

Multi-Agent Deep Reinforcement Learning for D2D-assisted MEC system with Energy Harvesting

Xin Mi

School of Computer and Engineering

University of Electronic Science and Technology of China

Chengdu, China

202021080226@std.uestc.edu.cn

Huaiwen He*

School of Computer

University of Electronic Science and Technology of China,
Zhongshan Institute, Zhongshan, China

he_huai_wen@aliyun.com

Abstract—It is challenging for delay-sensitive task offloading in a Device-to-Device(D2D) assisted MEC system with energy harvesting devices due to the dynamic load level at each edge node and the fluctuation of harvested energy. In this paper, we investigate the joint task assignment and frequency control problem for MEC system integrated with D2D communication and energy harvesting. D2D link is leveraged to reduce the traffic load of the cellular base station and improve the utilization of computing resources in MEC. The main objective is to minimize the long-term average task delay with battery energy constraint. We first formulate a large scale non-linear integer programming problem that makes task assignment and CPU frequency decisions for each active device at each time slot. To solve the sequential decision problem with huge multi-dimensional discrete action space, we propose a multi-agent reinforcement learning algorithm based on Multi-Agent Proximal Policy Optimization (MAPPO) technique. Our algorithm can work efficiently under non-stationary environment and convergence fast. Experimental results show that our proposed algorithm outperforms other baseline algorithms.

Index Terms—MEC, D2D communication, multi-agent reinforcement learning, energy harvesting, task offloading

I. Introduction

With the rapid development of software services, more and more services become computation-intensive and delay-sensitive, such as virtual reality, augmented reality, online games. In light of this development, Mobile cloud computing (MCC), which may introduce large communication delay, can no longer meet the requirement [1]. Therefore, Mobile Edge Computing (MEC), which can offload tasks to edge server closed to the end user and significantly reduce the service response delay, is considered to be a promising work and has attracted the attention of researchers [2].

Cao et al. [3] introduced the basic concept of MEC and reviewed main applications and existing fundamental works using various machine learning-based approaches. Ke et al. [4] designed a task computation offloading model in a heterogeneous vehicular network and proposed an adaptive computation offloading method based on deep reinforcement learning to obtain the trade-off between the cost of energy consumption and data transmission delay. Chen et al. [5] studied the joint optimization problem of computation offloading and resource allocation in MEC

network, and proposed a novel temporal attentional deterministic policy gradient that incorporated a temporal feature extraction network and a rank-based prioritized experience replay to tackle the problem.

In addition to the edge server, the computing resources of idle devices in the system cannot be ignored. Device to Device (D2D) technology allows users to offload tasks to adjacent idle devices, thereby utilizing the computing resources of idle devices to improve the overall computing efficiency of the system. Wang et al. [6] proposed a D2D-assisted MEC system, which can process a large number of independent computing tasks to reduce energy consumption and delay. Chai et al. [7] jointly studied computation offloading and resource allocation problem in cellular device-to-device mobile edge computing system and proposed a heuristic algorithm that successively solves computation offloading subproblem and resource allocation subproblem by Kuhn-Munkres algorithm and Lagrangian dual method, respectively.

The above works reduces the delay of tasks, but ignores that tasks may last for multiple time frames and have an impact on subsequent decisions. [8] considered non-divisible and delay-sensitive tasks as well as edge load dynamics and formulated a task offloading problem to minimize the expected long-term cost. [9] built a D2D-MEC model with user mobility and optimized the processing decision to save energy consumption in continuous time.

However, the energy in the above work is sufficient. In practice, the decision of offloading is usually subject to energy constraints. If the device runs out of energy, its service will be interrupted, seriously compromising the quality of service. In order to meet the demand for continuous service of mobile devices, one practical approach is Dynamic Voltage and Frequency Scaling (DVFS). Huang et al. [10] proposed a realtime reinforcement learning based offloading scheme, which was based on both MEC-based offloading and DVFS-based energy consumption reduction, and jointly learned the optimal offloading policy and DVFS-based scheduling method using double Q-learning algorithm. Liang et al. [11] considered offloading with precedence constraints among tasks and tried to

minimize makespan over a MEC center with multiple frequency-scalable server.

In addition to saving energy by adjusting the CPU frequency, Energy Harvesting (EH) technology has caught the attention of researchers by significantly extending the life of devices by capturing green energy. Xia et al. [12] 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. [13] investigated a green MEC with EH devices and developed an effective EH computation offloading scheme towards minimizing delay for green MEC.

Compared with above studies, for the task offloading paradigm, works [4][5][13] only considered MEC offloading while ignoring the computation resource of idle devices in the system. In terms of task attributes, works [5][6][12] studied partial offloading (tasks can be arbitrarily divided), while what introduced in this paper is more realistic full offloading (tasks can only be performed entirely). In particular, works [5][10] did not consider the underlying queue system but assumed that tasks could be completed within one time frame without affecting subsequent tasks. In the papers that introduced the DVFS technology, works [10][12][13] set the CPU frequency within a specific range instead of choosing different levels of CPU frequency as we introduced in this paper. Work [11] also controlled different levels of CPU frequency, but it controlled the CPU frequency of edge servers instead of mobile devices, so there was no explicit energy limit.

In the above work on sequential decision-making problems, many used reinforcement learning algorithms to solve the proposed problem [4][5][10]. However, when a single-agent reinforcement learning algorithm handles tasks involving multiple devices, there will be problems such as action space explosion and lazy agent, which will lead to difficulty in convergence and bad performance [14]. Lazy agent happens when one agent learns a useful policy, but a second agent is discouraged from learning because its exploration would hinder the first agent and lead to worse team reward.

In this paper, we consider the problem of delay minimization in D2D-assisted MEC networks under energy constraint, where devices can offload tasks to MEC edge servers via cellular links or to idle devices via D2D links, or execute locally, that the CPU frequency of devices can be controlled by DVFS technology to meet the energy constraints, and the energy can be supplemented by capturing green energy through the energy harvesting module. The proposed model can be applied to systems such as environmental monitoring IoT systems that exchange and process information under strict energy constraints. This problem is difficult to solve by traditional methods because the task generation, channel state, and green energy captured are stochastic and unpredictable. For each device, we need to solve the offloading decision

as well as the CPU frequency simultaneously, and this multi-dimensional discrete action space is also a challenge. Moreover, due to the need to make decisions for multiple devices at the same time, single-agent reinforcement learning algorithms are also difficult to solve. Therefore, we propose an algorithm based on multi-agent reinforcement learning, which can greatly reduce the action space of each agent, make the problem solvable, and ensure that the computational complexity is acceptable.

In particular, our key contributions are summarized as follows:

- **Model:** We simultaneously introduce DVFS and energy harvest technology into the task offloading framework in D2D-assisted MEC network to minimize the average task delay under strict energy constraints. The device can make decisions on CPU frequency and task load based on the current load and battery status.
- **Algorithm:** We model a highly complex non-convex optimization problem as a Markov decision problem and use a multi-agent reinforcement learning algorithm based on MAPPO to deal with the multi-dimensional discrete action spaces and complex cooperative relationships between devices.
- **Experiment:** We implement extensive simulations in the proposed framework to verify the effectiveness of the proposed algorithm. The numerical simulation results show that our proposed algorithm outperforms other baseline algorithms.

The remainder of this paper is organized as follows: Section II introduces our proposed model and basic formulas. Section III gives the mathematical model of the problem. Section IV gives the algorithm description and implementation details to solve the problem. In Section V, we compare the proposed algorithm with other baseline algorithms to verify the performance of our proposed algorithm. Finally, we draw the conclusions in Section VI.

II. System Model

In this paper, we consider a D2D-assisted MEC network which contains an edge server and a set of mobile devices $\mathcal{D}$. We assume that the system runs in a discrete time slot mode, and the whole period is divided into T time slots, which is denoted as $\mathcal{T} = \{1, 2, \dots, T\}$. The length of each time slot lasts for Δt which is small enough to ensure that at most one task is generated for each device within a slot. In the following subsections, we provide the details of device model, computation model and transmission model, delay model and energy model in the system.

A. Device Model

In the MEC system, the mobile devices $\mathcal{D}$ are divided into the active device set $\mathcal{M} = \{M_1, M_2, \dots, M_{|\mathcal{M}|}\}$, and the idle device set $\mathcal{N} = \{N_1, N_2, \dots, N_{|\mathcal{N}|}\}$. Here $\mathcal{D} = \mathcal{M} \cup \mathcal{N}$. Computation tasks only are generated in active mobile devices and can be offloaded to idle devices by D2D

link or the edge server by cellular link. Note that the idle devices do not generate computation tasks. Each mobile device $d \in \mathcal{D}$ employs a battery RB_d and integrates with an energy harvesting module. In each time slot $t \in \mathcal{T}$, each device d obtains energy e_d^h from environment, which is dynamic and hard to predict precisely. We assume that a computation task is generated in active device m at the t th time slot with a certain probability, denoted as $w(m, t) = (s(m, t), c(m, t))$, where $s(m, t)$ represents the size of task w , in 1-bit unit, and $c(m, t)$ represents the computation complexity of task w , in the number of CPU cycles required for a 1-bit operation. Herein we assume that the computation task is indivisible and can last for multiple time slots.

We adopt queueing system to model the process of task execution. Computation tasks will be pushed into the queue and wait to schedule in First Come First Serve (FCFS) manner. For each active device, there are two types of queue: one is computation queues Q_d^{comp} for task execution, the other is transmission queue Q_m^{tran} for task transmission. When a task is generated at an active mobile device m , the scheduler first decide the target place to execute it. There are three choices for task execution: executing locally or offloading to the edge server via cellular links or offloading the idle devices via D2D links. As shown in Fig. 1, $a_m \in \{-1, 0, 1, \dots, |\mathcal{M}|\}$ is used to denote task execute decision for task $w(m, t)$. If $a_m = -1$, the generated task w would enter the computation queue Q_d^{comp} and execute at the local device m ; if $a_m = 0$, computation task w would enter the transmission queue Q_m^{tran} and offload to the edge server; otherwise w would enter the transmission queue Q_m^{tran} and offload to the a_m th device in the idle devices set $\mathcal{N}$. There is only a computation queue Q^{comp} at the idle device and edge server. After task transmission is completed, the task will be pushed into the computation queue Q_n^{comp} of the target device to wait for execution. At the end of each time slot, the device broadcasts its state so that each active device can obtain global information and make action decisions accordingly. The power consumption of task transmission and execution are constrained by the limited power of mobile device energy.

We assume that active devices $m \in \mathcal{M}$ can adjust their CPU frequency $f_m \in \mathcal{F}$ via the dynamic voltage and frequency scaling (DVFS) technology to save power for the limited battery energy. Note that with the decreasing of CPU frequency, the power consumption would reduce but the task delay would increase and may violate the user's Service-Level Agreement (SLA). Here we adopt a discrete CPU frequency model and use $\mathcal{F} = \{f^1, f^2, \dots, f^K\}$ to denote the set of optional CPU frequencies for each active device. We suppose that CPU in idle devices runs at a fixed frequency f^{idle} and the edge server runs with a much higher CPU frequency f_0 with unlimited energy.

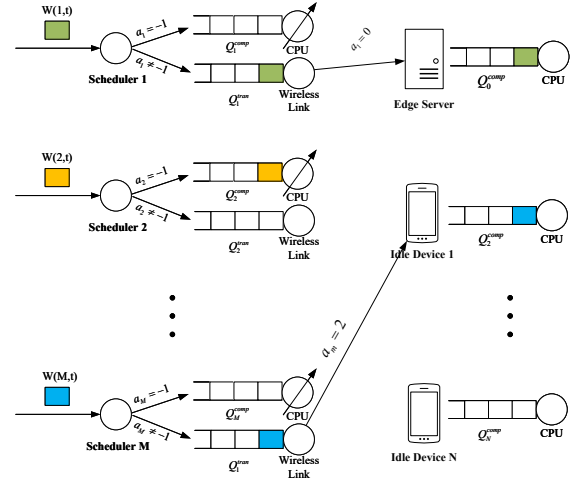


Fig. 1. Scheduling model of MEC system

B. Computation Model

For computation task $w(m, t)$ generated at mobile device m , it will be processed at m or transfer to the target device d' . The task $w(m, t)$ enters the computation queue and waits for processing. Let $\delta^{comp}(m, t)$ denote the waiting time of task $w(m, t)$ from entering computation queue to begin execution, and $l^{comp}(m, t)$ denote the completion time of the task $w(m, t)$ from entering computation queue to task completion. Since task $w(m, t)$ are scheduled to execute until the previous task is processed completely[8], we can derive the waiting time of $w(m, t)$ as follows:

$$\delta^{comp}(m, t) = \max_{w(m'', t'') \in Q_{d'}^{comp}} l^{comp}(m'', t'') \quad (1)$$

where $w(m'', t'')$ denotes the previous task of $w(m, t)$ in the computation queue $Q_{d'}^{comp}$, respectively. Note that $d' = m$ if task $w(m, t)$ executes locally. For example, suppose task $w(1, 2)$ is offloaded to the target device $d' = 4$ and enters the computation queue Q_4^{comp} with only reside task $w(3, 2)$, which task execution time is 5 time slot, we will obtain that the waiting time of task $w(1, 2)$ is $\delta_{d'}^{comp}(m, t) = 5$.

According to the above analysis, the completion time of task $w(m, t)$ is expressed as follows:

$$l^{comp}(m, t) = \delta^{comp}(m, t) + \lceil \frac{s(m, t)c(m, t)}{f_{d'}} \rceil \quad (2)$$

where $\lceil \cdot \rceil$ is the ceiling function since tasks only perform task switching at new time slot. $\lceil \frac{s(m, t)c(m, t)}{f_{d'}} \rceil$ represents the number of time slots required to process task $w(m, t)$.

C. Transmission Model

For each active mobile device $m \in \mathcal{M}$, a transmission queue Q_m^{tran} is maintained to offload task to the edge server or idle device. Scheduler m determine the target

device d' before task $w(m, t)$ enters the queues. When a task $w(m, t)$ enters the transmission queue, it will wait for transmission until the previous tasks are finished.

Let $\delta^{tran}(m, t)$ denote the waiting time of the task $w(m, t)$ in Q_m^{tran} before offloading to transfer, and $l^{tran}(m, t)$ denotes the transmission time of task $w(m, t)$, such that we have the following equations:

$$\delta^{tran}(m, t) = \max_{w(m, t'') \in Q_m^{tran}} l^{tran}(m, t'') \quad (3)$$

$$l^{tran}(m, t) = \delta^{tran}(m, t) + \lceil \frac{s(m, t)}{r_m} \rceil \quad (4)$$

where $w(m, t''), t'' \leq t$ represents the previous task of $w(m, t)$ in Q_m^{tran} and r_m denotes the transmission rate for task transmission. Based on Shannon's theorem, we derive the following equation:

$$r_m = B_m \log(1 + \frac{ph}{N_0}) \quad (5)$$

where B_m denotes the bandwidth, p denotes the transmission power, h denotes the channel gain between device m and the target device d' , and N_0 denotes the white noise of channel. Here we suppose that transmission power p is fixed for all active device as in [12][15] and the total bandwidth are allocated equally to each transfer pairs.

D. Delay Model

The total delay of task $w(m, t)$ can be expressed as follows:

$$l(m, t) = l^{comp}(m, t) + l^{tran}(m, t) \quad (6)$$

If task $w(m, t)$ is executed locally, the transmission delay $l^{tran}(m, t)$ is set to be 0.

E. Energy Model

At each time slot, device $d \in \mathcal{D}$ harvests a random amount of energy $e_d^h(t)$ by energy harvesting module and saves it in the battery RB_d . Notely, the remaining battery energy RB does not exceed the upper limit RB^{max} . However, because the computation and transmission of the device consume energy, similar to [16][17], the energy in the battery will also change, which is updated according to the following equation:

$$RB_d(t+1) = \min \{ \max \{ RB_d(t) - e_d^c(t), 0 \} + e_d^h(t), RB^{max} \} \quad (7)$$

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

$$e_d^c(t) = \begin{cases} \mathbb{I}(Q_d^{comp} \neq \emptyset) \epsilon f_d(t)^3 + \mathbb{I}(Q_d^{tran} \neq \emptyset) p \Delta t, & \forall d \in \mathcal{M} \\ \mathbb{I}(Q_d^{comp} \neq \emptyset) \epsilon f_d(t)^3, & \forall d \in \mathcal{N} \end{cases} \quad (8)$$

where $\epsilon > 0$ is a coefficient related to the CPU chip architecture, $\mathbb{I}(\cdot)$ is an indicator function that outputs 1 when $\cdot$ is true and 0 otherwise. 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 [12], if there is not enough energy to support computation or transmission, the task continues to wait in the queue until there is enough energy in the battery.

III. Problem Formulation

In this paper, we aim to minimize the time average task delay under energy constraint of mobile devices in a D2D-assisted MEC system. At each time slot t , MEC system makes task offloading decisions $A = \{a_1, a_2, \dots, a_{|\mathcal{M}|}\}$ for each generated task and CPU frequency decisions $F = \{f_1, f_2, \dots, f_{|\mathcal{M}|}\}$ for each active device to optimize the average task delay without knowing the future information. Here decision a_m is a discrete integer which represents the target device to execute the task $w(m, t)$ generated at slot t as mentioned above. Decision f_m is also an integer representing the corresponding CPU frequency for task execution at mobile device m at slot t . For example, $f_m = k$ means the frequency of device m is set to f^k .

Thus, the time average task delay minimization problem in D2D-assisted MEC system can be formulated as following problem P1.

$$\mathbf{P1} : \min_{A, F} \frac{1}{T} \sum_{t=1}^T \frac{1}{M} \sum_{d=1}^M l(m, t) \quad (9a)$$

s.t.

$$a_m \in \{-1, 0, 1, 2, \dots, N\}, \forall a_m \in A, \quad (9b)$$

$$f_m \in \{1, 2, \dots, K\}, \forall f_m \in F \quad (9c)$$

$$0 \leq RB_d(t) - e_d^c(t) + e_d^h(t) \leq RB^{max}, \forall d \in \mathcal{D} \quad (9d)$$

Eq. (9b) defines the domain of decision a_m . Note that $a_m = -1$ means generated task executes locally, $a_m = 0$ means task is offloaded to the edge server, and otherwise the value of a_m represents the index of idle device to execute the task. Eq. (9c) defines CPU frequency of active device m . Eq. (9d) is battery energy constraint for all mobile device.

The Problem P1 is a Non-Linear Integer Programming (NLP) problem even though all the offline environment information (generated task information, harvested energy and channel state) was known in advance. Generally, P1 is non-convex and NP-hard. The key challenges in solving the problem P1 lies in the following aspects: 1) The tasks generation, the amount of energy harvested, and the communication channel state are time-varying and hard to predict precisely and unpredictable, which needs to design algorithm runs in an online manner; 2) Energy constraint 9d make the current decisions couple with the future decision and the recursive expression of waiting time in queues make it more complexity. 3) The objective function is non-convex and the control decisions are discrete with large action space. Therefore, P1 is hard to be solved by directly using the previous algorithms proposed in [8] or simply extending it. Here we seek to use

a multi-agent reinforcement learning approach to reduce complexity and run in a polynomial time.

Problem complexity analysis: Assume that the number of active devices is M , the number of idle devices is N , and there are K kinds of CPU frequencies. Thus, the total number of combinations of decisions (a_m, f_m) made by each active device at slot t is $(N+2)K$. Furthermore, the whole action space at slot t is $((N+2)K)^M$. With the increasing of mobile devices, the action space will grow exponentially, which imposes great challenge to solve the problem by using a centralized network and a single-agent reinforcement learning algorithm which will incur curse of dimensionality and hard to converge.

IV. Algorithm

Based on the analysis in Section III, we seek to design a new algorithm by using multi-agent reinforcement learning technique to solve problem P1. We consider reinforcement learning that continuously conducts trial-and-error learning in the target environment and modifies the strategy through feedback results to maximize rewards. Facing huge action space, we adapt multi-agent reinforcement learning to allocate an independent actor-critic network for each active device to make decisions, thereby reducing the action space. However, this will lead to a nonstationary environment, so we adapt the MAPPO-based multi-agent reinforcement learning to enhance the stability of the network by clipping operation, which is similar to the IPPO algorithm, but uses global state information as input to enable better cooperation between agents.

First, we model P1 as a Markov decision process.

A. State

Since we model the problem as a Markov decision process, the Markov property must be satisfied, which means that state transition of the Markov process in the future depends only on the present state and does not depend on past history. The parameters that affect the decision variables are the properties of the currently generated task include task size $s(m, t)$ and task complexity $c(m, t)$, the task load of the device which includes the computation queue length Q_d^{comp} and the transmission queue length Q_m^{tran} , and the remaining energy of the device RB_d .

In summary, the global state information is $S(t) = (s(m, t), c(m, t), Q_0^{comp}, Q_d^{comp}(t), Q_m^{tran}(t), RB_d(t), \forall m \in \mathcal{M})$ is set to Without loss of generality, if no task is generated for active device m in the current time frame t , the task size $s(m, t)$, task complexity $c(m, t)$ are all set to 0. All agents use the same global information as the input to the network.

B. Action

As shown in Section III, the variables we decide on include the offloading decision for the current generating task $A = \{a_1, a_2, \dots, a_{|\mathcal{M}|}\}$ and the CPU frequency selection $F = \{f_1, f_2, \dots, f_{|\mathcal{M}|}\}$ for all active settings at the current time frame.

Algorithm 1: Our propose algorithm

```

Input: Active Device Set  $\mathcal{M}$ , total episode number
      N, episode length T,
1 for  $m \in \mathcal{M}$  do
2   Initialize the replay memory  $RM_m$ ;
3   Initialize the Actor network and Critic Network;
4 end
5 for n = 1 to N do
6   for t = 1 to T do
7     Incorporate the state of each device into a
      global state  $S(t)$ 
8     for  $m \in \mathcal{M}$  do
9       Compute the offloading decision  $a_m(t)$ 
      and CPU frequency decision  $f_m(t)$  by
      actor network;
10    end
11    Interact with the environment based on
      decisions  $A(t)$  and  $F(t)$ ;
12    for  $m \in \mathcal{M}$  do
13      Store transition into replay memory
       $RM_m$ ;
14    end
15  end
16  for  $m \in \mathcal{M}$  do
17    Sample random mini-batch of transitions
      from the replay memory  $RM_m$ ;
18    Update the actor network according to
      Eq.(12);
19    Update the critic network according to
      Eq.(13);
20  end
21 end

```

Since we use the MAPPO algorithm, the policy network only makes decisions on the corresponding device. For example, active device m only make decision $a_m \in \{-1, 0, 1, \dots, |N|\}$ and CPU frequency decision $f_m \in \{1, 2, \dots, K\}$ for itself.

C. Reward

As shown in Section III, our optimization goal is to minimize the average task completion delay, so our reward

$$r(S(t), A(t)) = \frac{1}{M} \sum_{d=1}^M l(m, t) \quad (10)$$

which denotes the average delay of tasks generated in time slot t .

D. Algorithm

For each device, its independent reinforcement learning algorithm is the same as the PPO algorithm. The PPO algorithm is based on the Actor-Critic framework. In order to improve the sample efficiency, importance sampling is introduced to make the samples reusable. In order to

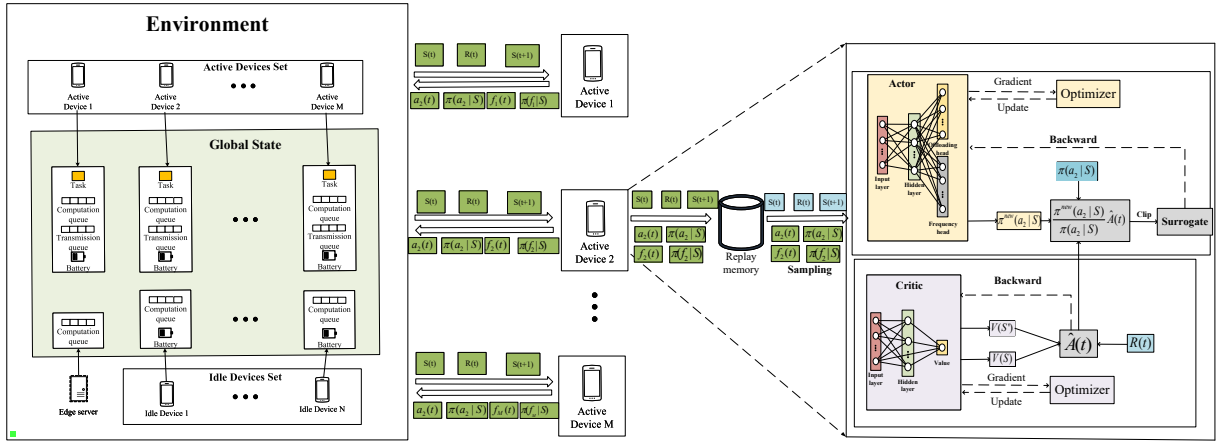


Fig. 2. Multi-agent reinforcement learning network architecture

prevent the network from being unstable or even crashing due to a violent network update, the policy clip operation is used to ensure that the network is only updated in a small amount.

Among them, the Actor network is used to output the policy, and the Critic network is used to estimate the current state to guide the learning of the Actor. The specific details are shown in Fig. 2: The state information of each device is integrated into the global state information and input into the input layer of the network. After being processed by two fully connected layers in the Actor network, the extracted features are sent to the offloading head and the frequency head respectively to obtain the corresponding action policy. When the action a_m and f_m returns to the agent m to interact with the environment, the agent m stores the transition $(S(t), a_m(t), \pi(a_m|S), f_m(t), \pi(f_m|S), r(t), S(t+1))$ into the replay memory RM_m for later training, where $\pi(a_m|S)$ and $\pi(f_m|S)$ denotes the probability corresponding to the action $a_m(t)$ and f_m respectively. In the training phase, we first sample data from the replay memory RB_m , compute the advantage value using the Critic network, and then compute the Actor's gradient based on the advantage value, action probability ratio, and clip operation.

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

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \quad (11)$$

where γ is the discount factor, it quantifies how much importance we give for future rewards, λ is a parameter, similar to $TD(\lambda)$, with a trade-off between variance and bias. δ_{t+l}^V is 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.

The Actor network loss function and the Critic network

TABLE I
The environment parameter settings

Parameter	Value
Active device number M	3
Idle device number N	3
Idle CPU frequency f^{idle}	$2 * 10^8$ cycles/s
Edge server CPU frequency f_0	$4 * 10^8$ cycles/s
Active device optional CPU frequency f_d	$[2 * 10^8, 2.5 * 10^8, 3 * 10^8]$ 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

loss function are computed as:

$$L_t(\theta^{actor}) = \min(r_t(\theta^{actor}) \hat{A}_t, \text{clip}(r_t(\theta^{actor}), 1-\epsilon, 1+\epsilon) \hat{A}_t) \quad (12)$$

$$L_t(\theta^{critic}) = \hat{A}_t^2 \quad (13)$$

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

In loss function of Actor network, the first item refers to conservative policy iteration [18], which would lead to an excessively large policy update. The second item, modifies the surrogate objective by clipping the probability ratio, which removes the incentive for moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$ [19]. And the minimization operation make the final objective is a lower bound on the unclipped objective. In this way, we make a small and safe update, i.e. when the action is good, we only slightly increase the probability of the good action.

Our proposed algorithm is summarized in Algorithm 1.

V. SIMULATION RESULTS

We build the reinforcement learning model with PyTorch framework, train our model on one computing server with four GeForce RTX 2080 Ti GPUs and one Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz 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 I. The parameters of the neural network will be introduced below: the batch size is set to 64, the learning rate is set to $1e-4$, 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. Meanwhile, we use Adam optimizer for network updating. We use simulators to produce training data online, and test data is pre-generated by simulators and is saved to eliminate the influence of simulators on experimental results.

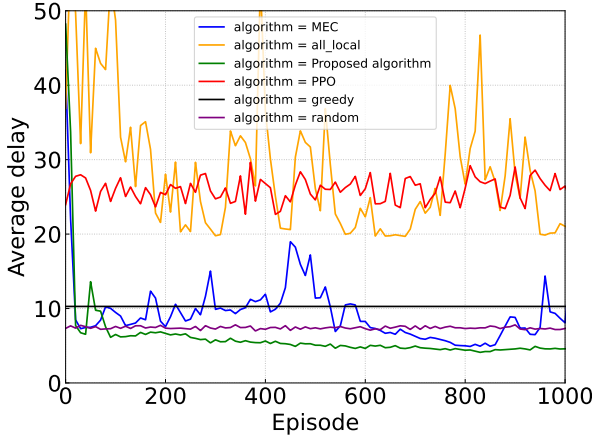


Fig. 3. Performance compared with baseline algorithms

Fig. 3 demonstrates the convergence ability of our proposed algorithm and compares it with other baseline algorithms. We compare the proposed algorithm with the single-agent reinforcement learning PPO, the MEC mode that only considers local computing and edge servers, the all-local mode that only considers local computing, the greedy algorithm, and the random algorithm. It can be seen that our proposed algorithm has significantly lower delay than other algorithms: 85% improvement compared to all local mode and PPO algorithms, 53% improvement compared to MEC mode, 65% improvement compared to greedy algorithm, and 43% improvement compared to random algorithm. And this illustration shows the powerful effect of mobile edge computing and D2D communication technology on delay.

Fig. 4 shows the convergence of the proposed algorithm under different learning rates. When the learning rate is too high, such as $5e-3$, the neural network cannot converge due to the parameters oscillating out of the valid range. When the learning rate is too low, such as $5e-6$, the algorithm is easy to converge to the local

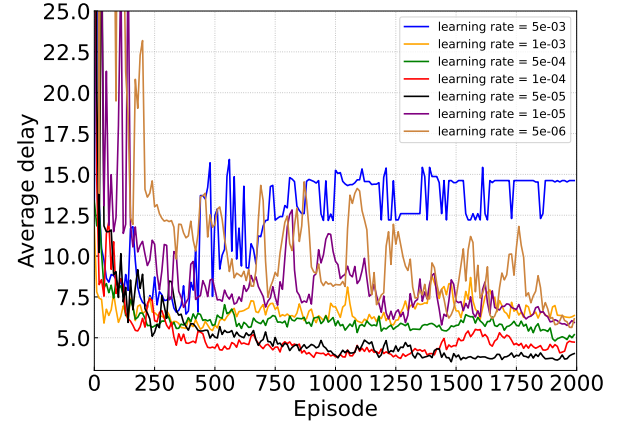


Fig. 4. Convergence at different learning rates

optimum which results suboptimal performance, and the convergence speed is slow.

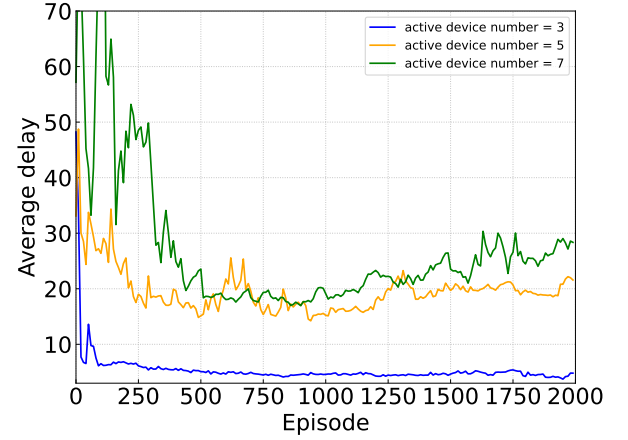


Fig. 5. Convergence under different active device numbers

Fig. 5 shows the performance of our proposed algorithm under different numbers of active devices. As the number of active devices increases, the number of tasks generated in each time slot in the system increases, and the system load increases when the edge server and idle devices remain unchanged. Facing increasingly complex environments, our proposed algorithm can also achieve convergence.

Fig. 6 shows the performance of our proposed algorithm under different harvest energy means. When the mean of harvested energy is higher, the energy restriction is looser. It can be seen from the Fig. 6 that when the mean is low, the device is severely constrained by energy and the performance fluctuates greatly, while when the mean is greater than or equal to 7, the energy is sufficient, resulting in little performance impact.

Fig. 7 shows the performance of our proposed algorithm under different gamma. Gamma is the discount factor. It quantifies how much importance we give for future rewards. Since at test phase, we only consider the average

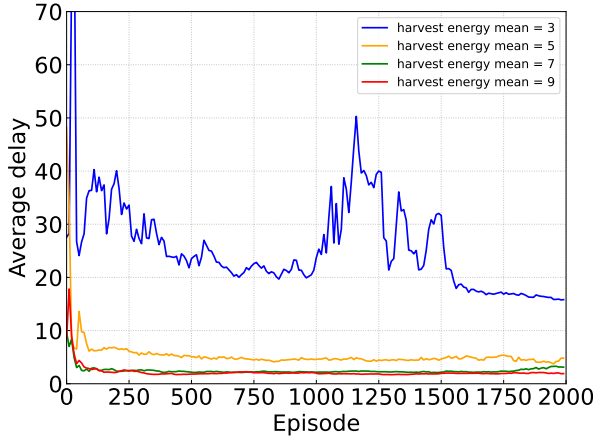


Fig. 6. Convergence under different harvest energy means

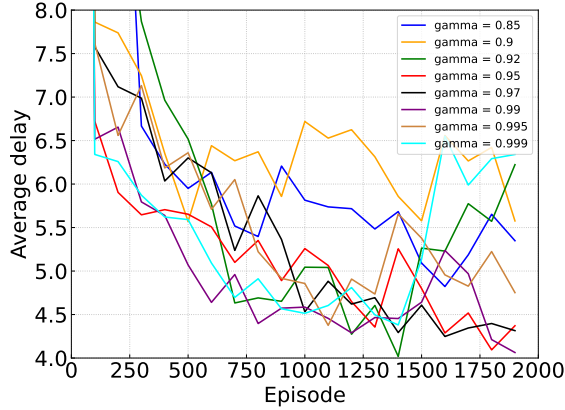


Fig. 7. Convergence under different gammas

task delay of the first 100 steps. Therefore, when the gamma is too small, such as 0.9, it cannot be considered long-term enough. When the gamma value is too high, such as 0.995, the returns of 100 steps will have bad performance.

VI. Conclusion

In this paper, we design a task offloading framework utilizing energy harvesting and Dynamic Voltage and Frequency Scaling (DVFS) techniques in D2D-assisted MEC networks to minimize task delay under energy constraints. We use MAPPO-based multi-agent reinforcement learning to make decisions on task offloading and CPU frequency adjustment. The experimental results show that after the introduction of MEC and D2D technology, the average task completion delay is greatly reduced, and our proposed algorithm achieves good performance improvements. In future research, wireless power transfer can also be considered to further improve the lifespan of the device and make the entire system more controllable.

References

- [1] Yun Li, Jie Liu, Bin Cao, and Chonggang Wang. Joint optimization of radio and virtual machine resources with uncertain user demands in mobile cloud computing. *IEEE Transactions on Multimedia*, 20(9): 2427–2438, 2018.
- [2] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2017.
- [3] Bin Cao, Long Zhang, Yun Li, Daquan Feng, and Wei Cao. Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework. *IEEE Communications Magazine*, 57(3):56–62, 2019.
- [4] Hongchang Ke, Jian Wang, Lingyue Deng, Yuming Ge, and Hui Wang. Deep reinforcement learning-based adaptive computation offloading for mec in heterogeneous vehicular networks. *IEEE Transactions on Vehicular Technology*, 69(7):7916–7929, 2020.
- [5] Juan Chen, Huanlai Xing, Zhiwen Xiao, Lexi Xu, and Tao Tao. A drl agent for jointly optimizing computation offloading and resource allocation in mec. *IEEE Internet of Things Journal*, 8(24):17508–17524, 2021.
- [6] 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.
- [7] 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.
- [8] Ming Tang and Vincent WS Wong. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing*, 2020.
- [9] Gaibin Li, Ming kai 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.
- [10] Hui Huang, Qiang Ye, and Hongwei Du. Reinforcement learning based offloading for realtime applications in mobile edge computing. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2020.
- [11] Jie Liang, Kenli Li, Chubo Liu, and Keqin Li. Joint offloading and scheduling decisions for dag applications in mobile edge computing. *Neurocomputing*, 424:160–171, 2021.
- [12] Shichao Xia, Zhixiu Yao, Yun Li, and Shiwen Mao. Online distributed offloading and computing resource management with energy harvesting for heterogeneous mec-enabled iot. *IEEE Transactions on Wire-*

- less Communications, 20(10):6743–6757, 2021.
- [13] Mian Guo, Qirui Li, Zhiping Peng, Xiushan Liu, and Delong Cui. Energy harvesting computation offloading game towards minimizing delay for mobile edge computing. *Computer Networks*, 204:108678, 2022.
 - [14] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.
 - [15] Ibrahim A Elgandy, Wei-Zhe Zhang, Hui He, Brij B Gupta, Abd El-Latif, and A Ahmed. Joint computation offloading and task caching for multi-user and multi-task mec systems: reinforcement learning-based algorithms. *Wireless Networks*, 27(3):2023–2038, 2021.
 - [16] Mengying Sun, Xiaodong Xu, Yuzhen Huang, Qihui Wu, Xiaofeng Tao, and Ping Zhang. Resource management for computation offloading in d2d-aided wireless powered mobile-edge computing networks. *IEEE Internet of Things Journal*, 8(10):8005–8020, 2020.
 - [17] Minghui Min, Liang Xiao, Ye Chen, Peng Cheng, Di Wu, and Weihua Zhuang. Learning-based computation offloading for iot devices with energy harvesting. *IEEE Transactions on Vehicular Technology*, 68(2):1930–1941, 2019.
 - [18] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer, 2002.
 - [19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.