge server makes predictions using the critic network, and calculates the temporal difference (TD) target and TD error. Subsequently, the TD error is broadcast to all active MDs' intelligent agents by the edge server, which also updates the parameters of the value network. (5) Upon receiving the TD error, each intelligent agent of the active MD updates its policy network.

For the global critic network, the advantage function is calculated using the reward $r(t)$, and then, the network parameters are updated as follows:

$$L_t(\theta^{critic}) = r(t) + \gamma V(S(t)) - V(S(t+1)) \tag{17}$$

For the actor network, we use the local state information of the device obtained from the replay memory $S_m(t)$ as input to calculate the current action policy output $\pi^{new}(a_t|S_t)$ by the network. We then use importance sampling to calculate the advantage value based on Equation (18) and perform clipping to ensure that the update magnitude is small enough. Finally, we backpropagate and update the actor network parameters.

$$L_t(\theta^{actor}) = \min(r_t(\theta^{actor})\hat{A}_t, clip(r_t(\theta^{actor}), 1-\epsilon, 1+\epsilon)\hat{A}_t) \tag{18}$$

where $r_t(\theta^{actor}) = \frac{\pi^{new}(a_t|s_t)}{\pi(a_t|s_t)}$ denotes the probability ratio, and ϵ is a hyperparameter that represents the trust region size.

In the loss function of the actor network, the first item refers to conservative policy iteration [34], which can result in an excessively large policy update. The second item modifies the surrogate objective by clipping the probability ratio, which removes the incentive to move r_t outside of the interval $[1-\epsilon, 1+\epsilon]$ [32]. By using the minimization operation, the final objective serves as a lower bound on the unclipped objective. As a result, we make a small and safe update, where we only slightly increase the probability of a good action.

Note that we use the generalized advantage estimation (GAE) trick to compute the advantage function, which is as follows:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{19}$$

where γ is the discount factor, which 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_{t+l}^V refers to the TD error, which is computed as $\delta_{t+l} = r(t+l) + \gamma V(t+l+1) - V(t+l)$, where V denotes the critic network operation.

Our proposed centralized training algorithm is summarized in Algorithm 1.

Algorithm 1: Multi-Agent Online Control algorithm (MAOC)

Input: Active Device Set $\mathcal{M}$,total episode number L, episode length T

for $m \in \mathcal{M}$ **do**
  Initialize the replay memory RM_m;
  Initialize the Actor network
end
Initialize the Global Critic Network;
for *n = 1 to L* **do**
  for *t = 1 to T* **do**
    for $m \in \mathcal{M}$ **do**
      Obtain the local state information $S_m(t)$ corresponding to the active device m;
      Compute the offloading decision a_m^t and CPU frequency decision f_m^t by actor network;
    end
    Interact with the environment based on decisions $A(t)$ and $F(t)$;
    for $m \in \mathcal{M}$ **do**
      Store transition into replay memory RM_m;
    end
  end
  for $m \in \mathcal{M}$ **do**
    Sample random mini-batch of transitions from the replay memory RM_m;
    Update the actor network according to Equation (18);
  end
  Randomly sample the same batch of transitions from the replay memory RM_m for each device m;
  Integrate local state information $S_m(t)$ of each device into global observation information $S(t)$;
  Update the global critic network according to Equation (17);
end

5.2.2. Decentralized Execution

At the end of each time slot t, idle devices transmit their own state information to the edge server. Then, the edge server broadcasts this information to active devices, paving the way for the next time slot. Thus, each active device can seamlessly integrate the broadcast information from the edge server and its own current state information into its local state. This local state is then fed into the actor network for inference to obtain the corresponding action policy $\pi(a_m|S_m)$ and $\pi(f_m|S_m)$. The action a_m and f_m are then obtained through sampling and used to interact with the environment.

5.2.3. Complexity Analysis of MAOC Algorithm

The complexity of the MAOC algorithm is a blend of the initialization and training phases.

During initialization, each active device establishes its replay memory and actor network, with complexity dependent on the number of active devices in set $|\mathcal{M}|$. This phase maintains a constant complexity level as it sets up these components.

In the training phase, iterative episodes unfold where devices engage with the environment, update their actor networks, and contribute to a global critic network. The algorithm complexity is analyzed from the following two aspects. (1) Time step operations: At each time step, tasks such as gathering local state information, decision making with the actor network, environment interaction, and transition storage in the replay memory occur. The complexity per time step is typically linear, influenced by factors like the number of active devices and state space size. (2) Network updates: Updating the actor networks and the global critic network involves sampling from the replay memory, calculating losses, and updating network parameters. The complexity of network updates is related to the size of the replay memory, network architecture, and the number of devices contributing to the global critic network.

6. Simulation Results

We construct the reinforcement learning model using the PyTorch framework and train it on a computing server with four GeForce RTX 2080 Ti GPUs and one Intel(R) Xeon(R) Silver 4116 CPU @ 2.10 GHz CPU with 48 cores. After the training, we evaluate the performance of the proposed algorithm through simulation experiments. The environment parameter settings are shown in Table 2 [13,35].

Table 2. The system parameter settings.

Parameter	Value
Active device number $\|\mathcal{M}\|$	10
Idle device number N	4
Idle CPU frequency f^{idle}	2×10^8 cycles/s
Edge server CPU frequency f_0	4×10^8 cycles/s
CPU power parameter δ	10^{-27} Watt$\cdot$ s^3/cycle^3
Active device optional CPU frequency f_d	$[2 \times 10^8, 2.5 \times 10^8, 3 \times 10^8]$ cycles/s
Total bandwidth B	3 MHz
White noise N_0	10^{-3} Watt
Task generation probability	0.5
Minimum task size s_{min}	2×10^5 bits
Maximum task size s_{max}	5×10^5 bits
Minimum task complexity c_{min}	500 cycles/bit
Maximum task complexity c_{min}	1000 cycles/bit
Device transmission power p_d	5 Watt
Maximum battery capacity RB^{max}	50 J

The neural network parameters are as follows: the batch size is set to 64, learning rate is set to 1×10^{-4}, the number of active devices is fixed at 3, the number of idle devices is fixed at 3, and the probability of task generation is fixed at 0.5. The actor network adopts

a three-layer multilayer perceptron, while the critic network adopts a four-layer multilayer perceptron. Both networks have a middle layer with 128 nodes, and the Adam optimizer is used to update the networks. We generate training data online using simulators, while test data are pre-generated by simulators and saved to eliminate the influence of simulators on the experimental results.

To evaluate our proposed algorithm, we compare it with the following baseline algorithms:

(1) No D2D mode: This scenario only considers offloading tasks to the edge server and executing tasks locally, without utilizing D2D technology.

(2) All local mode: In this mode, all tasks are executed only on the devices where they are generated, without any task offloading.

(3) PPO algorithm: This approach leverages a single-agent PPO algorithm in the edge server, which collects the state information of all MDs as the training network input. The actor network directly outputs joint actions and broadcasts them to the MDs.

(4) Centralized MAPPO: In this setup, all agents exchange their local state information during the execution phase. Each agent then uses the global state information as input for the actor network to make corresponding decision actions.

Figure 3 illustrates that our proposed MACO algorithm outperforms the other baseline algorithms. It can be observed that our proposed algorithm demonstrates convergence at around 150 episodes, indicating its effectiveness and feasibility. While our proposed algorithm's convergence speed may be marginally slower than that of the CM algorithm, its advantage lies in not requiring inter-agent communication during execution. This characteristic reduces data transmission overhead, enhancing its practicality for real-world scenarios. Compared to alternative algorithms, our proposed approach exhibits significant enhancements: an 81.2% improvement compared to all local mode, enhancement compared to PPO algorithms, and a 48.2% boost over the No D2D mode. This underscores the exceptional performance of our algorithm, emphasizing that more input information is not always better. In some cases, redundant information can have a detrimental effect on results [36].

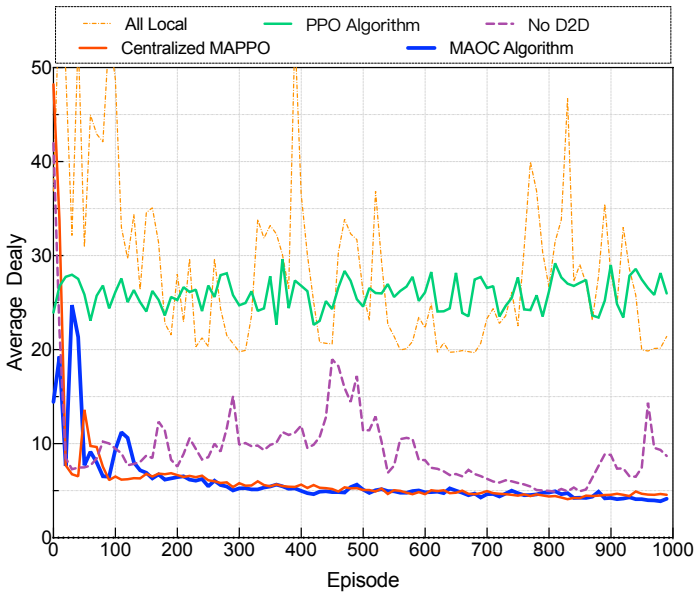

Figure 3. Performance compared with baseline algorithms.

Figure 4 illustrates the convergence of our proposed algorithm with varying learning rates. If the learning rate is excessively high, for instance, 5×10^{-3}, the neural network fails to converge, as the parameters oscillate beyond the acceptable range. Conversely, a learning rate that is excessively low, like 5×10^{-6}, results in the algorithm's performance hovering around local optima with minimal enhancements.

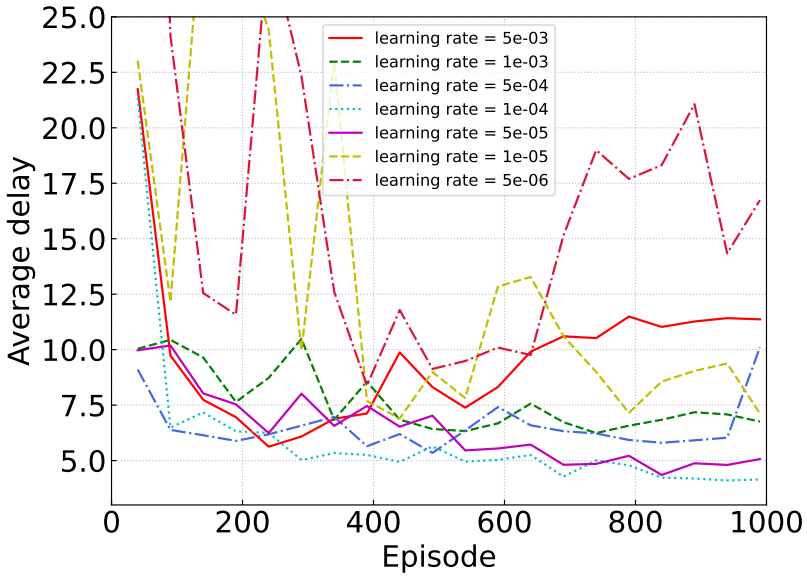

Figure 4. Convergence under different learning rates.

Figure 5 shows the performance of our proposed algorithm with varying numbers of active devices. As the number of active devices increases, the tasks generated per time slot also rise, consequently elevating the system load when the edge server and idle devices remain constant. This increase in system load contributes to a rise in average delay. Nonetheless, our proposed algorithm converges even in the face of these escalating challenges within the environment.

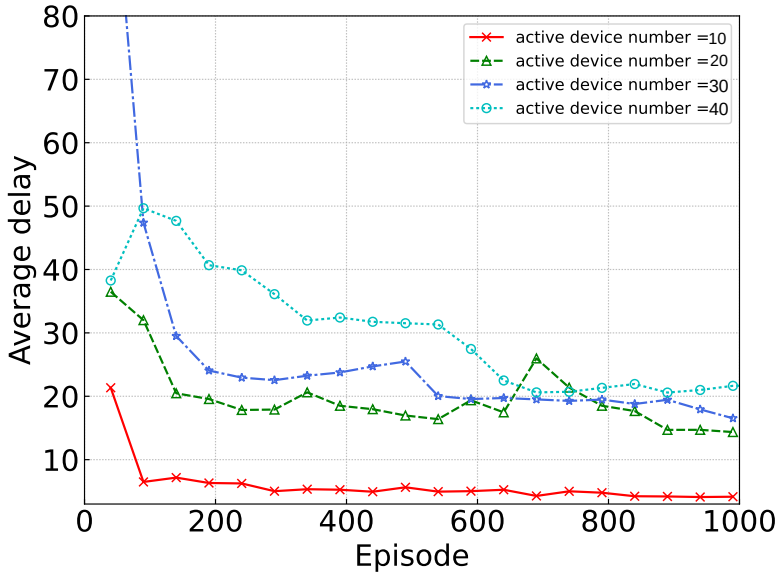

Figure 5. Convergence under different numbers of active devices.

Figure 6 shows the performance of our proposed algorithm under various mean values of harvested energy. A higher mean harvested energy value indicates a less stringent energy constraint. From Figure 6, it can be seen that our algorithm shows good convergence when the mean value is greater than 5. Moreover, as the mean value increases, the algorithm converges faster.Meanwhile, we can observe that when the average harvested energy surpasses 6 units, the captured energy proves adequate to fulfill the device's requirements,

with no additional benefit from further increases. Conversely, an insufficiency in harvested energy leads to temporary operational interruptions.

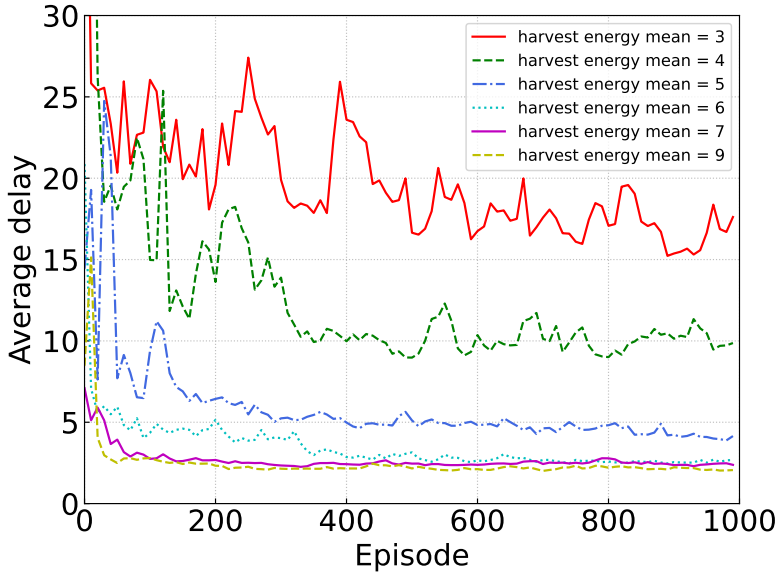

Figure 6. Performance under different harvested energy means.

Figure 7 shows the performance of our proposed algorithm under different battery capacities. It is evident that suboptimal outcomes are only evident when the battery capacity is limited, as seen with a value like 30. Conversely, when the battery capacity is substantial, the algorithm consistently delivers comparable performance, regardless of the specific capacity, underscoring its efficient utilization of the captured energy.

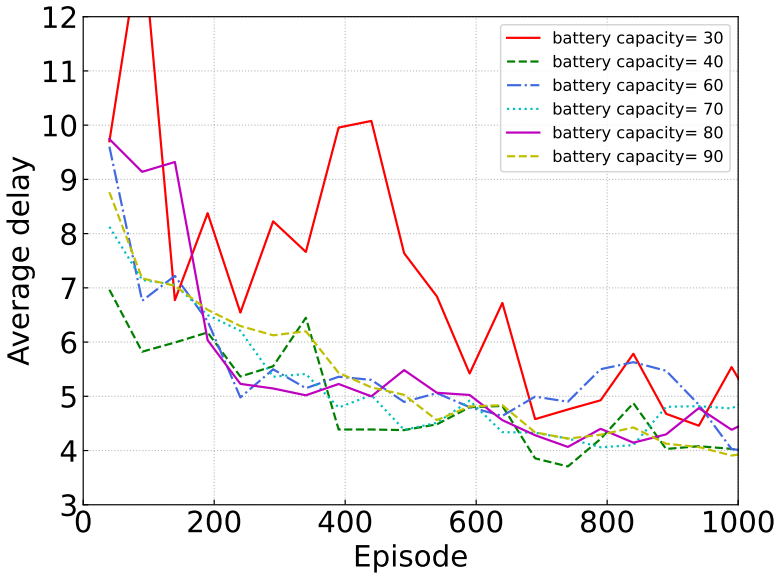

Figure 7. Performance under different battery capacities.

Figure 8 shows the performance of our proposed algorithm under varying CPU frequencies of idle devices. As the CPU frequency of idle devices increases, indicating a stronger processing capability, tasks offloaded to the device are completed more swiftly. However, due to stringent energy constraints on idle devices, higher CPU frequencies result in increased energy consumption. The harvested energy from the battery may prove insufficient to meet the energy demands of high CPU frequencies, leading to a rise in average delay rather than a reduction. It is evident from Figure 8 that the average delay is minimized when the CPU frequency is 2.5×10^8.

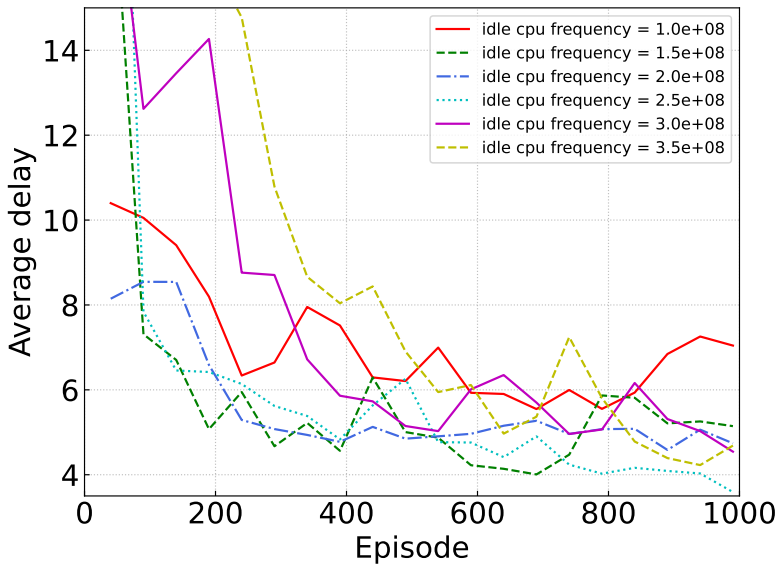

Figure 8. Performance under different idle MD CPU frequencies.

7. Conclusions

In this paper, we introduced a task-offloading framework that leverages energy harvesting and DVFS techniques in D2D-assisted MEC networks to minimize task delay within energy constraints. To tackle limited information exchange stemming from communication channels and user privacy concerns, we developed an MAPPO-based algorithm called MACO. MACO follows a centralized training with decentralized execution framework to determine offload targets and CPU frequencies. Our experimental results showcased the efficacy and resilience of our proposed algorithm. For future research, we aim to introduce a radio frequency-based charging method to enhance device battery life and enhance overall system controllability.

Author Contributions: Conceptualization, X.M. and H.H.; methodology, X.M. and H.H.; software, X.M.; validation, H.H. and H.S.; investigation, X.M.; resources, H.H.; writing—original draft preparation, X.M.; writing—review, H.H. 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 Huaiwen He.

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. Abbas, N.; Sharafeddine, S.; Mourad, A.; Abou-Rjeily, C.; Fawaz, W. Joint computing, communication and cost-aware task offloading in d2d-enabled het-mec. *Comput. Netw.* **2022**, *209*, 108900. [CrossRef]
2. Xiao, Z.; Shu, J.; Jiang, H.; Lui, J.C.S.; Min, G.; Liu, J.; Dustdar, S. Multi-objective parallel task offloading and content caching in d2d-aided mec networks. *IEEE Trans. Mob. Comput.* **2022**, *22*, 6599–6615. [CrossRef]
3. Ke, H.; Wang, J.; Deng, L.; Ge, Y.; Wang, H. Deep reinforcement learning-based adaptive computation offloading for mec in heterogeneous vehicular networks. *IEEE Trans. Veh. Technol.* **2020**, *69*, 7916–7929. [CrossRef]

4. Cao, B.; Zhang, L.; Li, Y.; Feng, D.; Cao, W. Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework. *IEEE Commun. Mag.* **2019**, *57*, 56–62. [CrossRef]
5. Mi, X.; He, H. Multi-agent deep reinforcement learning for d2d-assisted mec system with energy harvesting. In Proceedings of the 2023 25th International Conference on Advanced Communication Technology (ICACT), Pyeongchang, South Korea, 19 February 2023; pp. 145–153.
6. Salim, M.M.; Elsayed, H.A.; Abdalzaher, M.S. A survey on essential challenges in relay-aided d2d communication for next-generation cellular networks. *J. Netw. Comput. Appl.* **2023**, *216*, 103657. [CrossRef]
7. Wang, X.; Ye, J.; Lui, J.C.S. Mean field graph based d2d collaboration and offloading pricing in mobile edge computing. *IEEE/ACM Trans. Netw.* **2023**, *32*, 491–505. [CrossRef]
8. Wu, H.; Chen, J.; Nguyen, T.N.; Tang, H. Lyapunov-guided delay-aware energy efficient offloading in iiot-mec systems. *IEEE Trans. Ind. Inform.* **2022**, *19*, 2117–2128. [CrossRef]
9. Wang, H.; Lin, Z.; Lv, T. Energy and delay minimization of partial computing offloading for d2d-assisted mec systems. In Proceedings of the 2021 IEEE Wireless Communications and Networking Conference (WCNC), Nanjing, China, 29 March 2021; pp. 1–6.
10. Chen, J.; Xing, H.; Xiao, Z.; Xu, L.; Tao, T. A drl agent for jointly optimizing computation offloading and resource allocation in mec. *IEEE Internet Things J.* **2021**, *8*, 17508–17524. [CrossRef]
11. Zhao, P.; Tao, J.; Kangjie, L.; Zhang, G.; Gao, F. Deep reinforcement learning-based joint optimization of delay and privacy in multiple-user mec systems. *IEEE Trans. Cloud Comput.* **2022**, *11*, 1487–1499. [CrossRef]
12. Goudarzi, M.; Palaniswami, M.; Buyya, R. Scheduling iot applications in edge and fog computing environments: A taxonomy and future directions. *ACM Comput. Surv.* **2022**, *55*, 1–41. [CrossRef]
13. Tang, M.; Wong, V.W.S. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Trans. Mob. Comput.* **2020**, *21*, 1985–1997. [CrossRef]
14. Qiu, X.; Liu, L.; Chen, W.; Hong, Z.; Zheng, Z. Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing. *IEEE Trans. Veh. Technol.* **2019**, *68*, 8050–8062. [CrossRef]
15. Huang, H.; Ye, Q.; Du, H. Reinforcement learning based offloading for realtime applications in mobile edge computing. In Proceedings of the ICC 2020-2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7 June 2020; pp. 1–6.
16. Li, J.; Gao, H.; Lv, T.; Lu, Y. Deep reinforcement learning based computation offloading and resource allocation for mec. In Proceedings of the 2018 IEEE Wireless Communications and Networking Conference (WCNC), Barcelona, Spain, 15 April 2018; pp. 1–6.
17. Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W.M.; Zambaldi, V.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Leibo, J.Z.; Tuyls, K.; et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv* **2017**, arXiv:1706.05296.
18. Guo, M.; Li, Q.; Peng, Z.; Liu, X.; Cui, D. Energy harvesting computation offloading game towards minimizing delay for mobile edge computing. *Comput. Netw.* **2022**, *204*, 108678. [CrossRef]
19. Asim, M.; Affendi, M.E.L.; El-Latif, A.A.A. Multi-irs and multi-uav-assisted mec system for 5g/6g networks: Efficient joint trajectory optimization and passive beamforming framework. *IEEE Trans. Intell. Transp. Syst.* **2022**, *24*, 4553–4564. [CrossRef]
20. Chai, R.; Lin, J.; Chen, M.; Chen, Q. Task execution cost minimization-based joint computation offloading and resource allocation for cellular d2d mec systems. *IEEE Syst. J.* **2019**, *13*, 4110–4121. [CrossRef]
21. Liang, J.; Li, K.; Liu, C.; Li, K. Joint offloading and scheduling decisions for dag applications in mobile edge computing. *Neurocomputing* **2021**, *424*, 160–171. [CrossRef]
22. Xia, S.; Yao, Z.; Li, Y.; Mao, S. Online distributed offloading and computing resource management with energy harvesting for heterogeneous mec-enabled iot. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 6743–6757. [CrossRef]
23. Chen, X.; Zhang, H.; Wu, C.; Mao, S.; Ji, Y.; Bennis, M. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet Things J.* **2018**, *6*, 4005–4018. [CrossRef]
24. Huang, L.; Feng, X.; Zhang, C.; Qian, L.; Wu, Y. Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. *Digit. Commun. Networks* **2019**, *5*, 10–17. [CrossRef]
25. Huang, L.; Bi, S.; Zhang, Y.A. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. *IEEE Trans. Mob. Comput.* **2019**, *19*, 2581–2593. [CrossRef]
26. Hao, H.; Xu, C.; Zhang, W.; Yang, S.; Muntean, G.-M. Joint task offloading, resource allocation, and trajectory design for multi-uav cooperative edge computing with task priority. *IEEE Trans. Mob. Comput.* **2024**, *in press*. [CrossRef]
27. Dai, X.; Xiao, Z.; Jiang, H.; Alazab, M.; Lui, J.C.S.; Dustdar, S.; Liu, J. Task co-offloading for d2d-assisted mobile edge computing in industrial internet of things. *IEEE Trans. Ind. Inform.* **2022**, *19*, 480–490. [CrossRef]
28. Sun, M.; Xu, X.; Huang, Y.; Wu, Q.; Tao, X.; Zhang, P. Resource management for computation offloading in d2d-aided wireless powered mobile-edge computing networks. *IEEE Internet Things J.* **2020**, *8*, 8005–8020. [CrossRef]
29. Liu, Y.; Cai, Y.; Liu, A.; Zhao, M.; Hanzo, L. Latency minimization for mmwave d2d mobile edge computing systems: Joint task allocation and hybrid beamforming design. *IEEE Trans. Veh. Technol.* **2022**, *71*, 12206–12221. [CrossRef]
30. Elgendy, I.A.; Zhang, W.; He, H.; Gupta, B.B.; El-Latif, A.; Ahmed, A. Joint computation offloading and task caching for multi-user and multi-task mec systems: Reinforcement learning-based algorithms. *Wirel. Netw.* **2021**, *27*, 2023–2038. [CrossRef]
31. Zhang, J.; Du, J.; Shen, Y.; Wang, J. Dynamic computation offloading with energy harvesting devices: A hybrid-decision-based deep reinforcement learning approach. *IEEE Internet Things J.* **2020**, *7*, 9303–9317. [CrossRef]

32. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
33. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Proceedings of the International conference on machine learning, Stockholm, Sweden, 3 July 2018; pp. 1861–1870.
34. Kakade, S.; Langford, J. Approximately optimal approximate reinforcement learning. In Proceedings of the The 19th International Conference on Machine Learning, Sydney, Australia, 8 July 2002; pp. 267–274.
35. Li, G.; Chen, M.; Wei, X.; Qi, T.; Zhuang, W. Computation offloading with reinforcement learning in d2d-mec network. In Proceedings of the 2020 International Wireless Communications and Mobile Computing (IWCMC), Limassol, Cyprus, 15 June 2020; pp. 69–74.
36. Yu, C.; Velu, A.; Vinitsky, E.; Wang, Y.; Bayen, A.; Wu, Y. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv* **2021**, arXiv:2103.01955.