Journal of King Saud University Computer and Information Sciences (2025) 37:168
https://doi.org/10.1007/s44443-025-00160-w

ORIGINAL PAPER

Priority-aware task offloading for LEO satellite edge computing network: a multi-agent deep reinforcement learning-based approach

Juan Chen[1,2] · Jie Zhong[1] · Zongling Wu[3] · Di Tian[1] · Yujie Chen[1]

Received: 21 February 2025 / Accepted: 5 July 2025 / Published online: 18 August 2025

Abstract
In the current data-driven era, satellite edge computing (SatEC) emerges as a novel computing paradigm that addresses the coverage limitations of terrestrial networks in remote areas by deploying computational resources on satellite networks. This provides a new avenue for real-time data processing on a global scale. However, effectively scheduling tasks to optimize resource utilization, reduce processing delay, and ensure the timeliness of task execution has become a critical issue in SatEC. Existing task scheduling algorithms often fail to fully consider task deadlines and priority orders, leading to deficiencies in handling urgent tasks and optimizing overall computational efficiency. In response to these issues, we propose a multi-agent advantage actor-critic (MA2C) algorithm based on deep reinforcement learning, aiming to effectively address the task scheduling challenges in SatEC. The MA2C algorithm employs two cooperative agents to optimize scheduling strategies: one agent is responsible for prioritizing tasks, while the other focuses on task offloading decisions. These two agents interact in real-time, dynamically adjusting their strategies to adapt to environmental changes, effectively enhancing scheduling efficiency. Furthermore, the agent design adopts an encoder-decoder architecture, combined with self-attention (SA) mechanisms and temporal convolutional network (TCN) technology to extract environmental features, achieving real-time optimization of task scheduling. The simulation results indicate that the MA2C algorithm outperforms existing algorithms across various scenarios. Compared with the baseline algorithms BDQN, DRTO, and GA, the MA2C algorithm reduces the task failure rate by 9.1%, 11.2%, and 16.9% respectively.

Keywords Satellite edge computing · Task scheduling · Task priority · Optimization of task delay and failure rate · Deep reinforcement learning

1 Introduction

In today's data-driven era, computational power and data processing speed have become key factors constraining technological progress. With the rapid development of technologies such as the Internet of Things (IoT), artificial intelligence, and 6G communications, edge computing, as an emerging computing paradigm, is increasingly becoming a hot topic in the field of information technology (Ahmed et al. 2024). The core idea of edge computing is to push data processing and analysis to the edge of the network, thereby reducing data transmission latency, improving response speed, and alleviating the burden on central servers (Kong et al. 2022). However, across vast geographical areas, especially in remote regions, the construction and coverage of terrestrial network infrastructure still face numerous challenges and limitations (Zhao et al. 2022). Against this technological backdrop, satellite edge computing (SatEC) has emerged as an important branch of edge computing. In simple terms, SatEC is a computing model that combines the principles of edge computing with satellite communication technology. By deploying computational resources on satellite networks, it enables data processing and analysis to occur closer to the source of the data, making it possible even in areas where terrestrial networks are difficult to reach. The advent of SatEC offers new possibilities for real-time data processing and intelli-

✉ Zongling Wu
zl_wu@swjtu.edu.cn

1 School of Computer and Software Engineering, Xihua University, 610039 Chengdu, Sichuan, China

2 Yunnan Key Laboratory of Service Computing, Yunnan University of Finance and Economics, 650221 Kunming, Yunnan, China

3 School of Information Science and Technology, Southwest Jiaotong University, 611756 Chengdu, Sichuan, China

gent services on a global scale, particularly in environments with insufficient ground infrastructure such as oceans, polar regions, and mountainous areas, where its value and potential are particularly significant (Zhu and Jiang 2022).

In the SatEC environment, one of the important challenges we face is the task scheduling problem (Xie et al. 2020): that is, deciding whether tasks should be executed locally on IoT terminals or offloaded to satellite computing servers for processing. Relying entirely on the computational capabilities of local IoT devices, which typically have limited resources, may lead to severe computation delays. Conversely, if all tasks are transferred to satellite computing servers, given the limited parallel processing capabilities of satellite systems, inevitably long queuing delays will occur when handling a large number of concurrent tasks.

Additionally, in the problem of scheduling multiple tasks simultaneously, the order in which they are processed has a significant impact on both the latency of individual tasks and the total delay of the entire task set (Li et al. 2023). Due to the constraints of deadlines, high latency not only leads to a decrease in task processing efficiency but, more critically, it directly affects the success of the tasks (Chai et al. 2023). In many real-time application scenarios, tasks must be completed within the specified deadlines, or they may result in irreversible consequences. For instance, in fields such as emergency rescue and intelligent traffic control, the timely completion of tasks is crucial for ensuring the safety of personnel and the normal operation of systems. An effective scheduling algorithm needs to be able to identify which tasks have urgent deadlines and dynamically adjust the priority of tasks based on this information. Prioritizing tasks with urgent deadlines can ensure that critical tasks are completed on time while also allocating resources reasonably and reducing the overall task failure rate, thereby improving the system's reliability and stability. Therefore, there is an urgent need for an intelligent scheduling algorithm that can intelligently determine the priority execution order of tasks based on real-time task status information and decide whether tasks need to be offloaded.

In existing research, scholars have proposed various task scheduling algorithms based on game theory (Chen et al. 2024), convex optimization (Xi et al. 2024), heuristics (Xu et al. 2024), metaheuristics (Liu et al. 2022; Hu and Gong 2023), and deep reinforcement learning (DRL) (Zhang et al. 2024b; Li et al. 2024a) to determine whether tasks should be offloaded. However, these studies generally fail to fully consider two critical factors: task deadlines and the priority order of task processing.

Specifically, these studies either set overly relaxed deadlines, making it unlikely that tasks will time out whether they are executed on local IoT terminals or offloaded to computing servers. Additionally, some studies even completely overlook the importance of deadlines, a practice that undoubtedly undermines the practical application value of the research findings. Furthermore, existing research often overlooks the issue of task processing priority. In reality, the determination of task priorities has a direct impact on system performance and the quality of task execution. Given that different tasks may have varying levels of urgency and importance, a well-organized task processing sequence is crucial for ensuring the timely handling of critical tasks, reducing unnecessary waiting times, preventing resource wastage, and enhancing system responsiveness.

In our study, influenced by the current research status and literature (Xu et al. 2024), we not only consider the deadlines of tasks but also fully take into account the processing priority of tasks to ensure the full utilization of system resources, enhance overall computational efficiency, and meet task requirements. Subsequently, influenced by literature (Xiong et al. 2023), we propose a DRL-based multi-agent advantage actor-critic (MA2C) algorithm to address the task prioritization and offloading problem in dynamic SatEC environments. The main contributions of this paper are summarized as follows:

- We have proposed a multi-user multi-server SatEC model that considers task deadlines and task priority order, aiming to reduce the total task latency and task failure rate.
- We have proposed a MA2C algorithm based on DRL to perform task priority sorting and offloading decisions. Specifically, the algorithm consists of two cooperative agents: one focused on task priority sorting and the other responsible for offloading decisions. Both agents utilize an encoder-decoder architecture, where the encoder integrates a self-attention (SA) mechanism and temporal convolutional networks (TCN) to efficiently extract complex feature information in the SatEC environment. The decoder part is responsible for generating the corresponding task scheduling strategies based on the feature information extracted by the encoder.
- Extensive simulation experiments show that, compared with several existing DRL algorithms and genetic algorithms, our proposed MA2C algorithm achieves the best performance in reducing the total delay and task failure rate under varying task data sizes, different numbers of IoT terminal devices, different parallel computing capabilities of satellite systems, and different task deadlines.

This paper organizes the remaining sections as follows: Section 2 talks about the background research, Section 3 builds the system model and makes the research questions clear, Section 4 goes into detail about the MA2C algorithm, Section 5 tests and evaluates our MA2C algorithm's performance, Section 6 discusses the challenges of current research and future research directions, and Section 7 summarizes this article.

2 Related work

This section provides an overview of recent advancements in task scheduling research for SatEC. It reviews the related work from two different perspectives: first, the strategies employing traditional optimization algorithms, and second, the approaches based on DRL algorithms.

2.1 Traditional optimization methods

In the quest for effective task scheduling approaches in SatEC systems, traditional optimization methods have garnered significant attention from researchers due to their mature theoretical foundation and broad application scenarios. For instance, Chen et al. (2024) proposed a game theory-based joint unmanned aerial vehicle (UAV) and low earth orbit (LEO) satellite task offloading (JULTO) algorithm, which addresses the task offloading problem of multiple mobile user devices competing for limited resources in a UAV-assisted LEO satellite edge computing network. Xi et al. (2024) introduced a joint iterative algorithm based on convex optimization theory, which optimizes user association, task scheduling, power control, and satellite computing resource allocation to minimize energy consumption in a multi-satellite edge computing system. Xu et al. (2024) proposed a multi-applications schedule to multi-satellites (MAS-MS) algorithm, ensuring that tasks are completed within the established time constraints, effectively tackling the multi-task scheduling challenges in SatEC environments. Additionally, Liu et al. (2022) presented a task scheduling strategy based on genetic algorithms, solving an efficient task allocation problem in satellite networks. Hu and Gong (2023) proposed an offloading strategy based on the hybrid genetic binary particle swarm optimization (GABPSO) algorithm, addressing an optimization problem of minimizing latency and energy consumption in a dynamic LEO satellite network with consideration of high-load satellite constraints. Gao et al. (2023) introduced a metaheuristic algorithm based on k-medoids clustering and the non-dominated sorting genetic algorithm-II (NSGA-II), solving the joint optimization problem of server and service deployment in satellite-ground integrated edge computing networks.

Although traditional optimization algorithms have achieved significant accomplishments in the field of task scheduling for SatEC systems, they still face some challenges in practical applications. These methods often require exhaustive information and precise mathematical models, which can be difficult to satisfy in dynamic and uncertain satellite network environments. Additionally, traditional algorithms may lack sufficient flexibility in dealing with complex optimization problems (Zhang and Wang 2023), resulting in limited effectiveness when responding to real-time changes and unexpected events. More importantly, these algorithms may require a substantial amount of computational resources during the optimization process, which is a non-negligible issue in resource-constrained satellite systems.

2.2 Methods based on deep reinforcement learning

To overcome these limitations, researchers have turned to deep reinforcement learning algorithms. As a technique that combines deep learning and reinforcement learning, DRL algorithms represent decision-making policies through neural networks and optimize them using feedback obtained from interactions with the environment (Tang et al. 2020). With their unique structure and training mechanism, DRL algorithms demonstrate strong learning capabilities and adaptability, enabling effective decision-making in environments with incomplete information. DRL algorithms, on the other hand, can handle more complicated real-world problems than traditional ones. They do this by reducing the need for prior knowledge through an end-to-end learning process and keeping up high performance in environments that are always changing (Zhang et al. 2024a). Therefore, applying DRL algorithms to task scheduling in SatEC systems not only opens up new research avenues but also holds the promise of significantly enhancing the system's robustness and adaptability while ensuring task execution efficiency.

In recent years, researchers have explored various DRL-based task scheduling methods for SatEC. For instance, Zhang et al. (2024b) optimized the data collection time in space-to-ground integrated networks using the Q-learning algorithm. Waqar et al. (2022) employed the double deep Q-learning (DDQL) algorithm to effectively reduce computation and communication costs in MEC-assisted aerial-ground vehicular networks. Sun and He (2023) proposed a multi-branch deep Q network (BDQN) algorithm for time-varying edge computing environments, significantly decreasing task latency and failure rates. Furthermore, Zhang et al. (2023b) developed a multi-agent reinforcement learning approach based on the actor-critic framework to address heterogeneous computing task scheduling for the Internet of Remote Things in space-air-ground integrated networks (SAGIN). Tianhao and Zhiyong (2024) enhanced the feature extraction capability of state information by introducing a self-attention mechanism into the soft actor-critic (SAC) algorithm, effectively reducing the average latency and energy consumption of tasks. Cui et al. (2022) optimized task offloading and channel allocation in SatEC using a DRL algorithm based on proximal policy optimization (PPO), significantly decreasing task latency. Yang et al. (2024b) proposed a DRL-based task offloading (DRTO) framework, demonstrating great potential in reducing overall system latency and enhancing user experience. Additionally, Zhou et al. (2024) considered the mobility and resource constraints of LEO satellites and proposed a PPO algorithm incorporating spatio-temporal load

factors to optimize user-perceived latency and energy consumption. Yang et al. (2024a) addressed the joint access selection and computation offloading problem in LEO satellite edge computing networks by proposing the alternating dueling DQN (ADDQN) algorithm, which separately decides access selection and computation offloading to maximize the number of successful subtasks and quality of experience (QoE). Li et al. (2024a) introduced a dual-network structure algorithm based on deep reinforcement learning offloading (DRLO), effectively balancing latency and energy consumption costs and adapting to the dynamic changes of satellite networks, thereby significantly optimizing computation offloading decisions.

In Table 1, we comprehensively compare our work with existing studies, covering multiple dimensions such as optimization objectives, task priorities, deadlines, optimization

Table 1 Related works comparison

Articles	Optimization Objective	Task priority	Deadline	optimization strategy	Algorithm type
Chen et al. (2024)	Total delay and energy consumption	×	✓	Joint UAV and LEO satellite task offloading (JULTO) based on game theory.	
Xi et al. (2024)	Total energy consumption	×	✓	Joint iterative algorithm based on convex optimization theory.	
Xu et al. (2024)	Average delay	✓	✓	Multi-applications schedule to multi-satellites algorithm (MAS-MS).	
Liu et al. (2022)	Delay and task traffic	×	×	Genetic algorithm.	Traditional optimization
Hu and Gong (2023)	Total delay and energy consumption	×	×	Hybrid genetic binary particle swarm optimization algorithm (GABPSO).	algorithm
Gao et al. (2023)	Average delay and energy consumption	×	×	Non-dominated sorting genetic algorithm-II (NSGA-II).	
Zhang et al. (2024b)	Total delay	×	×	Q-learning.	
Waqar et al. (2022)	Total delay and energy consumption	×	✓	Double deep Q-learning (DDQL).	
Sun and He (2023)	Total delay	×	✓	Branch deep Q network (BDQN).	
Zhang et al. (2023b)	Average energy consumption	×	✓	Multi-agent reinforcement learning method based on actor–critic (AC) framework.	
Tianhao and Zhiyong (2024)	Average delay and energy consumption	×	×	Soft actor-critic (SAC) algorithm with self attention mechanism.	
Cui et al. (2022)	Total delay	×	×	Proximal policy optimization (PPO).	Deep reinforcement
Yang et al. (2024b)	Total delay	×	✓	DRL-based task offloading (DRTO) framework.	learning algorithm
Zhou et al. (2024)	Average delay and energy consumption	×	×	Proximal policy optimization (PPO) algorithm combining spatiotemporal load factor.	
Yang et al. (2024a)	Successful Sub-task Numbers and quality of experience (QoE)	×	×	Alternating dueling DQN (ADDQN).	
Li et al. (2024a)	Total delay and energy consumption	×	×	Dual network structure algorithm based on deep reinforcement learning offloading (DRLO).	
This paper	Total delay	✓	✓	A multi-agent advantage actor-critic (MA2C) reinforcement learning algorithm.	

strategies, and algorithm types. After in-depth analysis, we found that the majority of existing studies often set overly relaxed deadlines or even completely disregard them, which greatly diminishes the practical utility of these research findings in real-world applications.

Additionally, existing literature has not given sufficient consideration to the priority order of task execution, failing to effectively differentiate between the urgency and importance of different tasks. This oversight could lead to critical tasks not being completed within the set time, thereby affecting the overall performance and reliability of the system. In light of this, we believe that it is crucial to consider both the task deadlines and the priority execution order during the task scheduling process.

Therefore, our research aims to fill this gap by developing a new scheduling algorithm that ensures high-priority and urgent tasks are given precedence and processed before their deadlines, thereby enhancing the overall scheduling efficiency and responsiveness of the SatEC system.

3 System model and problem formation

This section mainly introduces the multi-user SatEC system model. In this section, we first introduce the network and task models, then introduce the communication model, delay model, priority sorting, and task processing process in the SatEC system. Finally, we formulate the optimization offloading problem that we intend to address.

3.1 Network and task model

As shown in Fig. 1, the network model consists of an IoT terminal plane composed of N IoT terminal users indexed by $\mathcal{N} = (1, 2, \ldots, N)$ and a LEO satellite plane composed of M LEO satellites indexed by $\mathcal{M} = (1, 2, \ldots, M)$. Each LEO satellite is equipped with a satellite computing server of the same computational capability. Satellites communicate with each other through intersatellite links (ISL). Each IoT terminal user generates an indivisible task in each time slot, which can either be computed locally or offloaded to the satellite computing server for processing via a wireless channel. IoT terminals and satellite computing servers can compute only one task at a time. When a satellite receives multiple tasks simultaneously, we can transfer them to neighboring idle satellites via ISL to achieve parallel computation of multiple tasks. However, due to the limitation of the number of LEO satellites in the satellite system, the system's parallel computing capability will not exceed M.

The entire SatEC system operates in a time-slot mode, which we denote as $\mathcal{T} = \{1, 2, \ldots, T\}$. Within any time slot $\tau \in \mathcal{T}$, each IoT terminal user generates a task $task_{\tau,n}$, $n \in \mathcal{N}$. We represent this task with a tuple $(d_{\tau,n}, t^{DL}_{\tau,n})$, where $d_{\tau,n}$ indicates the data size of the task, and $t^{DL}_{\tau,n}$ represents the deadline of the task.

The tasks generated by IoT terminal users can be executed either locally or offloaded to satellite computing servers. We use a binary offloading indicator $\beta_{\tau,n,m}$, $m \in (0, 1, 2, \ldots, M)$ to indicate whether $task_{\tau,n}$ is offloaded to

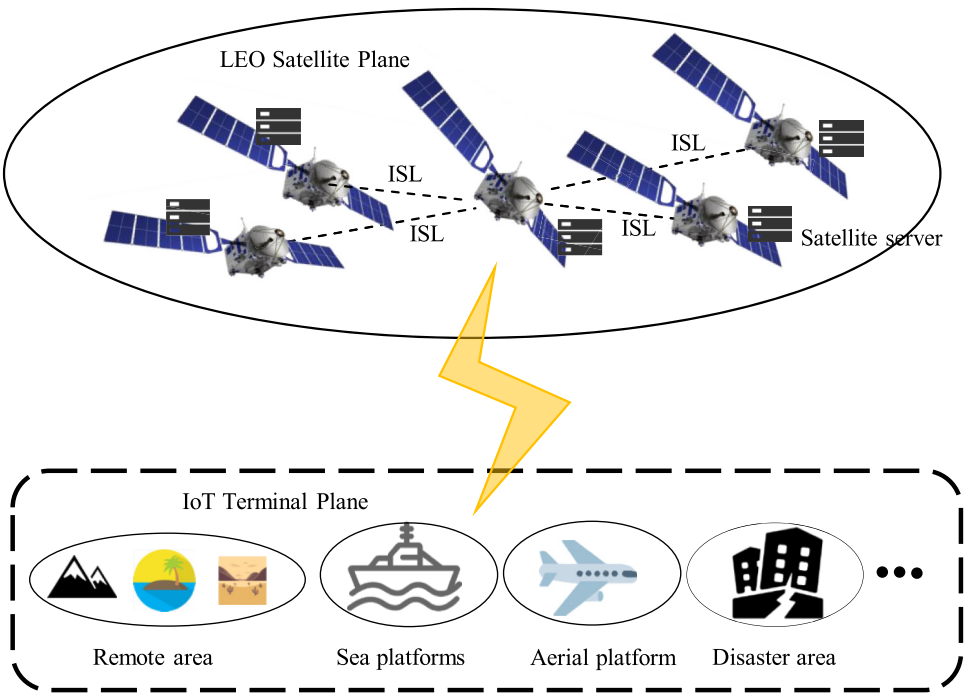


Fig. 1 Network model of SatEC system

satellite m for execution. $\beta_{\tau,n,m} = 1$ indicates that $task_{\tau,n}$ is offloaded to satellite m for execution, while $\beta_{\tau,n,0} = 1$ indicates that the $task_{\tau,n}$ is executed locally. Since each task can only be executed on a single satellite edge server or locally, the offloading indicators must satisfy the following constraints:

$$\beta_{\tau,n,0} + \sum_{m=1}^{M} \beta_{\tau,n,m} = 1, \ \forall m \in \mathcal{M} \tag{1}$$

Table 2 summarizes the main notations used in this paper and their meanings.

3.2 Communication model

1) *The access satellite is the computing satellite:* When IoT terminal user tasks are offloaded to on-board satellite computing, to precisely model the end-to-end latency, it is crucial to analyze the key performance metrics of the data transmission phase. According to Shannon's communication theory model (Shannon 1948), the data transmission rate $r_{\tau,n,m}$ between terminal user n and LEO satellite m can be mathematically characterized as:

$$r_{\tau,n,m} = b_{\tau,n} \log_2(1 + \frac{p_{\tau,n} \ g_{\tau,n,m}}{\sum_{i=1,i\neq n}^{N} \beta_{\tau,i,m} \ p_{\tau,i} \ g_{\tau,i,m} + \sigma^2}) \tag{2}$$

Table 2 Notations used in our formulation

Notation	Definition
N	The number of Iot terminals
M	The parallel computing capability of satellite systems
$\tau, \mathcal{T}$	Index and set of time slots
$d_{\tau,n}$	The data size of $task_{\tau,n}$
$t_{\tau,n}^{DL}$	The deadline for $task_{\tau,n}$
κ	The number of CPU cycles required to process one bit of data
$\beta_{\tau,n,m}$	A binary indicator indicating whether $task_{\tau,n}$ is offloaded to satellite m
$t_{\tau,n}^{local}$	The time taken for $task_{\tau,n}$ to be computed locally
$t_{\tau,n,m}^{tran}$	The transmission delay required for $task_{\tau,n}$ to be transmitted from the user n to the satellite m
$t_{\tau,n,m}^{prop}$	The propagation delay between IoT terminal user n and satellite m
$t_{\tau,n,m}^{com}$	The computation delay when the $task_{\tau,n}$ is computed on the satellite m
$D_{n,j}$	The distance from IoT terminal user n to access satellite j
$D_{j,m}$	The distance between the access satellite j and the computing satellite m
c	The propagation speed of wireless signals
f_n^{local}	The computing capability of the IoT terminal user
f_m^{sat}	The computing capability of the satellite server
$r_{\tau,n,m}$	Data transmission rate for $task_{\tau,n}$ from IoT terminal user n to satellite m
$R_{\tau,j,m}^{sat}$	Data transmission rate between access satellite j and computing satellite m
$b_{\tau,n}$	Communication bandwidth between the IoT terminal user n and the satellite
B_m^{sat}	The inter-satellite communication bandwidth
$p_{\tau,n}$	Transmission power of IoT terminal user n
p_m^{sat}	Transmission power of satellite m
$g_{\tau,n,m}$	Channel gain between the IoT terminal user n and the satellite m
$g_{\tau,j,m}^{sat}$	Channel gain between access satellite j and computing satellite m
σ^2	additive white Gaussian noise power
$st_{\tau,n}$	Task start execution time
$\mathcal{T}_{\tau,n}^{task}$	The $task_{\tau,n}$ readiness completion time
$\mathcal{T}_{\tau,m}^{server}$	The server m readiness completion time
R	Task failure rate

where $b_{\tau,n}$ and $p_{\tau,n}$ represent the communication bandwidth and transmission power of the IoT terminal user n, respectively, and $g^{sat}_{\tau,n,m}$ represents the channel gain between the IoT terminal user n and the LEO satellite m. $\sum_{i=1,i\neq n}^{N} \beta_{\tau,i,m}\ p_{\tau,i}\ g_{\tau,i,m}$ represents the total power of all interference signals, and σ^2 is the additive white Gaussian noise (AWGN) power.

2) *The access satellite is not the computing satellite:* If the access satellite j is currently executing other tasks and there are other idle satellites in the satellite system, then in this case, the task will be forwarded to the idle satellite for execution, i.e., ($j \neq m$). After the computation is completed, satellite m will first send the results to satellite j, and then back to the ground user n. Considering that the communication links between satellites are usually implemented by highly directional laser beams, interference between different links can be neglected (Zhong et al. 2025). Therefore, the data transmission rate $R^{sat}_{\tau,j,m}$ between satellite j and satellite m can be expressed as:

$$R^{sat}_{\tau,j,m} = B^{sat}_{m} \log_2(1 + \frac{p^{sat}_{m} g^{sat}_{\tau,j,m}}{\sigma^2}) \tag{3}$$

where B^{sat}_{m} represents the inter-satellite communication bandwidth, p^{sat}_{m} denotes the satellite's transmit power, $g^{sat}_{\tau,j,m}$ is the channel gain between LEO satellites.

3.3 Delay Model

1) *Local computing:* When a task is processed locally, it operates independently of the satellite computing server and relies solely on its own processing capabilities. Therefore, we can define the delay in local computation $t^{local}_{\tau,n}$ as follows:

$$t^{local}_{\tau,n} = \frac{d_{\tau,n} \cdot \kappa}{f^{local}_{n}} \tag{4}$$

where $d_{\tau,n}$ represents the data size of $task_{\tau,n}$, κ represents the number of CPU cycles required to process one bit of data, and $d_{\tau,n} \cdot \kappa$ indicates the total number of CPU cycles required for the task; f^{local}_{n} represents the processing speed of $task_{\tau,n}$ at the local IoT terminal user n, which is the number of CPU cycles that can be executed per second.

2) *Satellite computing:* When the $task_{\tau,n}$ is offloaded to the satellite for computation, we need to consider two scenarios: the first is when the access satellite j is the same as the computing satellite m, i.e., $j = m$; the second scenario is when the access satellite is different from the computing satellite, i.e., $j \neq m$. For the second case, the task will first be transmitted to the access satellite j and then forwarded to the computing satellite m by the access satellite. Furthermore, since the data volume of the task result is small, we ignore the transmission delay of the task result (Mao et al. 2017; Sun et al. 2023). Therefore, the transmission delay $t^{tran}_{\tau,n,m}$ of the offloading process can be expressed by the following formula:

$$t^{tran}_{\tau,n,m} = \begin{cases} \dfrac{d_{\tau,n}}{r_{\tau,n,j}}, & j = m \\ \dfrac{d_{\tau,n}}{r_{\tau,n,j}} + \dfrac{d_{\tau,n}}{R^{sat}_{\tau,j,m}}, & j \neq m \end{cases} \tag{5}$$

Next, due to the large distances between satellites and ground users, as well as between inter-satellite links, the corresponding propagation delays are appropriately considered (Qiu et al. 2024). These can be calculated using the following equation:

$$t^{prop}_{\tau,n,m} = \begin{cases} \dfrac{D_{n,j}}{c}, & j = m \\ \dfrac{D_{n,j}}{c} + \dfrac{D_{j,m}}{c}, & j \neq m \end{cases} \tag{6}$$

where $D_{n,j}$ represents the distance between the IoT terminal user n and the access satellite j, $D_{j,m}$ represents the distance between the access satellite j and the computing satellite m, and c represents the propagation speed of the wireless signal.

Assume the computing capability of the satellite computing server m is f^{sat}_{m}, the computation delay $t^{com}_{\tau,n,m}$ of $task_{\tau,n}$ on the satellite server can be calculated as follows:

$$t^{com}_{\tau,n,m} = \frac{d_{\tau,n} \cdot \kappa}{f^{sat}_{m}} \tag{7}$$

3.4 Priority ordering and task processing

In multi-user SatEC systems, tasks with long processing times and urgent deadlines must be given priority to reduce the risk of task failure. For tasks generated by IoT terminals in each time slot, we first determine their priority processing order based on their task characteristics, environmental features, deadlines, and so on. Then, after selecting the task with the highest priority, we decide whether to offload this task to a satellite computing server for processing. These two decision processes are interrelated, with the priority order established in the first decision having a significant impact on the second offloading decision. Furthermore, considering the rapid changes in task demands, resource states, communication conditions, and external environmental factors within the SatEC environment, there is an urgent need for a strategy that is both flexible and highly adaptive, capable of dynamically adjusting task priorities based on real-time

environmental conditions to make optimal offloading decisions and ensure maximum overall system performance and resource utilization efficiency.

To facilitate the description of the task processing procedure, we introduce a continuous variable $st_{\tau,n}$ to represent the start execution time of $task_{\tau,n}$. Specifically, if the task is executed on the local IoT terminal user, since there is no need to wait for satellite computing resources, its start execution time $st_{\tau,n}$ will be set to 0, meaning the task can begin execution immediately.

However, if the task is offloaded to the satellite computing server for processing, its start execution time will be influenced by the task readiness completion time $\mathfrak{T}_{\tau,n}^{task}$ and the server readiness completion time $\mathfrak{T}_{\tau,m}^{server}$. The task can only begin execution after both conditions are met, so we take the maximum of the two as the start execution time, which can be expressed as:

$$st_{\tau,n} = max\{\mathfrak{T}_{\tau,n}^{task}, \mathfrak{T}_{\tau,m}^{server}\} \tag{8}$$

where the task readiness completion time $\mathfrak{T}_{\tau,n}^{task}$ represents the time required for the task to be transmitted from the IoT terminal user n to the satellite m and complete all preprocessing, and can be defined as:

$$\mathfrak{T}_{\tau,n}^{re} = t_{\tau,n,m}^{tran} + t_{\tau,n,m}^{prop} \tag{9}$$

The server readiness completion time $\mathfrak{T}_{\tau,m}^{server}$ depends on whether there are idle computing resources in the satellite system. If there is an idle computing server in the satellite system, the task can start executing immediately.

If all computing servers in the current satellite system are busy, the task must wait, and the server readiness completion time $\mathfrak{T}_{\tau,m}^{server}$ is the earliest completion time of the tasks currently being executed in the system, which can be represented by the following formula:

$$\mathfrak{T}_{\tau,m}^{server} = min\{\mathfrak{T}_{\tau,x}^{end}\}, \quad x = (1, 2, \ldots, M) \tag{10}$$

where $\mathfrak{T}_{\tau,x}^{end}$ is the completion time of the task currently being executed on server x. It can be obtained by adding the start time $st_{\tau-1,n}$ of the previous task on the server to the computation time $t_{\tau-1,n,x}^{com}$.

3.5 Problem formulation

To deeply analyze the task scheduling problem proposed in this paper, the commonly adopted method is to construct a mathematical model and to analyze this model through a certain solution strategy. According to the above description, the delay $t_{\tau,n}$ from start to finish for $task_{\tau,n}$ can be defined as:

$$t_{\tau,n} = \beta_{\tau,n,0}\, t_{\tau,n}^{local} + \sum_{m=1}^{M} \beta_{\tau,n,m}(st_{\tau,n} + t_{\tau,n,m}^{com} + t_{\tau,n,m}^{prop}) \tag{11}$$

where $st_{\tau,n}$ denotes the start execution time of the $task_{\tau,n}$ on the satellite computing server, and $t_{\tau,n,m}^{prop}$ is the propagation delay it takes for the $task_{\tau,n}$ to return from the computing satellite m to the IoT terminal user n.

Our goal is to ensure that all tasks are completed before their respective deadlines while striving to minimize the total delay from start to finish for all tasks. With this objective in mind, we formulate the task scheduling problem as an optimization problem $\mathbb{P}$, which is described as follows:

$$\mathbb{P}: \min \frac{\sum_{n=1}^{N} t_{\tau,n}}{1-R} \tag{12}$$

$$s.t.$$

$$\beta_{\tau,n,0} + \sum_{m=1}^{M} \beta_{\tau,n,m} = 1, \ \forall m \in \mathcal{M} \tag{12a}$$

$$\beta_{\tau,n,0}, \beta_{\tau,n,m} \in \{0, 1\}, \forall n \in \mathcal{N}, \forall n \in \mathcal{M} \tag{12b}$$

$$t_{\tau,n} \le t_{\tau,n}^{DL}, \quad \forall n \in \mathcal{N} \tag{12c}$$

$$f_{min}^{local} \le f_{\tau,n}^{local} \le f_{max}^{local}, \quad \forall n \in \mathcal{N} \tag{12d}$$

$$p_{min} \le p_{\tau,n} \le p_{max}, \quad \forall n \in \mathcal{N} \tag{12e}$$

$$t_{min}^{DL} \le t_{\tau,n}^{DL} \le t_{max}^{DL}, \quad \forall n \in \mathcal{N} \tag{12f}$$

where R represents the failure rate of all tasks. Constraints (12a) and (12b) indicate that each task can only be executed on a single satellite edge server or locally. Constraint (12c) indicates that $task_{\tau,n}$ must be completed before the deadline $t_{\tau,n}^{DL}$; otherwise, $task_{\tau,n}$ is declared a failure. Constraints (12d) and (12e) provide the upper and lower bounds for the IoT terminal's computing capability and data transmission power. Constraint (12f) gives the maximum and minimum allowable deadlines for $task_{\tau,n}$.

The core of the aforementioned problem lies in the joint optimization of task priority ordering and offloading decisions. However, due to the limited availability of resources, the heterogeneity of tasks, and various dynamic constraints in the environment, this problem is essentially a complex nonlinear integer programming problem. More importantly, as the number of IoT terminal devices increases, the potential combination space of priority ordering and offloading decisions for each task exhibits combinatorial explosion, making it difficult to find a global optimal solution in polynomial time. Therefore, this problem is NP-hard.

Typically, traditional optimization algorithms face inefficiency and poor solution quality when dealing with such nonlinear programming problems, as they often rely on enumeration or heuristic searches, making it difficult to find the optimal solution within a reasonable time, especially when the problem scale is large (Li et al. 2024b)Li, Li, Shi, Yan, and Zhou). Meanwhile, single-agent reinforcement learning models also struggle to simultaneously handle highly coupled task priority and offloading decisions. Their single policy network is inadequate for effectively coordinating these two interdependent decision variables, thereby impacting overall optimization performance (Oroojlooy and Hajinezhad 2023).

Inspired by the work of Xiong et al. (2023), we decoupled task priority and offloading decisions and proposed a MA2C algorithm based on deep reinforcement learning to solve this collaborative optimization problem. Specifically, the algorithm employs two cooperative agents to achieve efficient task scheduling. The first agent, termed the task selection agent, is responsible for selecting the task with the highest priority from multiple pending tasks to be executed. The second agent, known as the offloading decision agent, based on the selection made by the task selection agent, further decides whether the task should be executed locally or offloaded to a satellite computing server for execution. The two agents work in cooperation, sharing information and learning experiences, to collectively drive the entire system towards the optimal solution, thereby achieving efficient task scheduling in the dynamically changing satellite edge computing environment. In the following sections, we will elaborate on the specific content and implementation details of this algorithm.

4 MA2C optimization algorithm

In this section, we first transform the problem into a multi-agent Markov decision process (MMDP). Next, we propose a multi-agent advantage actor-critic (MA2C) deep reinforcement learning algorithm that incorporates temporal convolutional networks (TCN) (Bai et al. 2018) and self-attention (SA) (Vaswani et al. 2017) mechanisms to formulate priority ranking and offloading strategies. Finally, we describe the training process of the MA2C algorithm in detail and point out the complexity and convergence of the algorithm.

4.1 Multi-agent Markov Decision Process

We approximate the problem $\mathbb{P}$ as a multi-agent Markov decision process, which can be described as a quintuple $\boldsymbol{M} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{\mathcal{P}}, r, \gamma)$, where $\boldsymbol{S}$, $\boldsymbol{A}$, $\boldsymbol{\mathcal{P}}$, r, and γ represent the state space, action space, state transition probability, reward function, and discount factor, respectively. In each time slot, the agent will choose an action $a_\tau \in \boldsymbol{A}$ based on the current environmental state $s_\tau \in \boldsymbol{S}$, and subsequently, the environmental state updates to the next state $s_{\tau+1} \in \boldsymbol{S}$. During this process, the agent receives an immediate reward r_τ, and this process is repeated over multiple episodes.

1) *State Space:* Since the state space of the MMDP is composed of the states of all agents, the state space of the MMDP at time slot τ can be described as $\boldsymbol{S}_\tau = \{\boldsymbol{S}_\tau^c, \boldsymbol{S}_\tau^o\}$, where $\boldsymbol{S}_\tau^c$ represents the state space of the task selection agent at time slot τ, and $\boldsymbol{S}_\tau^o$ represents the state space of the offloading decision agent at time slot τ. $\boldsymbol{S}_\tau^c$ can be defined as:

$$\boldsymbol{S}_\tau^c = \{d_{\tau,n}, f_{\tau,n}^{local}, p_{\tau,n}, g_{\tau,n,m}, t_{\tau,n}^{DL}\}, n = (1, 2, \ldots, N) \tag{13}$$

where $d_{\tau,n}$ denotes the data size of $task_{\tau,n}$, $f_{\tau,n}^{local}$ represents the computational capability of the IoT terminal user, $p_{\tau,n}$ indicates the data transmission power of the IoT terminal user, $g_{\tau,n,m}$ signifies the channel gain between the IoT terminal user n and the LEO satellite m, and $t_{\tau,n}^{DL}$ is the deadline for $task_{\tau,n}$.

It is worth noting that the state space of the offloading decision agent $\boldsymbol{S}_\tau^o$ needs to be defined based on the specific $task_{\tau,n}$ selected by the task selection agent. This space encompasses the start time and processing duration of tasks. Specifically, each task has two potential processing paths: one is to execute locally, and the other is to execute on the satellite computing server. To clarify these two scenarios, we define the following states: For local processing, the state consists of the task's start time $st_{\tau,n0}$ and processing duration $ex_{\tau,n0}$, where the start time is fixed at 0, and the processing duration equals the task's local computation time $t_{\tau,n}^{local}$. For satellite server processing, the state also consists of the task's start time $st_{\tau,n1}$ and processing duration $ex_{\tau,n1}$, where the start time is the greater value between the task readiness completion time $\mathfrak{T}_{\tau,n}^{task}$ and the server readiness completion time $\mathfrak{T}_{\tau,m}^{server}$, and the processing duration is the computation delay of the satellite server $t_{\tau,n,m}^{com}$. Therefore, the state space of the offloading decision agent $\boldsymbol{S}_\tau^o$ can be represented as a set of states containing these two processing paths:

$$\boldsymbol{S}_\tau^o = \{(st_{\tau,n0}, ex_{\tau,n0}) \ , \ (st_{\tau,n1}, ex_{\tau,n1})\} \tag{14}$$

2) *Action Space:* Similar to the state space, the action space of the MMDP is also composed of the actions of all agents and can be expressed as $\boldsymbol{A}_\tau = \{\boldsymbol{A}_\tau^c, \boldsymbol{A}_\tau^o\}$. The action space for the task selection agent is defined as $\boldsymbol{A}_\tau^c = \{1, 2, \ldots, N\}$, where each number represents the index of a task. This space reflects the responsibility of

the task selection agent, which is to prioritize the tasks generated by the IoT terminal and select the highest priority task for subsequent offloading decision processing. The action space for the offloading decision agent can be defined as $\boldsymbol{A}_\tau^o = \{\boldsymbol{\Pi}_{\tau,n}, n \in (1, 2, \ldots, N)\}$, where $\boldsymbol{\Pi}_{\tau,n} = \{\beta_{\tau,n,0}, \beta_{\tau,n,1}, \ldots, \beta_{\tau,n,M}\}$.

3) *State Transition Probability:* The state transition probability can be described as $\mathcal{P}(s_{\tau+1} \mid s_\tau, a_\tau)$, which gives the probability distribution of the system transitioning to the next state $s_{\tau+1}$ after taking action a_τ in the current state s_τ. The probability distribution is determined solely by the environment, and the agent has no prior knowledge of it Ye et al. (2019).
4) *Reward Function:* The design of the reward function is to guide the agent toward achieving our goals in the MMDP. We use r_τ to represent the reward obtained by performing action a_τ in state s_τ. By defining the reward function as the negative value of the objective function (12), specifically using 1-R as the denominator instead of linearly adding failure rates, we essentially construct a coupled optimization space for latency and failure rates. When the task failure rate R increases, the decrease in the denominator amplifies the negative impact of latency, prompting the agent to prioritize tasks with high failure risk. Conversely, when the task failure rate R approaches 0, the system focuses on latency optimization, achieving hierarchical optimization goals under constraints. The immediate reward, obtained by directly taking the negative of the objective function, transforms the original minimization problem into a maximization problem in reinforcement learning. This design ensures that the agent's decision-making direction aligns perfectly with the system's optimization objectives. Subsequently, each action (task priority sorting and offloading decisions) chosen by the agent will directly affect the reward value, thereby driving the policy to optimize in the direction of reducing latency and failure rates. When a task violates the deadline, a high-intensity negative reward is imposed through the penalty coefficient Ψ to continuously adjust the agent's optimization direction. In summary, we define the reward function as follows:

$$r_\tau = \begin{cases} -\dfrac{\sum_{n=1}^{N} t_{\tau,n}}{1-R}, & \text{(12a)} \sim \text{(12f) are satisfied} \\ -\Psi\dfrac{\sum_{n=1}^{N} t_{\tau,n}}{1-R}, & \text{else} \end{cases} \tag{15}$$

When conditions (12a) $\sim$ (12f) are met, the reward is the negative of the objective function. When these conditions are not met, we set the reward to a very small value as a penalty, which can be achieved by multiplying the objective function by a large penalty coefficient $\Psi = 1000$.

Based on the reward values obtained in each time slot, we can then derive the long-term reward through the following formula (16), thereby transforming the minimization problem mentioned in equation (12) into a maximization of long-term rewards problem in the MMDP. This can be described as:

$$\max\left\{\sum_{t=1}^{T}\sum_{n=1}^{N} -\gamma^t \frac{t_{\tau,n}}{1-R}\right\} \tag{16}$$

where γ is the discount factor, an important parameter in reinforcement learning algorithms that represents the significance of immediate rewards. Its value typically ranges between 0 and 1. When γ is close to 1, the agent will place more emphasis on long-term rewards; when γ is close to 0, the agent will tend to consider short-term rewards more.

4.2 The structure of the MA2C

In the satellite edge computing environment, due to the diversity of user demands and the uncertainty of computing resources, traditional single-agent reinforcement learning algorithms have certain limitations when dealing with such complex issues. Single-agent systems often struggle to optimize multiple interdependent tasks simultaneously, and their learning efficiency and strategy generalization ability are limited in dynamically changing environments. To overcome these shortcomings, we have adopted a multi-agent advantage actor-critic (MA2C) algorithm based on deep reinforcement learning. Through the collaborative learning of multiple agents, it can better handle the dependencies between multiple tasks, improving the adaptability and flexibility of the strategy. This multi-agent architecture allows each agent to focus on specific problem domains, thereby achieving overall better strategy optimization and resource management.

As shown in Fig. 2, the core of the MA2C architecture lies in the task selection agent and the offloading decision agent, which share a similar structure. Both incorporate an advanced encoder-decoder architecture, where the encoder integrates a temporal convolutional network (Bai et al. 2018) with an self-attention mechanism (Vaswani et al. 2017), focusing on extracting temporal sequence features from the data. The decoder, on the other hand, employs a pointer network (Vinyals et al. 2015) to output the probability distribution of actions. In the task selection agent, we can use a greedy strategy based on these probability distributions to select and execute the task with the highest priority. In the offloading decision agent, the probability distribution guides the making of the optimal offloading decisions. The collaboration of these two agents ensures the efficiency and adaptability of the system in dynamic environments. In the following descrip-

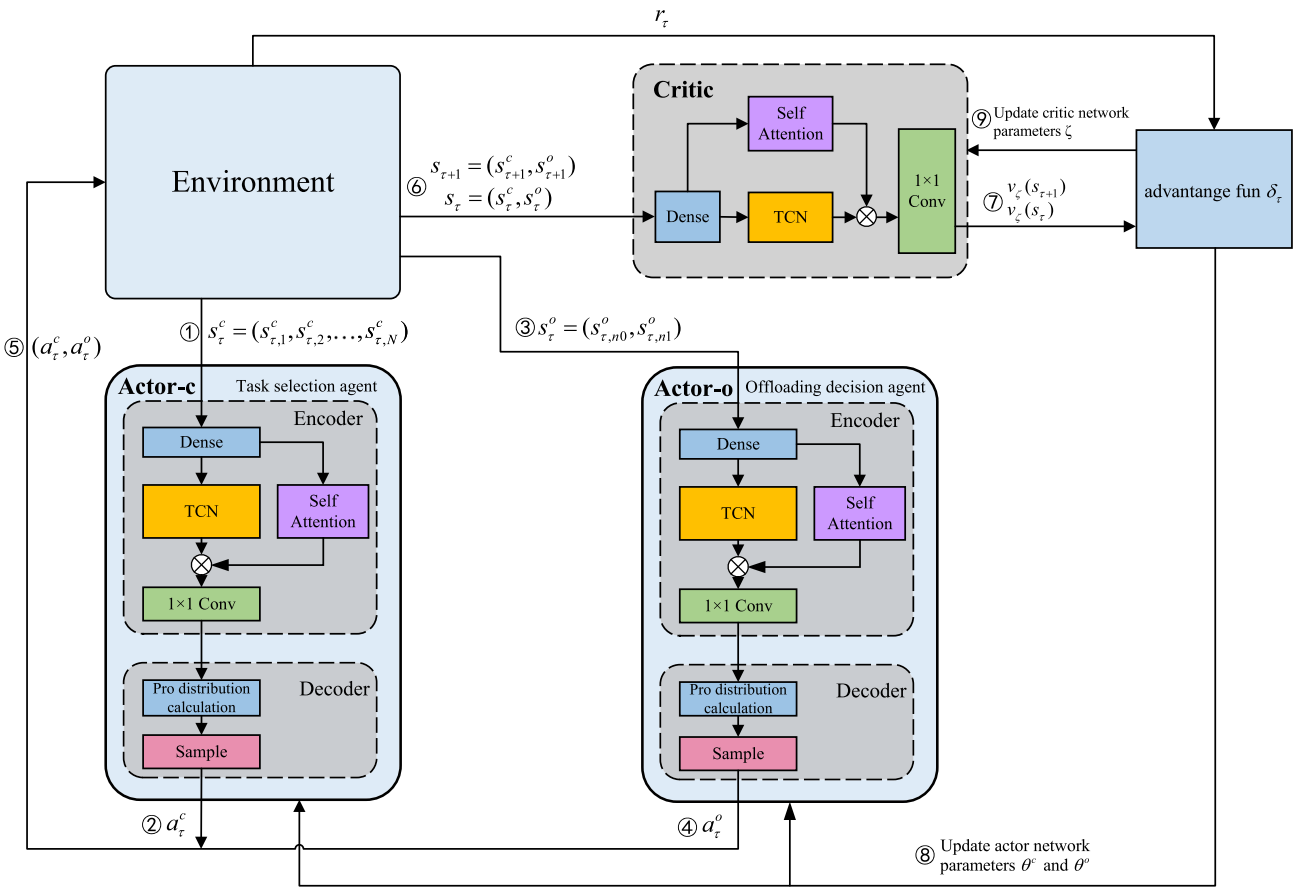


Fig. 2 The structure of the MA2C algorithm

tion, we will elaborate on the principles of the encoder and decoder in detail.

4.2.1 Encoder

The role of the encoder is to collect and integrate the state information from various IoT devices and perform key feature extraction on these input states. For the task selection agent, the inputs to its encoder are the states of the IoT devices $\boldsymbol{S}^c_\tau = (s^c_{\tau,1}, s^c_{\tau,2}, \ldots, s^c_{\tau,N})$, where $s^c_{\tau,n} = \{d_{\tau,n}, f^{local}_{\tau,n}, p_{\tau,n}, g_{\tau,n,m}, t^{DL}_{\tau,n}\}, n \in (1, 2, \ldots, N)$. For the offloading decision agent, the inputs to its encoder are the states $\boldsymbol{S}^o_\tau = (s^o_{\tau,n0}, s^o_{\tau,n1})$, where $s^o_{\tau,n0} = (st_{\tau,n0}, ex_{\tau,n0})$, and $s^o_{\tau,n1} = (st_{\tau,n1}, ex_{\tau,n1})$. The following describes the encoder structure design for the task selection agent, while the encoder for the offloading decision agent has a similar structure, with the only difference being the input state, hence it will not be described again to avoid redundancy.

In the encoding process, the input states $(s^c_{\tau,1}, s^c_{\tau,2}, \ldots, s^c_{\tau,N})$ are first preprocessed through a fully connected layer to obtain the initial d_{hid} dimensional embedding vectors $(e_{\tau,1}, e_{\tau,2}, \ldots, e_{\tau,N})$. The calculation formula for this step can be described as:

$$e_{\tau,n} = w_{\tau,n} \cdot s^c_{\tau,n} + b_{\tau,n}, n \in (1, 2, \ldots, N) \tag{17}$$

where $w_{\tau,n}$ and $b_{\tau,n}$ are both learnable parameters of the fully connected layer.

Following this, the encoder uses the output from the previous layer $(e_{\tau,1}, e_{\tau,2}, \ldots, e_{\tau,N})$ and employs a TCN and self-attention mechanisms to extract feature information.

1) *TCN:* To accurately predict and respond to the dynamic changes in user demand within the SatEC environment, we have adopted the TCN (Bai et al. 2018) as a key tool for strategy optimization. TCN is capable of processing and understanding complex patterns in time series data, particularly its ability to identify long-term dependencies between data points, which is crucial for capturing the evolving trends of user demand. In the context of SatEC, user demand not only changes over time but is also influenced by a variety of factors, exhibiting a high degree of uncertainty and dynamism. Therefore, the introduction of TCN not only helps to improve the accuracy of the model in predicting changes in user demand but also

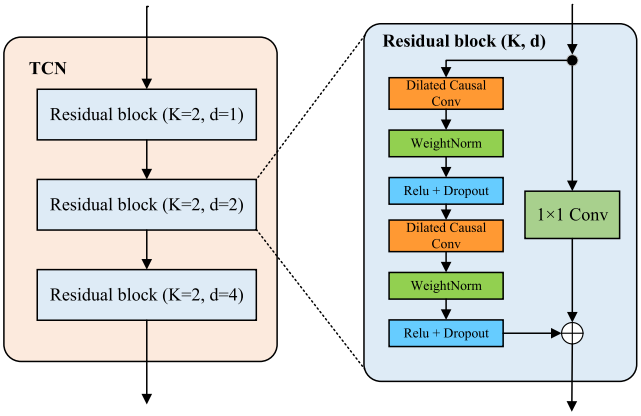


Fig. 3 The structure of the TCN network

enhances the model's adaptability and robustness in the face of complex environments.

As shown in Fig. 3, the TCN network in our paper consists of three residual convolutional blocks, each containing two dilated causal convolutional layers. The design of these layers ensures the correct order of time series data processing and employs residual connections to enhance the learning efficiency and stability of the network, thereby more effectively capturing and predicting the dynamic changes in user demand. The mathematical expression for dilated causal convolution can be represented as:

$$y_\tau = \sum_{k=0}^{K-1} f(k) \cdot x_{\tau - k \cdot d} \tag{18}$$

where $f(k)$ is the learnable weight vector used to slide over the input sequence and perform a weighted sum to produce the output sequence y_τ. $x_{\tau-k\cdot d}$ is the value of the input sequence x at time point $\tau - k \cdot d$. K and d represent the size of the convolution kernel and the dilation factor, respectively. The larger the dilation factor, the larger the receptive field, which can capture more long-term temporal dependencies. It is worth noting that when the dilation factor $d = 1$, this operation degenerates into a standard convolution.

Residual connections (He et al. 2016) are primarily aimed at preventing the vanishing or exploding of gradients. It is widely known that in traditional deep neural networks, as the number of layers increases, gradients may gradually vanish or explode during the backpropagation process, making the network challenging to train. Residual connections establish direct pathways between layers, allowing gradients to be more easily propagated back to the earlier layers, thus mitigating the issues of vanishing and exploding gradients. This improvement allows the model to be trained within deeper neural networks, thereby enhancing the model's predictive capabilities. In this paper, the process can be described as follows:

$$O^1_{\tau,n} = ReLU(Conv(e_{\tau,n}) + \Omega(e_{\tau,n})), n \in (1, 2, \ldots, N) \tag{19}$$

where $\Omega(x)$ is the result obtained after two layers of dilated causal convolution. $Conv$ is the convolution operation in the residual connection, with a kernel size of 1, used for dimension adjustment.

2) *Self Attention:* To further enhance the model's sensitivity and predictive accuracy to changes in user demand, we have also introduced an self-attention mechanism (Vaswani et al. 2017). The self-attention mechanism offers the model greater flexibility, enabling it to automatically focus on the most informative parts of the time series data, which is crucial for identifying and emphasizing the important features of user demand. In the scenario of SatEC, data from different time points have varying degrees of influence on user demand. The self-attention mechanism allows the model to dynamically adjust weights based on the importance of the data, thus capturing the subtle differences in demand changes with greater precision. Compared to TCN, the self-attention mechanism provides a different perspective for understanding the dynamic changes in time series. By incorporating self-attention, our model can better capture the multi-level features of time data, providing more information for the tasks that require prediction.

Specifically, the self-attention mechanism first generates a query vector $\boldsymbol{q}_{\tau,n} = \boldsymbol{w}_q \cdot e_{\tau,n}$, a key vector $\boldsymbol{k}_{\tau,n} = \boldsymbol{w}_k \cdot e_{\tau,n}$, and a value vector $\boldsymbol{v}_{\tau,n} = \boldsymbol{w}_v \cdot e_{\tau,n}$ for each data point in the embedded vector $(e_{\tau,1}, e_{\tau,2}, \ldots, e_{\tau,N})$. Subsequently, the self-attention mechanism computes the dot product between each query vector and all key vectors to obtain a set of attention scores. These scores reflect the similarity or relevance between the query vector and each key vector. To make the scores more stable and easier to handle, we use the softmax function to normalize these scores. The formula for the attention score $\xi^j_{\tau,n}$ can be described as:

$$\xi^j_{\tau,n} = softmax(\frac{\boldsymbol{q}_{\tau,n} \cdot \boldsymbol{k}^T_{\tau,j}}{\sqrt{d_k}}), j \in (1, 2, \ldots, N) \tag{20}$$

where $\boldsymbol{k}^T_{\tau,j}$ is the transpose of the key vector $\boldsymbol{k}_{\tau,j}$, and d_k is the dimension of the key vector $\boldsymbol{k}_{\tau,j}$.

After normalization, the output for each data point is obtained through a weighted sum, where the weights are the normalized attention scores. This way, each data point incorporates information from other data points in the

embedded vector. The formula can be described as:

$$O^2_{\tau,n} = \sum_{j=1}^{N} \xi^j_{\tau,n} \cdot \boldsymbol{v}_{\tau,j},\ j \in (1, 2, \ldots, N) \tag{21}$$

Finally, we sum the output results of the two branches and input them into a fully convolutional network with a convolution kernel size of 1, which gives us the output of the encoder. This process can be expressed as:

$$O_{\tau,n} = Conv(O^1_{\tau,n} + O^2_{\tau,n}), n \in (1, 2, \ldots, N) \tag{22}$$

4.2.2 Decoder

In task ordering problems, the elements of the output sequence are typically selected from the input sequence, and the order and elements of the output sequence have a direct correspondence with the input sequence. Influenced by the Pointer Network proposed by Vinyals et al. (2015), we use the output of the encoder as the input for the decoder, calculate the attention weights to obtain the probability distribution of actions, and then select the output action based on this probability distribution. Below, we will elaborate on the composition of the decoder.

1) *Probability Distribution Calculation:* First, we take the average of the encoder's output results ($O_{\tau,1}, O_{\tau,2}, \ldots, O_{\tau,N}$) to get $\bar{O}_\tau = \frac{1}{N}(O_{\tau,1} + O_{\tau,2} + \ldots + O_{\tau,N})$. Next, we perform embedding operations ($\boldsymbol{Pt}_\tau = \boldsymbol{w}_z \cdot \bar{O}_\tau$, $\boldsymbol{Ma}_{\tau,n} = \boldsymbol{w}_{\boldsymbol{m}} \cdot O_{\tau,n}$) on this average $\bar{O}_\tau$ and each output $O_{\tau,n}$ of the encoder to obtain the pointer vector $\boldsymbol{Pt}_\tau$ and the matching vector $\boldsymbol{Ma}_{\tau,n}$. Then, we calculate the attention weights $pro_{\tau,n}$ for the matching vector $\boldsymbol{Ma}_{\tau,n}$ based on this pointer vector $\boldsymbol{Pt}_\tau$, which are also the priorities mentioned above. The larger the attention weight, the higher the priority of the task. This can be described as:

$$pro_{\tau,n} = softmax(C \cdot tanh(u_{\tau,n})), n \in (1, 2, \ldots, N) \tag{23}$$

$$u_{\tau,n} = \begin{cases} \dfrac{\boldsymbol{Pt}^T_\tau \cdot \boldsymbol{Ma}_{\tau,n}}{\sqrt{d_{hid}}}, \text{If task n has not been selected} \\ -\infty, \quad \text{else} \end{cases} \tag{24}$$

In (23), $tanh(\cdot)$ is the activation function. C is a fixed hyperparameter, and $C \cdot tanh(u_{\tau,n})$ serves to clip the result of the activation function to the range $[-C, C]$. By limiting the contribution range of each $u_{\tau,n}$, it ensures that when calculating the attention weights $pro_{\tau,n}$, no single input will have an excessively large or small influence on the output result, which helps to improve the stability of numerical calculations. In (24), $\boldsymbol{Pt}^T_\tau$ is the transpose of the pointer vector $\boldsymbol{Pt}_\tau$. d_{hid} is the dimension of the matching vector $\boldsymbol{Ma}_{\tau,n}$.

In task ordering problems, each task can only be selected once, so we set the parameter $u_{\tau,n}$ for tasks that have already been selected to negative infinity to ensure that these tasks will not be selected again in subsequent scheduling steps. However, in the offloading decision agent, there is no issue of repeated selection, so the definition of the parameter $u_{\tau,n}$ is:

$$u_{\tau,n} = \frac{\boldsymbol{Pt}^T_\tau \cdot \boldsymbol{Ma}_{\tau,n}}{\sqrt{d_{hid}}} \tag{25}$$

2) *Action Selection:* After obtaining the probability distribution of tasks, we can select the output action based on these probabilities. In the task selection agent, the action corresponds to the index of the IoT terminal associated with the task. In the offloading decision agent, the action represents whether the task should be offloaded or not. During training, we randomly select actions based on the probability values $pro_{\tau,n}$ for each task, which helps the model learn a broader solution space and avoid getting trapped in local optima. In the testing phase, we employ a greedy strategy to determine the actions, which ensures that the model makes the most likely correct decisions.

4.3 Model training

The training process of the MA2C model achieves joint optimization of task prioritization and offloading decisions through a multi-agent collaborative framework. The model comprises two Actor networks (for task selection and offloading decisions, respectively) and a shared Critic network. Initially, all network parameters (θ^c, θ^o, and ζ) are initialized, followed by dynamic interaction with the environment to collect training data. At each timestep τ, the task selection agent Actor-c generates an action probability distribution $\pi_{\theta^c}(a^c_\tau \mid s^c_\tau)$ based on the current environmental state s^c_τ (including task data size, local computing capability, etc.), and samples the highest-priority task index a^c_τ. This action triggers a state update s^o_τ (containing timing features for local/satellite execution) for the offloading decision agent Actor-o, which then generates an offloading probability $\pi_{\theta^o}(a^o_\tau \mid s^o_\tau)$ and samples the decision a^o_τ. After executing the joint action (a^c_τ, a^o_τ), the environment returns an immediate reward r_τ and the next state $(s^c_{\tau+1}, s^o_{\tau+1})$. The Critic network evaluates the state value through $v_\zeta(s^c_\tau, s^o_\tau)$ and computes the advantage function δ_τ to measure the deviation between actual and predicted returns:

$$\delta_\tau = r_\tau + \gamma v_\zeta(s^c_{\tau+1}, s^o_{\tau+1}) - v_\zeta(s^c_\tau, s^o_\tau) \tag{26}$$

The Actor network updates parameters θ^c and θ^o through the policy gradient $\nabla L(\theta^c), \nabla L(\theta^o)$, and its gradient is driven by the product of the advantage function and the logarithmic probability of actions:

$$\nabla L(\theta^c) = -\delta_\tau \nabla log \pi_{\theta^c}(a_\tau^c \mid s_\tau^c) \tag{27}$$

$$\nabla L(\theta^o) = -\delta_\tau \nabla log \pi_{\theta^o}(a_\tau^o \mid s_\tau^o) \tag{28}$$

The Critic network optimizes the state value prediction by minimizing the temporal difference error and backpropagates to update the parameters ζ. Its loss function $L(\zeta)$ and gradient $\nabla L(\zeta)$ can be defined as:

$$L(\zeta) = \frac{1}{2}\delta_\tau^2 \tag{29}$$

$$\nabla L(\zeta) = -\delta_\tau \nabla v_\zeta(s_\tau^c, s_\tau^o) \tag{30}$$

After completing these updates, we transition the state to the next state and continue to the next iteration until all training epochs are completed. The training process of MA2C is shown in Algorithm 1.

Algorithm 1 The training process of MA2C.

Input: Status information, s; Total epoch number, E; Training times per epoch, T;
Output: The most appropriate offloading action a^c, a^o;
Initialize the parameters of the actor networks θ^c, θ^o and the parameters of the critic networks ζ;
for $epoch = 1$ **to** E **do**
  Initialize state $s^c = (s_1^c, s_2^c, \ldots, s_n^c)$;
  for $t = 1$ **to** T **do**
    Compute the action probability distribution $\pi_{\theta^c}(a_\tau^c \mid s_\tau^c)$ for the Actor-c network;
    Sample actions $a_\tau^c \sim \pi_{\theta^c}(a_\tau^c \mid s_\tau^c)$;
    Update the state $s_\tau^o = \{s_{\tau,n0}^o, s_{\tau,n1}^o\}$ of the Actor-o network based on action a_τ^c;
    Compute the action probability distribution $\pi_{\theta^o}(a_\tau^o \mid s_\tau^o)$ for the Actor-o network;
    Sample actions $a_\tau^o \sim \pi_{\theta^o}(a_\tau^o \mid s_\tau^o)$;
    Perform actions a_τ^c, a_τ^o in the environment to get a reward r_τ and the next state $(s_{\tau+1}^c, s_{\tau+1}^o)$;
    Calculate the advantage function $\delta_\tau = r_\tau + \gamma v_\zeta(s_{\tau+1}^c, s_{\tau+1}^o) - v_\zeta(s_\tau^c, s_\tau^o)$;
    Update the actor network $\theta^c = \theta^c + \alpha_{actor}\delta_\tau \nabla_{\theta^c} log \pi_{\theta^c}(a_\tau^c \mid s_\tau^c)$ $\theta^o = \theta^o + \alpha_{actor}\delta_\tau \nabla_{\theta^o} log \pi_{\theta^o}(a_\tau^o \mid s_\tau^o)$
    Update the critic network $\zeta = \zeta + \alpha_{critic}\delta_\tau \nabla v_\zeta(s_\tau^c, s_\tau^o)$
    $s_\tau^c, s_\tau^o \leftarrow s_{\tau+1}^c, s_{\tau+1}^o$;
  end for
end for

4.4 Complexity and convergence analysis

1) *Complexity Analysis:* The time complexity of the MA2C algorithm is mainly determined by the encoding and decoding process of the task selection agent and offloading decision agent, the multi-agent collaboration mechanism, and the model training process. In the encoder part, both the task selection agent and the offloading decision agent adopt the structure of TCN and SA. TCN extracts time-series features through multi-layer dilated causal convolutions, with each convolution operation having a time complexity of $\mathcal{O}(KNH^2)$. Here, K represents the kernel size, N is the number of task inputs, and H denotes the hidden layer dimension. Assuming that the TCN contains L residual blocks, the total complexity of the TCN is $\mathcal{O}(LKNH^2)$. Meanwhile, the SA mechanism calculates query, key, and value vectors along with attention scores for each task, introducing a complexity of $\mathcal{O}(N^2H)$. The decoder relies on a pointer network to generate action probability distributions. The complexity of calculating attention weights in each step is $\mathcal{O}(N^2H)$, while the complexity of discrete action sampling is $\mathcal{O}(N)$. During the training phase, two actor networks and one shared critic network need to be updated at each time step. Let the number of training rounds be E, and the number of time steps per round be T, then the total time complexity of the MA2C model is: $\mathcal{O}(E \cdot T \cdot (2(LKNH^2 + N^2H + N^2H + N) + LKNH^2 + N^2H))$.

2) *Convergence Analysis:*

Lemma 1 *(Convergence of the Critic) Given a fixed policy $\pi_{\theta^c}, \pi_{\theta^o}$ assume the Critic uses a linear function approximation for the value function $v_\zeta(s_\tau^c, s_\tau^o)$, with linearly independent feature vectors. If the step-size sequence α_τ satisfies the Robbins-Monro conditions $\sum_{\tau=0}^{\infty}\alpha_\tau = \infty$ and $\sum_{\tau=0}^{\infty}\alpha_\tau^2 < \infty$, then the Critic parameters ζ converge almost surely to the unique TD fixed point $\zeta^*(\theta^c, \theta^o)$, which satisfies the equation:*

$$\mathbb{E}_{s_\tau^c \sim p^c, s_\tau^o \sim p^o, a_\tau^c \sim \pi_{\theta^c}, a_\tau^o \sim \pi_{\theta^o}}[\delta_\tau \nabla v_\zeta(s_\tau^c, s_\tau^o)] = 0 \tag{31}$$

where $\delta_\tau = r_\tau + \gamma v_\zeta(s_{\tau+1}^c, s_{\tau+1}^o) - v_\zeta(s_\tau^c, s_\tau^o)$ is the temporal difference (TD) error, and p^c, p^o are the stationary state distribution induced by policies π_{θ^c} and π_{θ^o}, respectively.

Proof By the convergence theory of TD(0) algorithms (Tsitsiklis and Van Roy 1996), when the value function approximator is linear and the feature matrix has full column rank, the Critic's update is equivalent to solving a least-squares projection problem. The parameter ζ is updated via stochastic gradient descent:

$$\zeta_{\tau+1} = \zeta_\tau + \alpha_t \delta_\tau \nabla v_\zeta(s_\tau^c, s_\tau^o) \tag{32}$$

where δ_τ is the TD error at time τ. According to the linear TD convergence theorem, under the Robbins-Monro conditions, the iteration process converges almost surely to the unique solution $\zeta^*(\theta^c, \theta^o)$ that satisfies $\mathbb{E}[\delta_\tau \nabla v_\zeta(s_\tau^c, s_\tau^o)] = 0$. □

Lemma 2 *(Unbiasedness of the Actor's gradient) Assume the Critic parameters ζ converge to $\zeta^*(\theta^c, \theta^o)$, and the function approximator satisfies the compatibility condition $\nabla_\zeta Q(s_\tau^c, a_\tau^c) = \nabla_{\theta^c} log\pi_{\theta^c}(a_\tau^c \mid s_\tau^c)$, $\nabla_\zeta Q(s_\tau^o, a_\tau^o) = \nabla_{\theta^o} log\pi_{\theta^o}(a_\tau^o \mid s_\tau^o)$. Then, the Actor's update direction aligns with the true policy gradient:*

$$\nabla_{\theta^c} J(\theta^c) = \mathbb{E}_{s_\tau^c \sim p^c, a_\tau^c \sim \pi_{\theta^c}} \{\nabla_{\theta^c} log\pi_{\theta^c}(a_\tau^c \mid s_\tau^c) A_\zeta[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)]\} \tag{33}$$

$$\nabla_{\theta^o} J(\theta^o) = \mathbb{E}_{s_\tau^o \sim p^o, a_\tau^o \sim \pi_{\theta^o}} \{\nabla_{\theta^o} log\pi_{\theta^o}(a_\tau^o \mid s_\tau^o) A_\zeta[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)]\} \tag{34}$$

where $A_\zeta[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)] = Q_\zeta[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)] - v_\zeta(s_\tau^c, s_\tau^o)$ is the advantage function estimated by the Critic.

Proof By the policy gradient theorem (Sutton et al. 1999), the gradient of the objective function $J(\theta_\tau^c) = \mathbb{E}_{s_\tau^c \sim p^c}[v^{\pi_{\theta^c}}(s_\tau^c)]$ and $J(\theta_\tau^o) = \mathbb{E}_{s_\tau^o \sim p^o}[v^{\pi_{\theta^o}}(s_\tau^o)]$ are:

$$\nabla_{\theta^c} J(\theta^c) = \mathbb{E}_{s_\tau^c \sim p^c, a_\tau^c \sim \pi_{\theta^c}} \{\nabla_{\theta^c} log\pi_{\theta^c}(a_\tau^c \mid s_\tau^c) Q^{\pi_{\theta^c}, \pi_{\theta^o}}[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)]\} \tag{35}$$

$$\nabla_{\theta^o} J(\theta^o) = \mathbb{E}_{s_\tau^o \sim p^o, a_\tau^o \sim \pi_{\theta^o}} \{\nabla_{\theta^o} log\pi_{\theta^o}(a_\tau^o \mid s_\tau^o) Q^{\pi_{\theta^c}, \pi_{\theta^o}}[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)]\} \tag{36}$$

When the Critic converges to $\zeta^*(\theta^c, \theta^o)$, the advantage function estimate satisfies $A_\zeta[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)] = Q_\zeta[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)] - v_\zeta(s_\tau^c, s_\tau^o) \approx Q^{\pi_{\theta^c}, \pi_{\theta^o}}[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)] - v^{\pi_{\theta^c}, \pi_{\theta^o}}(s_\tau^c, s_\tau^o)$. The compatibility condition ensures that $Q_\zeta[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)]$ aligns with the policy gradient direction, and the approximation error $\epsilon[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)] = Q_\zeta[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)] - Q^{\pi_{\theta^c}, \pi_{\theta^o}}[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)]$ becomes orthogonal to $\nabla_{\theta^c} log\pi_{\theta^c}(a_\tau^c \mid s_\tau^c)$, $\nabla_{\theta^o} log\pi_{\theta^o}(a_\tau^o \mid s_\tau^o)$. Thus, the Actor's update direction $\mathbb{E}\{\nabla_{\theta^c} log\pi_{\theta^c}(a_\tau^c \mid s_\tau^c) A_\zeta[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)]\}$ matches $\nabla_{\theta^c} J(\theta^c)$ and $\mathbb{E}\{\nabla_{\theta^o} log\pi_{\theta^o}(a_\tau^o \mid s_\tau^o) A_\zeta[(s_\tau^c, s_\tau^o), (a_\tau^c, a_\tau^o)]\}$ matches $\nabla_{\theta^o} J(\theta^o)$. □

Theorem 1 *(Local convergence of MA2C algorithm) Let the step-size sequences $\{\alpha_{actor}\}$ for the Actor and $\{\alpha_{critic}\}$ for the Critic satisfy the two-timescale condition $\lim_{t \to \infty} \alpha_{actor}/\alpha_{critic} = 0$ and the Robbins-Monro conditions. Then, the joint parameter sequence $((\theta_\tau^c, \theta_\tau^o), \zeta_\tau)$ converges almost surely to a local optimal policy parameter $((\theta^{c*}, \theta^{o*}), \zeta^*(\theta^{c*}, \theta^{o*}))$.*

Proof By the two-timescale stochastic approximation theory (Konda and Tsitsiklis 1999), the Critic's updates (fast timescale) track the value function of the current policy $(\pi_{\theta_\tau^c}, \pi_{\theta_\tau^o})$ under quasi-static conditions. From Lemma 1, when $(\theta_\tau^c, \theta_\tau^o)$ evolves sufficiently slowly, ζ_τ converges almost surely to $\zeta^*(\theta_\tau^c, \theta_\tau^o)$. Meanwhile, the Actor's updates (slow timescale) approximate the ordinary differential equation (ODE):

$$\dot{\theta^c} = \nabla_{\theta^c} J(\theta^c) \tag{37}$$

$$\dot{\theta^o} = \nabla_{\theta^o} J(\theta^o) \tag{38}$$

whose equilibrium points correspond to local maxima of $J(\theta)$. Since gradient ascent dynamics monotonically improve the objective function, combined with the unbiased gradient direction in Lemma 2, the parameters $(\theta_\tau^c, \theta_\tau^o)$ converge to a local optimum $(\theta^{c*}, \theta^{o*})$. Consequently, the joint system $((\theta_\tau^c, \theta_\tau^o), \zeta_\tau)$ converges almost surely to $((\theta^{c*}, \theta^{o*}), \zeta^*(\theta^{c*}, \theta^{o*}))$, proving that the MA2C algorithm converges to a locally optimal policy. □

5 Simulation results and analysis

In this section, we first outline the dataset and parameter settings for the simulation experiments. Next, we experimentally analyze some key parameters in the MA2C algorithm and verify the convergence of the MA2C algorithm. Following this, we compare the performance with nine baseline algorithms in terms of task data size, the number of IoT terminals, the parallel computing capabilities of the satellite system, and task deadlines. Finally, we conduct ablation experiments on MA2C to demonstrate the correctness of our proposed method.

5.1 Simulation dataset generation and parameter settings

Since the real operational data and hardware parameters of the SatEC system are difficult to obtain directly, this study refers to the typical parameter ranges in authoritative literature (Tang et al. 2021; Jia et al. 2023; Leyva-Mayorga et al. 2023; Zhang et al. 2023a) in the field of satellite communication and edge computing. The dataset is constructed by combining uniform distribution sampling with fixed value assignment, and Python 3.9 and PyTorch 2.0.1 are used to build the simulation experiment environment.

In terms of task attributes, the amount of task data $d_{\tau,n}$ generated by each IoT terminal in each time slot follows a uniform distribution Unif(100, 2500) KB. This range references the data characteristics of typical sensing tasks in low-earth orbit satellite IoT scenarios (Tang et al. 2021), covering a variety of workloads from environmental monitoring to high-definition image transmission. The task deadline $t_{\tau,n}^{DL}$ is set

to Unif(0.2, 0.6) to ensure that the tasks have time-sensitive challenges.

For network parameter configuration, N IoT terminals are randomly and uniformly distributed within a 1200m×1200m area. The distance between the terminals and the access satellites is set to Unif(400, 2000) km to match the coverage characteristics of low-earth orbit satellites (Tang et al. 2021). In the channel parameters, the transmission bandwidth $b_{\tau,n} \in (1, 4)$ MHz adopts the Ku-band standard. The transmission power $p_{\tau,n} \in (1, 5)$ W conforms to the constraints of small terminals (Jia et al. 2023). The channel gain $g_{\tau,n,m} \in (-40, -20)$ dB is calculated based on the free-space path loss model (Leyva-Mayorga et al. 2023). The noise power $\sigma^2 = 1.0 \times 10^{-11}$ corresponds to the typical receiver noise coefficient.

For inter-satellite links configuration, the communication distance $D_{j,m}$ between satellites is set to Unif(600, 1200) km, which is consistent with the typical orbital spacing of low-earth orbit satellite constellations. The satellite transmission bandwidth B_m^{sat} uses the Ka-band laser communication link, with values randomly generated from the uniform distribution over the interval [400, 600] MHz. Referring to the parameters of the International Space Station's inter-satellite communication module (Zhang et al. 2023a), the satellite transmission power p_m^{sat} is randomly generated from the uniform distribution over the interval [25, 30] dBW. The channel gain $g_{\tau,j,m}^{sat}$ between satellites is also calculated based on the free-space path loss model, with values following a uniform distribution Unif(-35, -25) dB.

In terms of computing resource allocation, the computing capability of IoT terminals f_n^{local} is sampled from Unif(1, 5) GHz, simulating the differences in computing resources among heterogeneous terminal devices. The computing power of the satellite server f_m^{sat} is randomly generated from the uniform distribution over the interval [10, 20] GHz, simulating high-performance processors on satellites. The computational density $\kappa = 500$ matches x86 architecture instruction cycles.

Based on the above description, we construct the experimental dataset $s_\tau^c = (s_{\tau,1}^c, s_{\tau,2}^c, \ldots, s_{\tau,n}^c, \ldots, s_{\tau,N}^c), \tau \in \mathcal{T}$, where $s_{\tau,n}^c = \{d_{\tau,n}, f_{\tau,n}^{local}, p_{\tau,n}, g_{\tau,n,m}, t_{\tau,n}^{DL}\}, n \in \mathcal{N}$, for subsequent training. For testing, 100 time-series datasets are generated using the same method as the training set. To ensure reproducibility, the random seed is fixed at $seed = 12$. Table 3 provides a comprehensive summary of the simulation configuration parameters in this study.

In the MA2C model, we use self-attention mechanism and temporal convolutional network (TCN) as encoders to extract multi-dimensional feature information, and utilize a pointer network as the decoder to generate policy actions. The TCN consists of 3 residual blocks with dilated causal convolutions. Each block uses a convolution kernel size of 2 to balance local feature capture capability and computational efficiency. Additionally, a dropout rate of 0.2 is applied in the TCN layer to prevent overfitting and enhance generalization ability in dynamic scenarios. During the model training process, a total of 24 iterations are conducted, each containing 20,000 time steps. Throughout this process, hyperparameters such as learning rates, batch sizes, and hidden layer dimensions of the actor network and critic network are carefully tuned to promote stable convergence and optimal policy learning.

Table 3 Parameters of simulation experiment

Parameters	Value
The number of Iot terminals N	{10, 20, 30, 40, 50}
Parallel computing capability of satellite systems M	{1, 2, 3, 4, 5}
The data size of the task $d_{\tau,n}$	Unif(100,2500) KB
The deadline for the task $t_{\tau,n}^{DL}$	Unif(0.2, 0.6) s
The computational capability of task on the local Iot terminal f_n^{local}	Unif(1, 5) GHz
The computational capability of the satellite server f_m^{sat}	Unif(10, 20) GHz
CPU cucles to compute one-bit of data κ	500
Communication bandwidth between the IoT terminal user n and the satellite $b_{\tau,n}$	Unif(1, 4) MHz
The inter-satellite communication bandwidth B_m^{sat}	Unif(400, 600)MHz
Transmission power of IoT terminal user $p_{\tau,n}$	Unif(1, 5) W
Transmission power of satellite p_m^{sat}	Unif(25, 30) dBW
Channel gain between the IoT terminal user n and the satellite m $g_{\tau,n,m}$	Unif(-40, -20) dB
Channel gain between access satellite j and computing satellite m $g_{\tau,j,m}^{sat}$	Unif(-35, -25) dB
additive white Gaussian noise power σ^2	1.0×10^{-11} W
The distance from IoT terminal user n to access satellite j $D_{n,j}$	Unif(400,2000) Km
The distance between the access satellite j and the computing satellite m $D_{j,m}$	Unif(600,1200) Km
The propagation speed of wireless signals c	3.0×10^8 m/s

Table 4 Hyperparameter settings for MA2C

Hyperparameter	Value
Total epoch number E	24
Training times per epoch T	20000
Learning rate of actor network α_{actor}	5.0×10^{-6}
Learning rate of critic network α_{critic}	5.0×10^{-5}
Batch size	24
Hidden dimension H	128
Discount factor γ	0.99
Convolution kernel size of TCN K	2
dropout ratio	0.2
Clip parameter C	10

The specific settings of these hyperparameters are detailed in Table 4.

5.2 Hyperparameter and convergence analysis

To achieve the optimal scheduling performance of the MA2C algorithm in complex satellite edge computing environments and verify its robustness, this section conducts a systematic experiment to analyze the sensitivity of key hyperparameters and evaluates the convergence characteristics of the algorithm in different dynamic scenarios. The experiment adopts the control variable method, fixing the number of IoT terminals $N = 20$, satellite parallel computing capability $M = 2$, task data size $d_{\tau,n} \in (1000, 2000)$ KB, and deadline $t_{\tau,n}^{DL} \in (0.2, 0.6)$ s, focusing on the impact of learning rate, hidden dimension, and batch size on algorithm performance. In addition, to enhance the stability of experimental data and the interpretability of results, this study employs a moving average window method (window width $W = 20$), using the average reward value of 20 consecutive time steps as the core indicator for evaluating algorithm performance. This statistical method effectively eliminates the influence of instantaneous fluctuations on convergence trend analysis while maintaining the dynamic characteristics of time series data.

Firstly, this experiment systematically studies the efficiency of parameter updates for the Actor network. As shown in Fig. 4, the learning rate parameter has a significant impact on the training process: both excessively high and low learning rates can lead to poor results during the training process. When the learning rate is too high, such as $\alpha_{actor} = 0.00001$, the model's volatility during the convergence process is significantly greater than 0.000005. On the other hand, when the learning rate is too low, the rate of parameter updates decreases, causing the model's convergence speed to become very slow, or even fail to converge, as shown by 0.000001 and 0.0000001. Therefore, in this work, we ultimately chose a learning rate of 0.000005 to achieve rapid and stable convergence of the model. Experimental data shows that under this learning rate, the model's convergence curve presents a steady upward trend, with no significant oscillations, and the convergence speed is also relatively fast, enabling the model to reach a satisfactory solution within a reasonable time frame.

Next, we investigated the impact of hidden layer dimensions on the model's representational capacity. As shown in Fig. 5, the number of hidden nodes directly affects model performance: when the number of nodes is set to 64, the network structure is too simple to effectively capture the nonlinear features of the dataset, resulting in high system volatility. Conversely, expanding the hidden layer to 256 nodes provides the model with the ability to fit complex functions, but the excessive response to input noise and random features leads to a decline in generalization performance. After multiple comparative validations, it was determined

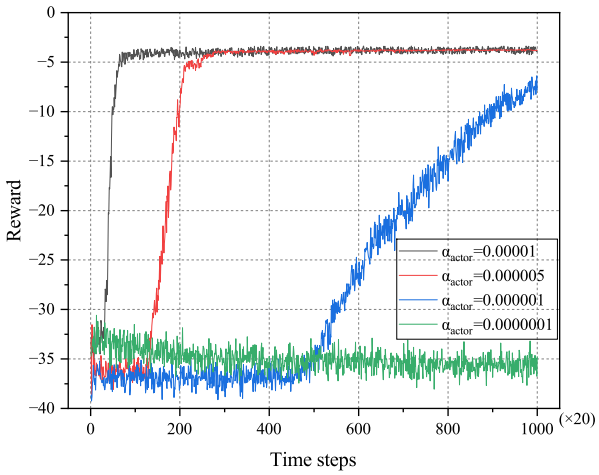


Fig. 4 Convergence performance at different learning rates

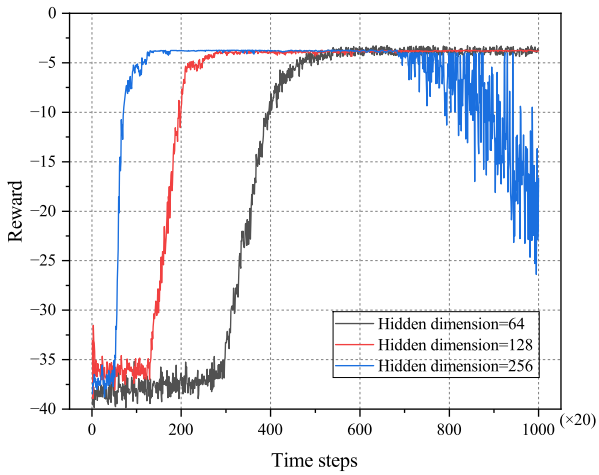


Fig. 5 Convergence performance at different hidden dimension

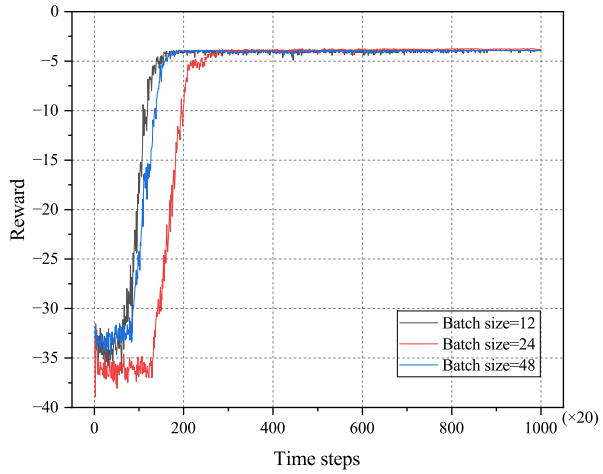


Fig. 6 Convergence performance at different batch size

that a configuration of 128 nodes strikes the best balance between model capacity and generalization requirements. Experimental data indicates that this parameter setting not only maintains a stable error trend during the training process but also effectively prevents overfitting. This design strategy successfully addresses the challenging optimization problem of balancing representational capacity and generalization performance in reinforcement learning models.

Finally, this study further investigated the impact mechanism of batch size on model optimization. As shown in Fig. 6, when training with small batches (e.g., Batch size = 24), gradient updates are susceptible to interference from sample sampling errors, resulting in violent fluctuations in the parameter iteration path. While large-scale batches can enhance the statistical stability of gradient estimation, they also lead to a linear increase in video memory consumption, which may cause computational bottlenecks, especially under limited hardware conditions. Considering the trade-off between training stability and resource constraints, a batch size of 24 was ultimately chosen as a compromise. This setting ensures the robustness of parameter updates while keeping video memory consumption within an acceptable range.

In addition, in order to verify the convergence stability of the MA2C algorithm under diverse scenarios, this study conducted convergence experiments with varying configurations including task data size (500-2000 KB), IoT terminal quantity (10-30), satellite system parallel computing capacity (2-4), and task deadlines (0.2-0.5 s). Figure 7 presents the average reward convergence curves of the MA2C algorithm during training under different parameter settings. Experimental results demonstrate that despite significant variations in task characteristics and resource constraints across different scenarios, the MA2C algorithm consistently achieves stable convergence within a reasonable number of iterations, demonstrating excellent environmental adaptability.

5.3 Performance analysis of MA2C

To evaluate the effectiveness of our proposed method, we compared it with the following nine baseline methods using the total delay for all tasks and the task failure rate as metrics:

- Gurobi: The Gurobi Optimizer is a high-performance mathematical programming optimization software developed by the Gurobi Company in the United States. It is generally regarded by the academic community as the authoritative solver for small to medium-scale optimization problems. By integrating exact algorithms such as branch-and-bound and cutting-plane methods with adaptive heuristic strategies like dynamic node selection policies and presolve strengthening techniques, along with a parallel computing architecture based on Benders decomposition (Xiong et al. 2022), the Gurobi Optimizer demonstrates significant advantages in solving small to medium-scale combinatorial optimization problems. In particular, for task scheduling problems like the one in this study, its unique conflict clause learning mechanism and Lagrangian relaxation bounding algorithms can effectively handle temporal constraints and resource conflicts (Jiao et al. 2024), a characteristic that has been verified in multiple empirical studies (Liu et al. 2024; Chen et al. 2025). However, the precise solving nature of Gurobi comes with the severe challenge of computational complexity. As the problem size increases, the number of branch-and-bound nodes grows exponentially, leading to a sharp increase in memory usage and computation time (Bischoff et al. 2024). This is unacceptable for scenarios like satellite task scheduling that require strong real-time performance. Therefore, in this study, the Gurobi Optimizer is only used as a benchmark to verify the effectiveness of the proposed algorithm.
- MA2C: The algorithm proposed in this paper.
- MA2C w/o TCN: The MA2C algorithm that removes TCN from the encoder.
- MA2C w/o SA: The MA2C algorithm that removes SA from the encoder.
- MA2C w/o TCN&SA: The MA2C algorithm that simultaneously removes TCN and SA from the encoder, leaving it composed solely of two fully connected layers.
- BDQN (Sun and He 2023): A deep Q-network algorithm based on a multi-branch network proposed by Sun et al. has shown good results in task offloading optimization problems in time-varying environments.
- DRTO (Yang et al. 2024b): A task offloading algorithm based on deep reinforcement learning proposed by Yang et al. can effectively reduce the overall system latency.
- GA (Liu et al. 2022): As a classical metaheuristic algorithm, Genetic Algorithm can effectively search for the

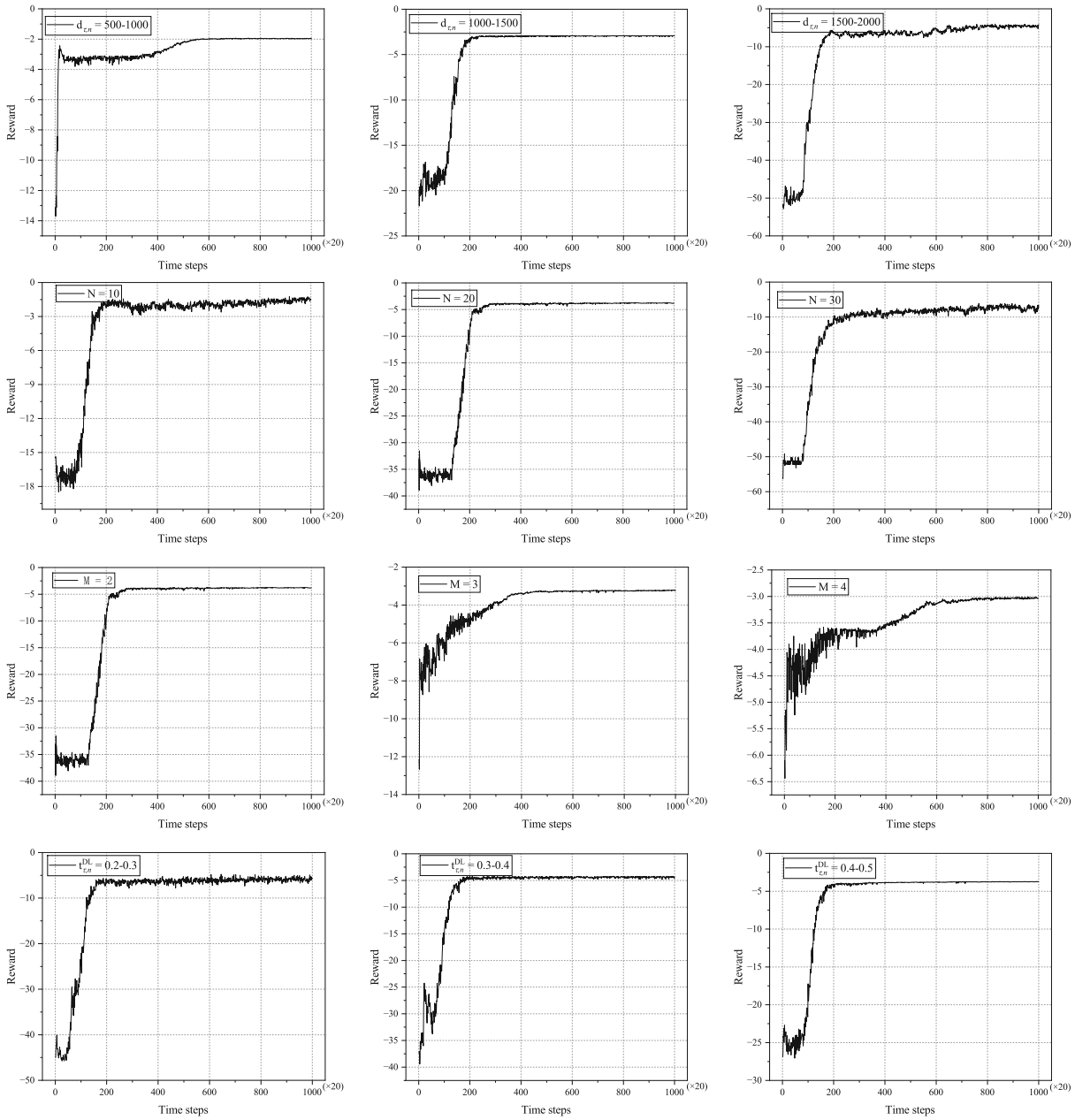
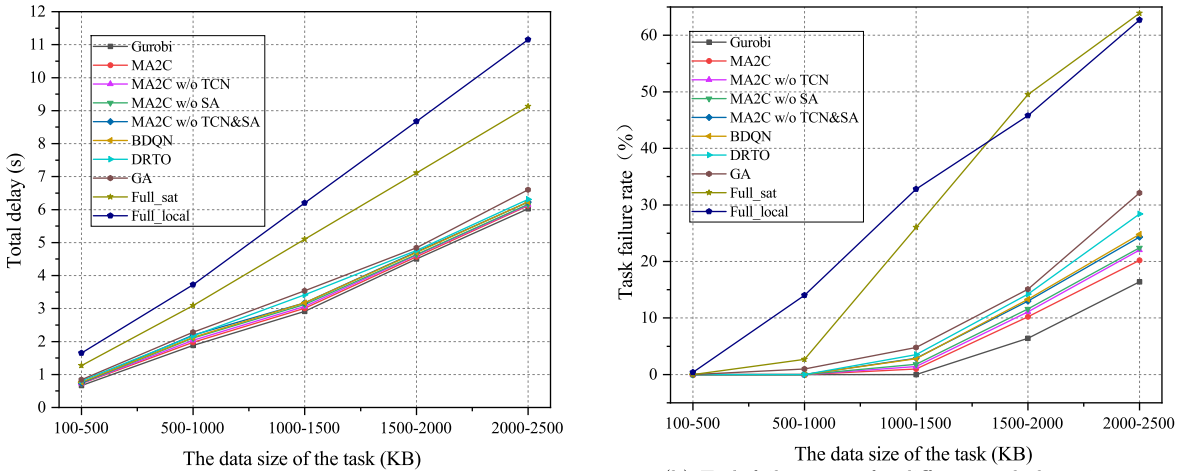


Fig. 7 The convergence curves of the MA2C algorithm in different scenarios

optimal or near-optimal solutions in task offloading problems, demonstrating significant optimization capabilities.

- Full_sat: All tasks are offloaded to the satellite server for computation.
- Full_local: All tasks are computed locally on the IoT terminal without offloading.

Figure 8 and Table 5 illustrates the relationship between task data size and total delay, as well as task failure rate, under different algorithms. In this experimental setup, the number of IoT terminals N is fixed at 20. The satellite system's parallel computing capability M is fixed at 2. Additionally, the deadline for tasks $t_{\tau,n}^{DL}$ is randomly generated from the uniform distribution over the interval [0.2, 0.6], i.e., $t_{\tau,n}^{DL} \sim U(0.2, 0.6)$. Experimental data indicates that as the task data size increases from 100-500KB to 2000-2500KB, the total delay and task failure rate of all algorithms show a stepwise upward trend, with Full_local and Full_sat strategies performing the worst. For instance, at 2000-2500KB, Full_local's delay reaches 11.15 s (failure rate of 62.7%), while Full_sat, although having slightly lower delay (9.13



(a) Total delay for different task data sizes.

(b) Task failure rates for different task data sizes.

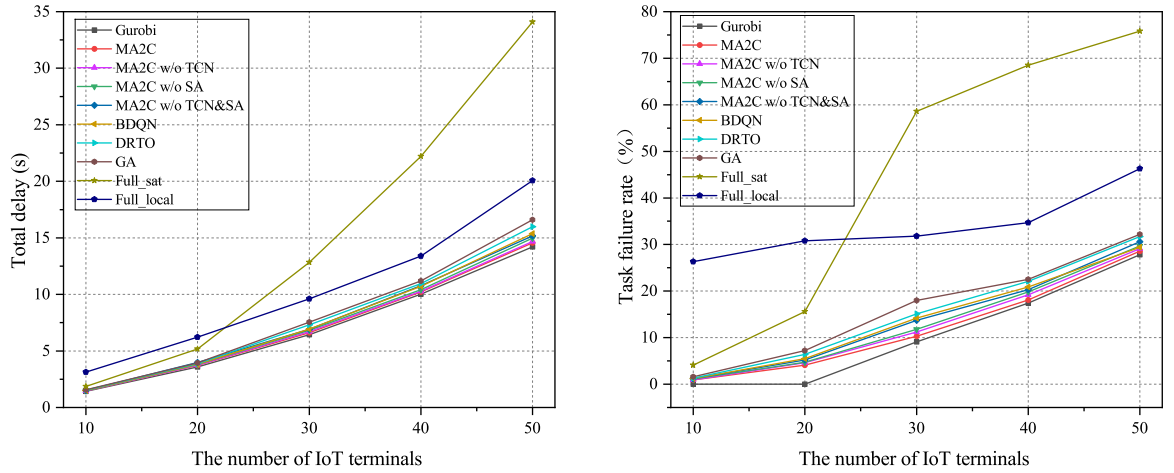
Fig. 8 The relationship between task data size and delay as well as failure rate

s), suffers from a high failure rate of 63.9% due to queuing congestion on satellite servers. In contrast, Gurobi, as an exact optimizer, consistently maintains the best performance (delay of 6.02 s/failure rate of 16.4%), but its longer solving time limits its real-time application. Among intelligent scheduling algorithms, MA2C stands out with the most prominent performance, showing significantly better delay (6.12 s) and failure rate (20.2%) than other baselines. Especially for tasks larger than 1500KB, its failure rate is 3-8 percentage points lower than algorithms like BDQN and DRTO, thanks to its multi-agent collaboration mechanism and dynamic feature extraction network that accurately models task characteristics and resource conflicts. Ablation study further validates the necessity of model components: Removing either TCN or SA from the MA2C algorithm increased failure rates by 1.8% and 2.2% respectively within the 2000-2500KB range, while simultaneous removal caused a 4.1% increase. This also demonstrates the crucial complementary role of SA and TCN in capturing spatiotemporal dependencies. Additionally, traditional methods like the GA struggle to adapt to dynamic environments, resulting in higher delay and failure rates across all stages compared to deep reinforcement learning methods, highlighting the advantages of reinforcement learning-based intelligent scheduling algorithms.

Figure 9 and Table 6 shows the relationship between the number of IoT terminals and the total delay, as well as the task failure rate, under different algorithms. In this experimental setup, the satellite system's parallel computing capability M is fixed at 2. The data size of tasks $d_{\tau,n}$ is randomly generated from the uniform distribution over the interval [1000, 2000], i.e., $d_{\tau,n} \sim U(1000, 2000)$. Additionally, the deadline for tasks $t_{\tau,n}^{DL}$ is randomly generated from the uniform distribution over the interval [0.2, 0.6], i.e., $t_{\tau,n}^{DL} \sim U(0.2, 0.6)$. The experimental results show that as the number of IoT terminals expands from 10 to 50, the total system delay increases significantly. For example, under the 50-terminal scenario, the Full_sat strategy's delay soared to 34.09 s (failure rate 75.8%), and its task accumulation phenomenon caused by computational resource bottlenecks is more severe than the computational limitations of the Full_local strategy (delay 20.07 s/failure rate 46.3%). Although Gurobi still maintains the lowest failure rate (17.4%) at 40 terminals, its characteristic of exponentially increasing solving time with the number of terminals makes it difficult to apply to practical satellite scheduling systems. Among the practical intelligent algorithms, MA2C achieves a balance between computational efficiency and optimization effect through a reinforcement learning architecture with priority sorting. When the number of terminals is 30, its failure rate (10.3%) is reduced by 3.9

Table 5 Task delay and failure rate under different task data sizes

Task data size	Gurobi	MA2C	MA2C w/o TCN	MA2C w/o SA	MA2C w/o TCN&SA	BDQN	DRTO	GA	Full_sat	Full_local
100-500	0.6633/0%	0.7273/0%	0.7316/0%	0.7381/0%	0.7504/0%	0.7665/0%	0.8078/0%	0.8428/0%	1.2709/0%	1.6470/0.4%
500-1000	1.8837/0%	1.9735/0%	2.0362/0%	2.1081/0%	2.1962/0%	2.1282/0%	2.1750/0%	2.2784/1%	3.0875/2.7%	3.7235/14%
1000-1500	2.9145/0%	3.0065/1%	3.0621/1.4%	3.1206/1.8%	3.1729/2.9%	3.1706/2.8%	3.4158/3.5%	3.5410/4.8%	5.1005/26%	6.1985/32.8%
1500-2000	4.5013/6.4%	4.5705/10.2%	4.6148/11.1%	4.6351/11.6%	4.7174/13%	4.7023/13.3%	4.7638/14.2%	4.8436/15.1%	7.1135/49.5%	8.6735/45.8%
2000-2500	6.0248/16.4%	6.1165/20.2%	6.1418/22%	6.1620/22.4%	6.2242/24.3%	6.2314/24.8%	6.3111/28.4%	6.6005/32.1%	9.1265/63.9%	11.1485/62.7%



(a) Total delay for different numbers of IoT terminals. (b) Task failure rates for different numbers of IoT terminals.

Fig. 9 The relationship between the number of IoT terminals and delay as well as failure rate

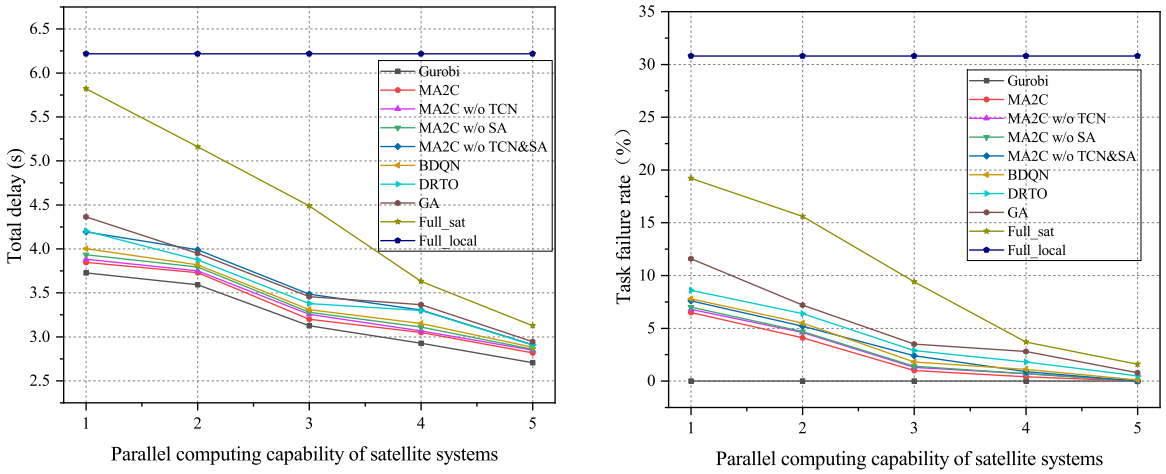
and 4.8 percentage points compared to BDQN (14.2%) and DRTO (15.1%), respectively, thanks to its multi-agent collaborative optimization mechanism, which can adjust task offloading decisions in real-time based on satellite load and task characteristics. The ablation study reveals the feature-capturing capabilities of the model's network components: In the 30-terminal scenario, removing the SA module caused the MA2C algorithm's latency to increase by 3.3% (from 6.62s to 6.84s). Removing the TCN module led to a 0.9% increase in the MA2C algorithm's failure rate (from 10.3% to 11.2%). When both modules were removed simultaneously, the failure rate increased even more significantly by 3.4% (from 10.3% to 13.7%). This further confirms the necessity of the combined SA and TCN modules for capturing temporal dependencies in satellite scheduling for time-sensitive task data. Compared to traditional methods, the MA2C algorithm reduces the failure rate by 3.6 percentage points (32.2%→28.6%) under 50 terminals compared to the GA.

Figure 10 and Table 7 provides an in-depth analysis of the interaction between the parallel computing capabilities of different algorithms in satellite systems and the total delay and task failure rate. In this experimental setup, the number of IoT terminals N is fixed at 20. The data size of tasks $d_{\tau,n}$ is randomly generated from the uniform distribution over the interval [1000, 2000], i.e., $d_{\tau,n} \sim U(1000, 2000)$. Additionally, the deadline for tasks $t_{\tau,n}^{DL}$ is randomly generated from the uniform distribution over the interval [0.2, 0.6], i.e., $t_{\tau,n}^{DL} \sim U(0.2, 0.6)$. Observing the chart, it can be seen that except for the Full_local algorithm, the delay and task failure rate of other algorithms significantly decrease with the enhancement of the satellite system's parallel computing capabilities. This is because as the number of servers involved in computation increases, the queuing waiting time for tasks within the satellite system is effectively reduced. The Full_local algorithm, which processes tasks only locally, is independent of the parallel computing capabilities of the satellite system, resulting in a horizontal line trend that remains unchanged in the chart. Similar to Figs. 8 and 9, thanks to the powerful feature extraction network and multi-agent collaborative optimization mechanism of the MA2C algorithm, the algorithm we proposed significantly outperforms other baseline algorithms in reducing delay and task failure rate, second only to the industry-leading Gurobi optimizer.

Finally, we explored the relationship between task deadlines and total delay as well as task failure rate under different algorithms. In this experimental setup, the number of IoT terminals N is fixed at 20. The satellite system's parallel computing capability M is fixed at 2. Additionally, the data size of tasks is randomly generated from the uniform distribution

Table 6 Task delay and failure rate under different number of IoT terminals

Number of terminals	Gurobi	MA2C	MA2C w/o TCN	MA2C w/o SA	MA2C w/o TCN&SA	BDQN	DRTO	GA	Full_sat	Full_local
10	1.4189/0%	1.4670/0.9%	1.4697/0.9%	1.4823/1.1%	1.5049/1.2%	1.5069/1.2%	1.5467/1.3%	1.5656/1.6%	1.8647/4.1%	3.1325/26.3%
20	3.5931/0%	3.7284/4.1%	3.7505/4.6%	3.7943/4.7%	3.9885/5.2%	3.8196/5.5%	3.8740/6.4%	3.9494/7.2%	5.1617/15.6%	6.2171/30.8%
30	6.4355/9.1%	6.6215/10.3%	6.7243/11.2%	6.8366/11.8%	6.9175/13.7%	6.9421/14.2%	7.3187/15.1%	7.5445/18.0%	12.8427/58.6%	9.6103/31.8%
40	10.0207/17.4%	10.2274/18.1%	10.2856/19.2%	10.4153/19.8%	10.7734/20.3%	10.7226/20.9%	10.9453/22.1%	11.1973/22.5%	22.1970/68.5%	13.3924/64.7%
50	14.2034/27.8%	14.5504/28.6%	14.6686/29.1%	14.9822/29.7%	15.1689/30.6%	15.3784/29.4%	15.9750/31.7%	16.5932/32.2%	34.0914/75.8%	20.0684/46.3%



(a) Total delay for different parallel computing capabilities. (b) Task failure rates for different parallel computing capabilities.

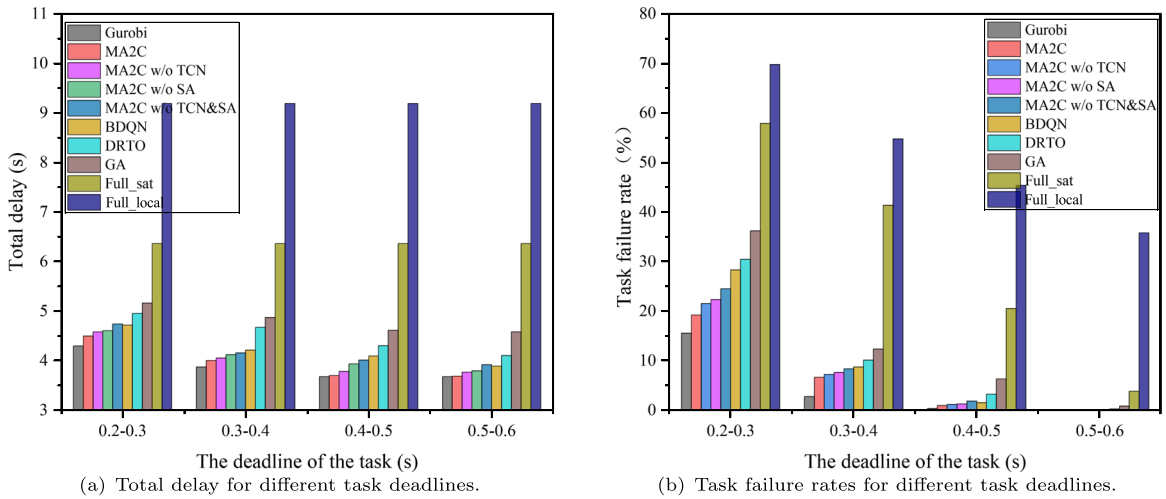
Fig. 10 The relationship between the parallel computing capability of the satellite system and the delay as well as failure rate

over the interval [1000, 2000], i.e., $d_{\tau,n} \sim U(1000, 2000)$. As shown in Fig. 11 and Table 8, intelligent scheduling algorithms exhibit significant performance differences under various task deadline constraints. As the deadline is relaxed from 0.2-0.3 seconds to 0.5-0.6 seconds, the task failure rates of all algorithms show a stepwise decline. For instance, the failure rate of the Full_sat strategy drops sharply from 57.9% to 3.9%. The MA2C algorithm demonstrates a significant advantage under strict time constraints (0.2-0.3 s), with its failure rate reduced by 9.1%, 11.2%, and 16.9% compared to BDQN, DRTO, and GA, respectively. This aligns with the previous analysis, where the MA2C model's ability to integrate spatiotemporal features accurately captures the nonlinear relationship between task urgency and resource occupancy. Concurrently, the collaborative action of multiple agents dynamically infers the competitive landscape of satellite resources to optimize task offloading decisions. Ablation experiments further reveal the complementary effects of algorithm components, once again validating the critical role of spatiotemporal dependency modeling in real-time decision-making.

In summary, the MA2C algorithm proposed in this study demonstrates significant advantages in satellite computing scenarios, with its core competitiveness stemming from multidimensional technological innovations in the algorithmic architecture. Firstly, by constructing a deep feature extraction network, the algorithm can accurately capture the multidimensional characteristics of tasks, effectively supporting decision-making modeling in dynamic environments. Secondly, the innovative multi-agent collaboration mechanism endows the algorithm with dual adaptability, allowing it to dynamically adjust optimization strategies based on both the dynamic changes of the satellite edge network scenarios and the urgency of tasks. This ensures optimal performance even when faced with fluctuations in the number of devices, computational loads, and communication conditions. More importantly, the algorithm embeds an autonomous evolution capability, continuously optimizing scheduling strategies through online reinforcement learning to achieve a dynamic balance among key indicators such as real-time performance, reliability, and computational efficiency. This intelligent evolution characteristic enables it to maintain superior scheduling efficacy even in satellite environments with stringent task deadlines and strong resource heterogeneity. It is the synergistic effect of these three technological advantages that allows the MA2C algorithm to consistently exhibit the lowest total task delay and optimal fault suppression capability in complex scenarios involving cross-scale tasks, heterogeneous computing units, and time-varying resource constraints, providing a groundbreak-

Table 7 Task delay and failure rate under different parallel computing capabilities of the satellite system

Parallel capabilities	Gurobi	MA2C	MA2C w/o TCN	MA2C w/o SA	MA2C w/o TCN&SA	BDQN	DRTO	GA	Full_sat	Full_local
1	3.7271/0%	3.8456/6.5%	3.8824/6.8%	3.9335/7%	4.1938/7.6%	4.0011/7.8%	4.2029/8.6%	4.3626/11.6%	5.8202/19.2%	6.2171/30.8%
2	3.5931/0%	3.7284/4.1%	3.7505/4.6%	3.7943/4.7%	3.9885/5.2%	3.8196/5.5%	3.8740/6.4%	3.9494/7.2%	5.1617/15.6%	6.2171/30.8%
3	3.1248/0%	3.2004/1%	3.2566/1.3%	3.2781/1.4%	3.4856/2.4%	3.3092/1.8%	3.3771/2.9%	3.4583/3.5%	4.4893/9.4%	6.2171/30.8%
4	2.9273/0%	3.0498/0.4%	3.0680/0.7%	3.1103/0.7%	3.3042/0.9%	3.1507/1.1%	3.2985/1.8%	3.3661/2.8%	3.6308/3.7%	6.2171/30.8%
5	2.7050/0%	2.8189/0%	2.8496/0%	2.8577/0%	2.9098/0%	2.8822/0.1%	2.9060/0.5%	2.9443/0.8%	3.1258/1.6%	6.2171/30.8%



(a) Total delay for different task deadlines. (b) Task failure rates for different task deadlines.

Fig. 11 The relationship between task deadlines and delay as well as failure rate

ing solution for building a highly reliable and highly resilient satellite-ground collaborative computing system. In contrast, algorithms such as BDQN, DRTO, and GA, due to the lack of a collaborative optimization strategy for task priority and task scheduling, rely solely on the autonomous regulation capabilities of reinforcement learning, and therefore are slightly inferior in terms of overall latency and task failure rate performance.

Although the MA2C algorithm demonstrates significant advantages in satellite edge computing scenarios, its technical architecture still faces certain challenges and limitations. For instance, while the integration of TCN and SA in the MA2C algorithm enhances feature extraction capabilities, the parallel computing requirements of multi-dimensional neural networks may lead to high consumption of computing resources. Under the constraints of limited computing power on satellite-borne equipment, this could potentially pose risks of delays in real-time task scheduling. In the future, we plan to adopt a collaborative optimization strategy involving knowledge distillation and dynamic parameter pruning to lighten the model.

5.4 Ablation experiment

1) *Network Model Evaluation:* To demonstrate the role of the feature extraction networks TCN and SA in our model, we conducted convergence analysis on models with and without TCN and SA. In this experimental setup, the number of IoT terminals N is fixed at 20. The satellite system's parallel computing capability M is fixed at 2. Additionally, the data size of tasks is randomly generated from the uniform distribution over the interval [1000, 2000], i.e., $d_{\tau,n} \sim U(1000, 2000)$, and the deadline for tasks is randomly generated from the uniform distribution over the interval [0.2, 0.6], i.e., $t_{\tau,n}^{DL} \sim U(0.2, 0.6)$.
As shown in Fig. 12, the MA2C model after adding TCN and SA has significantly improved in terms of stability and training accuracy. This improvement is mainly attributed to the following several key factors: First, the TCN captures the temporal dependencies in the data through its unique dilated causal convolution structure, which is crucial for understanding the evolution of task features over time and for the model's decision-making process. Moreover, the residual connections and batch normalization techniques in TCN effectively prevent issues such as vanishing or exploding gradients, thereby enhancing the training stability of the model. On the other hand, the SA mechanism captures global dependencies in the data by weighting elements at different positions in the sequence. Compared to TCN, SA offers a different perspective for understanding the dynamics of time series, which helps the model identify potential non-local correlations that are important for task scheduling decisions. Therefore, the introduction of SA not only enhances the model's

Table 8 Task delay and failure rate under different task deadlines

Task deadlines	Gurobi	MA2C	MA2C w/o TCN	MA2C w/o SA	MA2C w/o TCN&SA	BDQN	DRTO	GA	Full_sat	Full_local
0.2-0.3	4.3032/15.6%	4.5032/19.3%	4.5898/21.6%	4.6128/22.4%	4.7455/24.6%	4.7257/28.4%	4.9616/30.5%	5.1667/36.2%	6.3617/57.9%	9.1940/69.8%
0.3-0.4	3.8806/2.8%	4.0083/6.7%	4.0582/7.3%	4.1291/7.7%	4.1636/8.4%	4.2160/8.8%	4.6804/10.2%	4.8777/12.4%	6.3617/41.4%	9.1940/54.8%
0.4-0.5	3.6842/0.4%	3.7068/1%	3.7863/1.2%	3.9404/1.3%	4.0167/1.9%	4.0985/1.6%	4.3076/3.3%	4.6192/6.4%	6.3617/20.6%	9.1940/45.4%
0.5-0.6	3.6817/0%	3.6940/0%	3.7745/0%	3.8006/0%	3.9230/0%	3.8983/0%	4.1091/0.3%	4.5895/0.9%	6.3617/3.9%	9.1940/35.8%

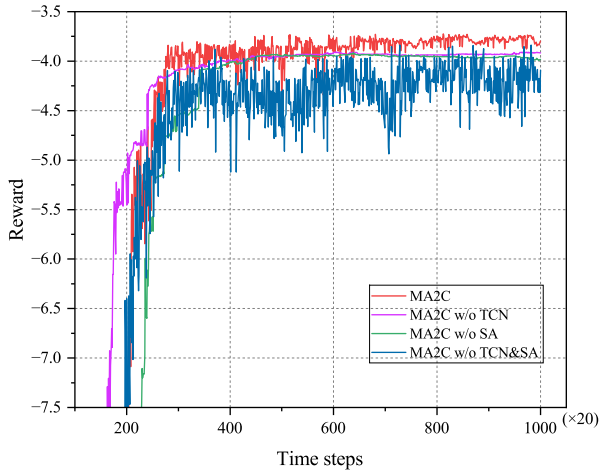


Fig. 12 Ablation experiments with or without TCN and SA

interpretive power of the data but also improves its generalization capability when dealing with complex time series.

However, it is noteworthy that the stability of the MA2C model slightly decreases when combining TCN and SA compared to using them individually. Experiments showed that the standard deviation of training reward fluctuations increases from 0.02 when TCN or SA is used alone to 0.09 when both are integrated, mainly due to feature competition between modules and dynamic parameter mismatches. TCN captures temporal-local dependencies through causal convolutions, while SA focuses on global contextual correlations. This creates complementary conflicts during feature extraction–TCN emphasizes local patterns whereas SA prioritizes global weight allocation, leading to inconsistent gradient update directions. Additionally, differing update rates between TCN's residual connections and SA's attention weights exacerbate parameter oscillations. Future improvements may include introducing an adaptive feature fusion mechanism, such as dynamic weighted gating, to balance the contributions of TCN and SA. Additionally, the optimizer could be further optimized, for instance, by using AdamW (Loshchilov and Hutter 2019) to decouple weight decay from gradient clipping. These approaches aim to mitigate module coordination challenges and enhance MA2C's robustness in complex satellite environments.

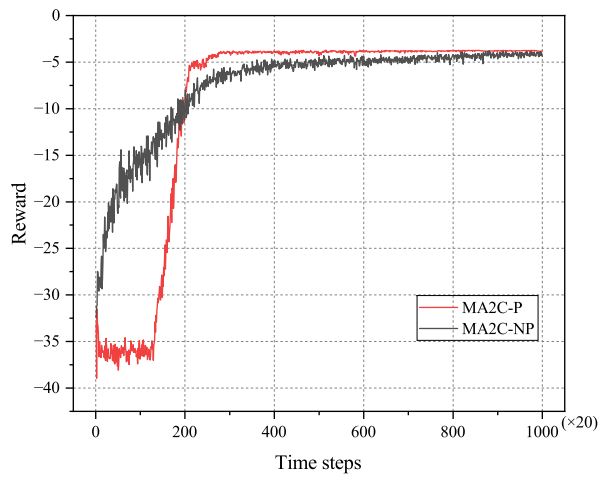


Fig. 13 Ablation experiments with or without task priority mechanism

2) *Priority Effectiveness Evaluation:* To validate the impact of the priority mechanism on task urgency and execution success, we conducted comparative experiments between the model with a priority mechanism (MA2C-P) and the baseline model without a priority mechanism (MA2C-NP). In this experimental setup, the number of IoT terminals N is fixed at 20. The satellite system's parallel computing capability M is fixed at 2. Additionally, the data size of tasks is randomly generated from the uniform distribution over the interval [1000, 2000], i.e., $d_{\tau,n} \sim U(1000, 2000)$, and the deadline for tasks is randomly generated from the uniform distribution over the interval [0.2, 0.6], i.e., $t_{\tau,n}^{DL} \sim U(0.2, 0.6)$. As shown in Fig. 13, the convergence speed of MA2C-P was 40% higher than that of MA2C-NP. The average reward value obtained by MA2C-P was 9.4% higher than that of MA2C-NP, and the fluctuation amplitude of its convergence process was also significantly smaller than that of MA2C-NP. As shown in Table 9, in the test experiment, the priority mechanism reduced the total system delay from 3.9363 s to 3.7284 s, a decrease of 5.3%; the overall task failure rate dropped sharply from 8.3% to 4.1%. Notably, emergency task success rate (deadline < 0.3 s) improved by 11.2 percentage points (82.6% $\rightarrow$ 93.8%). These data confirm that the priority sorting module, by establishing a dynamic mapping relationship between task characteristics and system status, realizes accurate identification of critical tasks. Combined with the task selection module, it achieves intelligent task scheduling, significantly improving the system's response capability to emergency tasks and overall scheduling efficiency.

Table 9 Ablation experiments of the priority module

Method	Total delay (s)	Task failure rate	Emergency task success rate
MA2C-P	3.7284	4.1%	93.8%
MA2C-NP	3.9363	8.3%	82.6%

6 Challenges and research directions

This study proposes a MA2C algorithm that significantly optimizes task scheduling efficiency in satellite edge computing scenarios by combining a multi-agent collaboration mechanism of task priority sorting and offloading decision-making. The algorithm adopts an encoder-decoder architecture and integrates TCN and SA mechanisms, enabling efficient extraction of dynamic environmental features and achieving dual optimization of total task latency and failure rate. Experimental results indicate that, compared to baseline algorithms, MA2C demonstrates stronger adaptability under various scenarios including task data scale, number of terminal devices, satellite parallel computing capabilities, and task deadlines, validating its effectiveness in multi-objective collaborative optimization. However, the current research still has the following limitations:

1) *Satellite mobility not considered:* The current model does not consider the impact of the rapid mobility of LEO satellites on communication links (e.g., frequent switching, Doppler effect), which could lead to communication quality fluctuations and task interruptions in highly dynamic scenarios. Future research could integrate real-time orbit prediction models and adaptive beamforming techniques, constructing time-varying channel state matrices and switching triggering feedforward mechanisms to anticipate topology changes, thereby reducing the impact of mobility on transmission stability. Simultaneously, Doppler compensation parameters and switching probability estimation could be embedded in the reinforcement learning state space to enhance the agent's perception of highly dynamic environments.
2) *Lack of real system constraints:* This study is currently validated in an idealized simulation environment and has not considered the engineering constraints faced by real satellite systems, such as actual satellite computing capacity limitations, dynamic bandwidth fluctuations, and protocol overhead. Future research could plan to promote the practical verification of the algorithm in phases: (a) Collaborate with satellite manufacturers to obtain real satellite computing resource parameters, measured channel datasets, and protocol overhead models to build a high-fidelity simulation platform. (b) Utilize the FLoRaSat open-source framework to establish a satellite-ground collaborative test environment, verifying the algorithm's robustness in extreme scenarios like channel mutations and resource contention through hardware-in-the-loop simulation, ultimately realizing on-orbit deployment verification of satellite edge computing nodes. This will systematically enhance the algorithm's engineering adaptability, providing a feasible technical solution for satellite-ground computing power collaboration.
3) *Insufficient energy efficiency and sustainability:* The current optimization objectives focus on task latency and failure rate without considering the impact of task scheduling on satellite energy consumption, which could affect satellite lifespan. Future research could introduce energy consumption index to construct a multi-objective optimization model and design a deep reinforcement learning algorithm based on the Pareto front. Additionally, energy-aware task migration strategies could be designed, integrated with satellite orbit illumination cycle predictions, prioritizing the scheduling of high-energy-consuming tasks to nodes with abundant sunlight.
4) *Insufficient flexibility of the task offloading model:* The current offloading decision model has certain limitations in terms of flexibility, primarily manifested in its binary decision-making mechanism that only supports either local execution or complete offloading, without yet supporting dynamic segmentation and collaborative processing of tasks across multiple nodes. Future research could construct a hybrid decision-making framework that integrates discrete actions for task prioritization with continuous actions for task offloading ratios. By introducing an adaptive collaborative node selection mechanism and a hybrid action space optimization algorithm (such as Hybrid SAC), it would be possible to jointly optimize task division granularity, collaborative node scheduling, and load balancing, thereby effectively addressing core challenges such as the adaptability of the hybrid decision framework. This approach would further enhance the flexibility of our offloading decision model and the refinement level of resource utilization in satellite edge computing environments.
5) *Limitations of dynamic environment modeling:* Experiments assume that task generation follows a uniform distribution and network topology is fixed, without considering burst task flows, topology dynamic changes caused by satellite mobility, or satellite node failures in extreme scenarios. Such dynamic factors could lead to resource allocation conflicts. Future designs should introduce topology dynamic evolution models to verify the algorithm's stability under complex disturbances.
6) *Lack of system fault tolerance verification:* This study has not systematically evaluated the task scheduling fault tolerance of satellite network topology anomalies (e.g., satellite node failures, inter-satellite link interruptions). In actual constellation operations, a single satellite failure could trigger cascading service degradation, and the current scheduling strategy may struggle to achieve rapid migration and resource reorganization of computing tasks. Future research could strengthen the system's fault tolerance from multiple dimensions: (a) Design a burst task flow generator based on the Poisson process to simulate traffic surges in scenarios like natural dis-

aster monitoring. (b) Construct a satellite node failure probability model and introduce a dedicated reinforcement learning architecture to enable autonomous learning of service degradation strategies. (c) Develop an inter-satellite link stability assessment module, incorporating topology robustness index into the reward function.

7 Conclusion

This paper primarily investigates the task scheduling problem in satellite edge computing (SatEC) environments, particularly the challenges posed by task deadlines and priority sequences. To address this challenge, we propose a multi-agent advantage actor-critic (MA2C) algorithm based on deep reinforcement learning. This algorithm employs two cooperating agents to achieve task priority ordering and offloading decisions, effectively reducing total task delay and task failure rates within the constraints of task deadlines. Simulation results demonstrate that the MA2C algorithm outperforms existing algorithms in various scenarios, especially in ensuring timely task completion, providing an efficient strategy for real-time data processing in SatEC environments.

In the future, we will expand our research to more complex SatEC scenarios, such as heterogeneous satellite constellations, multi-layer satellite network structures, and dynamically changing orbits. At the same time, we will also consider energy constraints and develop energy-saving inter-satellite task collaboration and computation offloading strategies to achieve sustainable SatEC. Moreover, we plan to conduct experiments using real satellite edge system data, which will help us more accurately evaluate algorithm performance and optimize algorithms for practical applications, ensuring they can adapt to the complex and variable satellite network environment and effectively support intelligent services.

Author Contributions Juan Chen contributed to methodology, supervision, writing-review, and editing. Jie Zhong contributed to methodology, conceptualization, simulation results, charts, and writing the original manuscript. Zongling Wu contributed to methodology, writing-review, and conceptualization. Di Tian contributed to editing and charts. Yujie Chen contributed to data curation and conceptualization.

Funding This research is partially supported by the Sichuan Province Science and Technology Program (2023JDRC0087) and the Yunnan Key Laboratory of Service Computing (YNSC24118).

Data Availability Data will be made available on reasonable request.

Declarations

Competing Interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Ahmed M, Raza S, Soofi AA et al (2024) A survey on reconfigurable intelligent surfaces assisted multi-access edge computing networks: State of the art and future challenges. Comput Sci Rev 54:100668

Bai S, Kolter JZ, Koltun V (2018) An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271

Bischoff T, Kasparick M, Tohidi E et al (2024) Real-time algorithms for combined embb and urllc scheduling. In: 2024 27th International workshop on smart antennas (WSA), pp 1–5

Chai Z, Hou H, Li Y (2023) A dynamic queuing model based distributed task offloading algorithm using deep reinforcement learning in mobile edge computing. Appl Intell 53(23):28832–28847

Chen Y, Zhao J, Wu Y et al (2024) Multi-user task offloading in uav-assisted leo satellite edge computing: A game-theoretic approach. IEEE Trans Mob Comput 24(1):363–378

Chen M, Xu J, Zhang W et al (2025) A new customer-oriented multi-task scheduling model for cloud manufacturing considering available periods of services using an improved hyper-heuristic algorithm. Expert Syst Appl 269:126419

Cui G, Duan P, Xu L et al (2022) Latency optimization for hybrid geo-leo satellite-assisted iot networks. IEEE Internet Things J 10(7):6286–6297

Gao Y, Yan Z, Zhao K et al (2023) Joint optimization of server and service selection in satellite-terrestrial integrated edge computing networks. IEEE Trans Veh Technol 73(2):2740–2754

He K, Zhang X, Ren S et al (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778

Hu Y, Gong W (2023) An on-orbit task-offloading strategy based on satellite edge computing. Sensors 23(9):4271

Jia M, Wu J, Zhang L et al (2023) Joint optimization communication and computing resource for leo satellites with edge computing. Chin J Electron 32(5):1011–1021

Jiao Z, Sha M, Zhang H et al (2024) City-leo: Toward transparent city management using llm with end-to-end optimization. arXiv:2406.10958

Konda V, Tsitsiklis J (1999) Actor-critic algorithms. Adv Neural Inf Process Syst 12

Kong X, Wu Y, Wang H et al (2022) Edge computing for internet of everything: A survey. IEEE Internet Things J 9(23):23472–23485

Leyva-Mayorga I, Martinez-Gost M, Moretti M et al (2023) Satellite edge computing for real-time and very-high resolution earth observation. IEEE Trans Commun 71(10):6180–6194

Li P, Xiao Z, Wang X et al (2023) Eptask: Deep reinforcement learning based energy-efficient and priority-aware task scheduling for

dynamic vehicular edge computing. IEEE Transactions on Intelligent Vehicles 9(1):1830–1846
Li D, Sun Y, Peng J et al (2024) Dual network computation offloading based on drl for satellite-terrestrial integrated networks. IEEE Trans Mob Comput 24(3):2270–2284
Li W, Li S, Shi H et al (2024) Uav-enabled fair offloading for mec networks: a drl approach based on actor-critic parallel architecture. Appl Intell 54(4):3529–3546
Liu Z, Dong X, Wang L et al (2022) Satellite network task deployment method based on sdn and icn. Sensors 22(14):5439
Liu J, Elsayed S, Essam D et al (2024) Large-scale project portfolio selection and scheduling problem: A comparison of exact solvers and metaheuristics. In: 2024 IEEE congress on evolutionary computation (CEC), pp 1–8
Loshchilov I, Hutter F (2019) Decoupled weight decay regularization. In: International conference on learning representations
Mao Y, You C, Zhang J et al (2017) A survey on mobile edge computing: The communication perspective. IEEE Commun Surv Tutor 19(4):2322–2358
Oroojlooy A, Hajinezhad D (2023) A review of cooperative multi-agent deep reinforcement learning. Appl Intell 53(11):13677–13722
Qiu L, Chen Q, Chen S et al (2024) Priority-aware parallel transmission towards dense satellite remote sensing and communication integrated networks. IEEE Trans Cognit Commun Netw 1–1
Shannon CE (1948) A mathematical theory of communication. Bell Syst Tech J 27(3):379–423
Sun Y, He Q (2023) Joint task offloading and resource allocation for multi-user and multi-server mec networks: A deep reinforcement learning approach with multi-branch architecture. Eng Appl Artif Intell 126:106790
Sun J, Wang H, Nie L et al (2023) A joint strategy for service deployment and task offloading in satellite-terrestrial iot. Comput Netw 225:109656
Sutton RS, McAllester D, Singh S et al (1999) Policy gradient methods for reinforcement learning with function approximation. Adv Neural Inf Process Syst 12:1057–1063
Tang Z, Jia W, Zhou X et al (2020) Representation and reinforcement learning for task scheduling in edge computing. IEEE Trans Big Data 8(3):795–808
Tang Q, Fei Z, Li B et al (2021) Computation offloading in leo satellite networks with hybrid cloud and edge computing. IEEE Int Things J 8(11):9164–9176
Tianhao L, Zhiyong L (2024) A self-attention based dynamic resource management for satellite-terrestrial networks. China Commun 21(4):136–150
Tsitsiklis J, Van Roy B (1996) Analysis of temporal-diffference learning with function approximation. Adv Neural Inf Process Syst 9
Vaswani A, Shazeer N, Parmar N et al (2017) Attention is all you need. Adv Neural Inf Process Syst pp 5998–6008
Vinyals O, Fortunato M, Jaitly N (2015) Pointer networks. Adv Neural Inf Process Syst 28
Waqar N, Hassan SA, Mahmood A et al (2022) Computation offloading and resource allocation in mec-enabled integrated aerial-terrestrial vehicular networks: A reinforcement learning approach. IEEE Trans Intell Transp Syst 23(11):21478–21491
Xi S, Shang B, Zhang H et al (2024) Energy optimization in multi-satellite-enabled edge computing systems. IEEE Int Things J 11(12):21715–21726
Xie R, Tang Q, Wang Q et al (2020) Satellite-terrestrial integrated edge computing networks: Architecture, challenges, and open issues. IEEE Netw 34(3):224–231
Xiong Z, Zhao M, Tan L et al (2022) Real-time power optimization for application server clusters based on mixed-integer programming. Futur Gen Comput Syst 137:260–273
Xiong J, Guo P, Wang Y et al (2023) Multi-agent deep reinforcement learning for task offloading in group distributed manufacturing systems. Eng Appl Artif Intell 118:105710
Xu X, Xia Y, Peng Q et al (2024) A novel structured task scheduling approach in satellite edge computing environments. In: 2024 IEEE international conference on web services (ICWS), IEEE, pp 718–727
Yang W, Liu Z, Liu X et al (2024) Deep reinforcement learning-based low-latency task offloading for mobile-edge computing networks. Appl Soft Comput 166:112164
Yang J, Zhang Y, Xiao Z et al (2024a) Joint access selection and computation offloading in leo ubiquitous edge computing networks: An alternating drl-based approach. IEEE transactions on cognitive communications and networking, pp 1–1
Ye H, Li GY, Juang BHF (2019) Deep reinforcement learning based resource allocation for v2v communications. IEEE Trans Veh Technol 68(4):3163–3173
Zhang X, Wang Y (2023) Deepmecagent: multi-agent computing resource allocation for uav-assisted mobile edge computing in distributed iot system. Appl Intell 53(1):1180–1191
Zhang H, Liu R, Kaushik A et al (2023) Satellite edge computing with collaborative computation offloading: An intelligent deep deterministic policy gradient approach. IEEE Internet Things J 10(10):9092–9107
Zhang S, Liu A, Han C et al (2023) Multiagent reinforcement learning-based orbital edge offloading in sagin supporting internet of remote things. IEEE Internet Things J 10(23):20472–20483
Zhang H, Zhao H, Liu R et al (2024) Collaborative task offloading optimization for satellite mobile edge computing using multi-agent deep reinforcement learning. IEEE Trans Veh Technol 73(10):15483–15498
Zhang S, Cai T, Wu D et al (2024) Iort data collection with leo satellite-assisted and cache-enabled uav: A deep reinforcement learning approach. IEEE Trans Veh Technol 73(4):5872–5884
Zhao M, Chen C, Liu L et al (2022) Orbital collaborative learning in 6g space-air-ground integrated networks. Neurocomputing 497:94–109
Zhong L, Li Y, Ge MF et al (2025) Joint task offloading and resource allocation for leo satellite-based mobile edge computing systems with heterogeneous task demands. IEEE transactions on vehicular technology, pp 1–15
Zhou J, Zhao Y, Zhao L et al (2024) Adaptive task offloading with spatiotemporal load awareness in satellite edge computing. IEEE Trans Netw Sci Eng 11(6):5311–5322
Zhu X, Jiang C (2022) Delay optimization for cooperative multi-tier computing in integrated satellite-terrestrial networks. IEEE J Select Areas Commun 41(2):366–380