electronics

Article

Multi-Queue-Based Offloading Strategy for Deep Reinforcement Learning Tasks

Ruize Huang [1], Xiaolan Xie [1,*] and Qiang Guo [2,*]

1 College of Computer Science and Engineering, Guilin University of Technology, Guilin 541006, China; 2120221102@glut.edu.cn
2 Guangxi Key Laboratory of Embedded Technology and Intelligent System, Guilin University of Technology, Guilin 541006, China
* Correspondence: xxl@glut.edu.cn (X.X.); 2005030@glut.edu.cn (Q.G.)

Abstract: With the boom in mobile internet services, computationally intensive applications such as virtual and augmented reality have emerged. Mobile edge computing (MEC) technology allows mobile devices to offload heavy computational tasks to edge servers, which are located at the edge of the network. This technique is considered an effective approach to help reduce the burden on devices and enable efficient task offloading. This paper addresses a dynamic real-time task-offloading problem within a stochastic multi-user MEC network, focusing on the long-term stability of system energy consumption and energy budget constraints. To solve this problem, a task-offloading strategy with long-term constraints is proposed, optimized through the construction of multiple queues to maintain users' long-term quality of experience (QoE). The problem is decoupled using Lyapunov theory into a single time-slot problem, modeled as a Markov decision process (MDP). A deep reinforcement learning (DRL)-based LMADDPG algorithm is introduced to solve the task-offloading decision. Finally, Experiments are conducted under the constraints of a limited MEC energy budget and the need to maintain the long-term energy stability of the system. The results from simulation experiments demonstrate that the algorithm outperforms other baseline algorithms in terms of task-offloading decisions.

Keywords: mobile edge computing; task offloading; multi-queue joint optimization; Lyapunov optimization; deep reinforcement learning

check for updates

Citation: Huang, R.; Xie, X.; Guo, Q. Multi-Queue-Based Offloading Strategy for Deep Reinforcement Learning Tasks. *Electronics* **2024**, *13*, 2307. https://doi.org/10.3390/electronics13122307

Academic Editor: Martin Reisslein

Received: 15 May 2024
Revised: 6 June 2024
Accepted: 11 June 2024
Published: 13 June 2024

1. Introduction

Over the past few years, the number of applications has grown exponentially, and the variety of innovative applications has exploded [1]. It has become increasingly challenging for computing resources to meet the demands of application devices, which cannot process tasks locally with low latency. The traditional way of offloading tasks in cloud computing suffers from high latency, network congestion, and long transmission distances, which no longer meet the demands of compute-intensive and latency-sensitive tasks [2]. MEC provides computing power closer to the user by pushing computing resources to the edge of the network, enabling faster and more responsive task processing [3–5]. Users can place tasks on edge servers for processing by means of task offloading, thus not only relying on local device computation [6]. In contrast to traditional cloud computing, MEC is less costly for data transfer and easier for task offloading [7].

Although MEC is gradually moving in a prominent direction, there are still many challenges. In mobile edge computing scenarios, the dynamic and random tasks to be processed are difficult to predict accurately. There is a need to consider how to offload tasks to MEC edge servers while maintaining system performance benefits [8]. When performing task offloading, the offloading incurs a series of overheads. The energy consumption and latency in performing task offloading are particularly important. Therefore, we need to find a task-offloading strategy to optimize energy consumption and latency.

In order to find a suitable task-offloading strategy, current research delves deeply and has conducted a large number of studies on computational task-offloading strategies. Typically, this involves solving mixed-integer nonlinear programming (MINLP) problems. Since decoupling these problems is very complex, most studies have focused on how to reduce the complexity of the algorithms, and many heuristic algorithms have been proposed [9–15]. However, finding a superior task-offloading strategy through heuristic algorithms requires complex and repetitive iteration. In a realistic scenario, if the parameters change, the MINLP problem needs to be solved all over again, resulting in computational redundancy. Therefore, the costs incurred by using traditional optimization algorithms in MEC environments with frequent dynamic changes are very high. However, the emergence of deep reinforcement learning now provides a state-of-the-art solution to address online computational offloading strategy.

In addition to the need to optimize performance, long-term system stability and staying within the MEC's energy budget need to be considered. Failure to consider local energy stability and the limited energy budget of the MEC may result in performance degradation of the offloading policy during a long-term task-offloading process, resulting in a reduced ability to make accurate decisions in real time. Yet most DRL-based research approaches nowadays give little consideration to the stability of local system energy consumption in the long term and the impact of energy budget constraints in MEC. There is no overall optimization of the system in the context of a long-term constraint, and most of the studies only introduce events that trigger task failures, such as [16–18] by introducing packet loss events.

In this paper, we study the dynamic task-offloading strategy in dynamic multi-user MEC scenarios and propose a strategy to solve task offloading with long-term constraints. The long-term constraints are represented by building multiple queues. And the LMADDPG algorithm is designed using a joint approach to its optimization, which can take full advantage of Lyapunov optimization and DRL. In a dynamic stochastic MEC scenario, an optimal task-offloading policy is found to ensure minimizing the task-offloading cost, which maximizes the user's QoE. The main contributions are as follows:

- We propose a real-time task-offloading problem with long-term local system energy consumption and energy budget constraints for MEC in a dynamic MEC scenario. The objective is to maximize the QoE while ensuring that the constraints can be satisfied in the long run. An optimal task-offloading strategy is found to minimize the weighted sum of delay and energy consumption in an unknown dynamic scenario.
- We propose a task-offloading strategy with long-term constraints. The long-term constraint states are represented by creating multiple queues that are jointly optimized with the optimization objective. Unlike other studies that directly add adverse behaviors to the penalty term, we model the problem as a problem with long-term constraints. The problem is decoupled using Lyapunov optimization and transformed into a MINLP problem with a single time slot, thus facilitating the solving of real-time optimization problems.
- We describe the problem as an MDP and then design an LMADDPG algorithm based on the union of Lyapunov optimization and deep reinforcement learning, which solves the real-time task-offloading problem by establishing the advantages of deep reinforcement learning.
- We testify the LMADDPG algorithm experimentally, proving its ability to find an optimal task-offloading strategy to ensure minimization of cost under long-term constraints, and comparing it with other baseline algorithms.

The rest of the paper is structured as follows: Section 2 discusses related work. Section 3 describes the system modeling. Section 4 describes the problem description and problem transformation process. Section 5 details the algorithmic design of the LMADDPG algorithm. Section 6 describes the experimental parameter settings and analyses the experimental simulation results. Section 7 concludes the paper.

2. Related Work

MEC has strong computing power, which can provide a platform for user devices to support them in offloading tasks to their servers for processing. In terms of server layout, MEC servers are located a short distance from the end devices. More emerging applications can use MEC to process tasks [19].

Hao et al. [20] proposed a MURL algorithm to minimize long-term average latency. Wu et al. [21] proposed a DAR-AC algorithm to optimize computational performance and energy consumption. Zhao et al. [22] proposed a branch-bounding method to minimize energy consumption. Zhuang et al. [23] proposed an OTDDPG algorithm to optimize energy consumption and delay. Xiao et al. [24] performed a modeling transformation of collaborative offloading into a MINLP problem formulated to minimize execution delay. Li et al. [25] successfully achieved more flexible energy savings in MEC environments by quantifying the correlation between statistical quality of service guarantees and task-offloading policies. Kim et al. [26] proposed a migration optimization scheme for user mobility by using integer linear programming and heuristic solution algorithms aimed at minimizing service provider costs and user latency. Lim et al. [27] considered optimizing latency, energy consumption, and the packet loss rate with DRL-OS based on the D3QN algorithm. Cao et al. [28] proposed the NSGA-II algorithm to solve the problem of minimizing the overall latency and energy consumption, taking into account time-varying networks and limited computational resources in realistic scenarios. The above work provides important reference criteria for the optimization objectives of task-offloading strategies to cope with realistic task offloading in different scenarios.

Wang et al. [29] investigated server failures, where mobility and power constraints caused tasks running on them to fail as well, and solved the problem by designing a new model. Jiang et al. [30] considered the case of limited edge server computational power and found a task-offloading strategy to effectively safeguard the QoE of end-users under the condition of limited edge server computational power. The above work has led us to recognize the need to consider the limitations of edge servers, which can cause task execution to fail if the server fails.

As the complexity of offloading tasks gradually increases, deep reinforcement learning (DRL) can be utilized to manage the burgeoning application tasks. In different scenarios, it can be optimized by learning strategies to find an excellent task-offloading strategy. Qiu et al. [31] proposed a DC-DRL algorithm to solve the problem, which can be trained in two ways: distributed and centralized training. Zou et al. [32] proposed a dual-offloading framework for realistic application scenarios by designing simulation experiments to simulate realistic task-offloading scenarios for dynamic regional resource scheduling. This is solved using an asynchronous advantageous participant–critic (A3C) algorithm to reduce energy and time costs. Alam et al. [33] proposed an autonomic management framework by considering the challenges surrounding mobility, heterogeneity, and the geographic distribution of mobile devices, which were eventually solved using DRL. Ren et al. [34] coordinated the offloading of software through multiple DRL agents on multiple edge nodes, which can deal with dynamic environments and enable the cost of task offloading to be optimized. Cao et al. [35] used deep reinforcement learning for optimization and a modified algorithm of DDPG for solving.

The aforementioned literature used DRL to solve the task-offloading problem and investigated various optimization objectives. The long-term optimization problem is crucial when solving the task-offloading problem. Some studies have already started to focus on long-term optimization. Some researchers have formulated the problem as an MDP and proposed algorithms to solve it [36,37]. In addition, some studies performed the solution by combining Lyapunov optimization with deep reinforcement learning [38,39]. However, most studies do not consider the design of representing long-term constraints by constructing multiple queues when using DRL methods to solve the task-offloading problem.

Unlike previous studies, this paper builds on existing research and investigates the problem of the dynamic task-offloading strategy with long-term constraints for multi-user

MEC dynamic scenarios. In a scenario with a dynamically changing system environment, the performance constraint—where the long-term system energy consumption is stable and does not exceed the long-term MEC energy budget—is considered, which is described as an optimization problem with long-term constraints. A task-offloading strategy with long-term constraints is proposed to co-optimize with the optimization objective by constructing multiple queues to represent the long-term constraint states. The optimization problem with long-term constrained task offloading is constructed as a MINLP problem. The MINLP problem is decoupled by using Lyapunov optimization. On this basis, the problem is described as a MDP. A LMADDPG algorithm is proposed for the solution. The algorithm represents the long-term constraints in the form of multiple queues and makes use of Lyapunov for optimization, which ensures that the long-term constraints find an optimal task-offloading policy to ensure that the task-offloading cost is minimized.

3. System Model

3.1. System Overview

As Figure 1 shows, in this paper, a scenario with multiple edge servers (ESs) and multiple types of user equipment (UE) is considered, and each user has its energy consumption queue and MEC energy queue. Each edge server has limited resources, and each user needs to execute multiple tasks, which are processed by task offloading, where an edge server is selected to offload part of the tasks and leave some to be executed locally.

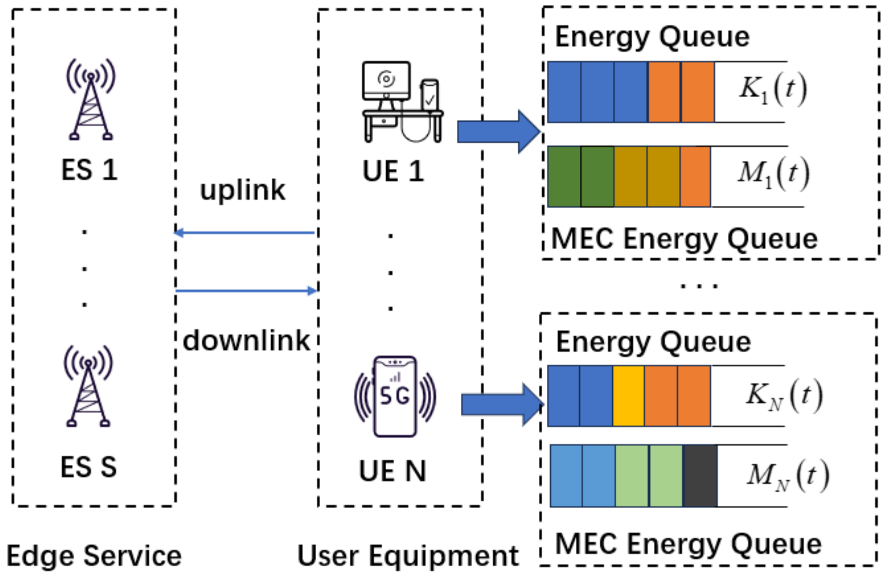

Figure 1. Scenario involving multiple edge servers with multiple user devices, each with its own energy queue and MEC energy queue.

Let $N = \{1, 2, \ldots, N\}$ denote the set of user devices, for each user device $i \in N$, computational tasks will be executed at each time slot. And these tasks need to be handed off to the edge server by way of offloading, or they can be executed on local devices. The task queue has a queuing form, which needs to queue until the end of the previous tasks before the subsequent tasks can be executed, and the latest processing end time of the whole task is its execution time. The set of edge servers is $S = \{1, 2, \ldots, S\}$, $j \in S$. In a continuous time, the edge server assists the user device in computing for a duration, T, set to a number of consecutive time slots, which we define as time slots $T = \{1, 2, \ldots, T\}$, where $t \in T$ is considered a time slot.

The aim of this paper is to consider the optimization of task-offloading costs under the constraints of guaranteeing long-term task energy stability, not exceeding the MEC energy budget. The optimal user QoE can be achieved by determining the best offloading strategy for each time slot. Therefore, the impact of task offloading to the ES for execution or local execution on the QoE needs to be considered.

3.2. Task Transfer and Computational Model

It is assumed that the wireless access system of the edge servers uses orthogonal frequency division multiple access (OFDMA). The network deployment uses the same common bandwidth B. $r_{\mathrm{ul}}(t)$ indicates the uplink transmission rate and $r_{\mathrm{dl}}(t)$ indicates the downlink transmission rate. The offloading method uses partial offloading, where the user device selects only one edge server at a time for offloading. f_i^{local} represents the local computational capacity for the ith UE (user equipment), and f_j^{edge} represents the computational capacity for the jth ES. P_{ij} represents the offload ratio of user i for the jth edge server, and $data_i$ represents the task data size. $c_i(t)$ represents the CPU cycles required to execute the task, $c_i(t) = \sigma \cdot data_i$, where σ represents the number of CPU cycles required to process 1 bit of data. Task processing methods include local execution and offloading to ES for execution.

(1) Local execution

The local computational delay $T_{ij}^{local}(t)$ is as follows:

$$T_{ij}^{local}(t) = \frac{(1 - P_{ij})c_i(t)}{f_i^{local}} \tag{1}$$

The local energy consumption $E_{ij}^{local}(t)$ is as follows:

$$E_{ij}^{local}(t) = \kappa(1 - P_{ij})c_i(t)\left(f_i^{local}\right)^2 \tag{2}$$

where κ denotes the energy consumption coefficient.

(2) Offloading execution

The transmission delay $T_{ij}^{tran}(t)$ is as follows:

$$T_{ij}^{tran}(t) = \frac{data_i(t)}{r_{\mathrm{ul}}(t)} + \frac{data_i(t)}{r_{dl}(t)} \tag{3}$$

Using Shannon's formula for uplink and downlink, respectively, $r_{\mathrm{ul}}(t)$ and $r_{\mathrm{dl}}(t)$ are denoted as follows:

$$r_{ul}(t) = \mathrm{Blog}_2\left(1 + \frac{p_{ue}h_{ul}}{\Gamma^2}\right) \tag{4}$$

$$r_{dl}(t) = B\log_2\left(1 + \frac{p_{edge}h_{dl}}{\Gamma^2}\right) \tag{5}$$

where p_{ue}, p_{edge} represent the transmission power of the UE and ES, respectively. h_{ul}, h_{dl} are the channel gains of the uplink and downlink, respectively, and Γ^2 is the noise power.

The edge server computational delay $T_{ij}^{edge}(t)$ is as follows:

$$T_{ij}^{edge}(t) = \frac{P_{ij}c_i(t)}{f_j^{edge}} \tag{6}$$

The edge offloading delay $T_{ij}^{ES}(t)$ is as follows:

$$T_{ij}^{ES}(t) = T_{ij}^{tran}(t) + T_{ij}^{edge}(t) \tag{7}$$

The transmission energy consumption $E_{ij}^{tran}(t)$ is as follows:

$$E_{ij}^{tran}(t) = p_{ue} \cdot \frac{data_i(t)}{r_{\mathrm{ul}}(t)} + p_{edge} \cdot \frac{data_i(t)}{r_{dl}(t)} \tag{8}$$

where p_{ue} and p_{edge} represent the transmission power of the UE and ES, respectively.

The edge server computing energy consumption $E_{ij}^{edge}(t)$ is as follows:

$$E_{ij}^{edge}(t) = \kappa P_{ij} c_i(t) \left(f_j^{edge}\right)^2 \tag{9}$$

The edge server energy consumption $E_{ij}^{ES}(t)$ is as follows:

$$E_{ij}^{ES}(t) = E_{ij}^{tran}(t) + E_{ij}^{edge}(t) \tag{10}$$

3.3. Queue Model

3.3.1. Energy Queue

To ensure long-term system energy consumption stabilization, we introduce N energy consumption queues $\{K_i(t)\}_{i=1}^N$ for each UE. We set $K_i(0) = 0$. The queue is dynamically updated as follows:

$$K_i(t+1) = \max\left\{K_i(t) + E_{ij}^{local}(t) - \gamma_i, 0\right\} \tag{11}$$

where $E_{ij}^{local}(t)$ is the energy consumption of the ith UE, and γ_i is the energy consumption threshold of the ith UE. When the energy consumption queue does not show an infinite increasing trend, i.e., the UE's energy consumption $E_{ij}^{local}(t)$ is less than or equal to the energy consumption threshold γ_i can be satisfied.

3.3.2. MEC Energy Queue

To ensure that the MEC energy budget is not exceeded when offloading for long-term tasks, we introduce N MEC energy queues $\{M_i(t)\}_{i=1}^N$. Each UE has a MEC energy queue, and the UEs incur different MEC energy consumption values under different offloading policies. We set $M_i(0) = 0$. The queue is dynamically updated as follows:

$$M_i(t+1) = \max\left\{M_i(t) + E_{ij}^{edge}(t) - \varsigma_i, 0\right\} \tag{12}$$

where $E_{ij}^{edge}(t)$ is the energy consumed on the edge server and ς_i is the average energy budget. When the ith user uses too much energy at the edge server, the MEC energy queue at time slot $t+1$ will expand and, thus, the queue can be used to measure the energy constraint of the MEC.

4. Problem Description and Transformation

The task-offloading cost is defined as $S_i(t)$, which is the weighted sum of task latency and user energy consumption, as follows:

$$S_i(t) = \lambda_1 \max\left\{T_{ij}^{local}(t), T_{ij}^{ES}(t)\right\} + \lambda_2\left(E_{ij}^{local}(t) + E_{ij}^{ES}(t)\right) \tag{13}$$

where λ_1, λ_2 denote the respective corresponding weights.

The aim of this paper is to consider the optimization of task-offloading costs under the constraints of guaranteeing long-term task energy stability, not exceeding the MEC energy budget. The optimal user QoE can be achieved by determining the best offloading strategy for each time slot. Task-offloading optimization with long-term constraints needs to be considered. Thus, the problem form is expressed as follows:

$$P1:\min S=\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{i=1}^{N}S_i(t) \tag{14}$$

$$\begin{aligned}
&s.t.\\
&C1:\lim_{T\to\infty}\tfrac{1}{T}\sum_{t=1}^{T-1}E\left[E_{ij}^{local}(t)\right]\le\gamma_i,\forall i\\
&C2:\lim_{T\to\infty}\tfrac{1}{T}\sum_{t=1}^{T-1}E\left[E_{ij}^{edge}(t)\right]\le\varsigma_i,\forall i
\end{aligned}$$

where $C1$ denotes the energy constraint of the local energy queue and $C2$ denotes the energy constraint of the MEC energy queue.

Making optimal task-offloading decisions under a long-term constraint is very difficult due to environment and task unknowns.

Lyapunov Optimization

The MINLP optimization problem is decoupled into a single time-slot deterministic problem through Lyapunov optimization, as the optimization objective, as well as the constraints of $P1$, have both long-term optimizations. In order to have control over the system energy queue and the MEC energy queue, we define $N(t)\overset{\Delta}{=}\{K(t),M(t)\}$, where $K(t)=\{K_i(t)\}_{i=1}^{N}$, and $M(t)=\{M_i(t)\}_{i=1}^{N}$, $N(t)=\{N_i(t)\}_{i=1}^{N}$. According to the Lyapunov optimization theory, the Lyapunov function $L(N_i(t))$, is denoted as follows:

$$L(N_i(t))\overset{\Delta}{=}\frac{1}{2}\left(\sum_{i=1}^{N}K_i(t)^2+\sum_{i=1}^{N}M_i(t)^2\right) \tag{15}$$

The change in value will be referred to as the Lyapunov drift function $\Delta(N_i(t))$, denoted as follows:

$$\Delta(N_i(t))\overset{\Delta}{=}E\{L(N_i(t+1))-L(N_i(t))|N_i(t)\} \tag{16}$$

Due to $N(t)\overset{\Delta}{=}\{K(t),M(t)\}$; therefore, $\Delta(N_i(t))$ can be derived from $K_i(t),M_i(t)$.

Theorem 1. *Given the dynamic relationship of the queues in the system, when* $\mathrm{E}\{L(N_i(t))\}<\infty$, *and there exist constants* $\mu>0,\varepsilon>0$, *we can obtain an upper bound on the Lyapunov drift* $\Delta(N(t))$ *as follows:*

$$\Delta(N_i(t))\le\mu+\varepsilon\sum_{i=1}^{N}N_i(t) \tag{17}$$

Proof of Theorem 1. Using $\{\max\{x,0\}\}^2\le x^2$, for the energy consumption queue, squaring Equation (11), we have the following:

$$\begin{aligned}
K_i(t+1)^2&=\left\{\max\left\{K_i(t)+E_{ij}^{local}(t)-\gamma_i,0\right\}\right\}^2\\
&\le\left(K_i(t)+E_{ij}^{local}(t)-\gamma_i\right)^2
\end{aligned} \tag{18}$$

The drift function is as follows:

$$\Delta(K_i(t))\overset{\Delta}{=}E\{L(K_i(t+1))-L(K_i(t))|K_i(t)\} \tag{19}$$

This can be obtained by substituting Equation (18) into Equation (19), as follows:

$$\begin{aligned}\Delta(K_i(t)) &= \frac{1}{2}\sum_{i=1}^{N} E\left\{K_i(t+1)^2|K_i(t)\right\} - \frac{1}{2}\sum_{i=1}^{N} E\left\{K_i(t)^2|K_i(t)\right\} \\ &\leq \frac{1}{2}\sum_{i=1}^{N} E\left\{\left(K_i(t) + E_{ij}^{local}(t) - \gamma_i\right)^2 - K_i(t)^2|K_i(t)\right\} \\ &\leq \frac{1}{2}\sum_{i=1}^{N} E\left\{\left(E_{ij}^{local}(t) - \gamma_i\right)^2|K_i(t)\right\} \\ &\quad + \sum_{i=1}^{N} E\left\{K_i(t)\left(E_{ij}^{local}(t) - \gamma_i\right)|K_i(t)\right\} \\ &\leq \mu_1 + \sum_{i=1}^{N} E\left\{K_i(t)\left(E_{ij}^{local}(t) - \gamma_i\right)|K_i(t)\right\}\end{aligned} \tag{20}$$

where μ_1 is a constant, $\mu_1 \overset{\Delta}{=} \frac{1}{2}\sum_{i=1}^{N}\left(E_{ij,\max}^{local}(t) - \gamma_i\right)^2$, holds for all $i \in N$ since all $E_{ij}^{local}(t) \leq E_{ij,\max}^{local}(t)$.

For the MEC energy queue, we square Equation (12) as follows:

$$\begin{aligned}M_i(t+1)^2 &= \left\{\max\left\{M_i(t) + E_{ij}^{edge}(t) - \varsigma_i, 0\right\}\right\}^2 \\ &\leq \left(M_i(t) + E_{ij}^{edge}(t) - \varsigma_i\right)^2\end{aligned} \tag{21}$$

The drift function of the MEC energy queue is calculated as follows:

$$\Delta(M_i(t)) \overset{\Delta}{=} E\{L(M_i(t+1)) - L(M_i(t))|M_i(t)\} \tag{22}$$

This can be obtained by substituting Equation (21) into Equation (22):

$$\begin{aligned}\Delta(M_i(t)) &= \frac{1}{2}\sum_{i=1}^{N} E\left\{M_i(t+1)^2 \mid M_i(t)\right\} - \frac{1}{2}\sum_{i=1}^{N} E\left\{M_i(t)^2 \mid M_i(t)\right\} \\ &\leq \frac{1}{2}\sum_{i=1}^{N} E\left\{\left(M_i(t) + E_{ij}^{edge}(t) - \varsigma_i\right)^2 - M_i(t)^2 \mid M_i(t)\right\} \\ &\leq \frac{1}{2}\sum_{i=1}^{N} E\left\{\left(E_{ij}^{edge}(t) - \varsigma_i\right)^2 \mid M_i(t)\right\} \\ &\quad + \sum_{i=1}^{N} E\left\{M_i(t)\left(E_{ij}^{edge}(t) - \varsigma_i\right) \mid M_i(t)\right\} \\ &\leq \mu_2 + \sum_{i=1}^{N} E\left\{M_i(t)\left(E_{ij}^{edge}(t) - \varsigma_i\right) \mid M_i(t)\right\}\end{aligned} \tag{23}$$

where μ_2 is a constant, $\mu_2 \overset{\Delta}{=} \frac{1}{2}\sum_{i=1}^{N}\left(E_{ij,\max}^{edge}(t) - \varsigma_i\right)^2$, holds for all $i \in N$ since all $E_{ij}^{edge}(t) \leq E_{ij,\max}^{edge}(t)$.

Summarizing Equations (20) and (23) yields the following:

$$\begin{aligned}\Delta(N_i(t)) &= \Delta(K_i(t)) + \Delta(M_i(t)) \\ &\le \mu + \sum_{i=1}^{N} E\left\{K_i(t)\left(E_{ij}^{local}(t) - \gamma_i\right) \mid N_i(t)\right\} \\ &\quad + \sum_{i=1}^{N} E\left\{M_i(t)\left(E_{ij}^{edge}(t) - \varsigma_i\right) \mid N_i(t)\right\}\end{aligned} \tag{24}$$

where $\mu = \mu_1 + \mu_2$. $\square$

This is optimized to ensure long-term task energy consumption stability without exceeding the constraints of the MEC energy budget. This is achieved using the Lyapunov drift plus penalty, minimizing its upper bound. The expression is given as follows:

$$\Delta(N_i(t)) + VE\left\{\sum_{i=1}^{N} S_i(t)|N_i(t)\right\} \tag{25}$$

where V is a hyperparameter and $V > 0$; the first term is the Lyapunov drift function containing the energy consumption queue and the MEC energy queue, and the second term $VE\left\{\sum_{i=1}^{N} S_i(t)|N_i(t)\right\}$ is the penalty term. The optimal QoE is found by minimizing the upper bound. According to Theorem 1, Equation (24) can be obtained by substituting into Equation (25):

$$\begin{aligned}&\Delta(N_i(t)) + VE\left\{\sum_{i=1}^{N} S_i(t)|N_i(t)\right\} \\ &\le \mu + \sum_{i=1}^{N} E\left\{K_i(t)\left(E_{ij}^{local}(t) - \gamma_i\right)|N_i(t)\right\} \\ &+ \sum_{i=1}^{N} E\left\{M_i(t)\left(E_{ij}^{edge}(t) - \varsigma_i\right)|N_i(t)\right\} \\ &+VE\left\{\sum_{i=1}^{N} S_i(t)|N_i(t)\right\}\end{aligned} \tag{26}$$

Therefore, the long-term optimization problem $P1$ is decoupled into a single time-slot MINLP subproblem, transforming the $P1$ problem into the following:

$$\begin{aligned}P2: \min R(t) = &\sum_{i=1}^{N} E\left\{K_i(t)\left(E_{ij}^{local}(t) - \gamma_i\right) \mid N_i(t)\right\} \\ &+ \sum_{i=1}^{N} E\left\{M_i(t)\left(E_{ij}^{edge}(t) - \varsigma_i\right) \mid N_i(t)\right\} \\ &+ VE\left\{\sum_{i=1}^{N} S_i(t) \mid N_i(t)\right\}\end{aligned} \tag{27}$$

When all $P2$ problems are solved, the $P1$ problem can be solved. Decisions within the current time slot are based on the current state without regard to the historical state.

5. Deep Reinforcement Learning-Based Solutions

5.1. Markov Decision Process

In the environment designed in this paper, the UE is viewed as the DRL-Agent. In the face of dynamically changing MEC environments, it is often difficult to go to the state transfer probability matrix P. Thus, it is not possible to rely on the full quaternion (S, a, R, P), which contains state, action, reward, and state transfer probability, to characterize the MDP.

Therefore, in this paper, we turn to the use of the ternary (S, a, R), which contains state, action, and reward, without considering state transfer probability.

(1) State space: Reinforcement learning involves the use of one's own powerful learning ability to improve one's decision-making by learning the information stored in the experienced replay buffer, which updates the strategy based on its improved decision-making. Therefore, defining an appropriate state space is crucial for overall performance improvement. In defining the state space, the complete environment needs to be added to it. For user device i, define the state space as follows:

$$s(t) = (s_1(t), s_2(t), \ldots, s_i(t)) \tag{28}$$

$$s_i(t) = \left(P(t), r^j_{\mathrm{ul}}(t), r^j_{\mathrm{dl}}(t), C^{loacl}_i, C^{edge}_j\right) \tag{29}$$

where C^{loacl}_i, C^{edge}_j represent the computational resources of UE and ES, respectively.

(2) Action space: When the DRL-Agent acquires state $s(t)$, it selects an action from the action space that determines the target server ES_j and the offload ratio P_{ij} for offloading tasks. This strategy aims to achieve a balanced allocation of tasks by processing some of them locally and offloading others to edge servers. For the user device i, define the action as follows:

$$a(t) = (a_1(t), a_2(t), \ldots, a_i(t)) \tag{30}$$

$$a_i(t) = [ES_j, P_{ij}] \tag{31}$$

(3) Reward function: The aim of this paper is to consider the optimization of task-offloading costs under the constraints of guaranteeing long-term task energy stability, not exceeding the MEC energy budget. However, the aim of reinforcement learning is to obtain the highest reward value, and the task-offloading cost is inversely proportional to the reward value. Therefore, the reward function for time slot t is as follows:

$$r(t) = reward(s(t), a(t)) = -R(t) \tag{32}$$

5.2. DRL-Based Algorithm Design

5.2.1. Deep Reinforcement Learning Algorithms

Traditional reinforcement learning algorithms typically use a Q-value table to select out the best action. However, as the complexity of the problem changes, it eventually makes the size of the Q-value table explode, resulting in a significant increase in storage and computational cost and, thus, is no longer advantageous in high-dimensional spaces. However, Deep Q-Network (DQN) introduces a neural network to approximate the Q-function, thus avoiding the need to directly store Q-values and being able to handle more complex problems. However, a single Q-learning or DQN cannot be used to implement complex interactions between multiple 'intelligences'. In this case, the use of the multi-agent deep deterministic policy gradient (MADDPG) algorithm offers significant advantages. The MADDPG algorithm facilitates co-learning among 'intelligences' and enhances overall performance through policy sharing, making it more effective in addressing multi-intelligence collaborative decision-making problems.

The MADDPG algorithm achieves better performance by utilizing deep learning and policy gradient methods that allow multiple 'intelligences' to learn and make decisions collaboratively in the environment. Its distributed learning feature makes the algorithm scalable and adaptable to large-scale multi-intelligent body systems. The algorithm uses the structure of the two networks joined so that the algorithm can be used in a dynamically changing environment, and can autonomously learn an advantageous strategy, through the learning of the strategy constantly updated and adjusted, in the face of dynamic tasks and different scenarios can be reasonably adjusted to the optimal strategy. In addition, the algorithm can further optimize and adjust the optimal strategy through the interaction between multiple 'intelligences', so that each intelligence can fully improve its own decision-making level.

5.2.2. Actor–Critic Network

The actor–network will maximize the reward value accumulated over time by receiving states as inputs and then outputting actions, and it learns the best action to choose in each state by continuously adjusting the network parameters. This network contains an evaluation network $\mu_i(s_i, \omega_i)$, and a target network $\mu'_i(s_i, {\omega'}_i)$, where ω_i, ω'_i denote the respective parameters of the two networks. The critic–network is used to evaluate the action's output by the actor–network and provide feedback to improve the strategy. The network receives states and actions as its inputs and outputs the payoff values of the selected actions. The critic–network then provides feedback to the actor–network by accurately estimating these values so that the actor–network can use the feedback to optimize its strategy. The network also includes an evaluation network $Q_i(s_i, a_i|\theta_i)$ and a target network $Q'_i(s_i, a_i|{\theta'}_i)$, where θ_i, θ'_i denote the parameters of each of the two networks.The updating of the network parameters between the two networks is done by gradient descent.

5.2.3. LMADDPG Algorithm Design

Each user device is individually defined as an intelligence, and each intelligence has two queues each, i.e., the system energy queue and the MEC energy queue, which are used to represent the long-term constraints. The use of the actor–critic network in this algorithm is used for learning to optimize the task-offloading decision. In order to reduce the data relevance of the input experience during the training process and to improve the data utilization, the LMADDPG algorithm employs an experience replay mechanism, which improves the performance of the algorithm by creating an experience replay buffer D. During training, samples are randomly selected from the experience replay buffer D. Each data sample includes information about the state, action, reward, and next state. It is assumed that Z data, each denoted as $\left(s_j, a_j, r_j, s_{j+1}\right)$, are randomly drawn from D for each training session.The corresponding loss values are computed using these randomly drawn data, and the computed loss values are used to update the parameters of the network. The critic–network at each intelligence interacts with each other through the 'intelligences' to update the network parameters by minimizing the computed loss function. Define the objective value as follows:

$$y_i^j = r_i^j + \gamma Q'\left(s'^j, a_1'^j, \ldots, {a'_N}^j \mid \omega_i\right)\Big|_{{a'}_i=\mu'_i\left(s_i'^j\right)} \tag{33}$$

where γ is the discount factor.

The loss function its expression is as follows:

$$L_i(\omega_i) = \frac{1}{Z}\sum_{j=1}^{Z}\left(y_i^j - Q_i\left(s^j, a_1^j, \ldots, a_N^j|\omega_i\right)\right)^2 \tag{34}$$

To update the actor–network using the policy gradient, the actor–network is used to compute the action in the current state, and then the gradient of the action value function computed by the critic–network is used to update the actor–network parameters with the following formula:

$$\nabla_{\theta_i} J(\mu_k) \approx \frac{1}{Z}\sum_{j=1}^{Z}\nabla_{\theta_i}\mu_k\left(s_i^j\right)\nabla_{a_i}Q_i\left(s^j, a_1^j, \ldots, a_N^j|\omega_i\right)\Big|_{a_i=\mu_i\left(s_i'^i\right)} \tag{35}$$

Use the soft update strategy to update the target actor–network and target critic–network parameters:

$$\omega'_i = \tau\omega_i + (1-\tau)\omega'_i \tag{36}$$

$$\theta'_i = \tau\theta_i + (1-\tau)\theta'_i \tag{37}$$

The reward function is optimized using Lyapunov optimization based on the multiple queues created. Each intelligent body obtains the reward value by executing an action after acquiring the environment information and accumulates experience data using the experience playback mechanism. Each intelligent body updates its policy network with the accumulated data, using a deep deterministic policy gradient approach to optimize its policy to maximize the expected cumulative reward. The queue information is updated at each training time slot. When updating the policy network, the intelligence also uses the target policy network to stabilize the training process by updating the parameters of the target network through a soft update approach. During the training process, the ‘intelligences’ learn from each other and optimize their respective strategies through collaboration and competition to achieve the global optimal solution. In this way, the LMADDPG algorithm enables multiple ‘intelligences’ to learn to work together effectively in a collaborative environment to achieve finding an optimal task-offloading decision under the performance constraint of stable long-term system energy consumption and not exceeding the long-term MEC energy budget. The overall execution process is shown in Algorithm 1.

Algorithm 1 LMADDPG task-offloading algorithm

```
1:  **Initialization:** Initialize the parameters of each agent's actor and critic evaluations and target networks. Initialize the replay buffer $D$, learning rate, the discount factor, the maximum learning epoch, steps.
2:  **for** $epoch = 1$ **to** $M$ **do**
3:    Initialize a random process $N$ for action exploration
4:    Initialize $K(0) = 0$, $M(0) = 0$
5:    **for** $t = 1$ **to** $T$ **do**
6:      For each agent $i$, select action $a(t)$, $a(t)$ based on the current observed state: $a(t) = \mu_i(s_i|\omega_i) + N(t)$
7:      Execute actions $a(t) = (a_1(t), a_2(t), \ldots, a_i(t))$, obtain the reward $r_i(t)$ based on Lyapunov drift-plus-penalty function, and the subsequent new state $s'_i(t)$
8:      For each agent $i$, input the new state $s'_i(t)$ to agent $i$
9:      Store the agent's information into the experience replay buffer $D$ as a tuple of four elements $(s, a, r, s')$
10:     $s_i = s'_i$
11:     **for** Each agent $i = 1$ **to** $N$ **do**
12:       Sample a random mini-batch of $Z$ samples from the replay buffer $D$
13:       $y_i^j = r_i^j + \gamma Q'\left(s'^j, a'^j_1, \ldots, a'^j_N|\omega_i\right)\Big|_{a'_i=\mu'_i\left(s'^j_i\right)}$
14:       Update the critic–network according to the following loss function: $L_i(\omega_i) = \frac{1}{Z}\sum_{j=1}^{Z}\left(y_i^j - Q_i\left(s^j, a_1^j, \ldots, a_N^j|\omega_i\right)\right)^2$
15:       Update the actor–network using the sampled gradient based on Equation (36)
16:       Update $K(t)$ using Equation (11)
17:       Update $M(t)$ using Equation (12)
18:     **end for**
19:     Update target network parameters for each agent with:
        $\omega'_i = \tau\omega_i + (1-\tau)\omega'_i$
        $\theta'_i = \tau\theta_i + (1-\tau)\theta'_i$
20:   **end for**
21: **end for**
```

5.2.4. Algorithm Complexity Analysis

In the LMADDPG algorithm, each intelligent body uses the actor–critic network architecture. The number of ‘intelligences’ is N. L^a denotes the number of network layers of actor–network and L^c denotes the number of network layers of critic–network. I_{l^a-1}, I_{l^a}

denotes the dimensions of inputs and outputs of the l^a layer of actor–network and $\mathrm{I}_{l^c-1}, \mathrm{I}_{l^c}$ denotes the dimensions of inputs and outputs of the l^c layer respectively. Therefore, the algorithm complexity of LMADDPG can be calculated as $O\left(2N \cdot \left(\sum_{l^a=1}^{L^a} \mathrm{I}_{l^a-1}\mathrm{I}_{l^a} + \sum_{l^c=1}^{L^c} \mathrm{I}_{l^c-1}\mathrm{I}_{l^c}\right)\right)$.

6. Simulation Experiment and Analysis

6.1. Experimental Parameters

This research conducted simulation experiments within a Python 3.10.7 environment on a Windows 11 operating system. In this paper, the simulation scenario is set up as a multi-user multi-MEC scenario, where each UE can transmit data to and from ES through channel. The size of each task $data_i$, follows a uniform distribution. The computational power of the UE of the mobile device is set up to be $[1.2, 1.8]$ Ghz, and that of the edge server is set up to be $[6, 7]$ Ghz. The task size is $[1.5, 2]$ Mb. The main parameters are shown in Table 1:

Table 1. Experimental parameter.

Parameters	Value
Number of users N	5
Number of edge servers S	7
Bandwidth B	40 Mhz
Lyapunov weights V	30
User equipment transmission power p_{ue}	0.01 W
Edge server transmission power p_{edge}	0.1 W
Gaussian noise power Γ^2	-174 dBm
The local computing capability f_i^{local}	$[1.2, 1.8]$ Ghz
The MEC computing capability f_j^{edge}	$[6, 7]$ Ghz
Actor–network learning rate	0.001
Critic–network learning rate	0.001
Size of mini-batch	32
Training epochs	1000
Time slots T	500
Optimizer	Adam

6.2. Comparative Experiments

6.2.1. Comparative Algorithms Overview

In order to prove the superiority and reliability of this paper's algorithm, this paper's algorithm is compared with five other baseline algorithms.

(1) DDQN-based algorithm [40]. Double DQN algorithm is an improved algorithm based on DQN. DDQN improves the performance by using two independent neural networks for selecting the action and evaluating the value of the action.

(2) D3QN-based algorithm [41]. D3QN estimates the function of state values and advantage separately by introducing the Double Q-Learning mechanism while using the Dueling Network architecture. It enables the Agent to learn the task-offloading strategy more accurately.

(3) MAPPO-based algorithm [42]. MAPPO is a multi-intelligence body algorithm. A centralized approach is used during training to acquire information from multiple 'intelligences' and optimize the decision making of the 'intelligences'. In the execution process the decision is based on local observation information only. The algorithm deals with the problem of policy learning in a multi-intelligent body environment through a proximal policy optimization approach.

(4) Local Computing Only (LOC) scheme. All the tasks are all computed locally without task offloading.

(5) Random scheme (Random). Directly randomly generate task-offloading strategies for task offloading.

6.2.2. Simulation Results and Analysis

Firstly, the convergence of the algorithm designed in this paper is evaluated, and this paper evaluates the convergence through two objects. The LMADDPG algorithm proposed in this paper performs task offloading through actor–critic network, so the first object for evaluation is the loss function values of the two networks. DRL-Agent will probabilistically choose the action in each step and, thus, the image will fluctuate continuously. Where Figure 2 represents the actor–network loss value and Figure 3 represents the critic–network loss value. As can be seen from Figures 2 and 3, with the increase in the number of iterations, the loss values of the two networks are decreasing and both of them begin to converge gradually, indicating that both networks are functioning properly.

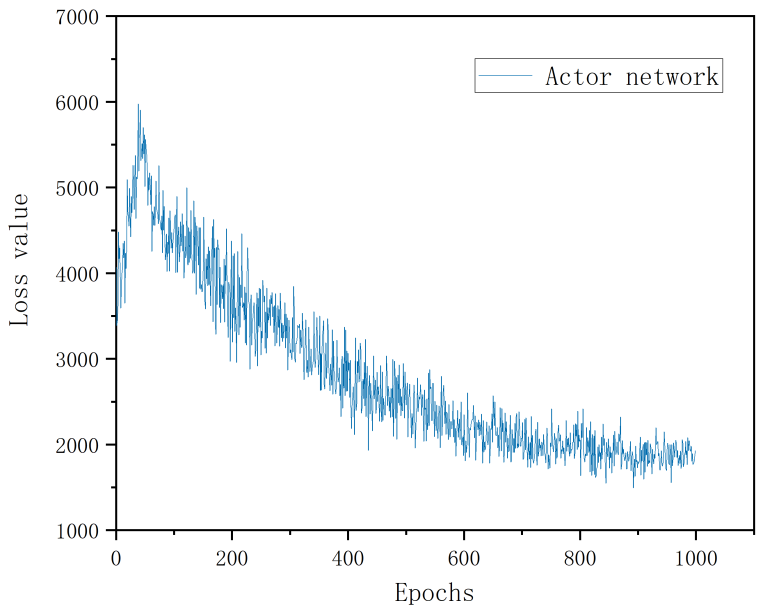

Figure 2. Actor-network loss value.

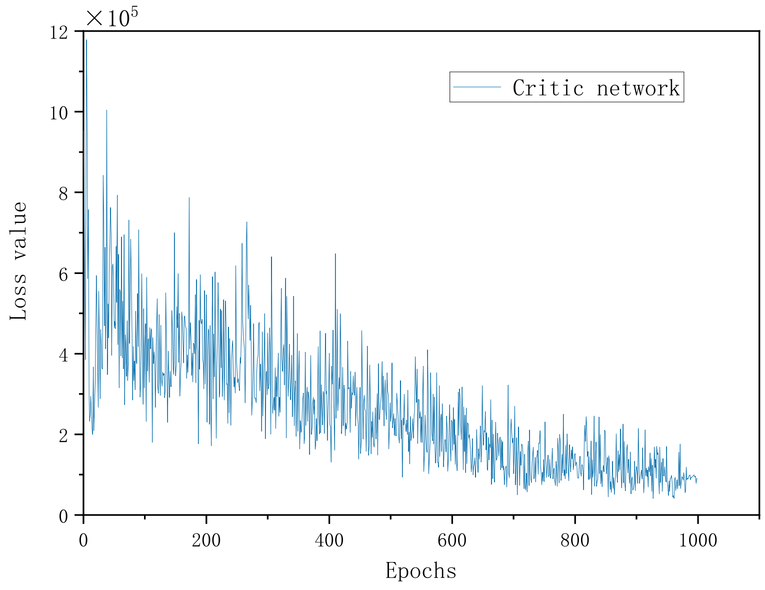

Figure 3. Critic-network loss value.

The LMADDPG algorithm reward value is used as the second evaluation object. In Figure 4, the convergence judgment of the LMADDPG algorithm is demonstrated. The algorithm, which undergoes a learning process, has a strong tendency to jitter in the initial phase due to the randomness of early reinforcement learning. However, the reward value obviously grows as the number of iterations increases. Through the learning process, an optimal task-offloading decision is finally learned, which leads to the stabilization of the reward value, i.e., the LMADDPG algorithm gradually reaches convergence. Moreover, the algorithm of this paper can reach convergence in a very short number of iterations, which can be attributed to its policy update. This update integrates the deterministic behavioral policy with the exploitation of the Q-value function. The effectiveness of the LMADDPG algorithm is based on the proof of convergence of the two evaluated objects described above.

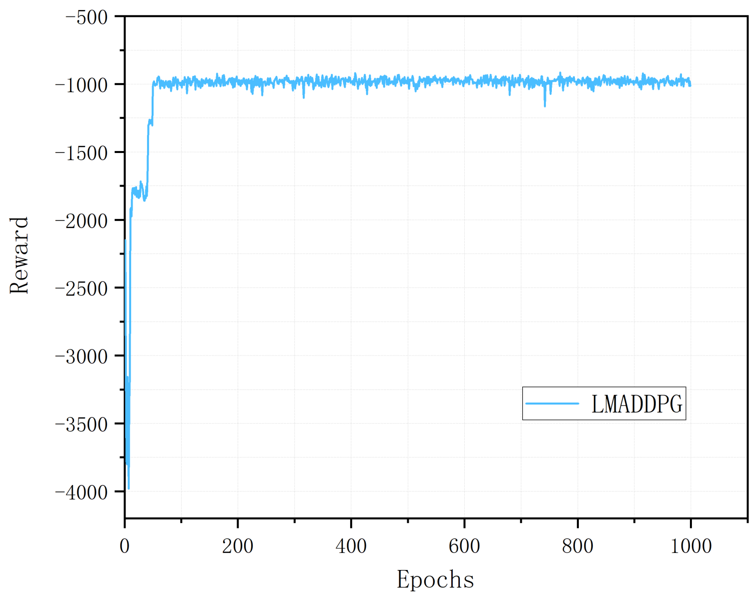

Figure 4. Convergence performance and reward of the proposed task-offloading algorithm for LMADDPG.

Then, the energy consumption is evaluated for the system energy queue and the MEC energy queue. In Figure 5 experiments are conducted for four different algorithms for the system energy queue, where the horizontal coordinate is the time slot and the vertical coordinate is the energy consumption. In this process, as the time slots increase, all algorithms through optimization simultaneously keep the system energy consumption queue stable and do not show an explosive rising trend. It means that in the environment of long-term task offloading, the system energy consumption queue will not be oversized. Instead, it shows a gradual reduction trend and eventually remains in a stable state. This is because the algorithm can maintain a relatively stable queue length after learning. In addition, comparing the LMADDPG algorithm with other baseline algorithms, the LMADDPG algorithm maintains the smallest queue length, i.e., the best optimization.

In Figure 6, the energy consumption of four different algorithms MEC energy queues are evaluated. It can be seen that the MEC energy queues of the optimized algorithms do not have an infinite rising trend. As the time slot progresses, it becomes evident that the MEC energy queue of the LMADDPG algorithm exhibits a more stable behavior compared to the other algorithms. Although the performance of LMADDPG will be lower than other algorithms in the initial process of task offloading, the performance of LMADDPG is more advantageous in the long-term process. The LMADDPG algorithm outperforms the other baseline algorithms more in keeping the MEC energy stable during long-term optimization.

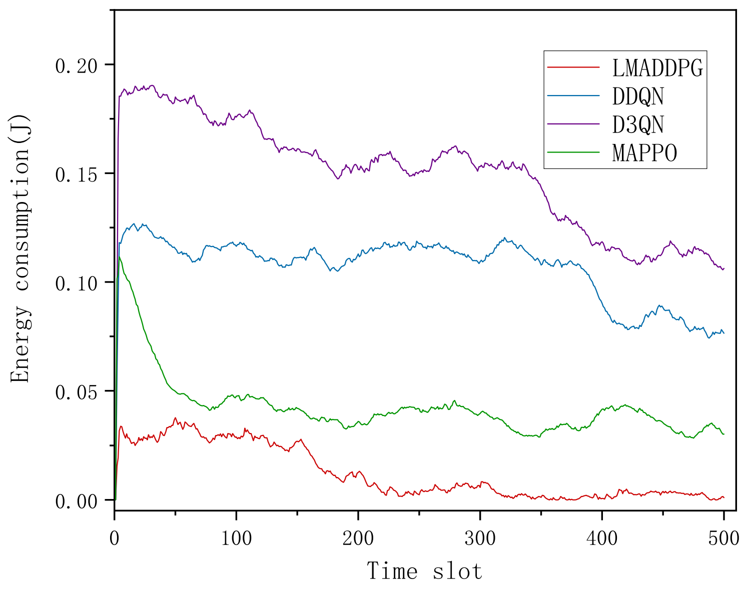

Figure 5. Energy consumption of the energy queue.

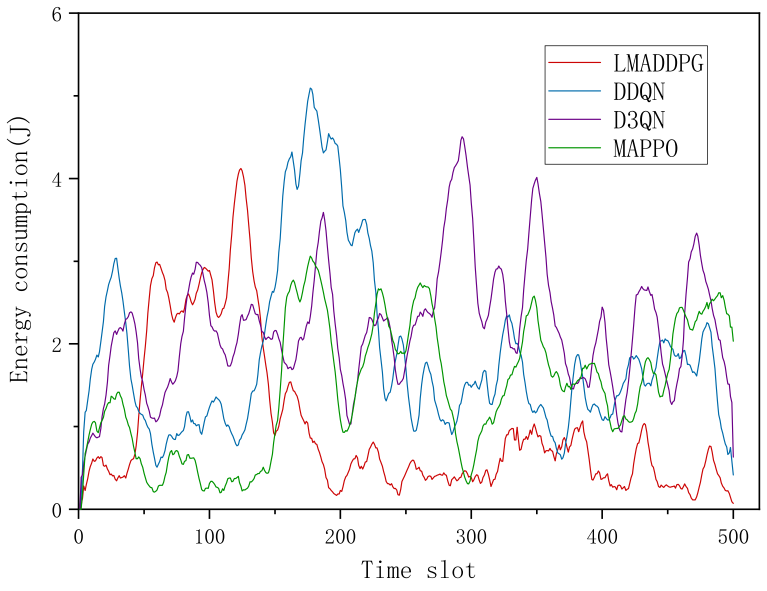

Figure 6. Energy consumption of MEC energy queue.

In addition, the effect of penalty coefficient adjustment average task-offloading cost is experimentally compared. The impact of Lyapunov penalty coefficient V on the performance of LMADDPG is further demonstrated in Figure 7, where the delay and energy weighted sum of tasks will be compared by adjusting the penalty coefficient V, setting $V \in [1, 10, 20, 30, 40, 50, 60]$. As V increases, the average cost is decreasing, which is due to the skewing of the fairness of the optimization strategy when the penalty factor is increasing. This trend is due to the fact that the higher the value of V gives more weight to the weighted delay and energy cost, which drives the algorithm towards cost optimization. Therefore, in practical scenarios, a suitable V value needs to be set to balance the average system energy consumption as well as the stability between the average MEC energy and the average cost.

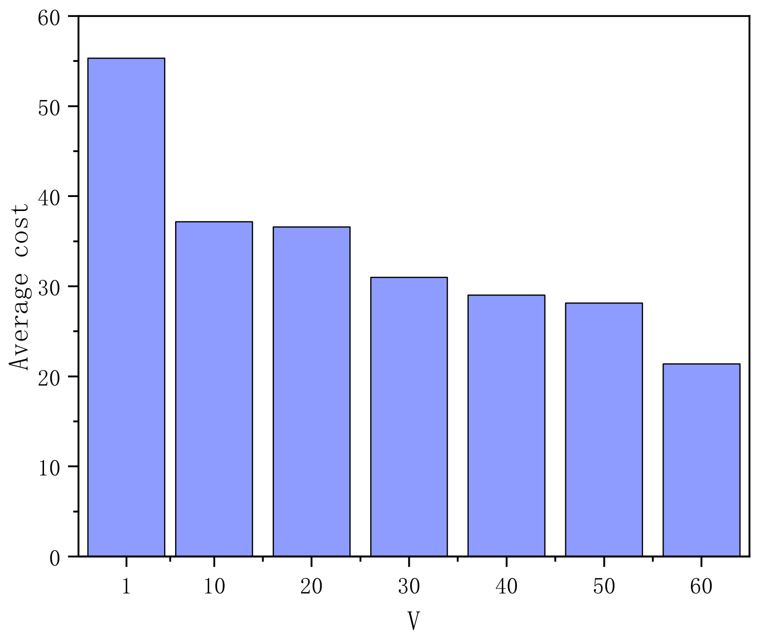

Figure 7. Effect of different parameters V on the weighted sum of average delay and energy consumption.

Figure 8 shows the performance comparison between the LMADDPG algorithm and the MADDPG algorithm using multi-queue optimization. Both algorithms perform task-offloading decisions separately and calculate the average task-offloading cost, i.e., the average cost. The proposed LMADDPG algorithm is able to reach stability in a very short number of iterations, converges better than the MADDPG algorithm, and produces a much smaller average cost than the MADDPG algorithm. This shows that the LMADDPG outperforms MADDPG in terms of performance.

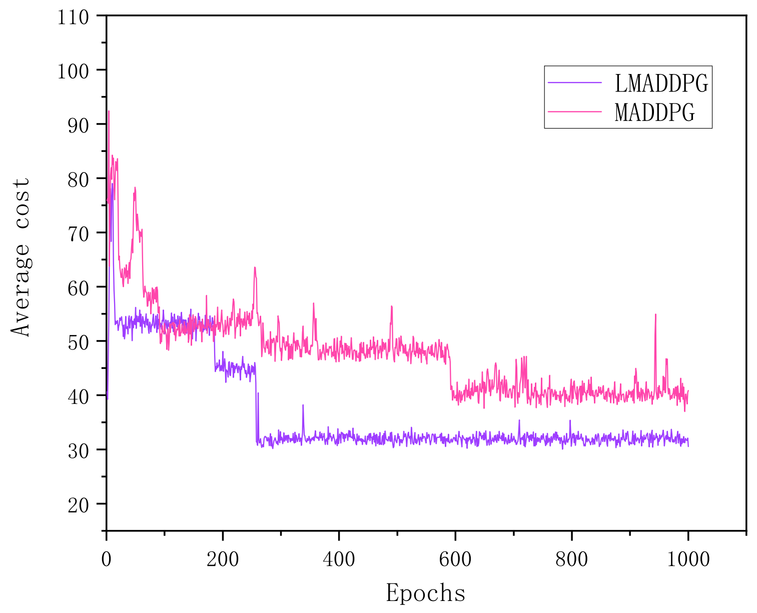

Figure 8. Comparison of average delay and energy weighted sum of LMADDPG algorithm and MADDPG algorithm.

Finally, the advantages of the algorithms over the baseline algorithms are verified, and the computational delay and energy weighted sums are compared for the different baseline algorithms. The results are shown in Figure 9, where the task-offloading strategy of LMADDPG is compared with other algorithms and the weighted sum of delay and energy consumption is calculated. It is found that our algorithm reduces the cost and finds the optimal task-offloading strategy faster compared to the other five baseline algorithms. Compared to other baseline algorithms, our algorithm is more advantageous.

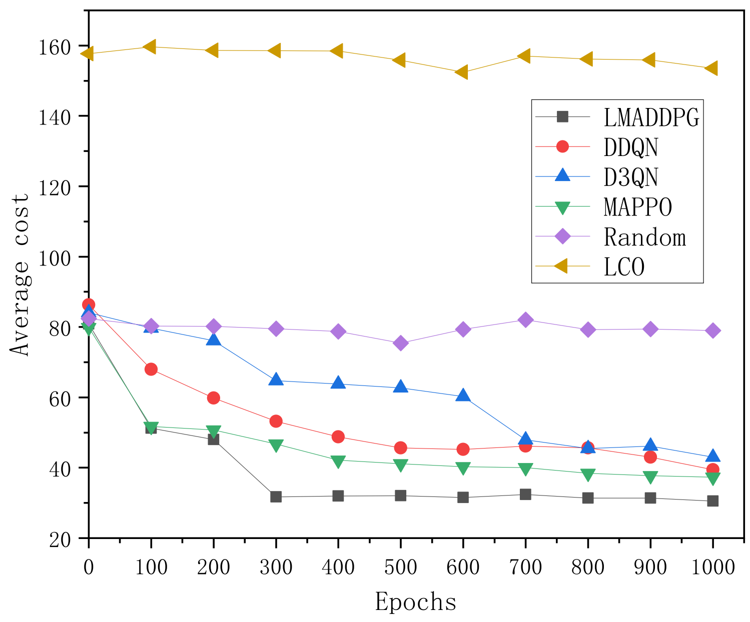

Figure 9. Comparison of LMADDPG algorithm with other baseline algorithms.

7. Conclusions

In this paper, we propose a multi-queue-based real-time task-offloading strategy for deep reinforcement learning. We establish multiple queues to represent the long-term constraint states and co-optimize with the optimization objective. The optimization problem with long-term constraints is decoupled into subproblems to be solved in a single time slot using Lyapunov optimization, which describes the problem as an MDP. We propose a DRL-based LMADDPG algorithm to solve the task-offloading decision problem. During the training process, all the 'intelligences' share a unified strategy network in order to utilize the experience of all the 'intelligences' during the training process. However, in the execution phase, each intelligence independently executes its own policy to interact with the environment. Finally, an optimal task-offloading strategy is found, which can effectively maintain long-term system energy consumption and MEC energy stability. The simulation results prove the effectiveness of the improved algorithm and it is more advantageous in comparison with other baseline algorithms.

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

Funding: This research was funded by the Guangxi key research and development program (grant number Guike AB23026004), the Guangxi key research and development program (grant number Guike AB23026036), and the National Natural Science Foundation of China (grant number NO. 62262011).

Data Availability Statement: Data are contained within the article.

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

References

1. Bai, T.; Pan, C.; Deng, Y.; Elkashlan, M.; Nallanathan, A.; Hanzo, L. Latency Minimization for Intelligent Reflecting Surface Aided Mobile Edge Computing. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 2666–2682. [Google Scholar][CrossRef]
2. Chatzopoulos, D.; Bermejo, C.; Kosta, S.; Hui, P. Offloading Computations to Mobile Devices and Cloudlets via an Upgraded NFC Communication Protocol. *IEEE Trans. Mob. Comput.* **2020**, *19*, 640–653. [Google Scholar][CrossRef]
3. Chen, N.; Zhang, S.; Qian, Z.; Wu, J.; Lu, S. When Learning Joins Edge: Real-Time Proportional Computation Offloading via Deep Reinforcement Learning. In Proceedings of the 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS), Tianjin, China, 4–6 December 2019; pp. 414–421. [Google Scholar][CrossRef]
4. Yu, R.; Li, P. Toward Resource-Efficient Federated Learning in Mobile Edge Computing. *IEEE Netw.* **2021**, *35*, 148–155. [Google Scholar][CrossRef]
5. Guo, Y.; Zhao, R.; Lai, S.; Fan, L.; Lei, X.; Karagiannidis, G.K. Distributed Machine Learning for Multiuser Mobile Edge Computing Systems. *IEEE J. Sel. Top. Signal Process.* **2022**, *16*, 460–473. [Google Scholar][CrossRef]
6. Liu, S.; Yu, Y.; Guo, L.; Yeoh, P.L.; Vucetic, B.; Li, Y.; Duong, T.Q. Satisfaction-Maximized Secure Computation Offloading in Multi-Eavesdropper MEC Networks. *IEEE Trans. Wirel. Commun.* **2022**, *21*, 4227–4241. [Google Scholar][CrossRef]
7. Badri, H.; Bahreini, T.; Grosu, D.; Yang, K. Energy-Aware Application Placement in Mobile Edge Computing: A Stochastic Optimization Approach. *IEEE Trans. Parallel. Distrib. Syst.* **2020**, *31*, 909–922. [Google Scholar][CrossRef]
8. Ma, Z.; Zhang, S.; Chen, Z.; Han, T.; Qian, Z.; Xiao, M.; Chen, N.; Wu, J.; Lu, S. Towards Revenue-Driven Multi-User Online Task Offloading in Edge Computing. *IEEE Trans. Parallel. Distrib. Syst.* **2022**, *33*, 1185–1198. [Google Scholar][CrossRef]
9. Kishor, A.; Chakarbarty, C. Task Offloading in Fog Computing for Using Smart Ant Colony Optimization. *Wirel. Pers. Commun.* **2021**, *127*, 1683–1704. [Google Scholar][CrossRef]
10. Chen, Y.; Zhang, S.; Jin, Y.; Qian, Z.; Xiao, M.; Ge, J.; Lu, S. LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment. *IEEE Trans. Parallel. Distrib. Syst.* **2022**, *33*, 1581–1592. [Google Scholar][CrossRef]
11. Babar, M.; Khan, M.S.; Din, A.; Ali, F.; Habib, U.; Kwak, K.S.; Cai, N. Intelligent Computation Offloading for IoT Applications in Scalable Edge Computing Using Artificial Bee Colony Optimization. *Complexity* **2021**, *2021*, 5563531. [Google Scholar][CrossRef]
12. Dong, S.; Xia, Y.; Kamruzzaman, J. Quantum Particle Swarm Optimization for Task Offloading in Mobile Edge Computing. *IEEE Trans. Industr. Inform.* **2023**, *19*, 9113–9122. [Google Scholar][CrossRef]
13. Jiang, F.; Wang, K.; Dong, L.; Pan, C.; Xu, W.; Yang, K. Deep-Learning-Based Joint Resource Scheduling Algorithms for Hybrid MEC Networks. *IEEE Internet Things J.* **2020**, *7*, 6252–6265. [Google Scholar][CrossRef]
14. Chen, C.; Chen, L.; Liu, L.; He, S.; Yuan, X.; Lan, D.; Chen, Z. Delay-Optimized V2V-Based Computation Offloading in Urban Vehicular Edge Computing and Networks. *IEEE Access* **2020**, *8*, 18863–18873. [Google Scholar][CrossRef]
15. Li, J.; Liang, W.; Xu, W.; Xu, Z.; Jia, X.; Zhou, W.; Zhao, J. Maximizing User Service Satisfaction for Delay-Sensitive IoT Applications in Edge Computing. *IEEE Trans. Parallel. Distrib. Syst.* **2022**, *33*, 1199–1212. [Google Scholar][CrossRef]
16. Braud, T.; Pengyuan, Z.; Kangasharju, J.; Pan, H. Multipath computation offloading for mobile augmented reality. In Proceedings of the 2020 IEEE International Conference on Pervasive Computing and Communications (PerCom), Austin, TX, USA, 23–27 March 2020; pp. 1–10. [Google Scholar]
17. Tang, M.; Wong, V.W.S. Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems. *IEEE Trans. Mob. Comput.* **2022**, *21*, 1985–1997. [Google Scholar][CrossRef]
18. Lv, Z.; Chen, D.; Wang, Q. Diversified Technologies in Internet of Vehicles Under Intelligent Edge Computing. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 2048–2059. [Google Scholar][CrossRef]
19. Li, W.; Chen, Z.; Gao, X.; Liu, W.; Wang, J. Multimodel Framework for Indoor Localization Under Mobile Edge Computing Environment. *IEEE Internet Things J.* **2019**, *6*, 4844–4853. [Google Scholar][CrossRef]
20. Hao, H.; Xu, C.; Zhong, L.; Muntean, G.M. A multi-update deep reinforcement learning algorithm for edge computing service offloading. In Proceedings of the 28th ACM International Conference on Multimedia, Seattle, WA, USA, 12–16 October 2020; pp. 3256–3264. [Google Scholar]
21. Wu, G.; Zhao, Y.; Shen, Y.; Zhang, H.; Shen, S.; Yu, S. DRL-based Resource Allocation Optimization for Computation Offloading in Mobile Edge Computing. In Proceedings of the IEEE INFOCOM 2022—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), New York, NY, USA, 2–5 May 2022; pp. 1–6. [Google Scholar]
22. Zhao, P.; Tian, H.; Qin, C.; Nie, G. Energy-Saving Offloading by Jointly Allocating Radio and Computational Resources for Mobile Edge Computing. *IEEE Access* **2017**, *5*, 11255–11268. [Google Scholar][CrossRef]
23. Zhuang, W.; Xing, F.; Lu, Y. Task Offloading Strategy for Unmanned Aerial Vehicle Power Inspection Based on Deep Reinforcement Learning. *Sensors* **2024**, *24*, 2070. [Google Scholar][CrossRef]
24. Xiao, Z.; Shu, J.; Jiang, H.; Min, G.; Chen, H.; Han, Z. Perception Task Offloading With Collaborative Computation for Autonomous Driving. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 457–473. [Google Scholar][CrossRef]
25. Li, Q.; Wang, S.; Zhou, A.; Ma, X.; Yang, F.; Liu, A.X. QoS Driven Task Offloading with Statistical Guarantee in Mobile Edge Computing. *IEEE Trans. Mob. Comput.* **2020**, *21*, 278–290. [Google Scholar][CrossRef]
26. Kim, T.; Sathyanarayana, S.D.; Chen, S.; Im, Y.; Zhang, X.; Ha, S.; Joe-Wong, C. MoDEMS: Optimizing Edge Computing Migrations for User Mobility. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 675–689. [Google Scholar][CrossRef]
27. Lim, D.; Lee, W.; Kim, W.T.; Joe, I. DRL-OS: A Deep Reinforcement Learning-Based Offloading Scheduler in Mobile Edge Computing. *Sensors* **2022**, *22*, 9212. [Google Scholar][CrossRef]

28. Cao, C.; Su, M.; Duan, S.; Dai, M.; Li, J.; Li, Y. QoS-Aware Joint Task Scheduling and Resource Allocation in Vehicular Edge Computing. *Sensors* **2022**, *22*, 9340. [Google Scholar][CrossRef]
29. Wang, H.; Xu, H.; Huang, H.; Chen, M.; Chen, S. Robust Task Offloading in Dynamic Edge Computing. *IEEE Trans. Mob. Comput.* **2023**, *22*, 500–514. [Google Scholar][CrossRef]
30. Jiang, H.; Dai, X.; Xiao, Z.; Iyengar, A. Joint Task Offloading and Resource Allocation for Energy-Constrained Mobile Edge Computing. *IEEE Trans. Mob. Comput.* **2023**, *22*, 4000–4015. [Google Scholar][CrossRef]
31. Qiu, X.; Zhang, W.; Chen, W.; Zheng, Z. Distributed and Collective Deep Reinforcement Learning for Computation Offloading: A Practical Perspective. *IEEE Trans. Parallel. Distrib. Syst.* **2021**, *32*, 1085–1101. [Google Scholar][CrossRef]
32. Zou, J.; Hao, T.; Yu, C.; Jin, H. A3C-DO: A Regional Resource Scheduling Framework Based on Deep Reinforcement Learning in Edge Scenario. *IEEE Trans. Comput.* **2021**, *70*, 228–239. [Google Scholar][CrossRef]
33. Alam, M.G.R.; Hassan, M.M.; Uddin, M.Z.; Almogren, A.; Fortino, G. Autonomic computation offloading in mobile edge for IoT applications. *Future Gener. Comput. Syst.* **2019**, *90*, 149–157. [Google Scholar][CrossRef]
34. Ren, J.; Wang, H.; Hou, T.; Zheng, S.; Tang, C. Federated Learning-Based Computation Offloading Optimization in Edge Computing-Supported Internet of Things. *IEEE Access* **2019**, *7*, 69194–69201. [Google Scholar][CrossRef]
35. Cao, S.; Chen, S.; Chen, H.; Zhang, H.; Zhan, Z.; Zhang, W. HCOME: Research on Hybrid Computation Offloading Strategy for MEC Based on DDPG. *Electronics* **2023**, *12*, 562. [Google Scholar][CrossRef]
36. Hao, H.; Xu, C.; Zhang, W.; Yang, S.; Muntean, G.M. Computing Offloading with Fairness Guarantee: A Deep Reinforcement Learning Method. *IEEE Trans. Circuits Syst. Video Technol.* **2023**, *33*, 6117–6130. [Google Scholar][CrossRef]
37. Hao, H.; Xu, C.; Zhang, W.; Yang, S.; Muntean, G.-M. Joint task offloading, resource allocation, and trajectory design for multi-uav cooperative edge computing with task priority. *IEEE Trans. Mob. Comput.* 2024, *in press*.[Google Scholar][CrossRef]
38. Li, N.; Zhu, X.; Li, Y.; Wang, L.; Zhai, L. Service Caching and Task Offloading of Internet of Things Devices Guided by Lyapunov Optimization. In Proceedings of the 2022 IEEE ISPA/BDCloud/SocialCom/SustainCom, Melbourne, Australia, 17–19 December 2022. [Google Scholar]
39. Wu, W.; Yang, P.; Zhang, W.; Zhou, C.; Shen, X. Accuracy-Guaranteed Collaborative DNN Inference in Industrial IoT via Deep Reinforcement Learning. *IEEE Trans. Ind. Inform.* **2021**, *17*, 4988–4998. [Google Scholar][CrossRef]
40. Tang, H.; Wu, H.; Qu, G.; Li, R. Double Deep Q-Network Based Dynamic Framing Offloading in Vehicular Edge Computing. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 1297–1310. [Google Scholar][CrossRef]
41. Chen, G.; Zhou, Y.; Xu, X.; Zeng, Q.; Zhang, Y.D. A multi-aerial base station assisted joint computation offloading algorithm based on D3QN in edge VANETs. *Ad Hoc Netw.* **2023**, *142*, 103098. [Google Scholar]
42. Liu, W.; Li, B.; Xie, W.; Dai, Y.; Fei, Z. Energy Efficient Computation Offloading in Aerial Edge Networks With Multi-Agent Cooperation. *IEEE Trans. Wirel. Commun.* **2023**, *22*, 5725–5739. [Google Scholar][CrossRef]