Dynamic Task Offloading Strategy for Multi-Agent Deep Reinforcement Learning Based on Lyapunov

Yang ZheXing
Guilin University of Technology
Xie XiaoLan
237290696@qq.com
Guilin University of Technology
Guo Qian
Guilin University of Technology
Tan ShuRu
Guilin University of Technology

Research Article

Keywords: Edge Computing, Task Offloading, Lyapunov, Deep Reinforcement Learning

Posted Date: June 5th, 2024

DOI: https://doi.org/10.21203/rs.3.rs-4447725/v1

Additional Declarations: No competing interests reported.

Dynamic Task Offloading Strategy for Multi-Agent Deep Reinforcement Learning Based on Lyapunov

Yang ZheXing[1], Xie XiaoLan[1,2], Guo Qian[1,2], Tan ShuRu[1]

[1]School of Computer Science and Engineering, Guilin University of Technology, Guilin, 541006, China.
[2]Key Laboratory of Embedded Technology and Intelligent System of Guangxi, Guilin University of Technology, Guilin, 541006, China.

Contributing authors: 294036881@qq.com; 237290696@qq.com; guoqiang121@163.com; 1020210976@glut.edu.cn;

Abstract

Aiming at the issue of communication overload and task backlog caused by the dynamic nature of the environment and the surge in the number of mobile devices in multi-user Mobile Edge Computing scenarios, a dynamic task offloading strategy based on Lyapunov-guided multi-agent deep reinforcement learning is proposed. This strategy aims to ensure long-term system stability while minimizing the average task processing latency and energy consumption. First, the dependency relationships among subtasks are modeled using a directed acyclic graph, and this is expressed as an optimization problem to minimize task offloading costs with long-term constraints. Then, using Lyapunov optimization theory, the long-term average problem is decoupled into deterministic problems for each time slot. Finally, the issue is further transformed into an optimal policy problem under a Markov Decision Process framework, and solved using the designed Lyapunov Multi-Agent Deep Deterministic Policy Gradient (Ly-MADDPG) algorithm. The simulation experiment results indicate that compared to the alternative algorithms, our proposed method reduces the task offloading cost while ensuring queue stability.

Keywords: Edge Computing, Task Offloading, Lyapunov, Deep Reinforcement Learning

1 Introduction

In recent years, there has been a rapid growth of intelligent terminal devices connected to wireless networks, driven by the development of cloud computing and the Internet of Things (IoT) [1]. Due to the limited capabilities of mobile devices and ever-increasing computing demands, it has become increasingly difficult to perform all tasks locally, especially for applications with ultralow latency and high resource consumption requirements, such as gesture, face recognition, health monitoring, and autonomous driving [2]. Conventional centralized cloud computing systems are facing significant challenges, including data transmission latency, excessive energy consumption, and low coverage [3].

Mobile edge computing (MEC) was introduced to address these challenges. By deploying servers at the network edge, close to users' intelligent terminal devices, MEC brings storage and computing capabilities from cloud data centres closer to the edge of user devices [4]. This enables users to offload some or all computationally intensive tasks to MEC hosts, effectively reducing task processing latency and device energy consumption and enhancing the quality of experience (QoE)[5].

Despite the many benefits of MEC, it also confronts several challenges, among them the pivotal concern of offloading computational tasks. An efficient offloading strategy can effectively reduce user latency and energy consumption and improve MEC system performance [6]. Many existing studies typically employ optimization methods based on heuristic algorithms. However, the optimization problem of task offloading often involves solving a mixed integer nonlinear programming (MINLP) problem [7], which is an NP-hard combinatorial optimization problem. Traditional heuristic optimization methods struggle to obtain optimal offloading decisions and have high computational complexity [8]. Furthermore, due to the dynamic nature of the system, a significant amount of human effort and expertise is required to adjust these heuristic models to adapt to new scenarios. As the complexity of MEC applications and system architectures increases, achieving long-term goal optimization while maintaining system stability is challenging due to its discrete and large action spaces [9]. Thus, making task offloading decisions in real time in complex dynamic multiuser MEC systems to efficiently utilize available computing resources and effectively perform complex task computations is a pressing issue, and its solution would have practical application value.

Recent advancements in deep reinforcement learning (DRL) have provided promising alternatives for addressing task offloading problems in mobile edge environments [10]. DRL uses deep neural networks (DNNs) to directly learn the optimal mapping from states to actions, maximizing rewards through interactions with the environment. It can effectively address complex decision-making problems with large-scale, high-dimensional state/action spaces and can dynamically learn from past experiences without the need for manually labelled training data samples. To date, DRL has achieved significant breakthroughs in various fields, including image processing [11], vehicular networks [12], and network management [13]. This has inspired further research into designing DRL-based methods for solving the optimization problem of task offloading decisions in mobile edge environments.

As shown in Fig 1, this paper examines the issue of dynamic task offloading strategy within a multiuser MEC scenario. We propose a Lyapunov-guided multiagent deep deterministic policy gradient (Ly-MADDPG) algorithm, which aims to minimize the average task processing latency and energy consumption while ensuring the stability of IoT device queues, thereby reducing task offloading costs. The main contributions of this study can be summarized as follows.

(1) We define a dynamic task offloading problem with latency constraints in a multiuser scenario, where both tasks and network states dynamically change in each time slot. The objective of the problem is to minimize the average task offloading cost while ensuring the long-term stability of the task queue.

(2) We model this problem as an optimization problem with long-term constraints, reflecting the limited computing resources in mobile devices and MEC, as well as the dependency requirements of subtasks in different applications. Then, by applying Lyapunov optimization theory, we transform the original optimization problem into a deterministic problem that only depends on the system state in the current time slot, and we further derive the upper bound of the Lyapunov drift plus the penalty function.

(3) Based on the above problem, we further transform the task offloading problem into an optimal policy problem that can be solved by the Markov decision process (MDP) by defining the state space, action space, and reward. We design a Ly-MADDPG algorithm, which can better handle the multiuser joint optimization problem.

(4) Through simulation experiments, we verify the effectiveness of the Ly-MADDPG algorithm and demonstrate its superiority to other algorithms.

This paper is organized as follows. In Section 2, we discuss related work. Section 3 introduces the system model and optimization problem description in the context of multiuser IoT-Edge scenarios. In Section 4, we transform the optimization problem and analyse its performance boundaries. Section 5 presents a solution based on deep reinforcement learning. Simulation results are provided in Section 6. Finally, Section 7 offers the conclusions of the paper.

2 Related Work

As an extension of cloud computing, mobile edge computing has garnered extensive research attention from both academia and industry due to its beneficial characteristics, such as proximity, low latency, and high bandwidth. By offloading computational tasks that cannot be locally supported to edge servers, the execution efficiency of these tasks can be significantly improved. In recent years, a variety of algorithms and frameworks have been proposed to address the optimization problem of offloading strategies in edge environments[14]. These solutions are primarily based on conventional offloading-decision approaches and intelligent offloading-decision approaches.

Geng et al. [15] considered the performance and energy trade-offs of computational offloading based on multicore systems, formulated the offloading problem as an NP-hard mixed integer nonlinear programming problem, and proposed a solution based on the critical path to address offloading decisions and task scheduling problems, effectively reducing device energy consumption. Yang et al. [16] considered competition

for the limited computing and communication resources of edge servers in multiuser environments and proposed a PFM-H algorithm that allocates data flow applications between mobile devices and edge servers to maximize system throughput. Liu et al. [17] proposed a one-dimensional search algorithm for task scheduling under the constraint of average energy consumption on mobile devices, considering the task queuing delay and incorporating Markov chain theory, which significantly reduced the average delay in various scenarios. Cui et al. [18] balanced energy consumption and latency to meet the diverse user needs of IoT applications, formulated the problem as a constrained multiobjective optimization problem, and solved it using an improved fast elite nominated sorting genetic algorithm II (NSGA-II), obtaining optimal solutions that reduced task computational energy consumption and latency.

The above work has provided important references for optimizing future task offloading strategies, considering different optimization objectives such as energy consumption, latency, and service response time. However, traditional offloading algorithms often require exponential iterations to obtain results, and it is challenging to find optimal solutions in increasingly complex and varied edge computing offloading problems. Deep reinforcement learning , which combines the characteristics of deep learning and reinforcement learning, has been applied to solve MEC task offloading problems. Huang et al. [19] proposed a DRL-based online offloading framework that learns binary offloading decisions from experience, eliminating the need to solve combinatorial optimization problems, greatly reducing computational complexity, and maximizing the weighted sum of computation rates for all user devices. Liu et al. [20] redefined the state space, action space, and rewards, transforming dependency task offloading into an optimal policy issue under the Markov decision process. By proposing an efficient dependency-aware task offloading scheme based on the deep deterministic policy gradient (DDPG), they leveraged the perceptual capabilities of deep learning and the decision-making abilities of reinforcement learning. This effectively reduces the average processing delay of tasks. Zou et al. [21] used an improved actor-critic (AC) algorithm in the deep deterministic policy gradient (DDPG) algorithm to balance edge server workloads and reduce task energy consumption and computation time. Wang et al. [22] combined DRL algorithms and sequence-to-sequence (seq2seq) neural networks to solve dependent task offloading problems and proposed the use of meta-reinforcement learning to enhance the adaptability of computation offloading to the environment, demonstrating a reduction in computation latency under various data transmission rates and task numbers.

However, mobile edge computing environments often involve a significant number of users and mobile devices. The previously mentioned deep reinforcement learning methods, which employ centralized decision-making, can lead to excessive communication overhead and an enlarged state space. To address these issues, multiagent reinforcement learning (MARL) has come into play. With centralized training and distributed execution, each agent can make independent decisions by observing the local environment, thereby achieving improved performance. Cao et al. [23] investigated a multichannel access and task offloading problem for MEC-enabled Industry 4.0. They proposed a multiagent deep reinforcement learning (MADRL) approach, which enables edge devices (EDs) to collaborate effectively. This approach significantly enhances the

successful access rate of channels in multiuser environments while concurrently reducing the computational delay of user tasks. Yang et al. [24] proposed a priority-driven cooperative task offloading algorithm based on multiagent deep reinforcement learning supplemented with a variational recurrent neural network (VRNN)-based global state sharing model. This model aims to reduce task processing time and enhance the utilization of edge computing resources.

The aforementioned literature employed DRL methods to solve dependent task offloading in MEC and investigated various optimization objectives. However, most DRL-based studies did not impose long-term performance constraints to ensure system stability and could not make dynamic offloading decisions based on the current time slot's state. Additionally, these studies assumed that each user only paid attention to their quality of experience (QoE) and not the total energy consumption of the system.

Based on existing research, this paper studies the dynamic task offloading problem for applications with dependencies in multiuser MEC systems. We consider the dynamic nature of the environment, system stability, and latency constraints of tasks and formulate this as a nonlinear optimization problem of minimizing task offloading costs with long-term constraints. We propose a dynamic task offloading strategy based on Lyapunov-guided multiagent deep reinforcement learning. First, we transform the problem into a deterministic problem for each time slot based on Lyapunov optimization theory and then into a Markov decision process. We solve this problem using Ly-MADDPG algorithm, which reduces task latency and mobile device energy consumption while ensuring queue stability.

3 System Model and Formal Description

3.1 System Overview

As shown in Fig 1, we consider a multiuser IoT-Edge scenario featuring a MEC server and a set of M mobile devices. Our goal is to efficiently utilize the computational resources of the mobile edge server to reduce user task latency and energy consumption while ensuring that deadline requirements are met, thereby enhancing system stability and user satisfaction.

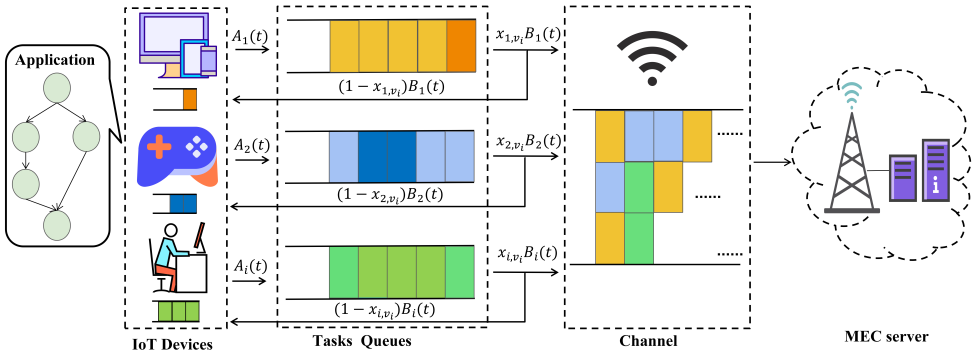


Fig. 1 Multi-user IoT-Edge scenario

Let $M = \{1, 2, \ldots, \mathrm{m}\}$ represent the set of mobile devices, and assume a discrete-time model for time; define a multiuser IoT-Edge system as a slotted system, with $\mathrm{t} \in \mathrm{T} = \{1, 2, \cdots\cdots, \mathrm{T}\}$ divided into T equal-length parts, each with slot length τ. In our scenario, we assume that each mobile device (MD) runs an application composed of a sequence of tasks with interdependencies. We posit that all application tasks generated by the mobile device can be decomposed into smaller subtasks. Each subtask can either be processed locally on the device or offloaded to the MEC server for execution.

We define an offloading decision set $\boldsymbol{\alpha}(t) = \left\{\alpha_{v_{m,k,0}}(t), \alpha_{v_{m,k,1}}(t), \ldots, \alpha_{v_{m,k,i+1}}(t)\right\}$ to represent the task offloading indicator sets within a given time slot. The element $\alpha_{v_{m,k,i}}(t)$ is a binary decision variable denoting that tasks in the time slot are offloaded to the edge server if $\alpha_{v_{m,k,i}}(t) = 1$; otherwise, $\alpha_{v_{m,k,i}}(t) = 0$.

To obtain the optimal offloading strategy and improve user experience, we need to carefully consider the dependencies among tasks and the impact of various processing options (local or MEC) on latency, energy consumption, and system stability.

3.2 Task and Queuing Model

In practice, many applications consist of multiple subtasks with dependencies. For an application $k \in \mathcal{M}$ originating from a mobile device, we describe its task dependencies using a directed acyclic graph (DAG) $G_{m,k}(t)$. We utilize $V_{m,k}(t)$ to signify the set of tasks within $G_{m,k}(t)$ and use $v_{m,k,i}(t)$ to represent the i-th task in $G_{m,k}(t)$. To offer a more generalized task model, we introduce two virtual tasks: $v_{m,k,0}(t)$ (for input) and $v_{m,k,i+1}(t)$ (for output).Task offloading from time slot is illustrated in Fig 2.

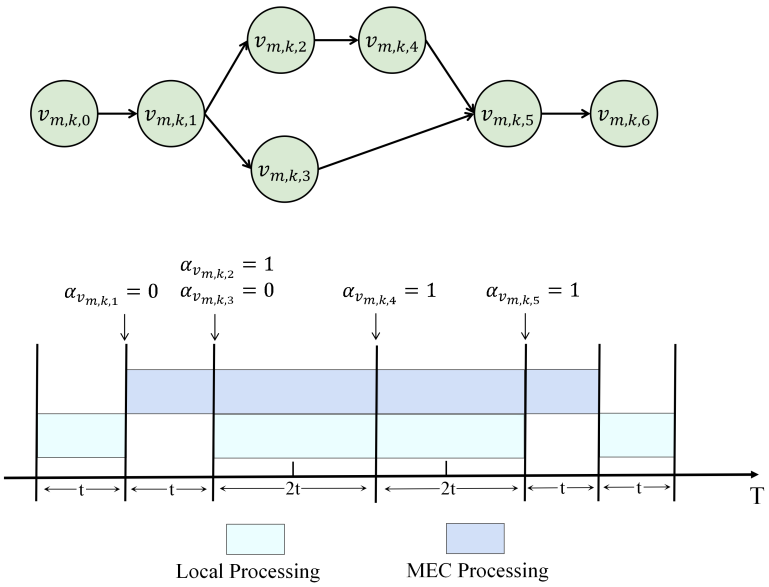


Fig. 2 Task offloading from time slot

For each subtask, we define it as $v_{m,k,i}(t) = \left[c_{v_{m,k,i}}(t), n_{v_{m,k,i}}(t), d_{v_{m,k,i}}(t)\right]$,where $c_{v_{m,k,i}}(t)$ represents the required computational resources for the task (e.g., CPU cycles), $n_{v_{m,k,i}}(t)$ denotes the data size of the task, and $d_{v_{m,k,i}}(t)$ is the maximum tolerable latency time. A directed edge (j, i) in the DAG of $G_{m,k}(t)$ is used to describe the dependency relationship between tasks, indicating that $v_{m,k,i}(t)$ must be executed after $v_{m,k,j}(t)$ is completed. Simultaneously, the set of direct predecessors for task $v_{m,k,i}(t)$ is represented as $pred\left(v_{m,k,i}(t)\right)$.

Each IoT device has a virtual queue to store arriving but unprocessed computational tasks. In each time slot, MDs generate mobile applications, such as real-time video processing, smart home applications, and certain augmented reality (AR) applications. The volume of tasks arriving at mobile device $m \in M$ is represented as $A_m(t)$, and the arrival process of computational tasks is independently and identically distributed. We define the processed computational tasks as $B_m(t)$, which consists of tasks completed locally and tasks offloaded to the MEC end, represented as $B_m(t) = B_{local,m}(t) + B_{mec,m}(t)$. Therefore, the task queue backlog at time slot t is defined as $Q_m(t) = \{Q_0(t), Q_1(t), \ldots, Q_m(t)\}$, which changes over time according to the following formula:

$$Q_m(t+1) = \max\left[Q_m(t) - B_m(t) + A_m(t), 0\right] \tag{1}$$

A discrete-time process $Q_m(t)$ is considered stable if:

$$\lim_{t\to\infty} \frac{\mathbb{E}\left\{|Q_m(t)|\right\}}{t} = 0 \tag{2}$$

All task queues for each mobile device must satisfy Eq. (2). Therefore, the queues will not grow indefinitely, ensuring that the system remains relatively stable.

3.3 Task Transmission Model

We assume that the base station's radio access system is based on orthogonal frequency-division multiple access (OFDMA). There are j channels within the system, which we denote as $J = \{1, 2, \ldots, j\}$ Each channel can connect to a single user device. Once the user device initially occupying the channel completes its transmission, other user devices can connect to the channel and offload tasks through it.

For each Internet of Things (IoT) device m, its transmission power is defined as $p_{m,j}(t)$, and the channel gain for the time slot is represented by $h_{m,j}(t)$. Then, according to Shannon's theorem, the transmission rate of mobile device k offloading tasks to the edge server through the j-th channel can be obtained as:

$$r_m(t) = B\log_2\left(1 + \frac{p_{m,j}(t)h_{m,j}(t)}{\tau^2}\right) \tag{3}$$

where τ^2 denotes the Gaussian noise power and B denotes the channel bandwidth.

3.4 Computational Model

When tasks are processed locally, the delay consists of the task processing time and the waiting time for readiness. Task $v_{m,k,i}$ can be executed only after its predecessor tasks $pred\left(v_{m,k,i}(t)\right)$ have been completed. We use $RT^{local}_{v_{m,k,i}}(t)$ to represent the waiting time for readiness at the local level, and $FT^{local}_{v_{m,k,i}}(t)$ denotes the completion time of $v_{m,k,i}$ when executed locally. Since tasks can be processed locally or offloaded to the MEC end and there may be more than one predecessor node, $RT^{local}_{v_{m,k,i}}(t)$ can be represented as:

$$RT^{local}_{v_{m,k,i}}(t) = \max_{v_{m,k,j} \in pred(v_{m,k,i})} \left\{ \max \left\{ FT^{local}_{v_{m,k,j}}(t), FT^{mec}_{v_{m,k,j}}(t) \right\} \right\} \tag{4}$$

Considering the processing capabilities of mobile user terminals and task computation requirements, the local execution delay can be expressed as:

$$T^{local}_{v_{m,k,i}}(t) = \frac{c_{v_{m,k,i}}(t)}{f_{local}} \tag{5}$$

where $c_{v_{m,k,i}}(t)$ is the amount of computation for task $v_{m,k,i}$ and f_{local} is the local computing capability.

The completion delay for tasks processed locally can thus be obtained as:

$$FT^{local}_{v_{m,k,i}}(t) = T^{local}_{v_{m,k,i}}(t) + RT^{local}_{v_{m,k,i}}(t) \tag{6}$$

The energy consumption for executing tasks locally can be expressed as:

$$E^{local}_{v_{m,k,i}}(t) = \kappa c_{v_{m,k,i}}(t) \left(f_{local}\right)^2 \tag{7}$$

where κ is the energy consumption coefficient.

The time spent offloading computational tasks to the MEC server for execution mainly consists of the computation time, transmission time, and waiting time for readiness.

The transmission delay can be expressed as:

$$T^{tran}_{v_{m,k,i}}(t) = \frac{n_{v_{m,k,i}}(t)}{r_m(t)} \tag{8}$$

The MEC-end execution delay can be expressed as:

$$T^{edge}_{v_{m,k,i}}(t) = \frac{c_{v_{m,k,i}}(t)}{f_{edge}} \tag{9}$$

where f_{edge} reflects the MEC computing capability.

The waiting time for task readiness at the MEC can be expressed as:

$$RT^{mec}_{v_{m,k,i}}(t) = \left\{ \max_{v_{m,k,j} \in pred(v_{m,k,i})} \left\{ FT^{local}_{v_{m,k,j}}(t), FT^{mec}_{v_{m,k,j}}(t) \right\} + T^{tran}_{v_{m,k,i}}(t) \right\} \tag{10}$$

The completion delay for tasks processed at the MEC server can thus be obtained as:

$$FT^{mec}_{v_{m,k,i}}(t) = T^{edge}_{v_{m,k,i}}(t) + RT^{mec}_{v_{m,k,i}}(t) \tag{11}$$

The energy consumption for offloading tasks to the MEC server for execution can be expressed as:

$$E^{mec}_{v_{k,m,i}}(t) = p_{m,j}(t) T^{tran}_{v_{m,k,i}}(t) \tag{12}$$

3.5 Problem Formulation

For the k-th DAG application from mobile device m, we define the offloading cost as:

$$C_{m,k}(t) = \gamma_1 \left(FT_{v_{m,k,i+1}}(t) - FT_{v_{m,k,0}}(t) \right) + \gamma_2 E_{m,k}(t) \tag{13}$$

$$\begin{aligned} E_{m,k}(t) = & \underbrace{\sum_{v_{m,k,i} \in V_{m,k}} \left(1 - \alpha_{v_{m,k,i}}(t)\right) E^{local}_{v_{m,k,i}}(t)}_{\text{local device}} \\ & + \underbrace{\sum_{v_{m,k,i} \in V_{m,k}} \alpha_{v_{m,k,i}}(t) E^{mec}_{v_{m,k,i}}(t)}_{\text{MEC server}} \end{aligned} \tag{14}$$

where γ_1 and γ_2 represent the weights corresponding to delay and energy consumption, respectively, and the term $E_{m,k}(t)$ denotes the total energy consumed for each application task.

This paper aims to consider both system resources and delay thresholds. By optimizing the offloading decisions, we aspire to achieve a relatively optimal value for the weighted average of energy consumption and delay while ensuring queue stability. Consequently, within each time slot, an optimal offloading strategy should be identified to minimize the average offloading cost in the system. The formal representation of this problem is as follows:

$$P1: \min \quad \mathbf{C} = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E} \left\{ \sum_{m=1}^{M} C_{m,k}(t) \right\} \tag{15}$$

$$\begin{aligned} \text{s.t.} \quad & C1: \ \alpha_{v_{m,k,i}}(t) \in \{0, 1\} \\ & C2: \ f_{\min} \leq f_{\text{local}}, f_{\text{edge}} \leq f_{\max} \\ & C3: \ FT_{v_{m,k,i}}(t) < d_{v_{m,k,i}}(t) \\ & C4: \ \max_{v_{m,k,j} \in pred(v_{m,k,i})} FT_{v_{m,k,j}}(t) \leq RT_{v_{m,k,i}}(t) \\ & C5: \ \lim_{t \to \infty} \frac{\mathbb{E}\{|\, Q_m(t)) \,|\}}{t} = 0 \end{aligned}$$

C1 indicates that each job can be processed only locally or at the edge; C2 requires the maximum computing capability of the cloud-edge end to be greater than the minimum capability needed for processing a job; C3 requires the job completion time

to be less than the job deadline; C4 represents the task dependency, indicating that a task can be processed only after all its predecessors are completed; and C5 ensures the long-term stability of all task queues.

4 Problem Transformation

4.1 Lyapunov optimization

Since problem P1 involves long-term average optimization in both the objective function and constraints, it is difficult for traditional optimization methods to solve it. P1 can be described as a Markov decision process. In contrast to MDP methods, Lyapunov optimization uses the current system state information to make optimal offloading decisions without relying on future information. It can decouple the long-term optimization objective into solving the optimal value for each time slot, thus optimizing offloading costs while maintaining queue stability.

According to Lyapunov optimization theory, we define the Lyapunov function as:

$$L\left(Q_m(t)\right) \triangleq \frac{1}{2} \sum_{m=1}^{M} Q_m(t)^2 \tag{16}$$

To ensure queue stability, we need to minimize the task queue backlog in each time slot t. We use $\Delta(Q_m(t))$ to represent the queue backlog expected change for mobile devices from time slot t to time slot $(t+1)$ and define the Lyapunov drift function as:

$$\Delta\left(Q_m(t)\right) \triangleq \mathbb{E}\left\{L\left(Q_m(t+1)\right) - L\left(Q_m(t)\right) \mid Q_m(t)\right\} \tag{17}$$

$\Delta\left(Q_m(t)\right)$ denotes the expected change in the Lyapunov function for a given time slot and serves as a crucial metric for maintaining system stability.

Lemma 1: Given the dynamic relationship of the queues in the system, when $E\left\{L\left(Q_m(t)\right)\right\} < \infty$ and there exist constants $\delta > 0$, $\varepsilon > 0$, we can obtain an upper bound for the Lyapunov drift $\Delta(Q_m(t))$ as:

$$\Delta\left(Q_m(t)\right) \leq \delta - \varepsilon \sum_{m=1}^{M} Q_m(t) \tag{18}$$

Proof: By squaring Equation (1) and using $\{\max[x, 0]\}^2 \leq x^2$, we obtain:

$$\begin{aligned} Q_m(t+1)^2 &= \left\{\max\left[Q_m(t) - B_m(t) + A_m(t), 0\right]\right\}^2 \\ &\leq \left(Q_m(t) - B_m(t) + A_m(t)\right)^2 \end{aligned} \tag{19}$$

By substituting Equation (1) and Equation (19) into Equation (17) and expanding, we obtain:

$$\begin{aligned}\Delta\left(Q_m(t)\right) =& \frac{1}{2}\sum_{m=1}^{M}\mathbb{E}\left[Q_m(t+1)^2 \mid Q_m(t)\right] \\ & -\frac{1}{2}\sum_{m=1}^{M}E\left[Q_m(t)^2 \mid Q_m(t)\right] \\ \leq & \frac{1}{2}\sum_{m=1}^{M}\mathbb{E}\Big[\left(Q_m(t)-B_m(t)+A_m(t)\right)^2 \\ & -Q_m(t)^2 \mid Q_m(t)\Big] \\ \leq & \frac{1}{2}\sum_{m=1}^{M}\mathbb{E}\left[\left(A_m(t)^2+B_m(t)^2\right) \mid Q_m(t)\right] \\ & +\sum_{m=1}^{M}\mathbb{E}\left[Q_m(t)\left(A_m(t)-B_m(t)\right) \mid Q_m(t)\right] \\ \leq & \delta+\sum_{m=1}^{M}\mathbb{E}\left[Q_m(t)\left(A_m(t)-B_m(t)\right) \mid Q_m(t)\right]\end{aligned} \tag{20}$$

where δ is a constant and $\delta=\frac{1}{2}\sum_{m=1}^{M}\left(A_m^{\max}(t)^2+B_m^{\max}(t)^2\right)$ holds for all $m\in M$, as we have $A_m(t)\leqslant A^{\max}(t)$ and $B_m(t)\leqslant B^{\max}(t)$.

If $Q_m(t)$ is stable, then $\lim_{t\to\infty}\frac{\mathbb{E}\{|Q_m(t))|\}}{t}=0$. For time slot $t\in\{1,\ldots,\mathrm{T}\}$, the law of weighted sums can be derived as:

$$\mathbb{E}\left[L\left(Q_m(t)\right)\right]-\mathbb{E}\left[L\left(Q_m(0)\right)\right]\leq\delta t-\varepsilon\sum_{t=1}^{T}\sum_{m=1}^{M}\mathbb{E}\left[Q_m(t)\right] \tag{21}$$

Dividing both sides of the inequality by t, letting $t\to\infty$ when $Q_m(0)=0$, we can obtain:

$$\frac{1}{t}\sum_{t=1}^{T}\sum_{k=1}^{\infty}\mathbb{E}\left[Q_m(t)\right]<\frac{\delta}{\varepsilon} \tag{22}$$

This implies that the backlog of the task queue for all IoT devices has an upper limit and cannot grow indefinitely, thereby ensuring the stability of the system.

Our objective is to minimize the task offloading cost and ensure the stability of the queue. Therefore, based on the Lyapunov optimization method, we introduce a task offloading cost function (penalty function) related to the value of V. By adjusting the size of V, we can find a trade-off between the queue backlog and task offloading cost, thereby achieving a balance of dual-objective optimization on the basis of system

stability. We define the drift and penalty function for each time period as:

$$\Delta\left(Q_m(t)\right) + V\mathbb{E}\left\{\sum_{m=1}^{M} C_{m,k}(t) \mid Q_m(t)\right\} \tag{23}$$

The first term reduces the backlog of the queue and maintains the stability of the system, and the second term is our optimization objective. We seek the optimal solution by minimizing the upper bound of the Lyapunov drift and penalty function, and we substitute Equation (20) into Equation (23) to obtain:

$$\begin{aligned}
&\Delta\left(Q_m(t)\right) + V\mathbb{E}\left\{\sum_{m=1}^{M} C_{m,k}(t) \mid Q_m(t)\right\} \\
&\leq \delta + \sum_{m=1}^{M} \mathbb{E}\left[Q_m(t)\left(A_m(t) - B_m(t)\right) \mid Q_m(t)\right] \\
&+ V\mathbb{E}\left\{\sum_{m=1}^{M} C_{m,k}(t) \mid Q_m(t)\right\}
\end{aligned} \tag{24}$$

Thus, the long-term optimization problem P1 is transformed into the problem of finding an offloading strategy for each time slot that minimizes the drift plus penalty function within that time slot. The objective function P1 is converted into:

$$\begin{aligned}
P2: \min \mathcal{L}(t) = \mathbb{E}\Bigg[&\delta + \sum_{m=1}^{M} Q_m(t)\left(A_m(t) - B_m(t)\right) \\
&+ V\sum_{m=1}^{M} C_{m,k}(t)\Bigg]
\end{aligned} \tag{25}$$

Once all the P2 solutions for all time slots have been determined, the solution to P1 can be obtained. The optimal offloading strategy for each time period can be calculated based only on the offloading decisions made within that time period, without considering the offloading decisions made in other time periods. This greatly simplifies the optimization problem P1. The drift plus penalty function can ensure that the long-term constraint of a stable queue is continuously maintained; the system's average offloading cost is inversely proportional to the weight coefficient V, while the average queue backlog is directly proportional to V.

4.2 Performance Bound Analysis

In the Lyapunov drift and penalty function, in addition to considering queue stability, we need to consider the penalty function. We hope that the average of this penalty function over time is less than or approaches some optimal target value, that is, the minimum offloading cost.

Lemma 2: For any V > 0, the average offloading cost and the average queue backlog satisfy:

$$\lim_{t\to\infty}\sup\frac{1}{T}\sum_{t=1}^{T}\sum_{m=1}^{M}\mathbb{E}\left[C_{m,k}(t)\right]\leq\frac{\delta}{V}+C^{*} \tag{26}$$

$$\lim_{\mathrm{t}\to\infty}\sup\frac{1}{t}\sum_{t=1}^{T}\sum_{m=1}^{M}\mathbb{E}\left[Q_{m}(\tau)\right]\leq\frac{\delta+V\left(C^{*}-\bar{C}\right)}{\varepsilon} \tag{27}$$

where C^{*} is the optimal solution to the optimization problem, i.e., the minimum offloading cost, and $\overline{C}$ is the average offloading cost, which can approach the minimum offloading cost C^{*} by increasing V.

Proof: Define all the state information in time slot t as $s(\mathrm{t})$, where $\alpha(t)$ is the offloading decision based on the observed $s(\mathrm{t})$. Suppose there exists a feasible decision $\alpha^{*}(t)$ such that:

$$\mathbb{E}\left[C\left(\alpha^{*}(t),s(t)\right)\right]=C^{*}\mid \tag{28}$$

$$\mathbb{E}\left[\left(A_{m}\left(\alpha^{*}(t),s(t)\right)-B_{m}\left(\alpha^{*}(t),s(t)\right)\right]\leq 0 \tag{29}$$

If we minimize the right side of Equation (24), we can reduce the average offloading cost. Thus, we have:

$$\begin{aligned}
&\Delta\left(Q_{m}(t)\right)+V\mathbb{E}\left\{\sum_{m=1}^{M}C_{m,k}(t)\mid Q_{m}(t)\right\}\\
&\leq\delta+\sum_{m=1}^{M}\mathbb{E}\left[Q_{m}(t)\left(A_{m}(t)-B_{m}(t)\right)\mid Q_{m}(t)\right]\\
&+V\mathbb{E}\left\{\sum_{m=1}^{M}C_{m,k}(t)\mid Q_{m}(t)\right\}\\
&\leq\delta+\sum_{m=1}^{M}\mathbb{E}\left[Q_{m}(t)\left(A_{m}\left(\alpha^{*}(t)\right)-B_{m}\left(\alpha^{*}(t)\right)\mid Q_{m}(t)\right]\right.\\
&+V\mathbb{E}\left\{C^{*}\mid Q_{m}(t)\right\}\\
&\leq\delta+VC^{*}
\end{aligned} \tag{30}$$

For time slot $\mathrm{t}\in\{1,2,\ldots,\mathrm{T}\}$, from the scaling and summing law, we derive:

$$\begin{aligned}
L\left(Q_{m}(t)\right)-&L\left(Q_{m}(0)\right)+V\sum_{t=1}^{T}\mathbb{E}\left\{\sum_{m=1}^{M}C_{m,k}(t)\mid Q_{m}(t)\right\}\\
&\leq\delta T+VTC^{*}
\end{aligned} \tag{31}$$

Letting $Q_{m}(0)=0$, dividing both sides of the inequality by t and letting $t\to\infty$, we obtain:

$$\lim_{t\to\infty}\sup\frac{1}{T}\sum_{t=1}^{T}\sum_{m=1}^{M}\mathbb{E}\left[C_{m,k}(t)\right]\leq\frac{\delta}{V}+C^{*} \tag{32}$$

Rearranging the terms, we obtain:

$$\lim_{t\to\infty}\sup\frac{1}{T}\sum_{t=1}^{T}\sum_{m=1}^{M}\mathbb{E}\left[C_{m,k}(t)\right]-C^{*}\leq\frac{\delta}{V} \tag{33}$$

The rearranged formula also indicates that the gap between the average offloading cost and the optimal value is less than the given range $\frac{\delta}{V}$.

Assume there exists a decision $\alpha^{*}(t)$ satisfying:

$$\mathbb{E}\left[\left(A_{m}\left(\alpha^{*}(t),s(\mathrm{t})\right)-B_{m}\left(\alpha^{*}(t),s(\mathrm{t})\right)\right]\leq-\varepsilon \tag{34}$$

Based on Equations (18) and (24), we deduce:

$$\begin{aligned}
&\Delta\left(Q_{m}(t)\right)+V\mathbb{E}\left\{\sum_{m=1}^{M}C_{m,k}(t)\mid Q_{m}(t)\right\}\\
&\leq\delta+VC\left(\alpha^{*}(t),s(t)\right)\\
&\quad+\sum_{m=1}^{M}\mathbb{E}\left[Q_{m}(t)\left(A_{m}\left(\alpha^{*}(t),s(t)\right)-B_{m}\left(\alpha^{*}(t),s(t)\right)\right]\right.\\
&\leq\delta+VC^{*}-\varepsilon\sum_{k=1}^{K}\mathbb{E}\left[Q_{m}(t)\right]
\end{aligned} \tag{35}$$

Therefore, analogously, we can employ the scaling and summing law to obtain:

$$\begin{aligned}
\frac{1}{t}\sum_{t=1}^{T}\sum_{k=1}^{K}\mathbb{E}\left[Q_{m}(t)\right]&\leq\frac{\delta+V\left(C^{*}-\mathbb{E}\left\{\sum_{m=1}^{M}C_{m,k}(t)\mid Q_{m}(t)\right\}\right)}{\varepsilon}\\
&\leq\frac{\delta+V\left(C^{*}-\bar{C}\right)}{\varepsilon}
\end{aligned} \tag{36}$$

5 Deep Reinforcement Learning-Based Solutions

5.1 Construction of the Markov Decision Model

Given that P2 is an NP-hard problem, it is challenging to efficiently solve the optimization problem using conventional methods. Therefore, we adopt deep reinforcement learning to address the proposed problem. We model the aforementioned optimization problem as a Markov decision process. Each mobile device is modelled as an agent, and actions are taken based on local observations and interactions with the VEC system.

To solve the MDP, we employ an algorithm based on the Lyapunov-guided multiagent deep deterministic policy gradient.

Our proposed Ly-MADDPG algorithm dynamically adjusts the decision variables to devise effective task offloading strategies, thereby minimizing task offloading costs. The state space, action space, and reward function in the Markov decision process are expressed as:

(1) State Space

The state space of mobile device m at time t is $\boldsymbol{s}_m(t)$. At time slot t, agent k observes the network environment, such as the available communication resources $\boldsymbol{w}(t)$ and computing resources $\boldsymbol{f}(t)$, task arrival $\boldsymbol{v}(t)$, and current queue vector $\boldsymbol{Q}(t)$, represented as:

$$\boldsymbol{s}(t) = (\boldsymbol{s}_1(t), \boldsymbol{s}_2(t), \cdots, \boldsymbol{s}_m(t)) \tag{37}$$

$$\boldsymbol{s}_m(t) = [\boldsymbol{w}(t), \boldsymbol{f}(t), \boldsymbol{v}(t), \boldsymbol{Q}(t))] \tag{38}$$

(2) Action Space

Based on the current offloading policy and the corresponding state, each agent selects an action from its action space. These actions describe the offloading decisions of all agents. Thus, the action space at time t is:

$$\boldsymbol{a}(t) = (\boldsymbol{a}_1(t), \boldsymbol{a}_2(t), \cdots, \boldsymbol{a}_m(t)) \tag{39}$$

$$\boldsymbol{a}_m(t) = [a_{m,1}(t), a_{m,2}(t), \cdots, a_{m,i}(t)] \tag{40}$$

(3) Rewards

Rewards are feedback from the environment that is given after agents perform actions. Specifically, our optimization goal is to maintain system stability and reduce the average task offloading cost while adhering to the task deadline constraints. The purpose of reinforcement learning is to achieve the highest reward value, and the average offloading cost is inversely proportional to the reward value. Therefore, the system-wide reward function at time slot t can be defined as $r(t) = R(S(t), A(t)) = -\mathcal{L}(t)$.

5.2 DRL-Based Algorithm Design

The MDP can be solved by either traditional dynamic programming (DP) algorithms or DRL methods. Conventional DP methods typically need to recompute the optimal policy for every environmental change. In contrast, DRL can perform online learning through interactions with the environment, progressively improving the policy. This makes DRL more adaptable to nonstatic environments and those that evolve during the training process, making it more suitable for dynamic multiagent task offloading problems. In this chapter's multiuser IoT-Edge system, the agents' actions are deterministic continuous actions, and the joint optimization problem is a mixed cooperative setting. Thus, we employ the Ly-MADDPG algorithm to address problem P2.

Traditional DRL methods, such as DQN or policy gradient algorithms, are not suitable for multiagent environments. As training progresses, the policy for each agent changes, making the environment seem unstable from the perspective of any single agent. When multiple agents coordinate, policy gradient methods typically exhibit

high variance. MADDPG is a MADRL algorithm based on the actor-critic framework and consists of heterogeneous networks and individual agents. Each agent contains a value-based critic network $\boldsymbol{Q}$ and a policy-based actor network $\boldsymbol{\mu}$. The actor network comprises two subnetworks: the evaluation network $\mu_m\left(\boldsymbol{s}_m \mid \omega_m\right)$ and the target network $\mu_m'\left(\boldsymbol{s}_m \mid \omega_m'\right)$, where ω_m and ω_m' are the respective network parameters. Analogous to the actor network, the critic network also consists of an evaluation network $\mathcal{Q}_k\left(\boldsymbol{s}_m, \boldsymbol{a}_m \mid \theta_m'\right)$ and a target network $\mathcal{Q}_k'\left(\boldsymbol{s}_m, \boldsymbol{a}_m \mid \theta_m'\right)$, with respective network parameters, where θ_m and θ_m' represent the parameters of the respective networks. The MADDPG algorithm updates the networks and parameters through centralized training and decentralized execution methods.

During the centralized training phase, each agent selects an action $\boldsymbol{a}_m = \mu_m\left(\boldsymbol{s}_m \mid \omega_m\right)$ based on the actor network. To help prevent becoming stuck in local optima, noise N is introduced to enhance exploration. At each time slot t, once the actor network selects the action, the agent will execute this action and send the newly obtained reward and state to the critic network. Information from all agents is stored in the experience replay buffer D in the form of quadruplets $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s}')$ for centralized training.

When training these agents, a batch of samples B is drawn for parameter updates, where the j-th sample is denoted as $\left(\boldsymbol{s}^j, \boldsymbol{a}^j, \boldsymbol{r}^j, \boldsymbol{s}'^j\right)$. Each agent's critic network computes a centralized Q-value based on observations and actions from all agents. The critic network value is updated by minimizing a loss function, whose expression is as follows:

$$\operatorname{Loss}_m\left(\omega_m\right) \approx \frac{1}{B} \sum_{j=1}^{B}\left(y_m^j - \mathcal{Q}_m\left(s^j, a_1^j, \cdots, a_m^j \mid \omega_m\right)\right)^2 \tag{41}$$

where y_m^j is the critic target, represented by:

$$y_m^j = r_m^j + \gamma \mathcal{Q}_m'\left(s'^j, a_1'^j, \cdots, a_m'^j \mid \omega_m\right)\Big|_{a_m'=\mu_m'\left(s_m'^j\right)} \tag{42}$$

where γ is the discount factor. The actor network is updated by minimizing the policy gradient of the agent, which can be expressed as:

$$\begin{aligned} &\nabla_{\theta_{\mathrm{k}}} J\left(\mu_m\right) \approx \\ &\frac{1}{B} \sum_{j=1}^{B} \nabla_{\theta_m} \mu_m\left(s_m^j\right) \nabla_{a_m} \mathcal{Q}_m\left(S^j, a_1^j, \ldots, a_m^j\right)\Big|_{a_m=\mu_m\left(s_m'^j\right)} \end{aligned} \tag{43}$$

As the parameters of the current actor and critic networks continuously update, both the actor target network and the critic target network also need to be updated using a soft update method. The expression for this is:

$$\omega_m' = \tau \omega_m + (1-\tau)\omega_m' \tag{44}$$

$$\theta_m' = \tau \theta_m + (1-\tau)\theta_m' \tag{45}$$

During the decentralized execution phase, the critic network is not involved; only the trained actor network operates in an online mode. Based on the actor network, each agent can output actions according to its local observation state.

Algorithm 1 Ly-MADDPG

Initialization: Initialize the parameters of each agent?s actor and critic evaluations and target networks. Initialize the replay buffer B, learning rate, the discount factor, the maximum learning episode, steps.
for episode = 1 to M **do**
    Initialize a random noise N
    for t = 1 to T **do**
        Each agent k generates a noise-added action based on the current observed state $\boldsymbol{a}(t) = \boldsymbol{\mu_k}(\boldsymbol{s}_m \mid \boldsymbol{\omega}_m) + N(t)$
        Execute actions $\boldsymbol{a}(t) = (\boldsymbol{a}_1(t), \boldsymbol{a}_2(t), \ldots, \boldsymbol{a}_m(t))$, obtain the reward $\boldsymbol{r}_m(t)$, and the subsequent new state $\boldsymbol{s}'_m(t)$
        Input the new state $\boldsymbol{s}'_m(t)$ to each agent k
        Store the agent's information into the experience replay buffer B as a tuple of four elements (s, a, r, s')
        $\boldsymbol{s}_m = \boldsymbol{s}'_m$
        for agent k = 1 to K **do**
            Sample a random mini-batch of B samples from the replay buffer B
            Update the critic network according to the following loss function based on Eq. (43)
            Update the actor network using the sampled gradient based on Eq. (44)
        end for
        Update target network parameters for each agent with $\omega'_m = \tau\omega_m + (1 - \tau)\omega'_m$ and $\theta'_m = \tau\theta_m + (1 - \tau)\theta'_m$
    end for
end for

6 Analysis of the Experimental Results

6.1 System Parameter Configuration

Simulation experiments are conducted within a Python environment on a Windows 10 operating system. The simulation scenario comprises multiple mobile devices (MDs) and a single edge server. Each MD can exchange data with the edge server via limited wireless radio links. The CPU frequency of the mobile devices is set at [1.2, 1.8] GHz, while the CPU frequency of the edge node is 7 GHz. The transmission power is 20 mW, the noise power is -100 dB, and the task size ranges between 0.5 and 1 MB.

To evaluate the performance of the proposed Ly-MADDPG algorithm, we compared it with the multiagent proximal policy optimization (MAPPO) algorithm and the multiagent advantage actor-critic (MAA2C) algorithm. Below is a brief overview of the mechanisms of each compared algorithm:

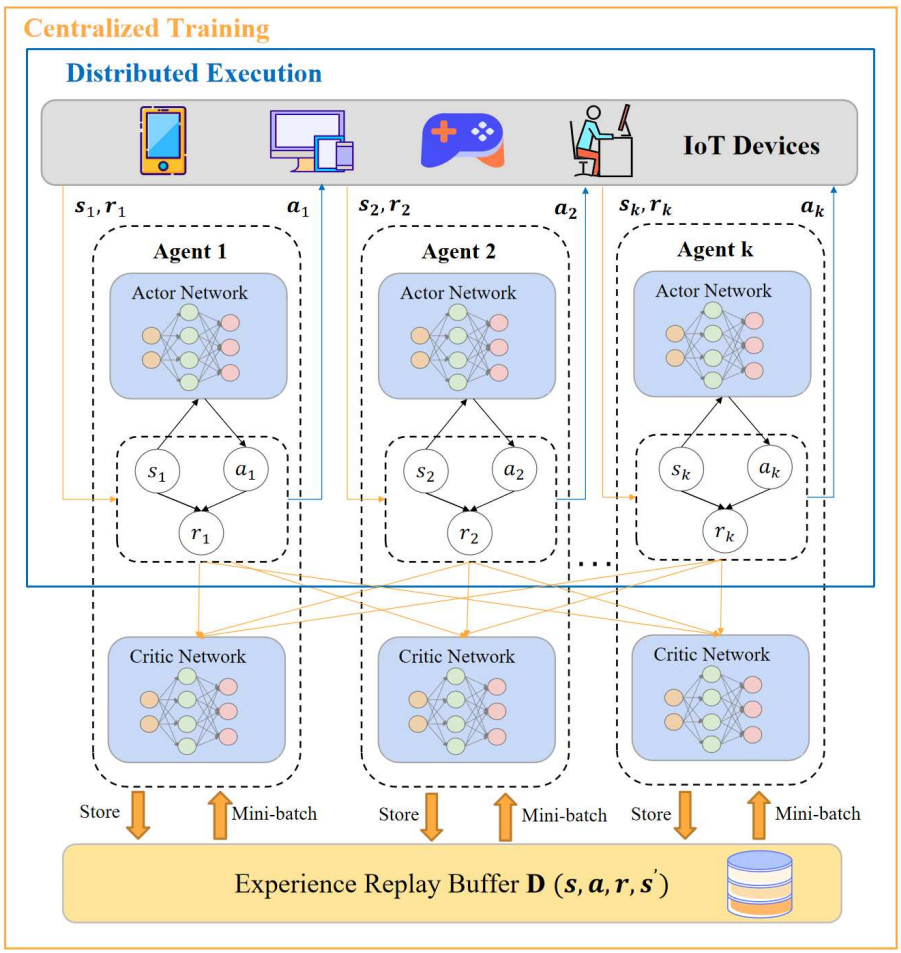


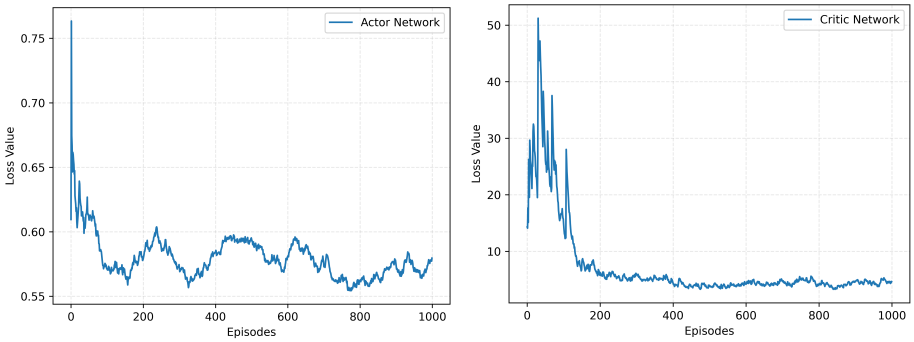
Fig. 3 The structure of the Ly-MADDPG framework

(1) MAPPO (Multiagent Proximal Policy Optimization): MAPPO [25] is a deep reinforcement learning algorithm optimized for multiagent proximal policies. By introducing an enhancement termed the "proximal" constraint, it aims to maintain good data efficiency while avoiding significant shifts during policy update steps.

(2) MAA2C (Multiagent Advantage Actor-Critic): MAA2C [26] is an actor-critic type of reinforcement learning algorithm. MAA2C employs a technique called the advantage function to update the agent's policy. The advantage function gauges the superiority of specific actions relative to the average ones. This allows MAA2C to efficiently update the policy so that beneficial actions are more likely to occur, while detrimental ones are suppressed.

Table 1 Main Notation Definition

Parameters	Value
Number of IoT devices M	10
Bandwidth B	50 MHz
Lyapunov weights V	30
Transmission power $p_{m,j}$	[0.5,1] W
Noise power τ^2	-100 dBm/Hz
The local computing capability f_{local}	$[1.2, 1.8]$ GHz
The MEC computing capability f_{edge}	7 GHz
Task arrival rate $A_m(t)$	[0.5,1] MB
Critic network learning rate	0.001
Actor network learning rate	0.01
The size of mini-batch	32
The size of replay buffer	10000
Actor hidden units	(64, 128)
Critic hidden units	(64, 128)
Training episodes	1000
Time slots T	500
Optimizer	Adam



(a) (b)

Fig. 4 Training loss values of the actor and critic network

(3) LCO (Local Computing Only): This algorithm enables all computational tasks requested by mobile devices to be completed locally, without any computational offloading.

(4) Random: When a task arrives, it can be randomly chosen to be executed either locally or on a mobile edge computing platform, and the wireless channel is also randomly allocated.

6.2 Simulation Results and Analysis

First, we give the convergence and reward results of the algorithm. The proposed Ly-MADDPG algorithm incorporates both an actor network and a critic network. Fig. 4 shows the convergence and average loss values of these two networks during the training process. The loss values for both networks gradually converge as the number of iterations increases, indicating that all networks function properly.

In Fig. 5, we present the learning curve performance of various algorithms during the training episodes. The x-axis represents the number of training iterations, while the y-axis indicates the rewards obtained by the agents during training. According to the figure, the rewards for MADDPG, MAPPO, and MAA2C increase progressively with the number of iterations until they converge. Compared to other algorithms, the MADDPG method proposed in this study converges more rapidly and achieves a greater reward. The convergence of the MAA2C algorithm is somewhat worse. This is because the policy updates in MAA2C are based on the current policy and value function estimations and do not employ a target network such as DDPG. Its learning objectives can be more easily perturbed by current experiences, leading to training instability and subsequently affecting algorithm performance. In contrast, MADDPG demonstrates a faster convergence rate and greater stability. This can be attributed to its strategy update method, which integrates deterministic behaviour policies with the Q-value function, showcasing enhanced performance.

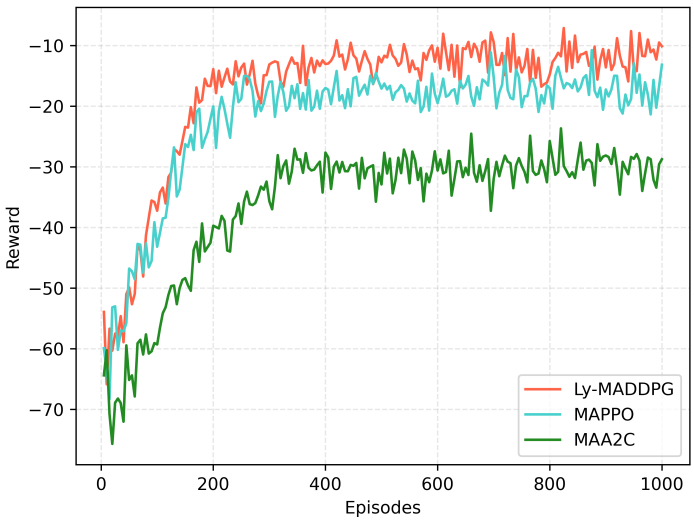


Fig. 5 The convergence and rewards of different algorithms

In Fig. 6, we show the average queue length versus time slot under different task arrival rates. We use values of 0.8, 1.0, and 1.2 for λ to signify that we are adjusting the task arrival rate from 80% to 120% of its default value. The experimental results indicate that the average queue length rapidly increases over time, eventually stabilizing and reaching a balanced state. Moreover, the average queue length increases with increasing task arrival rate. This is due to the limited wireless transmission rate and computational capacity. As the task arrival rate increases, the backlog of tasks in the queue of mobile devices correspondingly increases.

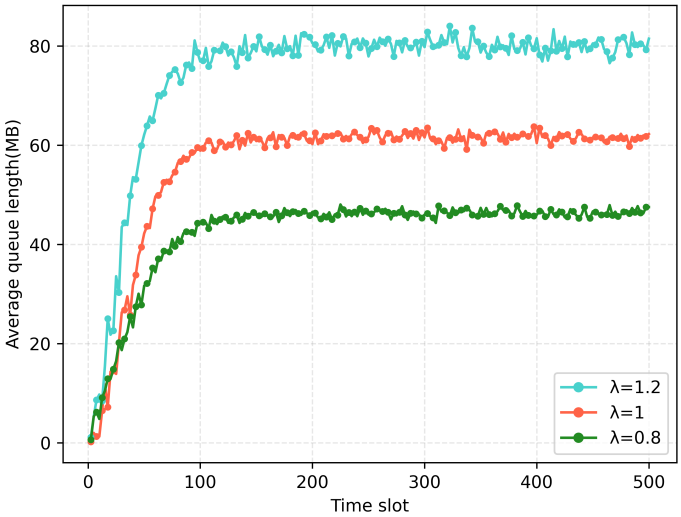
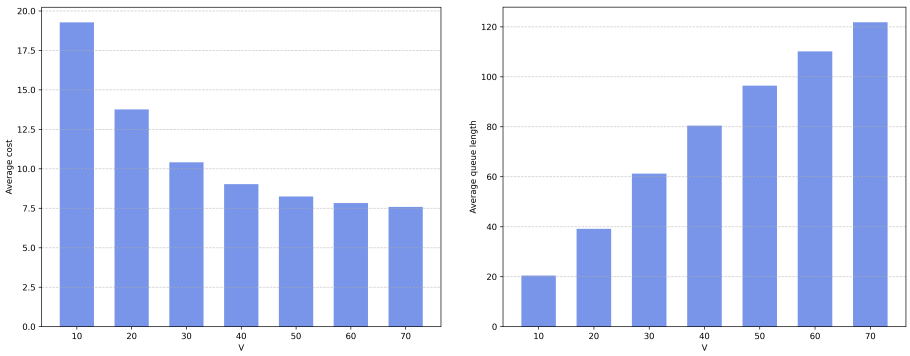


Fig. 6 Average queue length versus time slot



(a) (b)

Fig. 7 The relationship between V and the average cost and average queue length

Fig. 7 collectively shed light on the influence of varying V values on task offloading costs and respective queue lengths. As depicted in Fig. 7a, there is a discernible reduction in cost as V increases. This trend is ascribed to the higher weight that an increased V gives to offloading costs, thereby driving the algorithm to favour cost optimization. This inference aligns with the theory presented in Eq. (35). As shown in Fig. 7b, the task queue grows in proportion to the increase in V, in agreement with Eq. (38). Hence, in real-world scenarios, modulating V enables an optimal balance between offloading costs and queue stability.

Fig. 8 displays the weighted task offloading cost performance of each algorithm under varying communication bandwidths. In this figure, the range of the communication bandwidth is set from 20 MHz to 80 MHz. Through analysis, we find that the average costs of the MADDPG, MAPPO, MAA2C, and Random algorithms all decrease incrementally as the communication bandwidth increases. This is because with the increase in communication bandwidth, data transmission rates significantly increase, reducing the transmission time of tasks. Thus, more tasks with larger data sizes can be offloaded to edge servers, subsequently reducing the offloading cost. Furthermore, the MADDPG algorithm consistently outperforms MAPPO, MAA2C, Random and LCO in terms of average cost across different bandwidths. On the other hand, while MAPPO, MAA2C, and Random also show performance improvements with increased bandwidth, they still lag behind MADDPG in overall performance.

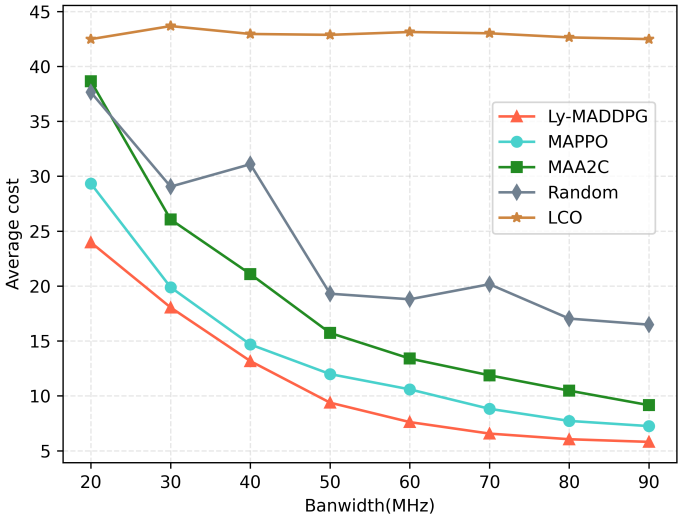


Fig. 8 Average cost versus bandwidth

Fig. 9 illustrates the weighted task offloading cost performance of each algorithm with variations in the number of mobile devices. Initially, as the number of users increases, the task offloading costs for all algorithms, excluding LCO, exhibit a corresponding increasing trend. Additionally, across varying mobile device counts, the performance of the MADDPG algorithm consistently surpasses that of the other algorithms. This is primarily because, with an increase in user count, there are more computational tasks to be processed. The computational resources that can be allocated to each device decrease proportionally, leading to an increase in both the offloading latency and energy consumption.

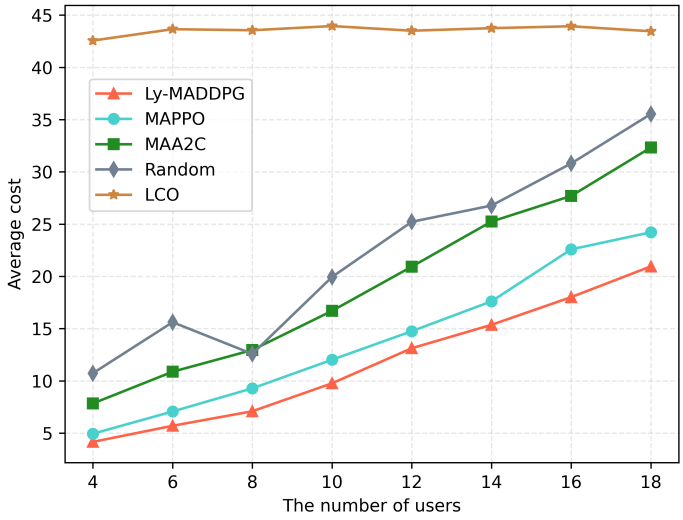


Fig. 9 Average cost versus the number of users

7 Conclusion

This paper presents a Lyapunov-guided multiagent deep reinforcement learning-based dynamic task offloading strategy, which decouples the optimization problem into a MDP that is solved frame by frame, thus reducing the computational difficulty. Furthermore, the Ly-MADDPG algorithm is introduced to solve this problem. The Ly-MADDPG algorithm aims to minimize average latency and energy consumption through centralized training and decentralized execution while ensuring overall stability. The simulation results demonstrate the efficacy of the proposed algorithm under varying system parameters, showing that it outperforms comparative algorithms.

References

[1] Liu, Y, Peng, M, Shou, G, Chen, Y, Chen, S: Toward edge intelligence: Multiaccess edge computing for 5g and internet of things. IEEE Internet of Things Journal **7**(8), 6722–6747 (2020)

[2] Guo, F, Yu, FR, Zhang, H, Li, X, Ji, H, Leung, VC: Enabling massive iot toward 6g: A comprehensive survey. IEEE Internet of Things Journal **8**(15), 11891–11915 (2021)

[3] Ai, Y, Peng, M, Zhang, K: Edge computing technologies for internet of things: a primer. Digital Communications and Networks **4**(2), 77–86 (2018)

[4] Jiang, K, Zhou, H, Chen, X, Zhang, H: Mobile edge computing for ultra-reliable

and low-latency communications. IEEE Communications Standards Magazine **5**(2), 68–75 (2021)

[5] He, X, Lu, H, Du, M, Mao, Y, Wang, K: Qoe-based task offloading with deep reinforcement learning in edge-enabled internet of vehicles. IEEE Transactions on Intelligent Transportation Systems **22**(4), 2252–2261 (2020)

[6] Ale, L, Zhang, N, Fang, X, Chen, X, Wu, S, Li, L: Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning. IEEE Transactions on Cognitive Communications and Networking **7**(3), 881–892 (2021)

[7] Chu, W, Yu, P, Yu, Z, Lui, JC, Lin, Y: Online optimal service selection, resource allocation and task offloading for multi-access edge computing: A utility-based approach. IEEE Transactions on Mobile Computing **22**(7), 4150–4167 (2022)

[8] Guo, M, Huang, X, Wang, W, Liang, B, Yang, Y, Zhang, L, Chen, L: Hagp: A heuristic algorithm based on greedy policy for task offloading with reliability of mds in mec of the industrial internet. Sensors **21**(10), 3513 (2021)

[9] Ale, L, King, SA, Zhang, N, Sattar, AR, Skandaraniyam, J: D3pg: Dirichlet ddpg for task partitioning and offloading with constrained hybrid action space in mobile-edge computing. IEEE Internet of Things Journal **9**(19), 19260–19272 (2022)

[10] Zabihi, Z, Eftekhari Moghadam, AM, Rezvani, MH: Reinforcement learning methods for computation offloading: A systematic review. ACM Computing Surveys **56**(1), 1–41 (2023)

[11] Shen, T, Gao, C, Xu, D: The analysis of intelligent real-time image recognition technology based on mobile edge computing and deep learning. Journal of Real-Time Image Processing **18**(4), 1157–1166 (2021)

[12] Liu, Z, Dai, P, Xing, H, Yu, Z, Zhang, W: A distributed algorithm for task offloading in vehicular networks with hybrid fog/cloud computing. IEEE Transactions on Systems, Man, and Cybernetics: Systems **52**(7), 4388–4401 (2021)

[13] Guo, H, Liu, J, Ren, J, Zhang, Y: Intelligent task offloading in vehicular edge computing networks. IEEE Wireless Communications **27**(4), 126–132 (2020)

[14] Alqarni, MM, Cherif, A, Alkayal, E: A survey of computational offloading in cloud/edge-based architectures: strategies, optimization models and challenges. KSII Transactions on Internet and Information Systems (TIIS) **15**(3), 952–973 (2021)

[15] Geng, Y, Yang, Y, Cao, G: Energy-efficient computation offloading for multicore-based mobile devices. IEEE INFOCOM 2018-IEEE Conference on Computer

Communications, 46–54 (2018)

[16] Yang, L, Cao, J, Wang, Z, Wu, W: Network aware mobile edge computation partitioning in multi-user environments. IEEE Transactions on Services Computing **14**(5), 1478–1491 (2018)

[17] Liu, J, Mao, Y, Zhang, J, Letaief, KB: Delay-optimal computation task scheduling for mobile-edge computing systems. In: 2016 IEEE International Symposium on Information Theory (ISIT), pp. 1451–1455 (2016)

[18] Cui, L, Xu, C, Yang, S, Huang, JZ, Li, J, Wang, X, Ming, Z, Lu, N: Joint optimization of energy consumption and latency in mobile edge computing for internet of things. IEEE Internet of Things Journal **6**(3), 4791–4803 (2018)

[19] Huang, L, Bi, S, Zhang, YqJA: Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Transactions on Mobile Computing **19**(11), 2581–2593 (2019)

[20] Liu, G, Dai, F, Huang, B, Qiang, Z, Wang, S, Li, L: A collaborative computation and dependency-aware task offloading method for vehicular edge computing: a reinforcement learning approach. Journal of Cloud Computing **11**(1), 68 (2022)

[21] Zou, J, Hao, T, Yu, C, *et al.*: A3c-do: A regional resource scheduling framework based on deep reinforcement learning in edge scenario. IEEE Transactions on Computers **70**(2), 228–239 (2020)

[22] Wang, J, Hu, J, Min, G, Zhan, W, Ni, Q, Georgalas, N: Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning. IEEE Communications Magazine **57**(5), 64–69 (2019)

[23] Cao, Z, Zhou, P, Li, R, Huang, S, Wu, D: Multiagent deep reinforcement learning for joint multichannel access and task offloading of mobile-edge computing in industry 4.0. IEEE Internet of Things Journal **7**(7), 6201–6213 (2020)

[24] Yang, J, Yuan, Q, Chen, S, He, H, Jiang, X, Tan, X: Cooperative task offloading for mobile edge computing based on multi-agent deep reinforcement learning. IEEE Transactions on Network and Service Management **20**(3), 3205–3219 (2023)

[25] Yu, C, Velu, A, Vinitsky, E, Gao, J, Wang, Y, Bayen, A, Wu, Y: The surprising effectiveness of ppo in cooperative multi-agent games. In: Advances in Neural Information Processing Systems, vol. 35, pp. 24611–24624 (2022)

[26] Zhan, Y, Guo, S, Li, P, Zhang, J: A deep reinforcement learning based offloading game in edge computing. IEEE Transactions on Computers **69**(6), 883–893 (2020)