electronics

Article

Computational Offloading for MEC Networks with Energy Harvesting: A Hierarchical Multi-Agent Reinforcement Learning Approach

Yu Sun † and Qijie He *,†

School of Computer and Electronics and Information, Guangxi University, Nanning 530004, China
* Correspondence: 2013391019@st.gxu.edu.cn
† These authors contributed equally to this work.

Abstract: Multi-access edge computing (MEC) is a novel computing paradigm that leverages nearby MEC servers to augment the computational capabilities of users with limited computational resources. In this paper, we investigate the computational offloading problem in multi-user multi-server MEC systems with energy harvesting, aiming to minimize both system latency and energy consumption by optimizing task offload location selection and task offload ratio.We propose a hierarchical computational offloading strategy based on multi-agent reinforcement learning (MARL). The proposed strategy decomposes the computational offloading problem into two sub-problems: a high-level task offloading location selection problem and a low-level task offloading ratio problem. The complexity of the problem is reduced by decoupling. To address these sub-problems, we propose a computational offloading framework based on multi-agent proximal policy optimization (MAPPO), where each agent generates actions based on its observed private state to avoid the problem of action space explosion due to the increasing number of user devices. Simulation results show that the proposed HDMAPPO strategy outperforms other baseline algorithms in terms of average task latency, energy consumption, and discard rate.

Keywords: multi-access edge computing; multi-agent reinforcement learning; energy harvesting; offloading strategy

Citation: Sun, Y.; He, Q. Computational Offloading for MEC Networks with Energy Harvesting: A Hierarchical Multi-Agent Reinforcement Learning Approach. *Electronics* **2023**, *12*, 1304. https://doi.org/10.3390/electronics12061304

Academic Editors: Jimmy Ming-Tai Wu, Matin Pirouz and Shahab Tayeb

Received: 15 February 2023
Revised: 3 March 2023
Accepted: 7 March 2023
Published: 9 March 2023

1. Introduction

Multi-access edge computing (MEC) is an innovative computing paradigm that has gained increasing attention in recent years. This computing model utilizes servers located at wireless access points (APs) to provide additional computing resources for mobile devices with limited computational capabilities [1]. The deployment of MEC servers facilitates the collaborative processing of computing tasks, reducing the computational latency and energy consumption of mobile devices. This, in turn, enhances the battery life of these devices and enables them to handle computationally intensive tasks more efficiently [2–4].

Computation offloading is a widely studied area in mobile computing which aims to address the computational demands of mobile devices by leveraging additional computing resources, thereby reducing their computational workloads. It has been widely recognized as an effective solution to resolve the computational constraints of mobile devices [5,6]. However, one of the major challenges associated with computation offloading is the limited energy of mobile devices powered by batteries, which can impact the local execution or offloading of tasks to MEC servers, resulting in tasks dropping due to timeouts. To mitigate this issue, researchers have proposed energy harvesting (EH) mechanisms [7]. Wireless power transfer (WPT) technology, in particular, allows for the transfer of electrical energy without the need for conductors as transmission links. The technology operates by generating a time-varying electromagnetic field, which transfers energy to a receiver device in the spatial field. The receiver device then extracts electrical energy from the field

to power electrical devices, effectively reducing the reliance on wires and batteries. WPT technology is therefore considered a promising solution to address the energy supply and access problems of Internet of Things (IoT) devices [8].

In order to explore the application of WPT in MEC computation offloading, this paper considers a wireless power transfer MEC system with multiple MEC servers and multiple users. The wireless access point provides power to the users through energy transmission beams and the users use the collected energy to execute tasks. Key issues that need to be addressed in computation offloading are: task offloading location selection: determining appropriate MEC servers for task offloading is crucial, as it is essential to consider not only the energy conditions but also the feasibility of offloading to avoid wasting of battery energy; offloading ratio optimization: the offloading ratio of a task can be determined by dividing it into independent parts, which can be executed locally and offloaded, respectively. However, optimizing computation offloading strategies in a dynamic MEC environment, where wireless radio signals are dynamic and task size is random, is challenging. The complex nature of the MEC environment further exacerbates this challenge [9,10], making the task of determining the optimal offloading ratio even more difficult.

In this paper, we study the computational offloading problem for multi-user multi-server MEC systems with energy harvesting with the aim of minimizing system latency and energy consumption by optimizing task offload location selection and task offload ratio.

We propose a reinforcement learning-based computational offloading approach that formulates the problem of balancing and minimizing system-average task latency, energy consumption, and task discard rate as a Markov decision process (MDP), and solves it using reinforcement learning. Our proposed approach differs from other reinforcement learning-based approaches in several ways: First, it is based on the observed partial information of each UE rather than global information. Second, it employs a multi-agent reinforcement learning approach, where each UE acts as an intelligent agent collaborating with others to achieve the goal. Third, it decomposes the computational offloading problem. The large state and action space in dynamic complex scenes require a computational offloading method based on hierarchical multi-agent reinforcement learning. The computational offloading problem is decomposed into two hierarchical sub-problems. The high-level subproblem is the offload location selection, which aims to control the task offload location of the UE, in order to improve MEC server resource utilization and preserve UE energy, thereby improving the reward sum in the medium and long term. The low-level subproblem is the control of the task offloading ratio of each user. To address this, we design a hierarchical double multi-agent proximal policy optimization (HDMAPPO) task offloading method, where the high level uses discrete MAPPO [11] to generate server location selection for each task and the low level uses continuous MAPPO to generate the task offloading ratio. Our proposed approach outperforms other algorithms. The main contribution of this paper can be summarized as follows:

- We design a multi-user multi-server MEC network with energy harvesting, and generate decisions on task offload location and task offload ratio considering the limited computing resources and battery power of UEs. We optimize the computation offloading decision to minimize the weighted sum of the average task delay, energy consumption, and task drop rate.
- We propose a computation offloading framework based on hierarchical multi-agent reinforcement learning to minimize system costs. The decision of task offloading location and ratio is optimized to minimize the weighted sum of the average task delay, energy consumption, and task drop rate, taking into account the limited computing resources and battery power of user devices. The problem is formulated as a MDP and solved using the HDMAPPO strategy, consisting of a high-level MAPPO algorithm and a low-level MAPPO algorithm. The high-level MAPPO algorithm determines the MEC server location for task offloading and the active dropping of tasks, while the low-level MAPPO algorithm determines the task offloading ratios. The state space of

the low-level problem is restricted by the output of the high-level problem, with the results of the high-level problem being part of the state of the low-level problem, thus reducing the complexity of the state.

- Experimental results demonstrate the effectiveness of our proposed hierarchical multi-agent reinforcement learning based computational offloading framework in reducing the weighted sum of the average task delay, energy consumption, and task drop rate compared to other baseline algorithms.

The remainder of this paper is organized as follows. Section 2 reviews the related work. We describe the system model and formulate the problem in Section 3, including the system architecture and computation model. In Section 4, we propose a hierarchical multi-agent reinforcement learning based computational offloading framework to solve the formulated problem. Section 5 presents simulation results. Finally, our paper is concluded in Section 6.

2. Related Work

The optimization of offloading decisions is a widely researched topic in edge computing, with the goal of reducing latency and energy consumption in MEC systems. The use of reinforcement learning as a tool for generating offloading decisions has been widely studied, as it allows for the agent to interact with the MEC environment by making adjustments to offloading decisions based on changes in the environment's state and reward values. The authors of [12] studied the joint optimization problem of offloading decision and computing resource allocation in the time-varying environment with a single MEC server and multiple users, and proposed a Q-learning based computing offloading method to minimize the delay and energy consumption cost for all UEs. The authors in [13] studied the task offloading problem in a multi-CAPs (computational access points) edge computing environment and proposed a DQN-based offloading strategy that dynamically adjusts the offloading ratio of tasks based on the states of the CAPs and tasks, in order to balance the overall delay and energy consumption of the system. The authors in [14] researched the task offloading problem in a heterogeneous vehicular network, considering the dynamic channel changes caused by vehicle movement and the random task arrival condition in an edge computing environment. They proposed a computation offloading method based on DDPG, with the aim of minimizing the overall energy consumption and task delay of the system. The authors in [15] studied the task offloading and resource allocation problem in a multi-site MEC RIoT environment, and proposed a hybrid hierarchical reinforcement learning method consisting of DDQN and DDPG. DDQN is responsible for generating subcarrier allocation decisions, while DDPG is responsible for offloading ratio, power allocation, and computing resource allocation decisions. The proposed method effectively reduces the weighted sum of energy consumption and delay. The authors in [16] studied a computation offloading problem with multiple users competing for resources, aiming to minimize the delay and energy consumption. The authors proposed a computation offloading method based on DDPG, which determines the offloading location and ratio for the task MEC. The authors in [17] proposed an actor–critic based computation offloading algorithm that generates offloading decisions and resource allocation in an energy harvesting computation offloading environment to maximize the number of offloaded tasks. The authors in [18] propose a DDQN algorithm based on an attention mechanism to generate task offloading strategies composed of computation resource allocation and power allocation. The goal is to minimize task completion delay and energy consumption in the long term. The authors in [19] propose a DQN-based action decomposition algorithm which decomposes the action space recursively into multiple actions and generates the decisions for server selection, offloading, and collaboration with multiple agents, in order to minimize delay cost. The authors in [20] study the problem of computational offloading with task dependencies and propose a sequence-to-sequence based deep reinforcement learning method for generating offloading decisions with the aim of minimizing latency and energy consumption. The increasing number of users leads to the growth of the system's state and action spaces,

resulting in a heightened complexity of single-agent reinforcement learning methods. In order to alleviate this complexity, this paper proposes a task offloading approach based on multi-agent reinforcement learning, in which each user is trained as an independent agent.

Many studies use the framework of MARL to optimize offloading decisions for multiple users. MARL can solve optimization problems in complex environments through mutual cooperation between multiple intelligent agents. The authors in [21] investigated the resource management problem in a vehicular network aided by MEC and UAV. They proposed a resource allocation algorithm based on multi-agent deep deterministic policy gradient (MADDPG), which treats each MEC server as an agent and enables the agents to collaborate in generating spectrum and computing and storage resources to meet the requirements of latency-sensitive tasks. The authors in [22] study the task offloading problem of an energy harvesting multi-user MEC system and propose a multi-agent based AC algorithm to solve the problem, where each user is an agent and the agents collaborate with each other to generate offloading decisions. The objective is to minimize the execution time of the task. The authors in [23] study the problem of computing offloading and resource allocation in a MEC system with multiple users and multiple MEC servers. Given the large number of users, the random arrival of tasks, and the time-varying nature of the environment, the authors proposed a multi-agent MADDQN-based offloading method for determining task offloading ratios and resource allocation decisions, with the aim of minimizing the weighted sum of delay and bandwidth. The authors in [24] propose a hierarchical multi-intelligence reinforcement learning framework to solve the computational offloading problem by decomposing the problem into two subproblems, beamforming strategy and task allocation ratio, and solving them using the MADDPG and single DDPG algorithms, respectively, with the aim of maximizing energy efficiency. The authors in [25] study a joint optimization problem based on computational offloading and interference coordination for smart small cell networks, and propose a MADDPG-based offloading method, while adding the idea of federal machine learning to MADDPG in order to reduce computational complexity, so that the model parameters can be reused by multiple agents. The objective is to reduce the energy consumption effectively while satisfying the delay requirements. The authors in [26] study the task offloading problem for NOMA multi-user MEC systems, aiming to minimize the weighted sum of long-term power consumption and latency, and propose a computational offloading method for multi-agent reinforcement learning based on MADDPG, where individual intelligences use the same policy network to reduce the complexity of training. The authors of [27] studied a multi-UAV and multi-MEC cooperative edge computing system. By jointly optimizing UAV trajectory, task allocation decisions, and resource management, the goal is to minimize the weighted sum of delay and energy consumption. Considering the high-dimensional continuous action space, the authors proposed a multi-agent reinforcement learning based MATD3 computation offloading method. The authors of [28] proposed a computation offloading framework based on MADDPG for solving resource allocation problems in an integrated MEC network for terrestrial applications. The authors in [29] proposed a computation offloading decision-making method based on QMIX multi-agent reinforcement learning to solve the problem of offloading server selection and task offloading ratio allocation in computation offloading. In this method, the agents make decisions based on both local observation information and global state, effectively reducing latency and energy consumption costs. The authors in [15] study the computational offloading and resource allocation problem for railroad IoT edge computing and propose a hybrid deep reinforcement learning offloading method. The method is integrated by DDQN and DDPG so that a mixture of action discrete and continuous policies can be learned, with DDQN used to make subcarrier allocation decisions and DDPG used to make offload rate, power, and computational resource allocation, effectively reducing the execution time. The authors in [30] study computational offloading in single-MEC server and multi-server scenarios and propose a MADDPG-based computational offloading approach to generate decisions such as task scheduling, transfer power and CPU cycles to minimize energy consumption and latency. In this paper, we

investigate the problem of task offloading location selection and offloading ratio selection in computational offloading, and propose a hierarchical multi-agent reinforcement learning based computational offloading framework to solve this problem. We summarize the advantages and disadvantages of the proposed approach compared to existing work in Table 1.

Table 1. Comparison of Approaches.

Approaches	Advantages	Disadvantages
Single-agent reinforcement learning based methods [12,13,19]	Simple model and easy to implement algorithm.	The growing number of user devices results in an explosive action space, impeding algorithmic learning.
Multi-agent reinforcement learning based methods [22,25,30]	Centralized training for decentralized implementation, able to solve cooperation or competition problems.	Unstable learning processes may occur in complex scenarios.
The proposed approach	Decomposing problems reduces complexity and can solve cooperation problems.	Poor adaptability, requires manual decomposition of problems.

3. System Model

In this paper, we propose a design for a multi-user multi-server MEC system, as depicted in Figure 1. The system consists of n UEs and m MEC servers, denoted by $\mathcal{N} = \{1, 2, 3, \ldots, n\}$ and $\mathcal{M} = \{1, 2, \ldots, m\}$, respectively. The UEs are randomly placed within the wireless network and are equipped with a battery that can be charged through energy harvesting from the beamforming signal of the nearby MEC server's AP. The computing capacity of each UE is denoted by $\mathcal{F}_{ue} = \{f_1, f_2, f_3 \ldots, f_n\}$ and the transmission bandwidth is W. The MEC servers are deployed on the APs, providing both wireless access and computational resources to the UEs, with a processing power denoted as f^{mec}. The time in the model is discrete and modeled as a sequence of time intervals of length τ, $\mathcal{T} = \{t_1, t_2, \ldots, t_k\}$, k represents the number of time slots. Due to the limited energy and computational resources of individual UEs, each UE can offload some of the computational workload to one of the nearby MEC servers to be processed simultaneously with the UE's local processing. UEs can be envisioned as wirelessly powered wearable devices for monitoring and analyzing healthcare information.

In time slot t, the set of tasks of users in the system is represented by A_t, $A_t = \{a_1^t, a_2^t, \ldots, a_n^t\}$. Task a_i^t is composed of two attributes: data size and computation density. The task generated by the i-th UE at time slot t is denoted by a_i^t, which can be denoted as $a_i^t = \{s_i^t, w_i^t\} (i \in N)$; s_i^t represents the data size of the task and w_i^t represents the computational intensity, i.e., the number of CPU cycles required to process a bit. The maximum tolerated delay for task a_i^t is T_{max}, which means that the task must be executed within T_{max} or it will be discarded.

At time slot t, the decision of the task offload location is denoted as $L_i^t \in \{0, 1, 2 \ldots, m\}$. $L_i^t = 0$ represents the active discard of the task and $L_i^t = [1, 2 \ldots, m]$ represents the selection of the L_i^t-th MEC server. The offloading ratio of tasks is represented by $X_i^t \in [0, 1]$. If $X_i^t = 1$, it means that all parts of the task are executed locally at the UE and if $X_i^t = 0$ it means that all parts of the task are offloaded to the MEC server. If the value of X_i^t is within $[0, 1]$, the part of X_i^t will be processed locally and $(1 - X_i^t)$ will be offloaded to the specified edge server.

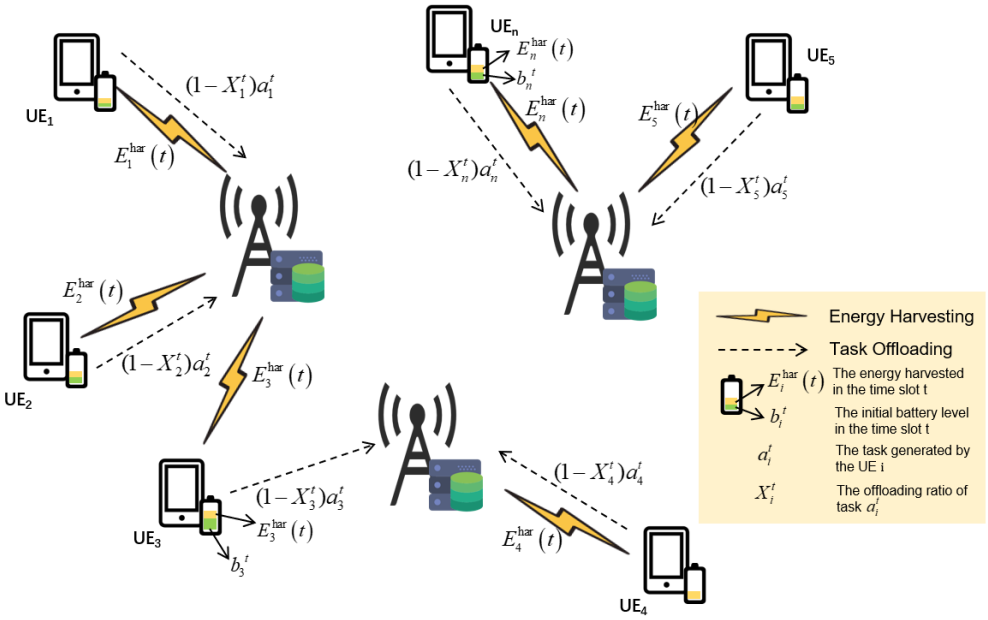


Figure 1. System model.

3.1. Local Computing Mode

The performance of the local computation, which refers to the execution of tasks by the UE itself, is influenced by the CPU frequency. The energy consumption associated with local computation is also correlated with the CPU frequency and the computational intensity of the tasks. The delay of local computation for the task a_i^t in time slot t can be expressed as:

$$D_{i,local}^t = \frac{X_i^t s_i^t w_i^t}{f_i}. \tag{1}$$

where w_i^t represents the number of CPU cycles required to process a bit and f_i represents the CPU frequency of the UE. At time slot t, the energy consumption from local computation is also given by the following formula:

$$E_{i,local}^t = k_e X_i^t s_i^t w_i^t f_i^2. \tag{2}$$

k_e is the effective capacitance coefficient, which is determined by the structure of the chip [31]. The energy consumption per CPU cycle can be characterized by $k_e f_i^2$. The CPU frequency required for local execution is $X_i^t s_i^t w_i^t$.

3.2. Mobile Edge Execution Model

According to the offloading decision, in time slot t, $(1 - X_i^t)s_i^t$ of the task data will be transmitted to the MEC server for processing and the frequency of the MEC can be represented as $f_{L_i^t}^{mec}$. Therefore, the processing time of the task on the MEC server can be represented as:

$$D_{i,mecexe}^t = \frac{(1 - X_i^t)s_i^t w_i^t}{f_{L_i^t}^{mec}}. \tag{3}$$

The execution time of task offloading to the MEC server consists of three stages: task uploading, task execution, and result return. In this paper, we assume that the result of the task execution is only a few characters, and the size of the task execution result is much smaller than the task data size, so the transmission delay of the result is ignored. The upload rate of the user device can be expressed as:

$$R_{i,L_i^t}^t = W \log_2(1 + \frac{P_i h_i^t}{I}). \tag{4}$$

where W is the bandwidth of the channel, P_i represents the transmission power of the UE, I represents the average interference power, and h_i^t represents the average channel gain. h_i^t can be expressed as:

$$h_i^t = A_d(\frac{3 \cdot 10^8}{4\pi f_c d_{i,L_i^t}})^{d_e}, \tag{5}$$

where d_{i,L_i^t} is the distance between UE and MEC server, A_d is the signal gain, f_c represents the carrier frequency, and d_e denotes the path loss. Then the transmission time of the task a_i^t can be calculated as:

$$D_{i,trans}^t = \frac{(1 - X_i^t)s_i^t}{R_{i,L_i^t}^t}. \tag{6}$$

Therefore the total execution time for task offloading is:

$$D_{i,mec}^t = D_{i,mecexe}^t + D_{i,trans}^t. \tag{7}$$

If the execution time of task a_i^t exceeds the length of time slot, it will be discarded. If the task is offloaded by the UE to the MEC server, the energy consumption is influenced by the transmission power and the time required for task transmission. This can be represented as:

$$E_{i,mec}^t = P_i D_{i,trans}^t. \tag{8}$$

3.3. Energy Harvesting

In time slot t, the energy obtained by the UE through wireless power transfer is represented by $E_i^{har}(t)$ and the total energy consumption is represented by $E_i^{tot}(t)$. The initial battery level at the beginning of the time slot can be represented by b_i^t and the remaining battery level of the UE can be expressed as:

$$b_i^{t+1} = max(0, b_i^t - E_i^{tot}(t) + E_i^{har}(t)). \tag{9}$$

If there is not enough power to complete the task, the task will be discarded and the remaining power set to

$$b_i^{t+1} = 0. \tag{10}$$

The energy collected in the time slot t can be expressed as

$$E_i^{har}(t) = v\eta(d_{i,L_i^t})^{-\xi}G. \tag{11}$$

where v represents the energy conversion efficiency, η represents the transmission power, ξ represents the path loss exponent, and G represents the combined gain of the frequency-radiating energy transmitter antenna and the user device antenna.

The task a_i^t execution time is the maximum of the local execution delay and the offload execution delay. Energy consumption is the total of the local execution and task offloading consumption.

$$D_i^t = max(D_{i,local}^t, D_{i,mec}^t). \tag{12}$$

$$E_i^t = E_{i,local}^t + E_{i,mec}^t. \tag{13}$$

If the task execution time exceeds the size of the time slot or the total energy consumption exceeds the remaining battery power, the task will be discarded.

3.4. Problem Formulation

In this paper, we study a MEC computation offloading problem based on energy harvesting, with the aim of minimizing the weighted sum of the average task delay, energy consumption, and task drop rate. The cost loss is composed of three parts: delay, energy consumption, and punishment for task drop, where punishment is used to describe the

scale of the system's task dropout rate. α, β, and ϕ represent the weights of latency, energy consumption, and task discard rate, respectively, The cost is formulated as follows:

$$\begin{aligned}
&\min_{L^t, X^t} \lim_{\tau \to \infty} \frac{1}{\tau} \sum_{t=0}^{\tau} [\alpha \frac{\sum_{i=1}^{N} D_i^t}{N} + \beta \frac{\sum_{i=1}^{N} E_i^t}{N} + \phi dropout_t] \\
&\text{s.t.} \\
&\text{C1} : L_i^t \in \{0, 1, 2, \ldots, m\}, \forall i \in N, \\
&\text{C2} : 0 \leq x_i^t \leq 1, \forall i \in N, \\
&\text{C3} : T_i^t < T_{max}, \forall i \in N, \\
&\text{C4} : E_i^{tot}(t) < b_i^t + E_i^{har}(t),
\end{aligned} \tag{14}$$

In the set of constraints, C1 indicates that each UE can only actively discard or select one of the MEC servers for offloading. C2 indicates that the task can be partially offloaded to the MEC server. C3 indicates that the delay of the task needs to be less than the maximum tolerable delay of the task. C4 indicates that the energy consumed by the current time slot execution needs to be less than the sum of the remaining energy of the previous time slot and the energy harvested by the current time slot.

The above optimization problem can be solved by optimizing the task offload location and task offload ratio decisions. The objective is to minimize the average task latency, energy consumption, and task discard rate in the system while satisfying the constraints. Since the complexity of the problem increases with the number of system UEs, this paper proposes a hierarchical multi-agent reinforcement learning-based computational offloading strategy to reduce complexity. To achieve this, the problem is first decomposed into two subproblems: a high-level task offloading location selection problem and a low-level task offloading ratio selection problem. Then, a MAPPO-based computational offloading method is proposed to solve these subproblems.

4. Computation Offloading Decision Strategy

The state and action spaces of the system are prone to growing in size due to the complexity and rapidly changing nature of the environment, as well as the large number of UEs. To overcome this challenge, this paper proposes a hierarchical multi-agent reinforcement learning based algorithm framework that decomposes the computation offloading problem into two sub-problems: server location selection and task offloading ratio. Each of these sub-problems is then separately solved by a distinct multi-agent reinforcement learning algorithm, as shown in Figure 2.

4.1. MAPPO Algorithm

The MAPPO is a variant of the PPO algorithm that has been adapted for use with multiple agents. PPO is a policy optimization algorithm that utilizes a stochastic actor–critic architecture. The strategy network, represented by $\pi_\theta(a_t|o_t)$, outputs the probability distribution of action a_t given the state observation o_t. The actions are then drawn from this distribution, resulting in a randomized process that enables exploration of the environment. This feature of PPO provides increased flexibility in determining the appropriate action in a given state.

The actor network, which approximates the policy, is represented by π_θ and the critic network, which approximates the value function, is represented by V_ω. The parameters of the actor network are denoted by θ, while the parameters of the critic network are denoted by ω. The discounted expected future reward is given by:

$$L(\pi_\theta) = \mathbb{E}_{s_0, s_1, \ldots} \left[\sum_{t=0}^{k} \gamma^t r_t \right]. \tag{15}$$

The value function $V_{\pi_\theta}(s)$, the state–action value function $Q_{\pi_\theta}(s,a)$, and the advantage function $A_{\pi_\theta}(s,a)$ are denoted as:

$$V_{\pi_\theta}(s) = \mathbb{E}_{a_t, s_{1+1}, \ldots}\left[\sum_{i=t}^{k} \gamma^{i-t} r_i | s_t = s\right]. \tag{16}$$

$$Q_{\pi_\theta}(s,a) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\left[\sum_{i=t}^{k} \gamma^{i-t} r_i | s_t = s, a_t = a\right]. \tag{17}$$

$$A_{\pi_\theta}(s,a) = Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s). \tag{18}$$

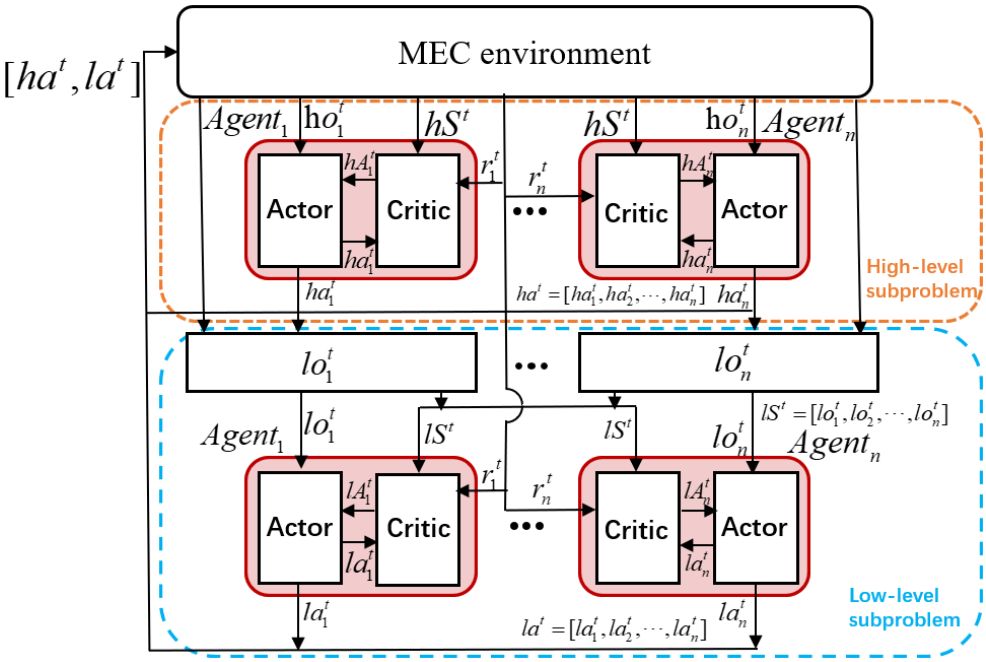


Figure 2. The HDMAPPO algorithm framework.

The PPO algorithm utilizes an actor–critic architecture to optimize the policy network, with the goal of maximizing the long-term rewards. The algorithm works by taking samples of interactions with the environment and adjusting the probabilities of actions based on the magnitude of the rewards received from the environment. The calculation of the policy gradient can be formulated as follows:

$$\begin{aligned} \nabla_\theta L(\pi_\theta) &= \mathbb{E}_{s\sim\rho_{\pi_\theta}, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A_{\pi_\theta}(s,a)] \\ &= \mathbb{E}_{s\sim\rho_{\pi_{\hat{\theta}}}, a\sim\pi_{\hat{\theta}}}\left[f_\theta \nabla_\theta \log \pi_\theta(s,a) A_{\pi_{\hat{\theta}}}(s,a)\right] \end{aligned} \tag{19}$$

$$f_\theta = \frac{\pi_\theta(a|s)}{\pi_{\hat{\theta}}(a|s)} \tag{20}$$

$\hat{\theta}$ is the vector of policy parameters before updating. In order to accelerate the convergence of the policy, PPO clips the policy gradient as:

$$\nabla_\theta L(\pi_\theta) = \mathbb{E}_{s\sim\rho_{\pi_{\hat{\theta}}}, a\sim\pi_{\hat{\theta}}}[\nabla_\theta \log \pi_\theta(s,a) C(s,a)] \tag{21}$$

$$C(s,a) = \min\left[f_\theta A_{\pi_{\hat{\theta}}}(s,a), clip(f_\theta, 1-\varepsilon, 1+\varepsilon) A_{\pi_{\hat{\theta}}}(s,a)\right] \tag{22}$$

where ε is a hyperparameter with a value of 0.2 [11] and the function clip (f_θ, $1-\varepsilon$, $1+\varepsilon$) operates to restrict the value of f_θ to remain within the interval $[1-\varepsilon, 1+\varepsilon]$. The approach adopted in this work involves taking the minimum value between the clipped

and unclipped targets, with the objective of preserving the original, unclipped target as the lower bound of the final target (pessimistic bound). The loss function for updating the critic is defined as:

$$\nabla_{\omega} L(V_{\omega}) = \mathbb{E}_{s\sim\rho_{\pi_\theta}}\left[\mathbb{E}_{s',a}\left[r_t + V_{\omega}(s')\right] - V_{\omega}(s)\right]^2 \tag{23}$$

where s' is the next state.

The MAPPO algorithm operates in a centralized training and decentralized execution manner. During the decentralized execution phase, each agent derives its actions based on its private state observations o_i^t. On the other hand, during the centralized training phase, the critic network aggregates global states S^t and global actions A^t to form a Q-value network that optimizes the reward function. The global states are denoted as $S^t = \{o_1^t, o_2^t, \ldots, o_n^t\}$ and global actions are denoted as $A^t = \{a_1^t, a_2^t, \ldots, a_n^t\}$. Each agent has its own actor network and critic network.

The gradient of actor i can be calculated as:

$$\nabla\theta_i = \frac{1}{Bn}\sum_{j=1}^{B}\sum_{k=1}^{n}\nabla_{\theta_i}\min\left[\frac{\pi_{\theta_i}(a_k^t|o_k^t)}{\pi_{\hat{\theta}_i}(a_k^t|o_k^t)}\hat{A}_i(t), clip(\frac{\pi_{\theta_i}(a_k^t|o_k^t)}{\pi_{\hat{\theta}_i}(a_k^t|o_k^t)}, 1-\varepsilon, 1+\varepsilon)\hat{A}_i(t)\right] \tag{24}$$

where $\hat{A}_i(t)$ is the estimation of the advantage function, which is defined as follows:

$$\hat{A}_i(t) = \sum_{l=0}^{\infty}(\gamma\lambda)^l(r_i(t+l) + \gamma V_i(S(t+1+l)) - V_i(S(t+l))) \tag{25}$$

where γ is the discount factor to balance the importance of current and future rewards, λ is the parameter of GAE for bias–variance tradeoff in estimation, and $V_i(S(t)) = \sum_{l=0}^{\infty}\gamma^l r_i(t+l)$ is the cumulative discounted reward, which also represents the state–value function estimated by the critic of agent i as V^{ω_i}(S(t)). The loss of critic i is given by:

$$\nabla\omega_i = \frac{1}{2}[V^{\omega_i}(S(t)) - V_i(S(t))]^2 \tag{26}$$

The algorithmic framework of MAPPO is given by Algorithm 1.

Algorithm 1 MAPPO for subproblems

Initialize Initialize actor networks θ_i, critic networks ω_i, replay buffer D;
for *episode* $= 1, 2, \ldots, E$ **do**
  Observe the initialization state $S(0) = \{o_1^t, o_2^t, \ldots, o_n^t\}$
  for $t = 1, 2, \ldots, T$ **do**
    for $n_i = 1, 2, \ldots, N$ **do**
      $a_i^t \sim \pi_{\theta_i}(a_i^t|o_i^t)$.
      Execution a_i^t get reward r_i^t and next state o_i^{t+1}.
    Store the trajectory $\tau = \{o_t, a_t, s_t, r_t, o_{t+1}, s_{t+1}\}$ to the buffer D.
    Compute advantage estimate $\hat{A}$ by (25).
  for $k = 1, 2, \ldots, K$ **do**
    Shuffle and renumber the data's order.
    for $j = 1, 2, \ldots, \frac{T}{B}$ **do**
      randomly choose B group data from D.
      for $n_i = 1, 2, \ldots, N$ **do**
        Update the actor and critic network parameters according to (23) and (26).
  Update $\hat{\theta} \leftarrow \theta$ and $\hat{\omega} \leftarrow \omega$ for each UE.
  Clear the replay buffer D.

4.2. Offloading Location Selection Strategy

The offloading location selection problem is a discrete problem where the goal of each agent aligns with the global goal. The high-level problem requires considering the states of multiple agents, and the size of both state space and action space increases exponentially with the number of UEs, thus creating the problem of curse of dimensionality. To solve the above problem the MAPPO algorithm is applied and the strategy is referred to as the high-MARL (HMARL) strategy. The HMARL strategy is designed such that each agent generates generative actions from its private states and updates the network parameters with the global states. The detailed design of the HMARL is listed as follows.

- States: The states consist of the system information observed by each agent at the beginning of each time slot. The UE can observe the agent's id n_i, the amount of data s_i^t, and the computational density w_i^t of task a_i^t as well as the remaining power b_i^t and the collected energy $E_i^{har}(t)$ of the UE in the current time slot. In addition information about the processing power of the MEC $F^{mec} = [f_1^{mec}, f_2^{mec}, \ldots, f_m^{mec}]$ is needed. Therefore, the observation of UE i at step t can be expressed as:

$$ho_i^t = \left\{ n_i, s_i^t, w_i^t, b_i^t, E_i^{har}(t), F^{mec} \right\} \tag{27}$$

- Action: In the offloading location selection problem, the action space is represented by ha_i^t, which has a range of $[0, m]$. A value of 0 indicates that the task should be directly discarded and no MEC server is selected for offloading, whereas values in the range of $[1, m]$ denote the identifier of the selected MEC server. The offloading ratio of the task is resolved in the lower-level subproblem, therefore the action space does not encompass the offloading ratio in this context.

$$ha_i^t = \left\{ L_i^t \right\} \tag{28}$$

- Reward: The reward function is designed to reflect the efficiency of the actions executed in the environment and must be consistent with the system design for optimal results. The reward function is designed to consider the task delay, energy consumption, and task drop rate, and can be formulated as follows:

$$r_i^t = \begin{cases} -penalt & D_i^t > T_{max} \quad or \quad ha_i^t = 0 \\ -(\alpha D_i^t + \beta E_i^t + \phi dropout_t) & else \end{cases} \tag{29}$$

The action of the higher-level subproblem will be treated as part of the state of the lower-level subproblem.

4.3. Task Offload Ratio Strategy

The low-level subproblem is the task offloading ratio problem. The algorithm treats each UE as an individual agent and makes offloading decisions based on local observations of the system and high-level subproblem action. The decision is to assign the ratio of tasks to be executed locally and on the server, and the assignment aims to reduce the task delay, energy consumption, and task drop rate. To improve the accuracy of the task assignment rate, the offload rate will select actions from a continuous action space, so MAPPO is still chosen as the low-level algorithm and the strategy is called low-MARL (LMARL).

- States: The state includes the decision of the HMARL and the local information observed by the agent, which is denoted as:

$$lo_i^t = \left\{ n_i, ha_i^t, s_i^t, w_i^t, f_i, b_t, E_t^{har}, p_i, F_{ha_i^t}^{mec}, h_i^t \right\} \tag{30}$$

where f_i is the the computing capacity of the UE i; p_i is the is the transmitted power of the UE; and h_i^t and ha_i^t are the channel gain and high-level subproblem action.

- Action: The action is to determine the task offloading ratio after determining the task offload server location. The range of the action is a continuous range of $[0, 1]$, X_i^t represents the proportion of task executed locally, and $1 - X_i^t$ represents the proportion of MEC offloads. The action of the low-level subproblem can be expressed as:

$$la_i^t = \{X_i^t\} \tag{31}$$

- Reward: The purpose of the low-level algorithm is to reduce the task latency, energy consumption, and task discard rate, which is consistent with the high-level algorithm, so the reward function of the low-level algorithm is the same as the reward function of the high-level algorithm. It is calculated by (29).

4.4. HDMAPPO Framework

In each time slot, the task computational offloading is decomposed into two subproblems: MEC server selection and task offloading ratio decision. HMARL is used in a high-level subproblem to solve the MEC server selection problem for task offloading and LMARL is used in the low-level subproblem to make the decision on the task offloading ratio. Both HMARL and LMARL are based on MAPPO. The HMARL starts at the beginning of each time slot, where each agent generates the action ha_i^t based on its observed private state. The LMARL, on the other hand, generates the continuous task offloading ratio la_i^t based on the observed information and the action ha_i^t obtained from the HMARL. The generated actions are executed in the environment and rewards are obtained as a result. MAPPO has the advantage of decentralized execution and centralized training, with the tracks of the high-level and low-level algorithms stored in their respective buffers as empirical samples for training. The detailed process of HDMAPPO is shown in Algorithm 2.

Algorithm 2 Proposed HDMAPPO algorithm

Initialize Initialize HMARL's actor networks $h\theta_i$, critic networks $h\omega_i$, replay buffer hD;
Initialize Initialize LMARL's actor networks $l\theta_i$, critic networks $l\omega_i$, replay buffer lD;
for $episode = 1, 2, \ldots, E$ **do**
  Observe the initialization state $hS_1 = \{ho_1^1, ho_2^1, \ldots, ho_n^1\}$
  for $t = 1, 2, \ldots, T$ **do**
    for $n_i = 1, 2, \ldots, N$ **do**
      $L_i^t \sim \pi_{h\theta_i}(a_i^t | ho_i^t)$ and $X_i^t \sim \pi_{l\theta_i}(a_i^t | lo_i^t)$.
    $L_t = \{L_1^t, L_2^t, \ldots, L_n^t\}$ and $X_t = \{X_1^t, X_2^t, \ldots, X_n^t\}$
    Execution L_t, X_t get reward r_t and next state $hS_{t+1} = \{ho_1^{t+1}, ho_2^{t+1}, \ldots, ho_n^{t+1}\}$.
    Store the trajectory $h\tau = \{ho_t, L_t, hS_t, r_t, ho_{t+1}, hS_{t+1}\}$ to the buffer hD.
    Store the trajectory $l\tau = \{lo_t, X_t, lS_t, r_t, lo_{t+1}, lS_{t+1}\}$ to the buffer lD.
    Compute advantage estimate $\hat{A}_h$ and $\hat{A}_l$ by (25) .
  for $k = 1, 2, \ldots, K$ **do**
    Shuffle and renumber the data's order.
    for $j = 1, 2, \ldots, \frac{T}{B}$ **do**
      randomly choose B_h group data from hD.
      randomly choose B_l group data from lD.
      for $n_i = 1, 2, \ldots, N$ **do**
        Update the actor parameters $h\theta_i$ and $l\theta_i$ according to (23).
        Update the critic parameters $h\omega_i$ and $l\omega_i$ according to (26).
  Update HMARL's parameters $h\hat{\theta} \leftarrow h\theta$ and $h\hat{\omega} \leftarrow h\omega$ for each UE.
  Update LMARL's parameters $l\hat{\theta} \leftarrow l\theta$ and $l\hat{\omega} \leftarrow l\omega$ for each UE.
  Clear the replay buffer hD and lD.

5. Performance Evaluation

In this section, we evaluate the performance of the proposed HDMAPPO algorithm. The simulation settings and numerical results are presented as follows.

5.1. Simulation Setup

We consider a multi-user, multi-MEC server edge computing network environment with three APs, each equipped with an MEC server. The UEs are randomly distributed within the coverage area of the APs and have varying computing capabilities, uniformly distributed within a range of $[1, 2]$ GHz. The CPU frequency of the edge server can be expressed as f^{mec}, uniformly distributed at $[2, 3]$ GHz. The system time is divided into discrete time slots and, in each time slot, each UE generates computational tasks with data sizes in the range of $[300, 500]$ Kbits and computational densities in the range of $[800, 1200]$ cycles/bit. The energy $E_i^{har}(t)$ collected by the UE at the beginning of the time slot is uniformly distributed in $[50, 150]mj$. The key system parameters are listed in Table 2.

Table 2. Simulation parameter settings.

Parameters	Value
CPU frequency of MEC server f^{mec}	$[2, 3]$ GHz
CPU frequency of UE f_i	$[1, 2]$ GHz
Bandwidth of channel W	10 MHz
Average interference power I	2×10^{-27} W
Average transmit power of UE P_i	0.5 W
The data size of the task s_i^t	$[300, 500]$ Kbits
Computational density of the task w_i^t	$[800, 1200]$ cycles/bit
The maximum tolerant delay of task T_{max}	1 s

In the HDMAPPO algorithm each UE has two policies for solving the subproblems of task offload location selection and task offload ratio, respectively. Each policy has an actor network and a critic network, and the parameters of actor network and critic network are listed in Table 3.

Table 3. Hyperparameters of HDMAPPO.

Component	Network Structure	Hyperparameter	Value
Actor	fc(state_dim,128),tanh	Learning rate of actor	0.0003
	fc(128,128),tanh	Learning rate of critic	0.0004
	fc(128,action_dim),softplus	Reward discount	0.99
Critic	fc(state_dim,128),tanh	Optimizer	Adam
	fc(128,128),tanh	K_epochs	10
	fc(128,1)	Clip_rate	0.2

We use latency, energy consumption, and task discard rate as judging metrics [7]; the task discard rate to some extent reflects the quality of service of the system. The proposed method is compared with the following benchmark algorithm to prove the simulation results.

- All Local (**AL**): All parts of the task are executed locally at UE.
- All MEC (**AM**): All parts of the task are randomly offloaded to one of the MEC servers.
- Random task offloading (**RTO**): Partial scale tasks will be randomly offloaded to one of the MEC servers and the percentage of tasks offloaded is random.
- Independent proximal policy optimization offloading (**IPPO**): The IPPO [32] algorithm considers each UE as a separate agent, with no direct interdependence between the individual agents. Each agent independently executes the PPO algorithm and generates both server selection and task offloading ratio decisions.
- Multi-agent deep deterministic policy gradient offloading (**MADDPG**): MADDPG [33] extends deep deterministic policy gradient (DDPG) to a multi-agent environment by

introducing a framework of centralized training and decentralized execution. Each UE individually trains an actor network to generate actions based on local private information and a centralized critic network is used to update policy parameters based on global information.

5.2. Performance Comparison

Figure 3 displays the performance of the reward function as the number of training iterations increases. In a multi-user multi-server edge computing network environment with 30 UEs and 3 MEC servers, HDMAPPO, IPPO, and MADDPG all converge to a stable reward value as the number of training rounds increase. The algorithm plateaus after 5000, 6000, and 8000 iterations for HDMAPPO, IPPO, and MADDPG, respectively. Our proposed method exhibits faster convergence and a higher reward value compared to the other algorithms. MADDPG converges the fastest, but with a lower reward value compared to HDMAPPO and IPPO. IPPO has a reward value close to that of the proposed method, but a slower convergence rate.

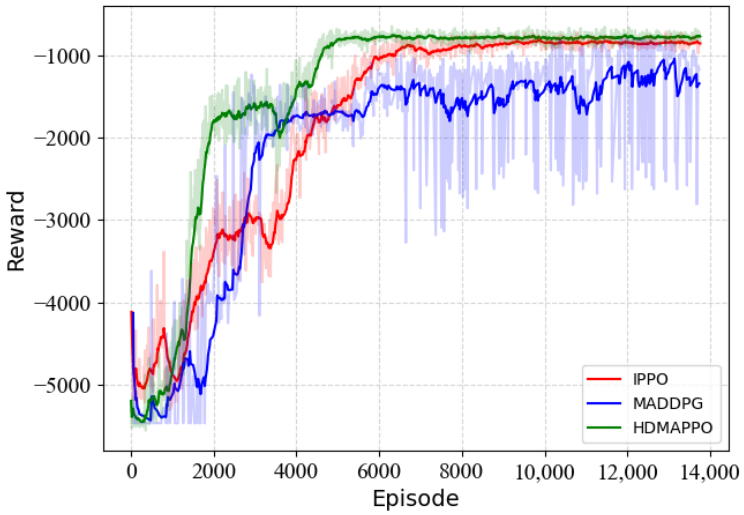
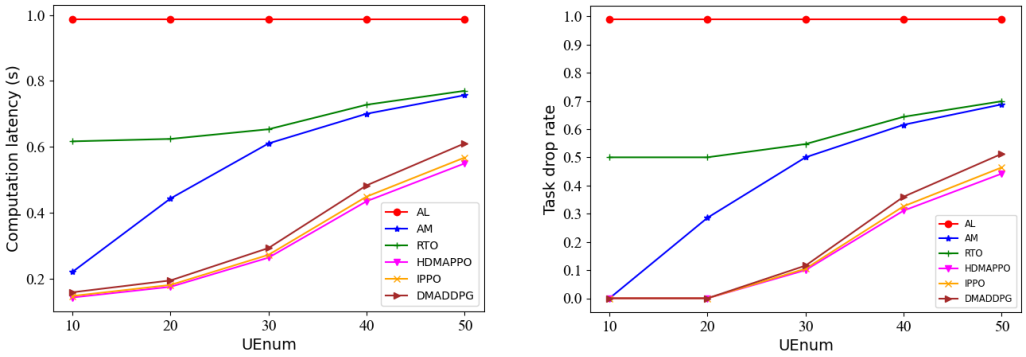


Figure 3. Convergence performance of reinforcement learning-based offloading strategies.

Figure 4 shows the trend of latency and task discard rate of the proposed algorithm and other baseline algorithms as the number of UEs increases, respectively. The increase in the number of UEs implies a more complex system environment, while the total number of edge server resources remains unchanged, leading to intense competition for resources among UEs. It is apparent that, as the number of UEs increases, the latency and task drop rate increase for all strategies except for AL. This is because the resources of the MEC server are limited and cannot accommodate the increased number of UEs. The three reinforcement learning-based computational offloading strategies, IPPO, MADDPG, and HDMAPPO, all outperform the other strategies, indicating that the DRL algorithm is able to optimize the system's latency. When the number of UEs in the system is 30, HDMAPPO reduces latency by 4.8%, 9.7%, 56.9%, 59.7%, and 73.3% compared to IPPO, MADDPG, AM, RTO, and AL. The task discard rate is reduced by 5.4%, 10.2%, 80%, 81.5%, and 89.8% compared to the same algorithms. This shows that the proposed strategy is effective in reducing the latency and task discard rate when the number of UEs increases.

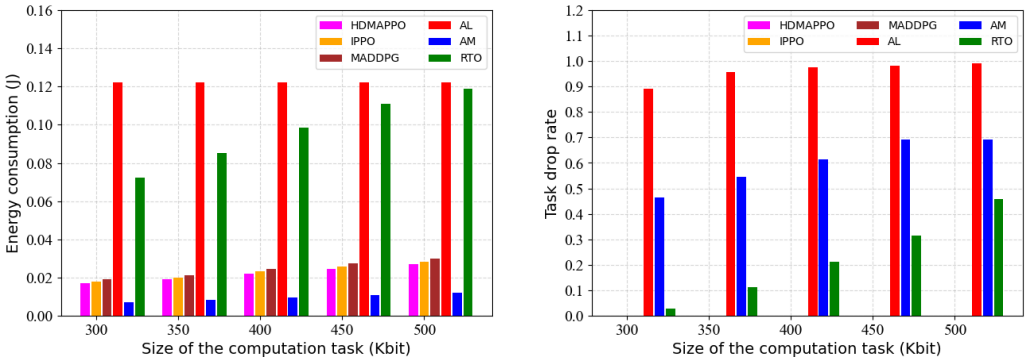
Figure 5 shows the performance of the proposed algorithm in terms of reducing energy consumption. The increase in task data size results in an increase in computational workload and necessitates balancing energy consumption between local computation and computation offloading. By increasing the data size of the task, the average energy consumption of the task for the computational offloading strategy is examined. In the system environment where the number of UEs is 40 and the computational power of MEC servers is normally distributed at 5 GHz, the computational density of tasks is

incremented in the range of [300, 500] kbits. It is obvious that the energy consumption of all policies except AL increases with the size of the task data. The energy consumption and task drop rate of the reinforcement learning-based approach are much better than the other three strategies. The low energy consumption of the AM is because the energy consumption of communication transmission is much lower than the energy consumption of computing tasks locally, but the computational resources of the MEC server cannot meet all computational demands, so there will be a higher discard rate. AL leads to high energy consumption and task drop rate due to insufficient computational resources of UEs. Three reinforcement learning-based strategies are able to reduce task energy consumption without discarding tasks, where HDMAPPO reduces energy consumption by 6.41% and 11.2% compared to IPPO and MADDPG, respectively, for the task data volume of 400 kbits.



(**a**) The average latency of UEs (**b**) The average task drop rate of UEs

Figure 4. Impact of the number of UEs on the average task latency and task drop rate.



(**a**) The average energy consumption of UEs (**b**) The average task drop rate of UEs

Figure 5. Impact of the data size of a task on the average latency of the task and the task dropout rate.

Figure 6 shows the trend of the reward value of each offloading strategy in the system as the computational density of the task increases. The increase in computational density results in an increased computational load for the task, which makes it challenging for the UE to complete the task within the maximum task tolerance time alone and causes an increase in the task discard rate. The computational density of the task increases from within [800, 1500] cycles/bit. It can be observe that the reward of all offloading strategies tends to decrease as the computational density of the task increases. This is due to the limited computational resources and the increase in computational density leading to a larger computational load, resulting in lower reward values. AL has the lowest reward value and the degree of change is not significant, because the UE's computational power and energy consumption makes it difficult to complete the task alone, and the drop rate of the task is high at low task computational density, which leads to a small change in reward value despite the change in computational density. The change in reward value of the three reinforcement learning-based strategies first remains relatively stable as the computational

density increases and then increases rapidly. This is because computational resources are limited and, when the computational load is low, the task can be satisfied by decision optimization but, when the load is higher than the amount of resources, only part of the computational demand of the task can be satisfied resulting in a decrease of the reward value. It is obvious that the proposed strategy is always better than the other strategies. The proposed strategy improves the reward value by 6.32% and 11.6% compared to IPPO and MADDPG for a computational density of 1300 cycles/bit of the task.

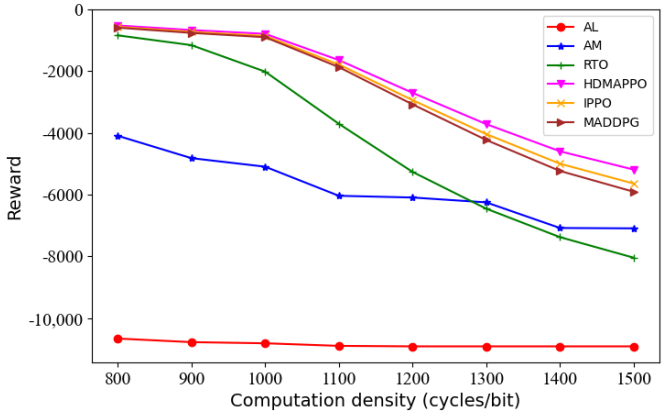


Figure 6. Impact of the calculation density of the task on the system's rewards.

6. Conclusions

In this paper, we investigate the computational offloading problem in MEC networks with energy harvesting, with the aim of optimizing the task offload location selection and task offload ratio to minimize the system latency, energy consumption, and task drop rate. To mitigate the issue of an oversized state space resulting from dynamic scenarios and a large number of UEs in MEC system, we propose a computational offloading strategy HDMAPPO based on multi-agent reinforcement learning. The strategy decomposes the offloading problem into two subproblems, i.e., task offloading selection and task offloading ratio allocation, and solves each subproblem using a multi-agent reinforcement learning algorithm. This decoupled approach reduces the complexity of the problem. The numerical results demonstrate that the proposed method significantly reduces latency, energy consumption, and task drop rate compared to other benchmarks. Our proposed approach has some limitations; it cannot be adapted to all edge computing scenarios. The state information and actions need to be adjusted for different kinds of edge computing scenarios. In the future, we will investigate the application of multi-agent reinforcement learning to the task offloading problem in vehicular edge computing.

Author Contributions: Conceptualization, Y.S. and Q.H.; methodology, Q.H.; validation, Q.H.; formal analysis, Y.S. and Q.H.; investigation, Q.H.; resources, Q.H.; data curation, Q.H.; writing—original draft preparation, Q.H.; writing—review and editing, Q.H. and Y.S.; visualization, Q.H.; project administration, Y.S.; funding acquisition, Y.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the National Natural Science Foundation of China (Grant Nos. 61763002 and 62072124), Guangxi Major projects of science and technology (Grants No. 2020AA21077007).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

Abbreviations

The following abbreviations are used in this manuscript:

MEC	Multi-access edge computing
AP	Access points
MARL	Multi-agent reinforcement learning
MAPPO	Multi-agent proximal policy optimization
HDMAPPO	Hierarchical double multi-agent proximal policy
WPT	Wireless power transfer
EH	Energy harvesting
IoT	Internet of Things
MDP	Markov decision process
MADDPG	Multi-agent deep deterministic policy gradient
AL	All local
AM	All MEC
RTO	Random task offloading
IPPO	Independent proximal policy optimization offloading

References

1. Mach, P.; Becvar, Z. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
2. Zhao, Y.; Hou, F.; Lin, B.; Sun, Y. Joint Offloading and Resource Allocation with Diverse Battery Level Consideration in MEC System. *IEEE Trans. Green Commun. Netw.* **2023**. [CrossRef]
3. Guo, S.; Liu, J.; Yang, Y.; Xiao, B.; Li, Z. Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing. *IEEE Trans. Mob. Comput.* **2018**, *18*, 319–333. [CrossRef]
4. Yi, C.; Cai, J.; Su, Z. A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications. *IEEE Trans. Mob. Comput.* **2019**, *19*, 29–43. [CrossRef]
5. Kumar, K.; Liu, J.; Lu, Y.H.; Bhargava, B. A survey of computation offloading for mobile systems. *Mob. Netw. Appl.* **2013**, *18*, 129–140. [CrossRef]
6. Lin, H.; Zeadally, S.; Chen, Z.; Labiod, H.; Wang, L. A survey on computation offloading modeling for edge computing. *J. Netw. Comput. Appl.* **2020**, *169*, 102781. [CrossRef]
7. Min, M.; Xiao, L.; Chen, Y.; Cheng, P.; Wu, D.; Zhuang, W. Learning-based computation offloading for IoT devices with energy harvesting. *IEEE Trans. Veh. Technol.* **2019**, *68*, 1930–1941. [CrossRef]
8. Choi, K.W.; Aziz, A.A.; Setiawan, D.; Tran, N.M.; Ginting, L.; Kim, D.I. Distributed wireless power transfer system for Internet of Things devices. *IEEE Internet Things J.* **2018**, *5*, 2657–2671. [CrossRef]
9. Zaman, S.K.U.; Jehangiri, A.I.; Maqsood, T.; Umar, A.I.; Khan, M.A.; Jhanjhi, N.Z.; Shorfuzzaman, M.; Masud, M. COME-UP: Computation offloading in mobile edge computing with LSTM based user direction prediction. *Appl. Sci.* **2022**, *12*, 3312. [CrossRef]
10. Zaman, S.K.u.; Jehangiri, A.I.; Maqsood, T.; Haq, N.u.; Umar, A.I.; Shuja, J.; Ahmad, Z.; Dhaou, I.B.; Alsharekh, M.F. LiMPO: Lightweight mobility prediction and offloading framework using machine learning for mobile edge computing. *Clust. Comput.* **2022**, *26*, 99–117. [CrossRef]
11. 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.
12. 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), IEEE, Barcelona, Spain, 15–18 April 2018; pp. 1–6.
13. Li, C.; Xia, J.; Liu, F.; Li, D.; Fan, L.; Karagiannidis, G.K.; Nallanathan, A. Dynamic offloading for multiuser muti-CAP MEC networks: A deep reinforcement learning approach. *IEEE Trans. Veh. Technol.* **2021**, *70*, 2922–2927. [CrossRef]
14. 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]
15. Xu, J.; Ai, B.; Chen, L.; Cui, Y.; Wang, N. Deep Reinforcement Learning for Computation and Communication Resource Allocation in Multiaccess MEC Assisted Railway IoT Networks. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 23797–23808. [CrossRef]
16. Qu, B.; Bai, Y.; Chu, Y.; Wang, L.e.; Yu, F.; Li, X. Resource allocation for MEC system with multi-users resource competition based on deep reinforcement learning approach. *Comput. Netw.* **2022**, *215*, 109181. [CrossRef]
17. Zhang, Z.; Yu, F.R.; Fu, F.; Yan, Q.; Wang, Z. Joint offloading and resource allocation in mobile edge computing systems: An actor-critic approach. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), IEEE, Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6.
18. Liu, T.; Zhang, Y.; Zhu, Y.; Tong, W.; Yang, Y. Online computation offloading and resource scheduling in mobile-edge computing. *IEEE Internet Things J.* **2021**, *8*, 6649–6664. [CrossRef]

19. Ho, T.M.; Nguyen, K.K. Joint server selection, cooperative offloading and handover in multi-access edge computing wireless network: A deep reinforcement learning approach. *IEEE Trans. Mob. Comput.* **2020**, *21*, 2421–2435. [CrossRef]
20. Wang, J.; Hu, J.; Min, G.; Zhan, W.; Zomaya, A.Y.; Georgalas, N. Dependent task offloading for edge computing based on deep reinforcement learning. *IEEE Trans. Comput.* **2021**, *71*, 2449–2461. [CrossRef]
21. Peng, H.; Shen, X. Multi-agent reinforcement learning based resource management in MEC-and UAV-assisted vehicular networks. *IEEE J. Sel. Areas Commun.* **2020**, *39*, 131–141. [CrossRef]
22. Liu, C.; Tang, F.; Hu, Y.; Li, K.; Tang, Z.; Li, K. Distributed task migration optimization in MEC by extending multi-agent deep reinforcement learning approach. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 1603–1614. [CrossRef]
23. Ke, H.; Wang, H.; Sun, H. Multi-Agent Deep Reinforcement Learning-Based Partial Task Offloading and Resource Allocation in Edge Computing Environment. *Electronics* **2022**, *11*, 2394. [CrossRef]
24. Zhou, H.; Long, Y.; Gong, S.; Zhu, K.; Hoang, D.T.; Niyato, D. Hierarchical Multi-Agent Deep Reinforcement Learning for Energy-Efficient Hybrid Computation Offloading. *IEEE Trans. Veh. Technol.* **2022**, *72*, 986–1001. [CrossRef]
25. Huang, X.; Leng, S.; Maharjan, S.; Zhang, Y. Multi-agent deep reinforcement learning for computation offloading and interference coordination in small cell networks. *IEEE Trans. Veh. Technol.* **2021**, *70*, 9282–9293. [CrossRef]
26. Chen, Z.; Zhang, L.; Pei, Y.; Jiang, C.; Yin, L. NOMA-based multi-user mobile edge computation offloading via cooperative multi-agent deep reinforcement learning. *IEEE Trans. Cogn. Commun. Netw.* **2021**, *8*, 350–364. [CrossRef]
27. Zhao, N.; Ye, Z.; Pei, Y.; Liang, Y.C.; Niyato, D. Multi-agent deep reinforcement learning for task offloading in UAV-assisted mobile edge computing. *IEEE Trans. Wirel. Commun.* **2022**, *21*, 6949–6960. [CrossRef]
28. Lin, W.; Ma, H.; Li, L.; Han, Z. Computing Assistance From the Sky: Decentralized Computation Efficiency Optimization for Air-Ground Integrated MEC Networks. *IEEE Wirel. Commun. Lett.* **2022**, *11*, 2420–2424. [CrossRef]
29. Gan, Z.; Lin, R.; Zou, H. A Multi-Agent Deep Reinforcement Learning Approach for Computation Offloading in 5G Mobile Edge Computing. In Proceedings of the 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), IEEE, Taormina, Italy, 16–19 May 2022; pp. 645–648.
30. Gong, Y.; Yao, H.; Wang, J.; Jiang, L.; Yu, F.R. Multi-agent driven resource allocation and interference management for deep edge networks. *IEEE Trans. Veh. Technol.* **2021**, *71*, 2018–2030. [CrossRef]
31. Guo, S.; Xiao, B.; Yang, Y.; Yang, Y. Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing. In Proceedings of the IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE, San Francisco, CA, USA, 10–14 April 2016; pp. 1–9.
32. de Witt, C.S.; Gupta, T.; Makoviichuk, D.; Makoviychuk, V.; Torr, P.H.; Sun, M.; Whiteson, S. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv* **2020**, arXiv:2011.09533.
33. Chen, X.; Liu, G. Energy-efficient task offloading and resource allocation via deep reinforcement learning for augmented reality in mobile edge networks. *IEEE Internet Things J.* **2021**, *8*, 10843–10856. [CrossRef]