Computer Networks 252 (2024) 110665

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Multi-agent DRL for edge computing: A real-time proportional compute offloading[☆]

Kunkun Jia [a], Hui Xia [a,*], Rui Zhang [a], Yue Sun [a], Kai Wang [b]

[a] College of Computer Science and Technology, Ocean University of China, Qingdao 266100, China
[b] School of Computer Science and Technology, Harbin Institute of Technology, Weihai 264200, China

ARTICLE INFO

Keywords:
Computation offloading
Edge computing
Deep reinforcement learning
Orthogonal frequency division multiple-access

ABSTRACT

In the Industrial Internet of Things, devices with limited computing power and energy storage often rely on offloading tasks to edge servers for processing. However, existing methods are plagued by the high cost of device communication and unstable training processes. Consequently, Deep reinforcement learning (DRL) has emerged as a promising solution to tackle the computation offloading problem. In this paper, we propose a framework called multi-agent twin delayed shared deep deterministic policy gradient algorithm (MASTD3) based on DRL. Firstly, we formulate the task offloading conundrum as a long-term optimization problem, which aids in mitigating the challenge of deciding between local or remote task execution by a device, leading to more effective task offloading management. Secondly, we enhance MASTD3 by introducing a priority experience replay buffer mechanism and a model sample replay buffer mechanism, thus improving sample utilization and overcoming the cold-start problem associated with long-term optimization. Moreover, we refine the actor-critic structure, enabling all agents to share the same critic network. This modification accelerates convergence speed during the training process and reduces computational costs during runtime. Finally, experimental results demonstrate that MASTD3 effectively addresses the proportional offloading problem, which is optimized by 44.32%, 29.26%, and 17.47% compared to DDPQN, MADDPG, and FLoadNet.

1. Introduction

The Industrial Internet of Things (IIoT) stands as a transformative technology, ushering in substantial changes within the realms of production and manufacturing [1–3]. Offering flexibility and enhanced agents, IIoT empowers devices with unprecedented availability [4]. However, as its application proliferates, several challenges have begun to surface. Many IIoT terminal devices grapple with inadequate computing prowess and limited energy reserves, presenting hurdles in meeting the pressing demands of industrial production, particularly in real-time processing and handling large-scale data [5,6]. Edge computing serves as a remedy, circumventing the constraints of conventional cloud computing by decentralizing computational capabilities to the network's periphery, thereby curtailing data transmission latency [7] (see Fig. 1).

Hence, we propose the integration of edge computing to mitigate potential data latency and bandwidth constraints encountered by the IIoT, thereby enhancing the efficient utilization of computational resources.

While edge-based computing offloading methods hold the potential to augment user computing capabilities notably, it is imperative to acknowledge the finite nature of hardware resources within edge servers [8]. The simultaneous offloading of tasks by numerous users may surpass the connection bandwidth of the edge server, consequently prolonging task response times. Hence, the pivotal question emerges: How can we devise an efficacious offloading strategy to discern which task segments should be directed towards the edge server?

In the realm of computational offloading within mobile edge computing, selecting an appropriate offloading strategy is a persistent and pivotal challenge [9]. Currently, existing offloading strategies predominantly fall into two categories: those rooted in traditional methodologies and those based on deep reinforcement learning (DRL).

Traditional computational offloading methods are usually rule-based or empirical and use static modeling methods such as queuing models or non-cooperative games [10,11], which focus on a single element only and lack the ability to consider multiple factors comprehensively. They perform poorly when facing real-time, fast-changing

☆ This research is supported by the National Natural Science Foundation of China (NSFC) under grant number 62172377, the Taishan Scholars Program of Shandong province under grant number tsqn202312102, and the Startup Research Foundation for Distinguished Scholars under grant number 202112016.

* Corresponding author.
E-mail address: xiahui@ouc.edu.cn (H. Xia).

https://doi.org/10.1016/j.comnet.2024.110665
Received 15 April 2024; Received in revised form 14 June 2024; Accepted 17 July 2024
Available online 25 July 2024

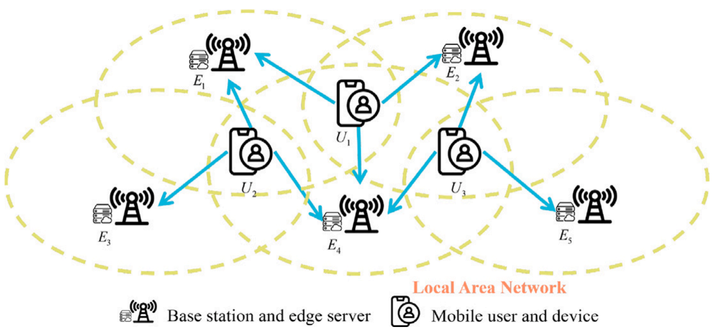


Fig. 1. Explanation of computational offloading. Within an ellipse, users have the ability to reach the respective base station, thereby transferring a segment of the ongoing application to one of them.

computational offloading decision problems [12]. Therefore, we focus on DRL-based computational offloading and categorize it into single-agent DRL (SADRL) and multi-agent DRL (MADRL). DRL [13,14] is characterized by self-learning, adaptability, and global search capability, which effectively solves the shortcomings of heuristic algorithms. Computational offloading based on DRL can be further categorized into SADRL computational offloading and MADRL computational offloading. Compared to SADRL, the learning environment provided by MADRL is inherently unstable.

However, SADRL is difficult to scale in a huge state and action space [15]. In MADRL, there may be cooperation or competition between agents [16], and SADRL is not suitable for such interactions. Therefore, there is a complex interaction process between multiple agents. When all the agents are adapting their strategies, the stability of the environment is affected, which makes the convergence of the training less clear. Therefore, it is important to design a mechanism that can adaptively update its own strategy to ensure coordination and cooperation among multiple agents. In addition, in the environment of a multi-agent system, due to the lack of comprehensive information sharing and centralized control mechanisms, each agent can only make decisions based on its own local data without knowing anything about the behavioral patterns and current states of the other agents, and thus the development of an efficient learning framework becomes particularly important.

This paper delves into the execution of combined task division and power management for multi-agent partial offloading within an edge computing network, which includes various IIoT devices and edge servers [17]. As Fig. 1 shows, each IIoT device partitions computing tasks into multiple sub-tasks based on the available communication and computing resources of edge servers, the computing resources of IIoT devices, and the task requirements, and then determines the transmission power for transmitting sub-tasks to edge servers. Next, the IIoT device and the edge server work together to execute subtasks aimed at reducing the latency and energy consumption of the execution while strictly following the latency and energy consumption limits of the tasks. We first model this problem as a long-term optimization problem for multiple agents [18]. Then, we propose a multi-agent twin delayed shared deep deterministic policy gradient algorithm (MASTD3), discussing offloading decisions for each IIoT device, including task partitioning strategies and transmission power strategies, to optimize task offloading policies. MASTD3 not only considers offloading decisions but also considers the proportion of task offloading. To improve algorithm efficiency, we introduce prioritized experience replay [19] and model sample buffer mechanisms to enhance training efficiency and stability, increase sample utilization, and overcome cold start issues, improving system efficiency and availability. We then improve the actor-critic structure of MASTD3, with all agents sharing the same critic network, accelerating convergence during training and reducing computational costs during operation.

In summary, this paper contributes the following:

1. In order to minimize mobile device runtime and energy use, we need to clarify whether offloading should be done and which computing resources should be allocated to the offloading task of the Mobile edge servers (MES). Given the computational power and communication bandwidth of the device, the issue of computational offloading is structured as a problem of long-term optimization.
2. Based on the real-time requirements and large-scale data processing needs of IIoT, we construct a smart lightweight task offloading framework that fully considers issues such as the selfishness of IIoT terminal devices, limited computing bandwidth, and low energy storage, which is suitable for IIoT deployment and applications.
3. We propose MASTD3 and according to the randomness of input state information, we correspondingly construct a network structure to address how to determine offloading decisions and proportions. MASTD3 considers the long-term optimization of task offloading to balance real-time performance and energy efficiency.
4. Numerous experimental findings have shown the algorithm's effective convergence. Simulation results show that compared to other traditional strategies such as no offloading strategy, random offloading strategy, or full offloading strategy, the algorithm has fast convergence and stable performance, providing higher performance and efficiency for IIoT applications.

The subsequent chapters of this paper are structured in this manner: Section 2 presents pertinent research in the field of computational offloading and DRL; Section 3 describes the problem studied in this paper and establishes the corresponding mathematical model; Section 4 presents the MASTD3; Section 5 validates the performance of our algorithm under various scenarios; finally, Section 6 wraps up and explores prospective paths.

2. Related work

In this section, we reviewed the current state of research in DRL-based computational offloading schemes.

2.1. Single-agent deep reinforcement learning computational offloading scenarios

Huang et al. [20] implemented a binary-based task offloading decision, utilizing a DRL-based online offloading framework (DROO) to optimize task offloading and resource allocation in mobile edge computing networks. Moreover, [21,22] focused on single-agent wirelessly powered mobile edge computing (MEC) systems. Zhang et al. [21] proposed the task offloading decision. To maximize the total computation rate by mastering the near-optimal Wireless power transfer duration

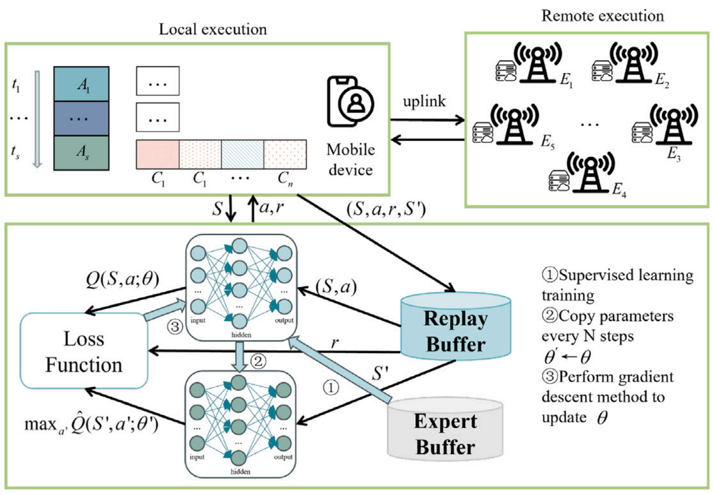


Fig. 2. Synopsis of the System. During every time interval, a MD breaks down an application into several separate parts. It then employs MASTD3 to determine the target server and allocation proportions, continuously updating network parameters using batch gradient descent.

through DRL, addressing the challenge of energy shortage and computational power limitations of IIoT nodes. Similarly, Wang et al. [22] proposed a joint energy and task allocation design, taking into account energy and task causality constraints in wireless-operated mobile edge computing systems. This design aimed to reduce the overall transmission energy consumption while ensuring the successful execution of tasks by users. Under the MEC system, this approach enabled the mobile device (MD) to select the device, power of transmission, and offload rate under unknown circumstances.

2.2. *Multi-agent deep reinforcement learning computational offloading scenarios*

The combined challenge of dividing tasks and distributing resources in multi-user, multi-server systems has been acknowledged for its complexity [23], especially in light of the diverse nature and behavior of network resources. Xiao et al. [24] proposed a mobile offloading scheme based on DRL, enabling the MD to autonomously select edge servers, determine offloading rates, and adjust transmission power to enhance its utility while mitigating interference and jamming. Gao et al. [25] introduced a decentralized approach using the attention-weighted recursive Multi-agent Actor Family Critic (ARMAAC) in a vast, hybrid, cooperative, and competitive environment for mobile edge computing. The method showcased enhanced proficiency in forecasting the future conditions of edge servers in terms of dynamic resource distribution. Heydari et al. [26] tackled a scenario of non-cooperative offloading within a system that includes various MDs and edge devices. The goal was to create an ideal offloading strategy aimed at reducing the rates of task loss and delay in execution, independent of pre-existing understanding of task arrival models and channel features. Additionally, [27,28] presented an innovative algorithm based on imitation learning for task offloading. Under this method, every MD had the ability to replicate an expert's strategy for delegating tasks, even in the absence of comprehensive information. Baek et al. [29] formalized the problem as a gaming problem and proposed an actor-critic reinforcement learning framework called FLoadNet, but the complexity and communication overhead may increase significantly as the network size increases.

To address these challenges, we propose MASTD3, designed to efficiently tackle the computational offloading problem posed by multi-agent bodies. Our approach integrates a prioritized replay buffer and a multi-agent network. Through these innovative designs, we anticipate enhancing the performance and stability of the algorithm.

Table 1
Notations used in our formulation.

Notation	Description
h_{ul}	Channel fading coefficients for uplinks
N_0, B	Noise power and total bandwidth
g_{ul}	Target BER for uplink
β_l	Path loss index
$\mathcal{B}, \mathcal{W}$	Total input data bytes and total load
ω	Number of clock cycles per byte cycle/byte (cpb) for microprocessor execution
x_{ij}, y_{ij}	Target server index and target ratio
$p_{y_{ij}}$	Percentage of applications S_{ij} that have been uninstalled to the edge server $E_{x_{ij}}$
f_{ij}^{loc}	local computing power
e_l	Energy consumption per byte
t_q^{loc}	local queuing delay
$f_{x_{ij}}^{mec}$	$E_{x_{ij}}$ computing power
e_r	Server energy consumption per byte
t_q^{mec}	Remote queuing delay
λ, β	Balancing the weight of latency and energy consumption
γ	Diminishing reward
K	Episode
D	replay buffer size
m	Mini-batch size
f	Update network weight
ε	Greed index

3. Preliminary

This part presents the problem model being examined, encompassing a network model, an application model, a local execution model, and a remote execution model. Following this, we outline the issue of long-term optimization being analyzed.

3.1. *Network model*

Within the existing Mobile Edge Computing framework, the edge server takes charge of managing resources and virtualizing via virtual machines, whereas the network's implementation depends on orthogonal frequency division multiple access. It is assumed that the overall bandwidth B is divided among $\mathcal{N}$ subcarriers, with the present count of subcarriers indicated as $k \in \mathcal{N} = \{1, 2, \dots, N\}$. Echoing other research, numerous applications demonstrate minimal response times to computational outcomes that are significantly less than the size of the input data. As a result, our focus is exclusively on the network's uplink transmission duration, ignoring the downlink transmission time. Representing p_u as the MD's transmission capability, it is presumed that

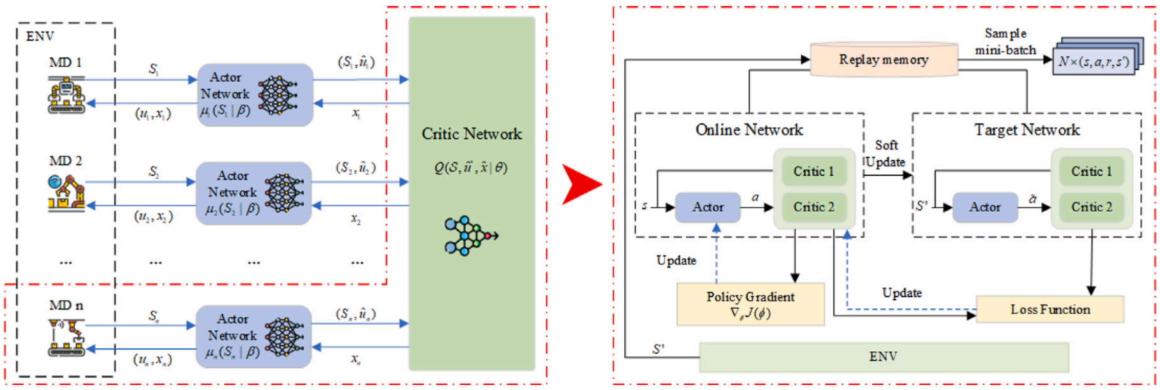


Fig. 3. The framework of the MASTD3 approach.

each user has a zero mean and variance for the extra Gaussian white noise σ^2, maintaining uniform variances. Consequently, it is possible to ascertain the maximum achievable rate (in bits per second) for the uplink across an additive white Gaussian noise channel as

$$r_{ul} = k\frac{B}{N}\log\left(1+\frac{p_u|h_{ul}|^2}{\Gamma(g_{ul})\, d^{\beta_l} N_0}\right), \tag{1}$$

where d represents the gap between the server and MD, N_0 denotes the power of noise, β_l is the exponent for path loss, h_{ul} is the coefficient for channel attenuation in the uplink, and g_{ul} indicates the target—the uplink's Bit Error Rate (BER).

3.2. Application model

At every time interval t, the MD produces an application $\mathcal{G}$ that demands significant computing power. The set $\mathcal{G}$ can be divisible into various tasks, denoted as $\{c_1, c_2, \dots, c_n\}$, which are independent of each other (i.e., fine-grained partitioning). Every component has the capability to run independently on-site or be transferred to an edge server. Take, for example, a real-time monitoring system that concurrently examines data from multiple sensor devices, where the information produced by each sensor operates independently. Consequently, the surveillance system is divisible into various autonomous functions, each tasked with the analysis of data from a distinct sensor unit. By assigning each task to handle real-time data from the same sensor device, the system's resources are utilized more efficiently, thereby enhancing the efficiency of real-time monitoring. This task decomposition approach optimizes the processing and analysis of real-time data in IIoT systems.

The application's workload is measured by the input data's size, labeled as $\mathcal{W}$, and the aggregate byte count of the input data as $\mathcal{B}$. With a specified value of $\mathcal{B}$, it is possible to compute $\mathcal{W} = \omega\mathcal{B}$, with ω symbolizing the CPU *cycles/byte(cpb)*, denoting the computation's clock cycle count by the microprocessor. The variable ω varies based on the application's characteristics, shaped by elements like the intricacy of time and space. As a result, the byte ratio for each task, indicated as $\{p_1, \dots, p_n\}$, $\sum_{i=1}^{n} p_i = 1$, can be ascertained by the byte count per task.

3.3. Local and remote execution model

Maintaining generality, our focus remains on latency and energy usage as the key metrics. The application $\mathcal{G}$ is segmented into multiple distinct tasks $\{c_1, c_2, \dots, c_n\}$, for calculating $\{p_1, \dots, p_n\}$, $\sum_{i=1}^{n} p_i = 1$. Subsequently, we split the tasks into two segments to ascertain their respective proportions. The workload carried out on the server or local device is represented as $\{(A_1, B_1), (A_2, B_2), \dots, (A_N, B_N)\}$, where regarding $A_i + B_i = 1$. Suppose the set of users is $\{U_1, U_2, \dots, U_Z\}$, the configuration of edge servers is $\mathcal{M} = \{E_1, E_2, \dots, E_M\}$, and time $\mathcal{T}$ is partitioned into $\{t_1, t_2, \dots, t_s\}$. The set U_i is required to handle a unique application S_{ij} during each interval t_j. If the application S_{ij} is not created, set $\mathcal{B}_{ij} = 0$. Define x_{ij} as the target server's index, and y_{ij} as the index for the target ratio. Subsequently, $P_{y_{ij}}$ denotes the proportion of applications S_{ij} transferred to the edge server $E_{x_{ij}}$, where $x_{ij} = \{1, 2, \dots, M\}$, and $y_{ij} = \{1, 2, \dots, N\}$. This research reveals that the execution of the application involves two concurrent elements: execution on-site and execution remotely. For every application, our assessment focuses on two key metrics: the time delay and the energy consumption.

3.3.1. Local execution

The model for local execution is quite direct, concentrating exclusively on the delay in local computation and energy usage. Consequently, for a user denoted as U_i in the time slot t_j, the time slot T_{ij}^{loc} and the energy usage E_{ij}^{loc} for the execution at a local level are

$$T_{ij}^{loc} = D_{ij}^{loc} + t_q^{loc} = \frac{w\mathcal{B}_{ij}Q_{y_{ij}}}{f_{ij}^{loc}} + t_q^{loc}, \tag{2}$$

and

$$E_{ij}^{loc} = e_l \mathcal{B}_{ij} Q_{y_{ij}}, \tag{3}$$

where f_{ij}^{loc} represents the local computational capacity, e_l denotes the energy usage per byte, t_q^{loc} indicates the delay in local queuing, and $\mathcal{B}_{ij}$ denotes the input size.

3.3.2. Remote execution

Edge servers employ a far more intricate remote execution approach compared to MDs. This complexity arises from the necessity to consider data transfer latency in the network, particularly for applications with large data sizes. In practice, computation latency, computation energy consumption, and queuing delays are generally much smaller for edge servers than for MDs. Consequently, we can compute the computation delay T_{ij}^{mec} and energy consumption E_{ij}^{mec} as follows

$$T_{ij}^{mec} = D_{ij}^{mec} + t_q^{mec} = \frac{\omega\mathcal{B}_{ij}P_{y_{ij}}}{f_{ij}^{mec}} + t_q^{mec} + \frac{\mathcal{B}_{ij}P_{y_{ij}}}{r_{ul}^{(ij)}}, \tag{4}$$

and

$$E_{ij}^{mec} = e_r \mathcal{B}_{ij} P_{y_{ij}}, \tag{5}$$

where $f_{x_{ij}}^{mec}$ represents the edge server's computational capacity $E_{x_{ij}}^{mec}$, e_r denotes the server's energy usage per byte, and t_q^{mec} signifies the delay in remote queuing.

Summing up we can get the total energy consumption as

$$E_{ij}^{total} = E_{ij}^{mec} + E_{ij}^{loc}. \tag{6}$$

By combining the assessments of the previously mentioned models, we determine the combined weighted sum of delay and energy consumption as follows

$$W_{ij} = \lambda \max\left(T_{ij}^{mec}, T_{ij}^{loc}\right) + \beta E_{ij}^{total}, \tag{7}$$

where λ and β represent the balancing factors for delay and energy usage, with $\lambda, \beta \in (0, 1)$. In practice, users typically prioritize lower latency, so we prefer to assign a larger value to λ than to β.

3.4. Problem formulation

After simulating the operation of an individual application, our attention now turns to the deployment of multiple applications, as depicted in Fig. 2. The duration $\mathcal{T}$ is segmented into intervals $\{t_1, t_2, \dots, t_s\}$. In every time interval t_j, each user U_i on the MD is required to manage a distinct application S_{ij}. We aim to ascertain the ideal values for x_{ij} and y_{ij}. Consequently, issue $\mathcal{P}_1$ is as follows

$$\begin{aligned} \mathcal{P}_1 : \quad & \min \textstyle\sum_{t_j \in \mathcal{T}} \left[\lambda \max \left(T_{ij}^{mec}, T_{ij}^{loc} \right) + \beta E_{ij}^{total} \right] \\ \text{s.t.} \quad & f_{x_{ij}}^{mec} \le \mathcal{F}_{x_{ij}}^{mec}, f_{x_{ij}}^{loc} \le \mathcal{F}_{i}^{loc} \\ & r_{ul}^{(ij)} \le \mathcal{R}_{ul}^{(i)} \\ & x_{ij} \in \{1, 2, \dots, m\}, y_{ij} \in \{1, 2, \dots, n\}, \end{aligned} \tag{8}$$

where $\mathcal{F}_{x_{ij}}^{mec}$ and $\mathcal{F}_{i}^{loc}$ represent the maximum available computing power of $E_{x_{ij}}$ and U_i, respectively, and $\mathcal{R}_{ul}^{(i)}$ is the maximum transport rate of the uplink.

We realize the balance between real-time performance and energy efficiency by adjusting the parameters to meet the needs of different application scenarios. In IIoT, it is necessary to respond quickly to user requests or process real-time data, and the requirement for real-time performance is more critical. Therefore, by setting λ to a higher value, we can pay more attention to real-time performance to ensure that the system can respond to user demands in a timely manner.

Clearly, problem $\mathcal{P}_1$ is an integer programming problem aimed at determining the optimal values of x_{ij} and y_{ij} to obtain the target server $E_{x_{ij}}$ and the target ratio $P_{y_{ij}}$. Nonetheless, this issue is classified as NP-hard, representing an expansion of the Knapsack problem. Traditional optimization methods are inadequate for solving such NP-hard problems. Consequently, our suggestion is to employ a reinforcement learning method to directly derive the ideal $E_{x_{ij}}$ and $P_{y_{ij}}$.

4. Algorithm design

This part extensively explores the computational offloading technique that relies on DRL. Firstly, we define the multi-agent Markov decision process (MAMDP) as a quaternion. Subsequently, we present our improved MASTD3.

4.1. Preliminaries

In this case, the computational offloading issue is represented as a MAMDP through the training of each device's offloading policy. This document characterizes the MAMDP issue as a quartet $(\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{R})$, where in $\mathcal{N}, \mathcal{S}, \mathcal{A}$ and $\mathcal{R}$ symbolize the agent space, agents' state space, agents' action space, and their reward function, in that sequence.

Agents Space $\mathcal{N}$: $\mathcal{N} = \{1, 2, \dots, N\}$, where N is the number of agents, each of which is placed on an IIoT device.

State Space $\mathcal{S}$: DRL aims to persistently acquire tactics from past data to achieve an ideal viewpoint. Therefore, a comprehensive state definition is crucial for decision-making efficiency. We take into account the conditions of the application, the computing power of the users, the resource status of the edge servers, and the network condition and establish the condition at time slot t as

$$s_t = (\mathcal{B}, r_{ul}^1, \dots, r_{ul}^m, C_l, C_1, \dots, C_m)_t, \tag{9}$$

where $\mathcal{B}$ signifies the input data's magnitude, C_l denotes the local device's computing power, C_i signifies the computing capacity of edge server i, while r_{ul}^m denotes the uplink speed of server m.

Action Space $\mathcal{A}$: Within our offloading context, the MD is regarded as an RL agent tasked with deciding on the target server E_{tar} and the target ratio P_b when the state s is received. The decision to offload is incorporated with calculating the ratio and characterizing the action as a vector

$$a_t = (E_{tar}, P_b)_t. \tag{10}$$

Reward function $\mathcal{R}$: During every time interval t, the agent is rewarded with $R(s_t, a_t)$ in a particular state s_t following the action a_t. Ideally, there should be a positive correlation between the reward function and the objective function. Section 3 aims to reduce the combined total latency and energy usage across all MDs, in contrast to reinforcement learning, which seeks to optimize long-term benefits. Consequently, there exists a negative correlation between the reward function to the aggregate of latency and energy usage, that is, $r_j = -W_{ij}$. The reward is designated as $\frac{W_l - W(s,a)}{W_l}$, with a negative reward when $W(s,a) > W_l$, motivating RL agents to steer clear of adverse outcomes behaviors.

Be aware that the MD is uninformed about the immediate bandwidth available (i.e., r_{ul}) and the kernel of the server (i.e., C_i). In response, we suggest implementing the server broadcasting system. Every mobile gadget upholds its own information. Displayed on a table are the records of computing resources accessible locally, the uplink transport rate, and the computing resources of each server available to the user. During every time interval t, every edge server dispatches a heartbeat packet to the MDs in its area, encompassing details about accessible computing resources. Subsequently, every user revises their table in response to the disseminated data. Training of the model occurs on the server, with regular updates to the model. By regularly downloading the most recent model from the server, users are empowered to make informed decisions using the latest data at hand.

4.2. Multiagent Twin Delayed Shared Deep Deterministic policy gradient algorithm (MASTD3)

The MASTD3 employs an actor-critic framework comprising an actor network (μ), a critic network (Q), and their respective target networks. It utilizes a gradient algorithm to update the network parameters. Significantly, its design incorporates the use of both two critic networks and two target critic networks.

In edge cloud computing systems, each agent changes its policy, which makes the network environment change over time, and the dynamic network makes the performance of traditional MADRL unstable. To ensure convergence, as shown in Fig. 3, we use a collaborative framework based on MASTD3. Meanwhile, we use the standard paradigm of centralized training and decentralized execution in MADRL. That is, each MD plays an actor, while all MDs share a common criticism for evaluating the value of a given state–action pair. By doing so, MDs share continuous decisions, improve communication efficiency, and achieve overall performance improvement. It uses two independent critic networks to reduce estimation errors, introduces delays when updating the target network, and stabilizes the training process. The detailed process of the MASTD3 is summarized in Algorithm 1.

Generally, MASTD3 is composed of two primary stages: gathering experiences and conducting training. During the gathering phase, a new action a_t arises by integrating random Gaussian noise into the output of the actor network within the defined s_t.

$$a_t = \mu(s_t, \theta^\mu) + \sigma^2, \tag{11}$$

where the parameter θ^μ represents the actor's home network, and σ^2 denotes the additional Gaussian noise. This randomized element facilitates wider exploration of the action space and thus promotes strategic learning.

Subsequently, the environment rewards r_t, leading to the realization of the subsequent state s_{t+1}, influenced by current state and action (s_t, a_t). Aiming to enhance the decision-making capabilities via training aided by previous experiences, the tuple (s_t, a_t, r_t, s_{t+1}) is archived as past experiences in the prioritized replay buffer. Throughout the training phase, a specific quantity of tuples is randomly selected from the experience replay buffer for the purpose of training.

Typically, the sample size of model sample data is restricted, thus impacting only a minor portion of the state–action values. Additionally, insufficient coverage of as many states as feasible could adversely affect the model. In response to these matters, the defined objective function for supervised learning is defined in the following manner

$$J_E(\theta^\mu) = \max_{a\in\mathcal{A}} \left[Q(s,a) + l\left(a_E, a\right)\right] - Q\left(s, a_E\right), \tag{12}$$

where a_E represents the model sample's action, and $l(x,y)$ serves as an indicator function. When $a = a_E$, it follows that $l\left(a, a_E\right) = 0$ and $J_E(Q) = 0$, signifying the model's choice aligns with that of the model sample decision. When $a \neq a_E$, it suggests that the worth of a different action is comparably good to that of the model sample's action. Eq. (12) minimizes the difference between the predicted and actual values of the critic network by adjusting the parameters of the actor network. This optimization ensures that the actions chosen by the actor network are consistent with those that maximize the value function, thus improving the overall decision-making process. The objective function introduces supervised learning into the reinforcement learning framework, ensuring that the actor network learns from a diverse set of actions, not just the actions initially considered optimal.

In the actor's primary network training progress, a group of tuples is retrieved from the priority replay buffer. The actor network generates a novel action $a'_n = \mu(s_n, \theta^\mu)$ derived from the state s_n. Concurrently, should the policy a'_n diverge from the current policy a_n in the experience replay buffer, upon inputting s_n and a'_n into the critic network (such as Q_1), the network generates $q_n = Q_1(s_n, \mu(s_n, \theta^\mu), \theta^{Q_1})$, with θ^{Q_1} representing a critical network parameter. Upon achieving all q_n, the anticipated mathematical outcome is

$$J_{TD3}(\theta^\mu) = \mathbb{E}\left[Q_1\left(\mathcal{S}, \mu(\mathcal{S}, \theta^\mu), \theta^{Q_1}\right)\right], \tag{13}$$

where $\mathcal{S} = \{s_n | n \in \mathcal{N}\}$. The goal of this objective function is to maximize the expectation of Q under the current policy, to find a set of parameters θ^μ that maximize the Q value corresponding to the output of the action by the actor network in a given state. Then, the gradient of strategy for the function J in relation to θ^μ can be described as

$$\nabla_{\theta^\mu} J_{TD3} = \mathbb{E}\left[\nabla_{\mathcal{A}} Q_1\left(\mathcal{S}, \mathcal{A}, \theta^{Q_1}\right) \nabla_{\theta^\mu} \mu(\mathcal{S}, \theta^\mu)\right], \tag{14}$$

among them, $\mathcal{A} = \{a_n | n \in \mathcal{N}\}$. By calculating this gradient, we can update the parameters of the actor network θ^μ so that the actor network outputs better and better actions for a given state.

It is worth noting that the computed gradient needs to be gradient-decimated, which can avoid numerical instability caused by too large a gradient, such as skipping the optimal solution. The computed policy gradient will be used to further update the parameters of the actor network to make the parameter update more stable and efficient. We assume that the learning rate of the actor network is β^μ, and use the adaptive estimation technique (Adam) method commonly used in DRL to obtain the optimal θ^μ.

In the critic network undergoes a training process, the target action μ^- network estimates the target action $a'_t = \mu^-(s'_t, \theta^{\mu^-}) + \hat{\sigma}^2$ from the state at the next time step, where $\hat{\sigma}^2$ represents the strategy noise, specifically, pruned additive Gaussian noise. Following this, the next action a'_n and the next state s'_n are inputted into the target critic network Q_1^- and the critic network Q_2^-, in that order, using the parameters $\theta^{Q_1^-}$ and $\theta^{Q_2^-}$. The output of these networks is $\tilde{q}_{n,1}$ and $\tilde{q}_{n,2}$, in that order. Subsequently, the Bellman equation is used to estimate the value of Q, denoted as $\tilde{q}_n = \min(\tilde{q}_{n,1}, \tilde{q}_{n,2})$, with γ being a variable represents the discount rate. Concurrently, both action a_n and state s_n are fed into the critic networks Q_1 and critic Q_2, with θ^{Q_1} and θ^{Q_2} as their respective parameters, resulting in $q_{n,1}$ and $q_{n,2}$ in that order. Ultimately, for every $\bar{q}_n$, as per the mean square error theorem, the anticipated squared loss function between $Q_1(\mathcal{S}, \mathcal{A}, \theta^{Q_1})$ and $\bar{Q}$ is computed

$$L_1\left(\theta^{Q_1}\right) = 0.5\mathbb{E}\left[\left(Q_1\left(\mathcal{S}, \mathcal{A}, \theta^{Q_1}\right) - \bar{Q}\right)^2\right]. \tag{15}$$

Algorithm 1 MASTD3: Multi-agent Twin Delayed Shared Deep Deterministic Policy Gradient

```
1: Initialization:
   Initialize replay buffer $D$
   Initialize critic network and actor network $Q_{\theta_1}$, $Q_{\theta_2}$, $\pi_\phi$
2: for $episode = 1$ to $M$ do
3:   Choose a random initial state $s_i$
4:   Getting the status of preprocessing $\phi_i = \phi(s_i)$
5:   for $step = 1$ to $T$ do
6:     Extracting higher weights minibatch of transitions
       $(s_j, a_j, r_j, s_{j+1})$ from $D$ model sample data
7:     $y_j = \begin{cases} r_j & \text{terminates} \\ r_j + \gamma \max_{a'} J(\hat{Q}(\varphi_{j+1}, a'; \theta)) & \text{otherwise} \end{cases}$
8:     $loss = (y_j - J(Q(\varphi_j, a_j; \theta)))^2$
9:     Execute a gradient descent on $\theta_1$, $\theta_2$, $\theta_3$
10:    if $step \% n == 0$ then
11:      $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\theta'_3 \leftarrow \theta_3$
12:      for $agent\ i = 1$ to $N$ do
13:        Randomly select $H$ from $D$ to form minibatch
14:        respectively, through Eq. (15) and Eq. (16) Updating the
           Critics' Network and the Actors' Network
15:      end for
16:      Updating the target network by Eq. (19), Eq. (20) and
         Eq. (21)
17:    end if
18:    for $t = 1$ to $T$ do
19:      Randomize a stochastic process $\sigma$, using Eq. (11) that
         facilitates continuous action exploration
20:      Otherwise select $a_t = \arg\max_a Q(\varphi(s_t), a; \theta)$
21:      Executing actions in the simulator $a_t$
22:      Observe the reward $r_t$ and the next state $s_{t+1}$
23:      preprocessing $\varphi_{t+1} = \varphi(s_{t+1})$
24:      Store the conversion $(\varphi_t, a_t, r_t, \varphi_{t+1})$ in $D$
25:      if $D$ reaches the capacity limit and $t \% m == 0$ then
26:        Sample a small random batch from $D$ with a prioritized
           replay caching mechanism: $(\varphi_j, a_j, r_j, \varphi_{j+1})$
27:        Performs the same operation as in for $step$
28:      end if
29:    end for
30:  end for
31: end for
```

The anticipated squared loss between $Q_2(\mathcal{S}, \mathcal{A}, \theta^{Q_2})$ and $\bar{Q}$ is

$$L_2\left(\theta^{Q_2}\right) = 0.5\mathbb{E}\left[\left(Q_2\left(\mathcal{S}, \mathcal{A}, \theta^{Q_2}\right) - \bar{Q}\right)^2\right], \tag{16}$$

where $\bar{Q} = \{\bar{q}_n | n \in \mathcal{N}\}$, the objective is to minimize the mean square error between the actual Q value and the target Q value, followed by the gradient of the loss function $L_1\left(\theta^{Q_1}\right)$ in relation to θ^{Q_1} is

$$\nabla_{\theta^{Q_1}} L_1 = \mathbb{E}\left[\left(Q_1\left(\mathcal{S}, \mathcal{A}, \theta^{Q_1}\right) - \bar{Q}\right) \nabla_{\theta^{Q_1}} Q_1\left(\mathcal{S}, \mathcal{A}, \theta^{Q_1}\right)\right]. \tag{17}$$

The slope of the loss function $L_2\left(\theta^{Q_2}\right)$ in relation to θ^{Q_2} is

$$\nabla_{\theta^{Q_2}} L_2 = \mathbb{E}\left[\left(Q_2\left(\mathcal{S}, \mathcal{A}, \theta^{Q_2}\right) - \bar{Q}\right) \nabla_{\theta^{Q_2}} Q_2\left(\mathcal{S}, \mathcal{A}, \theta^{Q_2}\right)\right]. \tag{18}$$

Similarly to computing the policy gradient, there is no significant difference between using Eqs. (17) and (18). These gradients are used to update the parameters of the value network to incrementally improve the estimation of the value of state–action pairs. Gradient clipping is also required after computing the gradient. Furthermore, β^Q symbolizes the critic network's learning pace, with the parameters of both critic networks being modified through Adam's algorithm. For improved stability in learning, the critical target network's parameters also necessitate modifications via the soft update technique. This process entails the incremental modification of the critic target network's parameters to align with those of the critic network, as described by

$$\theta^{\mu^-} = \lambda\theta^{\mu} + (1-\lambda)\theta^{\mu^-}, \tag{19}$$

$$\theta^{Q_1^-} = \lambda\theta^{Q_1} + (1-\lambda)\theta^{Q_1^-}, \tag{20}$$

$$\theta^{Q_2^-} = \lambda\theta^{Q_2} + (1-\lambda)\theta^{Q_2^-}, \tag{21}$$

where λ represents the rate at which the target network learns. This document employs a reduced frequency for updating the network, necessitating multiple batch network training to maintain the stability of the Q-value. Subsequently, we proceed to update the network of strategies.

The goal function of MASTD3 combines several of these learning objectives

$$J(\theta^{\mu}) = J_{TD3}(\theta^{\mu}) + \lambda_1 J_E(\theta^{\mu}) + \lambda_2(L_1\left(\theta^{Q_1}\right) + L_2\left(\theta^{Q_2}\right)), \tag{22}$$

where $J_{TD3}(\theta^{\mu})$ is the target function of $Q(s,a)$, $J_E(\theta^{\mu})$ is the target function of supervised learning, and λ_1 and λ_2 are the weighting coefficients to balance these target functions. It is used to ensure the effectiveness and stability of the algorithm in optimizing the strategy and value estimation.

where $J_{TD3}(\theta^{\mu})$ is the objective function of $Q(s,a)$, $J_E(\theta^{\mu})$, which improves the quality of decision making by optimizing the strategy, $J_E(\theta^{\mu})$ is the objective function of supervised learning, which helps the algorithm to converge to a better strategy faster by utilizing the prior knowledge or labeled data, $L_1\left(\theta^{Q_1}\right)+L_2\left(\theta^{Q_2}\right)$ denotes the loss function of the two Q-networks, which is aimed at reducing the prediction error and improving the accuracy of the value function estimation, and λ_1 and λ_2 is the weighting factor that balances these objective functions, and through appropriate weight allocation, a balance can be found between different learning objectives, which can be used to ensure the effectiveness and stability of the algorithm in optimizing the strategy and value estimation. This multi-objective optimization strategy improves the performance and adaptability of the overall algorithm by using the complementary advantages of each.

4.3. Summary

At any given time slot t, the MD segments the application $\mathcal{G}$ into several distinct tasks, each linked to a particular ratio. Subsequently, it is up to the client to determine the parameters for offloading, namely the target server and the desired proportion. Our suggested MASTD3 involves eliminating $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$. Training of the model takes place on an edge server, where the user regularly retrieves the most recent model. Subsequently, the client utilizes the state $\mathcal{S}$ into the model to derive Q-values for every action, represented as $\{Q(\mathcal{S},a_1), Q(\mathcal{S},a_2),\dots,Q(\mathcal{S},a_n)\}$. The operation related to the highest Q-value, denoted as $a' = \arg\max_{a\in\mathcal{A}} Q(\mathcal{S},a)$ gets chosen. Following this procedure, indicated as $a' = (E_{tar}, P_b)$, the client transfers the application to the designated server E_{tar} in accordance with the desired ratio P_b.

Initially, the replay buffer D is set up with model sample data, the behavioral network Q with θ, and the target network $\hat{Q}$ with θ'. Subsequently, we proceed to comparably optimal weights θ_1 and θ_1' via guided learning on the sampled model data. Subsequently, the behavioral network is employed to engage with the RL agent (namely, the MD) and gather a sequence of specimens, archiving them in the replay buffer. Upon attaining a predetermined number of samples, we intermittently choose favored samples $(\phi_j, a_j, r_j, \phi_{j+1})$ from D at regular intervals to train the network until the convergence of both networks occurs.

The impact of network dynamics refers to the effect of changes in the network environment on the performance of computational offloading algorithms. It mainly includes: bandwidth fluctuation that affects data transmission speed and latency, server availability changes that increase system scheduling overhead, network congestion that leads to packet loss and retransmission increases latency and energy consumption, and latency changes that affect the real-time processing capability of tasks.

The MASTD3 copes with the effects of network dynamics and provides a flexible task-offloading solution by the following methods. First, DRL is utilized to implement an adaptive offloading strategy that monitors and adjusts network conditions in real-time. Second, a prioritized empirical replay buffer is introduced to improve the model's adaptability to dynamic changes and alleviate the cold-start problem. All agents share a public evaluation network to promote information sharing and collaboration, improve system stability, and perform well even with the increase of devices and users. Finally, a real-time monitoring and feedback mechanism is integrated to adjust the task offloading strategy in a timely manner to ensure efficient operation of the system in different network dynamic environments. The detailed flow of the MASTD3 is outlined in Algorithm 1.

The composite objective function design in MASTD3 combines the reinforcement learning and supervised learning objectives of TD3, which are balanced by weighted coefficients to enable the algorithm to excel in a variety of metrics such as reward, latency, and energy consumption. The pruned Gaussian noise treatment reduces the random fluctuations in policy updates and improves stability and robustness. MASTD3 significantly improves the multi-objective optimization capability through these several key innovations.

In MADRL, there are also scenarios where the agents do not share a critical network. For example, the agents work in different environments, and different agents perceive different states of the environment. This situation requires each agent to perform value estimation and policy updates independently. If it is not shared, the problem can be solved using MATD3, which is used to handle scenarios where multiple agents do not share a critical network.

Table 2
Setup parameters.

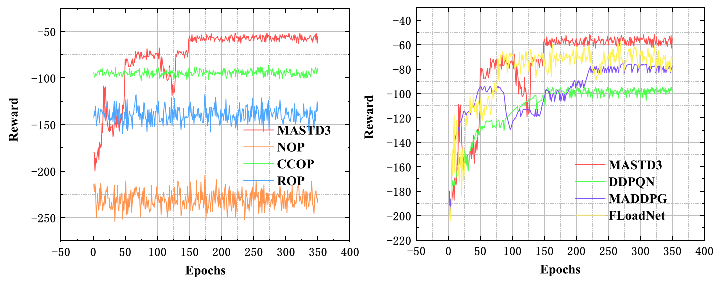
Parameter	Value	Parameter	Value
p_u	0.01 W	N	256
g_{ul}	10^{-3}	β_2	0.1
B	3	λ	0.8
d	10	λ_1	0.9
β_l	−2	λ_2	0.4
α	0.001	β	0.001
γ	0.9	ε	0.9
K	350	D	2000
m	50	f	200

5. Experiment

This part evaluates the efficacy of the MASTD3 we suggest. In our assessment, we focus on creating simulation tests that revolve around real-time video analysis tools, assigning distinct values to various parameters. Subsequently, we evaluate the practicality of employing MASTD3 in these situations. Furthermore, multiple control groups were formed to confirm the efficiency of the MASTD3.

5.1. Parameter setup

Our simulation experiment mirrors a real-life scenario in a real-time monitoring system, where end devices receive simultaneous real-time data from various sensor devices. Since the data from these sensors are independent of each other, we can divide the monitoring system into multiple independent tasks, each tasked with analyzing data from a specific sensor device. If the end devices' computational capabilities are inadequate, a specific proportion of data must be transferred to the respective edge server for processing. As a result, the final device



(a) Comparison with static methods. (b) Comparison with other algorithm.

Fig. 4. Changes in reward for different methods.

Table 3
Changes in various algorithmic rewards as the number of MESs changes.

The number of MESs	NOP	CCOP	ROP	DDPQN [30]	MADDPG [31]	FLoadNet [29]	MASTD3
3	235.127128	125.999806	139.169395	96.024980	118.576540	83.711629	**81.649920**
5	232.986406	123.011159	134.875464	107.090790	110.920730	89.494631	**80.463030**
7	217.194178	119.213036	130.753300	70.749970	104.302350	101.368773	**55.102914**
9	226.350736	126.236699	135.169395	90.107290	118.682090	97.594187	**76.031120**
10	242.187325	135.700859	150.394199	95.964800	125.945170	104.541259	**88.991590**

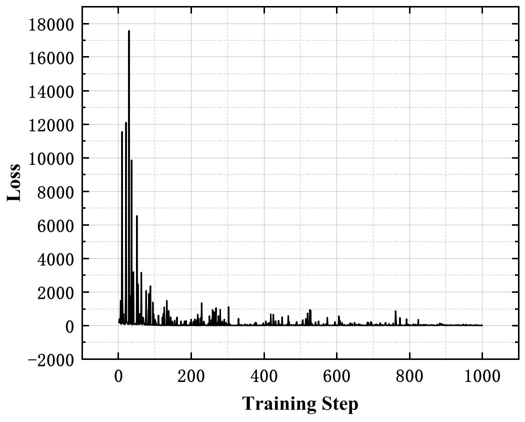


Fig. 5. Impact of the number of mobile edge servers on total losses.

must identify the desired edge server and decide the amount of data to transfer.

Presuming every terminal device handles data from 8:00 a.m. to 5:00 p.m., producing a single data point every five minutes, steady growth in data volume is noted from 8:00 a.m. to 11:00 a.m. during business hours. Accompanying this rise signifies a rise in the accessible computing resources on both local and server levels, coupled with a reduction in the link bandwidth available. The interval from 11:00 a.m. to 2:00 p.m. marks the interval of rest, leading to a trend of fluctuation that contrasts with that of the morning. Lastly, starting at 2:00 p.m., the pattern of each element reverts to its initial morning configuration.

In general, we can set the parameters according to the scenario described above. Initially, data size B adheres to a consistent distribution, denoted as $\mathcal{B}$ ranging from $[2000, 3000]$. The count of initially accessible subcarriers k and the computational assets (namely, cores) f_s and f_l exhibit a consistent distribution, with k ranging between [160,200]. with f_s ranging from [25,32] and f_l from [6,8], in that order. During every time interval t, the fluctuations $\Delta\mathcal{B}$, Δk, Δf_l, Δf_s adhere to a Poisson distribution characterized by parameters $\mathcal{P}$ = 30,10,2 and 5, in that order. Additional fixed parameters can be found in Table 1.

5.2. Comparison design

For confirming MASTD3's efficacy, a variety of tactics were used as standard benchmarks:

- No Offload Policy (NOP): This policy entails handling all processes locally, disregarding link bandwidth and server resources.
- Computational Capacity Optimal Policy (CCOP): At each time slot t, the MD will choose the server with the largest amount of available computing resources and migrate the entire application to it.
- Random Offload Policy (ROP): At every time slot t, the MD selects a random $a \in \mathcal{A}$ server and a ratio for offloading.
- MATD3: The critic network in MATD3 is not shared, and it is used to handle situations that require value estimation and strategy updating to be performed independently by each agent.
- DDPQN [30]: The Dual Duel Priority Deep Q-Network (DDPQN) algorithm allocates computational resources rationally in local-edge-cloud collaborative environments, aiming for an efficient offloading strategy with low latency, energy consumption, and cost.
- MADDPG [31]: This algorithm minimizes the total cost for all mobile users while meeting delay and resource constraints using the Multi-agent Deep Deterministic Policy Gradient (MADDPG) approach.
- FLoadNet [29]: The algorithm presents an actor-critic reinforcement learning framework that significantly reduces computational cost for optimizing joint multi-subject strategies.

5.3. Simulation results

First, our feasibility of MASTD3 is demonstrated. The convergence of the network can be ensured because we use deep neural networks to evaluate the Q-value. Originally, the inaccurate estimation would seriously influence the decision-making performance in the later stage. Additional parameters can be found in Table 2.

During the training process, we learn using different methods: NOP, CCOP, ROP, and our proposed MASTD3. We observe the performance of each agent body in terms of average reward with a gradually increasing number of training sets.

As shown in Fig. 4(a), the rewards of the NOP, CCOP, and ROP remain almost unchanged throughout the training process, indicating the effectiveness of the DRL-based algorithms in the learning task. At the beginning of training, MASTD3 is only marginally better than ROP and much worse than the other three strategies, mainly due to the high degree of randomness in action selection. However, MASTD3 gradually increases the probability of selecting the optimal action as the

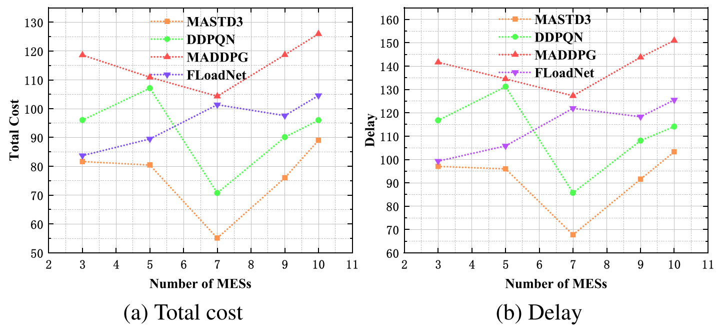


Fig. 6. Impact of number of users on total loss and delay.

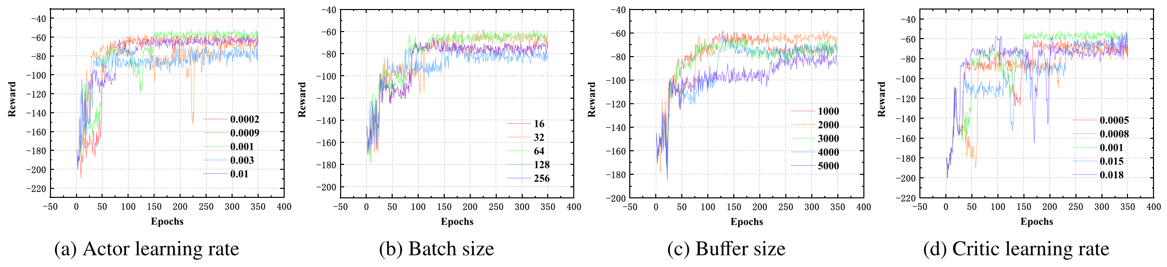


Fig. 7. Effect of learning rate, model sample buffer size, and batch size on the performance of MASTD3.

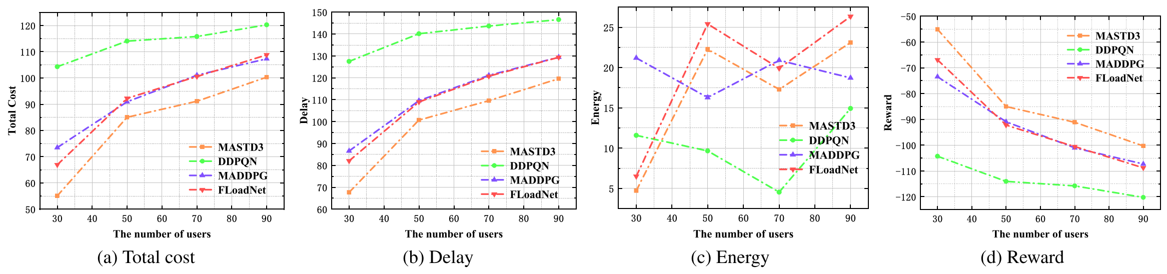


Fig. 8. The total cost, delay, and reward as the number of users changes.

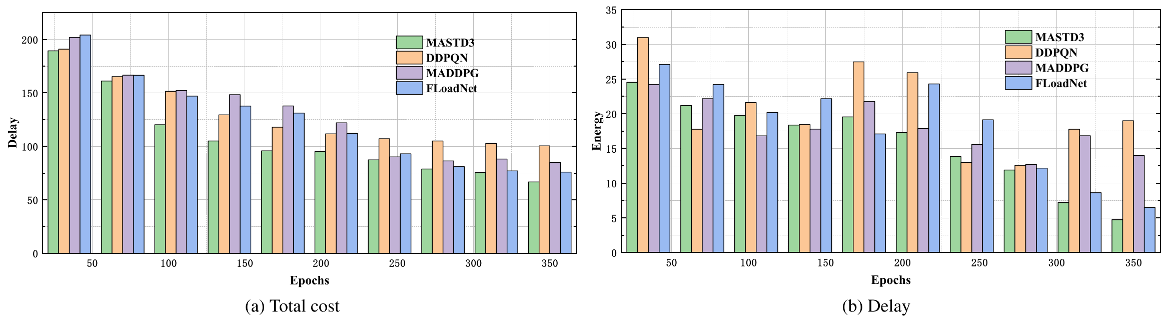


Fig. 9. Impact of number of users on total loss and delay.

number of training steps increases, bringing the decision closer to the ultimate choice each time. CCOP unilaterally pursues the optimization of the resources (i.e., subcarriers k or kernels f), a valid approach in many cases, but the consideration of the problem is too one-sided. The performance of CCOP is weak when the network is crowded. As a result, the whole performance is marginally worse than that of MASTD3. On the other hand, ROP is too random, and the convergence speed of MASTD3.

We also provide a comparison of the behavior of MASTD3 with other tactics. In a similar way to the approach described earlier, the rewards of the DRL-based algorithms are clearly demonstrated in Fig. 4(b) as the number of training sets proceeds. MASTD3 converges faster and obtains higher rewards in the same rounds. This indicates that MASTD3 not only achieves a stable performance during the learning process, making it a superior choice to cope with the computational offloading problem under mobile edge networks.

Fig. 5 depicts the change in the network's loss during the training process, which stabilizes after about 150 episodes, indicating that MASTD3 learns an effective strategy for resource allocation and adapts quickly to the changing environment. The figure clearly illustrates that with an increase in training steps, there is a gradual reduction in training loss, ultimately leading to zero convergence, indicating convergence. This suggests the practicality of estimating the Q-value using neural networks. Variations in network loss arise due to the

Table 4
Losses of different algorithms for different failure rates and failure time slots.

	MASTD3			DDPQN [30]			MADDPG [31]			FLoadNet [29]		
	5	10	15	5	10	15	5	10	15	5	10	15
0	55.10	–	–	98.96	–	–	77.90	–	–	66.77	–	–
1	59.70	61.29	74.19	104.04	105.82	107.74	92.20	95.46	100.09	74.83	75.10	76.10
2	60.34	66.58	88.43	105.76	106.89	109.84	105.51	104.86	122.32	74.98	75.05	76.18
3	61.31	67.87	77.29	106.71	107.09	108.59	105.06	109.98	110.38	76.25	77.49	78.58
4	73.73	74.64	76.44	105.39	106.24	107.44	106.00	109.63	111.64	78.04	79.58	80.04
5	74.96	75.17	92.15	106.50	107.41	108.36	107.62	110.18	110.41	79.64	80.18	81.18
10	75.60	77.64	101.67	107.36	114.92	115.14	110.00	112.90	115.82	80.86	82.72	83.15
15	76.83	85.79	90.22	108.53	109.43	110.48	112.40	114.65	116.15	81.83	82.72	83.14
20	81.99	83.59	88.89	109.30	111.68	112.71	113.30	115.94	118.35	82.72	83.19	84.65
25	84.85	108.02	110.71	110.87	112.44	114.21	114.57	116.10	119.11	84.85	85.85	86.65

reinforcement learning agent's probabilistic decision-making at every stage. Consequently, if the action is suboptimal (or beneficial), the agent will be rewarded with either a low (or high) amount, resulting in variations during the training defeat.

In Fig. 6, we compare various amounts of mobile edge servers. As the quantity of mobile edge servers equals 7, we observe that the workloads are distributed more efficiently, the resources are allocated more appropriately, and the system reaches an optimal equilibrium point, which ultimately leads to the minimization of system losses.

Increasing the number of edge servers may provide several benefits. First, a reasonable increase in the number of edge servers can better satisfy user or device requests and reduce the latency of data transmission and processing. Second, a moderate increase in the number of edge servers can effectively share the workload and reduce the average power consumption of the system because each server is running at relative ease. However, too many edge servers can have some negative effects. Too many servers may lead to over-dispersal of the system, adding additional communication and task scheduling overheads, and thus increasing latency. In addition, coordination and synchronization between edge servers takes time, and too many servers can also increase management and scheduling complexity. In addition, too many servers add additional overhead for system communication and data transfer, which in turn increases the overall energy consumption of the system.

As the number of edge devices increases, MASTD3 is able to effectively cope with more device requests and task assignment needs. Its DRL-based architecture has a certain degree of parallelism, which enables it to handle multiple device requests in large-scale deployments and make appropriate decisions in a shorter period of time.

The analysis of the impact of various parameters on the performance of MASTD3 is crucial for optimizing its effectiveness. Here are the observations from the analysis:

Fig. 7(a) demonstrates that a combination of high learning rates does not inevitably speed up the acceleration of convergence. MASTD3 with a learning rate of $\alpha = 0.001$ achieves the highest return, indicating optimal convergence. Higher learning rates, such as $\alpha = 0.003$, result in slower convergence, and further increases may prevent convergence altogether. Fig. 7(b) reflects the rewards of MASTD3 at different batch sizes. Similarly, $N = 64$ demonstrates the fastest convergence and achieves the highest reward for MASTD3. Batch size does not significantly change effectiveness As shown in Fig. 7(c), increasing the model sample buffer size initially enhances performance, but the improvement is not linearly related to the buffer size. Beyond a certain threshold, further increases in buffer size do not significantly improve performance. Therefore, it is crucial to carefully determine the size of the model sample buffer. $D = 2000$ exhibits higher performance. As shown in Fig. 7(d), consistent with the findings for the actor's home network, a learning rate $\alpha = 0.001$ yields the largest reward for MASTD3 when considering the actor family network. Overall, these analyses highlight the importance of parameter selection in optimizing the performance of MASTD3. Moderate learning rates and appropriate buffer sizes are crucial for achieving optimal convergence and performance in computational offloading scenarios.

We investigate how the number of MDs affects these computational offloading algorithms. Fig. 8 shows the comparison of total loss, delay, energy consumption, and reward under different numbers of users. With the rise in the quantity of MUs, MASTD3 consistently achieves lesser losses compared to these algorithms. With the rise in the quantity of MUs, the overall expense of each algorithm escalates due to heightened competition for transmission and computational resources. Given that MASTD3 takes into account the uncertain distribution of resources (i.e., computational and transmission resources) at the level of ESs, it effectively orchestrates the operations of each MU. Although the system cost increases as the number of MUs increases, the system cost as well as the time delay of our approach is lower than other algorithms. You can also see more data in Table 3.

As the number of users increases, MASTD3 can better cope with the growth of network load and communication demand. Its algorithm design takes into account the interaction and competition among users, and by learning and optimizing the task offloading strategy, it can effectively manage the resource allocation and network traffic when the user scale expands, thus improving the overall efficiency and performance of the system. In addition, MASTD3 is flexible and adaptive, able to adjust and optimize according to changes in the environment and demand increases. It can dynamically adapt to IIoT deployments of varying size and complexity, and flexibly respond to various network conditions and environmental changes while maintaining stability and performance.

As depicted in Fig. 9, we compare the four algorithms, MASTD3, DDPQN, MADDPG, and FLoadNet, in terms of time delay and energy consumption during training, our algorithm is significantly lower than the other two algorithms in terms of computational time delay and energy consumption, and it can be seen that the effect of DDPQN is the worst, and MASTD3 has had a change in the metric of energy consumption, but after that it still find a better strategy, using MASTD3 as a benchmark, the time delay is 30.62% lower than DDPQN, 20.3% lower than MADDPG, and 12.17% lower than FLoadNet.

At the same time, we investigate the robustness of the model under server availability. We conduct a comparative study on the performance of different algorithms in terms of server failure rate and failure time slots. The experiments include four algorithms; MASTD3, DDPQN, MADDPG, and FLoadNet. We observe the performance of each algorithm by modifying the server failure rate as well as the failure time slot. In each scenario, we recorded the performance of each algorithm at different failure rates (0%, 1%, 2%, 3%, 4%, 5%, 10%, 15%, 20%, 25%) and different failure time slots (5, 10, 15). The results are summarized in the following table, which contains data on the performance of each algorithm in each scenario.

With 5, 10, and 15 in the second row denoting the failure time slots and the first column denoting the failure probability, it can be seen that the loss, delay, and energy consumption of the algorithms increases overall with the increase in the failure rate and the failure time slots, but there are exceptions. These exceptions show that the algorithms are able to optimize and adjust their performance in the face of complex and dynamic environments through a variety of mechanisms so that they can still maintain a high level of performance in certain specific

Table 5
Delay of different algorithms for different failure rates and failure time slots.

	MASTD3			DDPQN [30]			MADDPG [31]			FLoadNet [29]		
	5	10	15	5	10	15	5	10	15	5	10	15
0	67.75	–	–	118.48	–	–	93.54	–	–	80.02	–	–
1	73.49	75.29	91.00	124.83	127.25	129.16	110.02	113.19	118.40	88.29	89.05	88.81
2	74.20	81.86	108.69	128.05	129.32	131.61	125.92	125.22	146.13	88.81	88.85	90.03
3	75.22	83.41	94.32	128.56	128.79	130.18	124.96	130.80	131.20	90.54	91.81	92.91
4	90.71	91.84	93.08	126.42	128.24	129.03	126.65	131.75	133.38	93.52	94.32	94.53
5	92.17	92.40	112.51	127.26	128.20	129.72	128.83	131.94	132.14	94.85	95.06	96.32
10	92.73	95.24	122.55	128.30	136.58	137.55	131.79	135.37	138.55	96.19	100.10	105.48
15	94.12	104.92	110.17	129.45	132.84	133.95	134.53	136.93	138.86	96.99	98.01	98.42
20	99.87	102.37	104.45	130.39	133.33	134.31	135.56	138.42	141.26	97.20	105.31	119.42
25	103.27	130.37	132.77	131.57	134.06	136.10	136.93	138.53	142.06	109.23	131.08	138.64

Table 6
Energy of different algorithms for different failure rates and failure time slots.

	MASTD3			DDPQN [30]			MADDPG [31]			FLoadNet [29]		
	5	10	15	5	10	15	5	10	15	5	10	15
0	4.52	–	–	20.87	–	–	15.34	–	–	13.81	–	–
1	4.84	5.30	6.99	20.86	20.11	15.29	20.94	24.55	26.83	16.70	17.95	20.27
2	4.88	5.44	7.40	16.61	17.17	16.55	23.85	23.40	27.08	19.69	19.86	20.79
3	5.69	5.73	9.18	19.33	19.89	22.21	25.48	26.74	27.10	19.11	20.24	21.28
4	5.80	5.87	9.88	21.25	18.27	20.84	23.43	21.15	24.67	19.35	20.63	22.09
5	6.13	6.22	10.71	23.43	24.25	22.88	22.78	23.17	23.51	20.33	20.68	23.24
10	7.10	7.28	18.15	23.59	28.31	25.47	22.84	23.02	24.89	19.57	21.54	23.77
15	7.67	9.27	10.44	24.85	15.80	16.60	23.85	25.53	25.32	19.80	21.58	22.07
20	10.48	8.44	11.64	24.96	25.04	26.34	24.22	26.01	26.74	20.60	25.36	22.88
25	11.16	18.59	22.48	25.57	25.96	26.61	25.09	26.37	27.31	22.27	25.74	28.72

situations. This also reflects the adaptability and robustness of the algorithm in complex environments. MASTD3 has relatively low loss values at all fault time slots and especially performs best with 5 time slots. In contrast, the loss of DDPQN increases significantly with increasing fault time slots, while MADDPG has the highest loss at high fault rates. FLoadNet performs between MASTD3 and the other algorithms, but the loss is still high at high fault rates. As can be understood from Table 5, MASTD3 has significantly lower latency than the other algorithms. From Table 6, it can be understood that the energy consumption of MASTD3 is also lower than the other algorithms. The experimental results show that MASTD3 performs well under the conditions of changing server availability, verifying its robustness and effectiveness in this aspect of the application.

With 5, 10, and 15 in the second row denoting the failure time slots and the first column denoting the failure probability, it can be seen that the loss, delay, and energy consumption of the algorithms increases overall with the increase in the failure rate and the failure time slots, but there are exceptions. These exceptions show that the algorithms are able to optimize and adjust their performance in the face of complex and dynamic environments through a variety of mechanisms so that they can still maintain a high level of performance in certain specific situations. This also reflects the adaptability and robustness of the algorithm in complex environments. In Table 4, MASTD3 has relatively low loss values at all fault time slots and especially performs best with 5 time slots. In contrast, the loss of DDPQN increases significantly with increasing fault time slots, while MADDPG has the highest loss at high fault rates. FLoadNet performs between MASTD3 and the other algorithms, but the loss is still high at high fault rates. As can be understood from Table 5, MASTD3 has significantly lower latency than the other algorithms. From Table 6, it can be understood that the energy consumption of MASTD3 is also lower than the other algorithms. The experimental results show that MASTD3 performs well under the conditions of changing server availability, verifying its robustness and effectiveness in this aspect of the application.

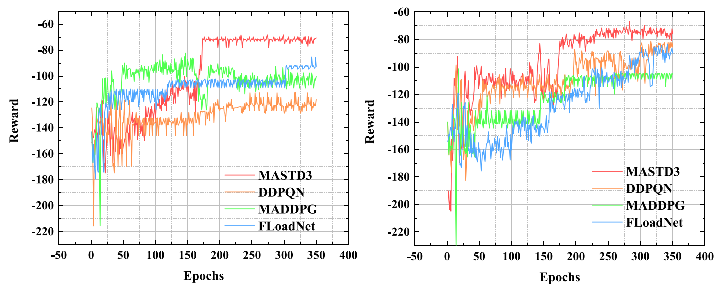
5.4. Real world data results

We use real traces of 4G/LTE networks to construct an evaluation environment. The first trace dataset contains 4G network bandwidth measurements for several routes in and around the city of Ghent, Belgium between 2015-12-16 and 2016-02-04. These data are collected from a Huawei smartphone running in six situations: walking, cycling, bus, streetcar, train, and car. We randomly select three walking datasets for our experiments. The results are shown in Fig. 10(a). The second tracking dataset contains 4G network bandwidth measurements for several routes in and around Jinan, Shandong Province, China between 2022-06-16 and 2023-07-17. These data are collected from Xiaomi smartphones running in five situations: walking, bicycling, bus, high-speed rail, and car. We randomly selected three HSR datasets in our experiment. The results are shown in Fig. 10(b) The other experimental parameters are set the same as the previous experimental parameters.

MASTD3 has the highest and most stable reward values throughout the training process, indicating its optimal performance on this task. FLoadNet performs second only to MASTD3. MADDPG, although it improves in the later stages of the training, does not perform as well as MASTD3 and FLoadNet overall. MADDPG, as well as DDPQN, have unstable reward values. This graph shows that the MASTD3 still has a significant advantage in real world data.

6. Conclusion

The proposed MASTD3 addresses the computational offloading problem in real-time multi-user, multi-server scenarios. By minimizing the weighted sum of latency and energy consumption, it aims to optimize the allocation of workload to edge servers. Key enhancements include leveraging the time-varying nature of the edge environment, integrating priority replay buffer and model sample buffer mechanisms, and sharing a common critic network among all agents. Experimental results demonstrate the practicality and efficiency of MASTD3. However, training DRL networks in real world scenarios with consistent tracking and minimal errors can be challenging. In the IIoT, MASTD3 is able to be generalized to other networks, and its generalization and applicability depend on the characteristics of the network and the setup. There are many different application scenarios in IIoT, such as smart manufacturing, remote monitoring, etc. These scenarios have different requirements for task offloading and resource management. the design of MASTD3 is flexible and can be adjusted according to



(a) Ghent, Belgium (b) Jinan, Shandong Province, China

Fig. 10. Real world data results.

specific application scenarios to optimize task scheduling strategies in different scenarios. Therefore, we will go a step further and design a DRL model that is more adapted to the real world.

CRediT authorship contribution statement

Kunkun Jia: Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Hui Xia:** Writing – review & editing, Supervision, Funding acquisition. **Rui Zhang:** Writing – review & editing, Supervision, Conceptualization. **Yue Sun:** Writing – review & editing, Supervision, Investigation. **Kai Wang:** Writing – original draft, Supervision.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Hui Xia reports financial support was provided by National Natural Science Foundation of China (NSFC) under grant number 62172377. Hui Xia reports financial support was provided by Taishan Scholars Program of Shandong province under grant number tsqn202312102. Hui Xia reports financial support was provided by Startup Research Foundation for Distnguished Scholars under grant number 202112016. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] G.S.S. Chalapathi, V. Chamola, A. Vaish, R. Buyya, Industrial Internet of Things (IIoT) applications of edge and fog computing: A review and future directions CoRR abs/1912.00595, 2019.
[2] W. Chen, X. Qiu, T. Cai, H. Dai, Z. Zheng, Y. Zhang, Deep reinforcement learning for Internet of Things: A comprehensive survey, IEEE Commun. Surv. Tutor. 23 (3) (2021) 1659–1692.
[3] A. Heidari, M.A.J. Jamali, N.J. Navimipour, S. Akbarpour, Internet of Things offloading: Ongoing issues, opportunities, and future challenges, Int. J. Commun. Syst. 33 (14) (2020).
[4] M. Kumar, G.K. Walia, H. Shingare, S. Singh, S.S. Gill, AI-based sustainable and intelligent offloading framework for iIoT in collaborative cloud-fog environments, IEEE Trans. Consum. Electron. (2023).
[5] C. Qiu, H. Yao, C. Jiang, S. Guo, F. Xu, Cloud computing assisted blockchain-enabled Internet of Things, IEEE Trans. Cloud Comput. 10 (1) (2022) 247–257.
[6] R. Almutairi, G. Bergami, G. Morgan, Advancements and challenges in IoT simulators: A comprehensive review, Sensors 24 (5) (2024) 1511.
[7] J. Niu, S. Zhang, K. Chi, G. Shen, W. Gao, Deep learning for online computation offloading and resource allocation in NOMA, Comput. Netw. 216 (2022) 109238.
[8] A. Hazra, A. Kalita, M. Gurusamy, Meeting the requirements of Internet of Things: The promise of edge computing, IEEE Internet Things J. 11 (5) (2024) 7474–7498.
[9] Y. Li, S. Cheng, H. Zhang, J. Liu, Dynamic adaptive workload offloading strategy in mobile edge computing networks, Comput. Netw. 233 (2023) 109878.
[10] S. Long, W. Long, Z. Li, K. Li, Y. Xia, Z. Tang, A game-based approach for cost-aware task assignment with QoS constraint in collaborative edge and cloud environments, IEEE Trans. Parallel Distrib. Syst. 32 (7) (2021) 1629–1640.
[11] L. Ma, X. Wang, X. Wang, L. Wang, Y. Shi, M. Huang, TCDA: Truthful combinatorial double auctions for mobile edge computing in industrial internet of things, IEEE Trans. Mob. Comput. 21 (11) (2022) 4125–4138.
[12] A. Shakarami, M. Ghobaei-Arani, A. Shahidinejad, A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective, Comput. Netw. 182 (2020) 107496.
[13] Z. Zabihi, A. Eftekhari-Moghadam, M.H. Rezvani, Reinforcement learning methods for computation offloading: A systematic review, ACM Comput. Surv. 56 (1) (2024) 17:1–17:41.
[14] J. Zhang, J. Du, Y. Shen, J. Wang, Dynamic computation offloading with energy harvesting devices: A hybrid-decision-based deep reinforcement learning approach, IEEE Internet Things J. 7 (10) (2020) 9303–9317.
[15] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, L. Li, Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning, IEEE Trans. Cogn. Commun. Netw. 7 (3) (2021) 881–892.
[16] G. Wu, Z. Xu, H. Zhang, S. Shen, S. Yu, Multi-agent DRL for joint completion delay and energy consumption with queuing theory in MEC-based IIoT, J. Parallel Distrib. Comput. 176 (2023) 80–94.
[17] L. Qian, Y. Wu, F. Jiang, N. Yu, W. Lu, B. Lin, NOMA assisted multi-task multi-access mobile edge computing via deep reinforcement learning for industrial internet of things, IEEE Trans. Ind. Inform. 17 (8) (2021) 5688–5698.
[18] X. Yao, N. Chen, X. Yuan, P. Ou, Performance optimization of serverless edge computing function offloading based on deep reinforcement learning, Future Gener. Comput. Syst. 139 (2023) 74–86.
[19] B. Saglam, F.B. Mutlu, D.C. Cicek, S.S. Kozat, Actor prioritized experience replay, J. Artificial Intelligence Res. 78 (2023) 639–672.
[20] L. Huang, S. Bi, Y.A. Zhang, Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks, IEEE Trans. Mob. Comput. 19 (11) (2020) 2581–2593.
[21] S. Zhang, H. Gu, K. Chi, L. Huang, K. Yu, S. Mumtaz, DRL-based partial offloading for maximizing sum computation rate of wireless powered mobile edge computing network, IEEE Trans. Wirel. Commun. 21 (12) (2022) 10934–10948.
[22] F. Wang, J. Xu, S. Cui, Optimal energy allocation and task offloading policy for wireless powered mobile edge computing systems, IEEE Trans. Wirel. Commun. 19 (4) (2020) 2443–2459.
[23] X. Liu, J. Liu, A truthful mechanism for multi-access multi-server multi-task resource allocation in mobile edge computing, Peer-to-Peer Netw. Appl. 17 (1) (2024) 532–548.
[24] L. Xiao, X. Lu, T. Xu, X. Wan, W. Ji, Y. Zhang, Reinforcement learning-based mobile offloading for edge computing against jamming and interference, IEEE Trans. Commun. 68 (10) (2020) 6114–6126.
[25] Z. Gao, L. Yang, Y. Dai, Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing, IEEE Trans. Mob. Comput. 22 (6) (2023) 3425–3443.
[26] J. Heydari, V. Ganapathy, M. Shah, Dynamic task offloading in multi-agent mobile edge computing networks, in: 2019 IEEE Global Communications Conference, GLOBECOM 2019, Waikoloa, HI, USA, December 9-13, 2019, IEEE, 2019, pp. 1–6.
[27] B. Shi, Z. Chen, Z. Xu, A deep reinforcement learning based approach for optimizing trajectory and frequency in energy constrained multi-UAV assisted MEC system, IEEE Trans. Netw. Serv. Manag. (2024).
[28] X. Wang, Z. Ning, S. Guo, Multi-agent imitation learning for pervasive edge computing: A decentralized computation offloading algorithm, IEEE Trans. Parallel Distrib. Syst. 32 (2) (2021) 411–425.
[29] J. Baek, G. Kaddoum, FLoadNet: Load balancing in fog networks with cooperative multiagent using actor-critic method, IEEE Trans. Netw. Serv. Manag. 20 (1) (2023) 400–414.

[30] M. Xue, H. Wu, G. Peng, K. Wolter, DDPQN: An efficient DNN offloading strategy in local-edge-cloud collaborative environments, IEEE Trans. Serv. Comput. 15 (2) (2022) 640–655.

[31] L. Chen, G. Gong, K. Jiang, H. Zhou, R. Chen, DDPG-based computation offloading and service caching in mobile edge computing, in: IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops, INFOCOM 2022 - Workshops, New York, NY, USA, May 2-5, 2022, IEEE, 2022, pp. 1–6.

Kunkun Jia earned her B.S. degree in Computer Science and Technology from Qingdao University in 2018. She is currently working towards her M.S. degree in Electronic Information at Ocean University of China in Qingdao, China. Her current research interests include Edge Computing and Computation Offloading.

Hui Xia received his Ph.D. degree in computer science at School of Computer Science and Technology, Shandong University, China, in June 2013. From July 2013 to June 2020, he worked as a lecturer and associate professor at College of Computer Science and Technology, Qingdao University, China. Since July 2020, he is currently a Full Professor and a Ph.D. Supervisor at the College of Computer Science and Technology, Ocean University of China. He was a visiting scholar at the Department of Computer Science, The George Washington University, U.S. (2017-2018). His current research interests focus on IoT security, AI security, Privacy protection, Federated learning, Edge computing and Smart healthcare. He has published over 50 scientific papers, and his research is sponsored by the Natural Science Foundation of China (NSFC) under Grant No. 62172377, the Taishan Scholars Program of Shandong province under Grant No. tsqn202312102, and the Startup Research Foundation for Distinguished Scholars under Grant No. 202112016. He is a member of the ACM and the IEEE.

Rui Zhang received the M.S. degree in network security from Qingdao University, in 2018. She is currently pursuing a Ph.D. in Artificial Intelligence at Ocean University of China, Qingdao, China. Her current research interests include artificial intelligence and adversarial attacks.

Yue Sun earned her B.S. degree in Computer Science and Technology from Qingdao University in 2018. She is currently working towards her M.S. degree in Computer Science and Technology at Ocean University of China in Qingdao, China. Her current research interests include Edge Computing and Resource Allocation.

Kai Wang received the B.S. and Ph.D. degrees from Beijing Jiaotong University. He is currently a Professor with the School of Computer Science and Technology, Harbin Institute of Technology (HIT), Weihai, China. Before joined HIT, he was a postdoc researcher in computer science and technology with Tsinghua University. He has published more than 40 papers in prestigious international journals and conferences, including IEEE TITS, IEEE TNSM, ACM TOIT, ACM TIST, etc. His research interest is in the area of Cyber Physical Systems Security (CPSSEC), with a focus on topics in Intrusion Detection using Lightweight Deep Learning Models. He is a Member of the IEEE and a Senior Member of the China Computer Federation (CCF).