Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.072505

ARTICLE

Heterogeneous Computing Power Scheduling Method Based on Distributed Deep Reinforcement Learning in Cloud-Edge-End Environments

Jinwei Mao[1,2], Wang Luo[1,2,*], Jiangtao Xu[3], Daohua Zhu[3], Wei Liang[3], Zhechen Huang[3], Bao Feng[1,2] and Shuang Yang[1,2]

[1]State Grid Electric Power Research Institute Co., Ltd., Nanjing, China
[2]Nanjing Nari Information & Communication Technology Co., Ltd., Nanjing, China
[3]State Grid Jiangsu Electric Power Co., Ltd. Research Institute, Nanjing, China

*Corresponding Author: Wang Luo. Email: luowang@sgepri.sgcc.com.cn

Received: 28 August 2025; Accepted: 15 December 2025; Published: 12 March 2026

ABSTRACT: With the rapid development of power Internet of Things (IoT) scenarios such as smart factories and smart homes, numerous intelligent terminal devices and real-time interactive applications impose higher demands on computing latency and resource supply efficiency. Multi-access edge computing technology deploys cloud computing capabilities at the network edge; constructs distributed computing nodes and multi-access systems and offers infrastructure support for services with low latency and high reliability. Existing research relies on a strong assumption that the environmental state is fully observable and fails to thoroughly consider the continuous time-varying features of edge server load fluctuations, leading to insufficient adaptability of the model in a heterogeneous dynamic environment. Thus, this paper establishes a framework for end-edge collaborative task offloading based on a partially observable Markov decision-making process (POMDP) and proposes a method for end-edge collaborative task offloading in heterogeneous scenarios. It achieves time-series modeling of the historical load characteristics of edge servers and endows the agent with the ability to be aware of the load in dynamic environmental states. Moreover, by dynamically assessing the exploration value of historical trajectories in the central trajectory pool and adjusting the sample weight distribution, directional exploration and strategy optimization of high-value trajectories are realized. Experimental results indicate that the proposed method exhibits distinct advantages compared with existing methods in terms of average delay and task failure rate and also verifies the method's robustness in a dynamic environment.

KEYWORDS: Edge computing; end-edge collaboration; heterogeneous computing power scheduling; resource allocation

1 Introduction

With the rapid advancement of next-generation information technologies, exemplified by applications like smart factories, smart homes, and the Internet of Vehicles, the number of terminal devices worldwide has grown exponentially [1]. According to International Data Corporation, the number of global Internet of Things (IoT) device connections is projected to exceed 41 billion by 2027 [2]. This explosive growth in device scale spurs complex and diverse network access requirements: industrial robots on production floors enable real-time control via ultra-reliable low-latency communication (URLLC); smart home appliances rely on Wi-Fi to construct high-density IoT networks; and autonomous vehicles on urban roads require integration of cellular networks and roadside units for collaborative perception [3]. The data deluge from massive terminal devices presents additional challenges to the existing cloud computing paradigm. Although

Copyright © 2026 The Authors. Published by Tech Science Press. This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

the cloud computing model offers powerful centralized processing capabilities, its inherent geographical distribution causes significant increases in transmission latency, failing to adequately support real-time-sensitive applications [4]. To address these issues, multi-access edge computing (MEC) has emerged as a computing paradigm that complements traditional cloud computing models [5]. MEC's core concept is to offload computing tasks from terminal devices to the network edge via diverse access technologies, enabling flexible computing power scheduling to meet the low-latency requirements for compute-intensive applications in multi-access scenarios (e.g., smart factories, smart homes) [6]. It distributes data processing across edge servers [7], allowing users to compute near data sources via multiple access methods, thus reducing transmission latency and accelerating computation. By distributing computing resources at the network edge, MEC is better able to adapt to diverse service requirements and reduces the risk of network outages [8].

While MEC exhibits significant development potential, it faces substantial challenges in practical applications, with heterogeneous computing power scheduling being the most prominent [9]. In smart scenarios, massive data generated by terminal devices no longer needs to be uploaded to the cloud for processing-reducing the core network's transmission burden but imposing higher demands on the resource management of edge nodes [10]. MEC nodes have limited resources yet serve a large number of terminal devices [11]; tasks must be distributed and scheduled based on their type and scale to prevent excessive loads on specific edge nodes from degrading system performance and user experience.

Significant progress has been achieved in optimizing heterogeneous computing power scheduling in MEC environments, with related studies continuously refining scheduling algorithms via deep reinforcement learning (DRL) and other methods [12]. As a specialized machine learning paradigm, DRL models systems based on Markov decision processes (MDPs) [13], enabling algorithms to adjust decisions based on environmental feedback during continuous trial-and-error-exhibiting strong performance in decision-making tasks. In MEC scenarios, DRL algorithms can dynamically adjust heterogeneous computing power scheduling strategies based on real-time edge node load, task urgency, and other key factors to achieve efficient resource utilization [14].

However, with the continuous emergence of diverse smart devices and the increasing number of terminal devices, edge networks exhibit highly dynamic environments, where edge server status (e.g., load) changes continuously over time. Against this backdrop, existing research has limitations: most algorithms assume that edge server load status can be fully acquired and updated in real time-an assumption hard to realize in practice due to monitoring overhead and privacy protection constraints. Thus, this paper leverages DRL to investigate heterogeneous computing power scheduling in MEC environments, aiming to optimize scheduling efficiency and service continuity while improving the quality of service (QoS).

2 Related Work

As the core technical support of the MEC system, the optimization level of the heterogeneous computing power scheduling strategy directly determines the quality of system services [15]. The heterogeneous computing power scheduling strategy not only needs to comprehensively consider multi-dimensional parameters such as the real-time load status of edge servers, network transmission quality, and task computing latency, but also needs to achieve a fine balance between resource utilization, cost-effectiveness, and service quality. Heterogeneous computing power scheduling enables edge servers and users to focus on their respective core businesses, thereby maximizing resource utilization, reducing costs, and enhancing overall service performance [16].

In recent years, machine learning techniques have shown significant advantages in the field of load optimization for complex systems. Especially in the edge computing scenario, the task offloading decision

faces a highly dynamic and heterogeneous complex system environment, involving the contradiction between multi-dimensional resource constraints (of terminal devices, communication links, and edge nodes) and real-time requirements, which poses a modeling bottleneck for meta-heuristic algorithms. In this context, DRL can simultaneously model the relationships among network bandwidth fluctuations, server load, and task priority by constructing an end-to-end "state-action-reward" decision-making optimization framework, and its online learning characteristics provide a new technical path for adaptive task offloading in complex edge environments.

Zhang et al. [9] proposed a heterogeneous task collaborative offloading scheme for satellite edge computing, which effectively reduces satellite energy consumption and improves task processing efficiency through a ground-satellite-cloud multi-layer architecture and dynamic threshold strategy. Gao et al. [17] proposed a DRL-based algorithm with continuous convex approximation for hybrid UAV-assisted systems, optimizing service assignment, task segmentation, UAV trajectories, resources, and transmission power to reduce energy consumption. Cai et al. [18] designed a framework based on the Double Deep *Q*-Network (DDQN), along with a resource tree model and multi-granular task decomposition, for cloud-edge-device networks, aiming to optimize task decomposition, hierarchical scheduling, and maximize latency-energy utility. Chen et al. [19] developed an online convolutional DRL algorithm for Non-Orthogonal Multiple Access (NOMA) enabled edge networks, decoupling the minimization of system computing completion time into duration and resource allocation sub-problems. Fan et al. [20] presented a dual-delay deep deterministic policy gradient algorithm for in-vehicle edge computing, using road side unit (RSU) collaboration to optimize task offloading and resource allocation, cutting down delay and energy use. Guo et al. [21] put forward a fuzzy trust-enhanced, DRL-driven scheme for vehicle-mounted edge networks, optimizing offloading delay and task processing credibility. Li et al. [22] built a pre-determined multi-agent DRL offloading mechanism for resource-constrained edges to maximize user experience quality. Li et al. [23] proposed a Lyapunov- and multi-agent deep deterministic policy gradient-based two-stage offloading architecture for distributed environments, minimizing subsystem delay through joint initial and RSU peer-to-peer offloading while considering RSU energy constraints. Lin et al. [24] formalized decentralized edge task offloading as partially observable Markov decision-making process (POMDP), leveraging heuristic logs to warm-up and fine-tune a DRL model, enhancing task success and solving cold-start issues. Liu et al. [25] constructed a two-time-scale multi-dimensional resource optimization model with a dual-actor deep deterministic strategy gradient for edge networks, reducing long-term delay and cache costs. Liu et al. [26] proposed a predictive Deep *Q*-Network (DQN) for resource allocation and computation offloading (PQ-RACO) algorithm and a multi-agent DQN for resource allocation and computation offloading (MQ-RACO) algorithm for the dynamic edge networks, which optimize offloading decisions and power allocation via agent behavior prediction and explicit reward design to maximize computing rates.

When applying DRL to construct a task offloading model, the existing research fails to effectively model the dynamic characteristics in large-scale terminal device scenarios:

(1) When the number of terminal devices surges, the load fluctuation of edge servers, the instantaneous change of network link state, and the heterogeneity of computing resources will lead to the high dynamic time variation of the system, which significantly reduces the stability of the state transition probability in the Markov Decision Process (MDP) based framework and makes it challenging to ensure the strategy convergence.

(2) The state space dimensional explosion and the complexity of action space combination caused by massive device access have forced the algorithm to frequently retrain the model and update the policy parameters, which not only causes the exponential increase of computing overhead, but also aggravates the system's dynamic imbalance due to the real-time response lag. The over-reliance on the dynamic Markov

assumption and the scalability defect of the high-dimensional state-action space seriously restrict the practical value of the algorithm in ultra-large-scale edge computing scenarios.

3 System Model

This section abstracts the MEC system into three layers, as illustrated in Fig. 1. The first layer is a third-party trusted authority, geographically remote from terminal devices, which primarily stores task offloading-related terminal device information. The second layer comprises edge nodes, including edge servers, base stations, and other edge network access points, where edge servers are equipped with sufficient computing resources; base stations are responsible for direct communication with terminal devices within their coverage area, handling wireless signal transmission and reception; and edge network access points integrate fixed-network-connected devices into the edge network. The third layer consists of a large number of terminal devices-typically resource-constrained devices enabled by the Industrial IoT and smart city initiatives. As shown in the figure, devices in the MEC environment include IIoT terminal devices and mobile robots connected via 5G, smart home devices connected via Wi-Fi access points, environmental monitoring sensors in smart cities connected via wired networks, and other smart devices using fixed broadband or Wi-Fi.

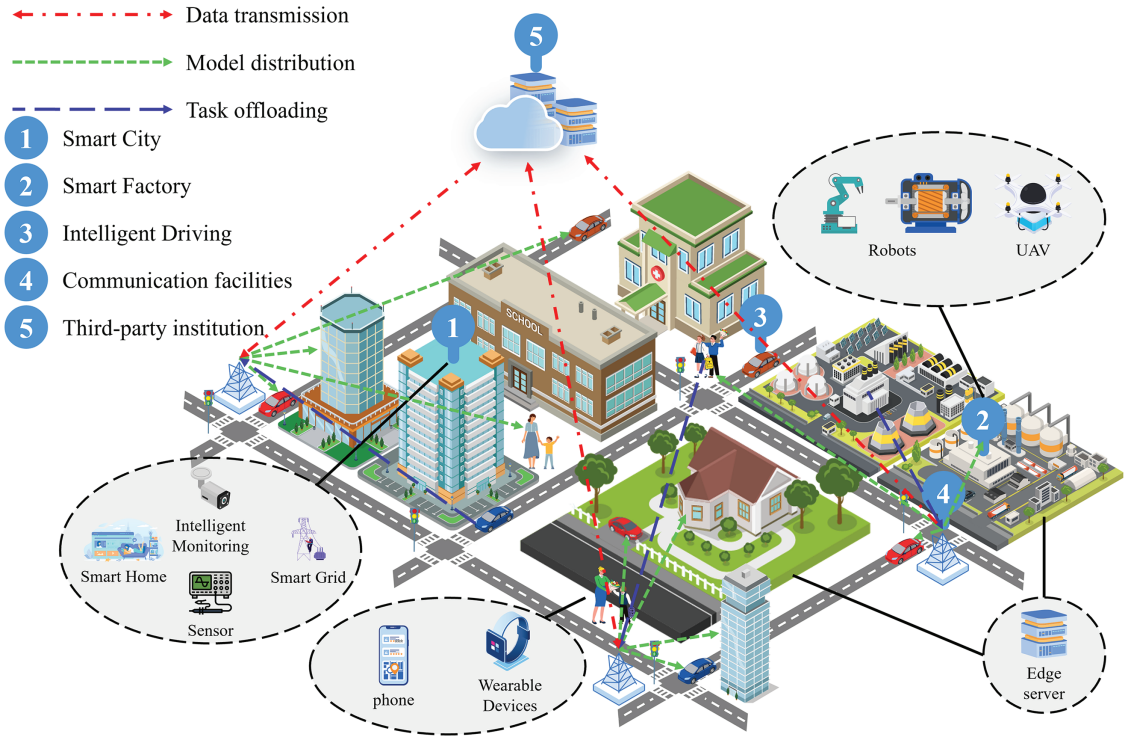


Figure 1: An overall cloud-edge-end collaboration task offloading scenario.

This section focuses on a MEC environment involving a set of edge servers denoted as $E = \{1, 2, \cdots, e\}$ and a set of terminal devices denoted as $D = \{1, 2, \cdots, d\}$. It specifically addresses task offloading over a set of time slots $T = \{1, \cdots, t\}$, where each time slot has a short duration δ (in seconds).

3.1 Task Model

Assume a new task arrives at a terminal device at the start of a time slot. Let $I_d(t)$ denote the unique index of the task arriving at the d-th terminal device during time slot t. For generality, $I_d(t) = 0$ if no new task arrives at terminal d in that time slot.

Let $\eta_d(t)$ denote the data size (in bits) of newly arrived task $I_d(t)$, with $\eta_d(t) = 0$ if no new task arrives at terminal d. Tasks have distinct processing densities $\rho_{d,t}$ (CPU cycles per bit). Each task also has a latency constraint $\tau_{d,t}$ (in time slots): task $I_d(t)$ is discarded if not fully processed by the end of time slot $t + \tau_{d,t} - 1$.

3.2 Transmission Model

Edge servers use non-overlapping frequency bands, eliminating uplink interference between them. Let $|h_{d,e}|^2$ represent the channel gain from terminal d to edge server e, and $p_{d,e}$ denote terminal d's transmission power when communicating with edge server e.

The transmission rate from terminal d to edge server e is $O_{d,e}^{tran}$, given by:

$$O_{d,e}^{tran} = B \log_2 \left(1 + \frac{|h_{d,e}|^2 p_{d,e}}{\sigma^2}\right), \quad d \in D,\ e \in E \tag{1}$$

where B is the channel bandwidth and σ^2 is the channel noise power.

At the start of a time slot, if the local computing capacity of terminal d is insufficient for task $I_d(t)$, $T_d^{tran}(t)$ denotes the time slot when the task is transmitted (to an edge server) or discarded (if offloading fails). If the task is processed locally or no task arrives, $T_d^{tran}(t) = 0$ (with $T_d^{tran}(0) = 0$).

Let $\mu_d^{tran}(t)$ (in time slots) represent the waiting time for transmission of task $I_d(t)$, determined by prior transmission slots $T_d^{tran}(t')$ (where $t' < t$) as:

$$\mu_d^{tran}(t) = \max\left\{\sup_{t' \in \{0,1,\ldots,t-1\}} T_d^{tran}(t') - t + 1,\ 0\right\} \tag{2}$$

If task $I_d(t)$ starts waiting for transmission in a time slot, $T_d^{tran}(t)$ (the time slot when transmission completes or the task is discarded) is calculated as:

$$T_d^{tran}(t) = \min\left\{t + \mu_d^{tran}(t) + \left\lfloor \sum_{e \in E} \frac{y_{d,e}(t)\eta_d(t)}{\delta O_{d,e}^{tran}} \right\rfloor - 1,\ t + \tau_{d,t} - 1\right\} \tag{3}$$

where $y_{d,e}(t) = 1$ indicates that task $I_d(t)$ is offloaded to edge server e, and $y_{d,e}(t) = 0$ otherwise.

3.3 Computational Models

When a new task $I_d(t)$ arrives at terminal d in a time slot, the terminal must decide whether to process the task locally or offload it to an edge server.

The indicator $x_d(t)$ specifies whether task $I_d(t)$ is processed locally ($x_d(t) = 1$) or offloaded to the edge ($x_d(t) = 0$). Here, $\eta_d(t)x_d(t)$ represents the data volume processed locally, and $\eta_d(t)(1 - x_d(t))$ represents the data volume transmitted to edge servers.

If task $I_d(t)$ is offloaded to edge server e, $y_{d,e}(t) = 1$ (0 otherwise). Each task can only be offloaded to one edge server, so:

$$\sum_{e \in E} y_{d,e}(t) = \begin{cases} 1 & \text{if } x_d(t) = 0, \\ 0 & \text{if } x_d(t) \neq 0. \end{cases} \tag{4}$$

3.3.1 Local Computing

Let f_d denote the computing capacity of terminal d. For time slot t, $T_d^{comp}(t)$ denotes the time slot when task $I_d(t)$ is processed locally or discarded; $T_d^{comp}(t) = 0$ if the task is not processed locally (e.g., $I_d(t) = 0$).

Let $\mu_d^{comp}(t)$ represent the waiting time for local processing of the task $I_d(t)$, determined by prior local processing slots $T_d^{comp}(t')$ (where $t' < t$) as:

$$\mu_d^{comp}(t) = \max\left\{ \sup_{t' \in \{0,1,\ldots,t-1\}} T_d^{comp}(t') - t + 1,\ 0 \right\} \tag{5}$$

If task $I_d(t)$ starts waiting for local processing in a time slot, $T_d^{comp}(t)$ (the time slot when processing completes or the task is discarded) is:

$$T_d^{comp}(t) = \min\left\{ t + \mu_d^{comp}(t) + \left\lfloor \frac{\eta_d(t)\rho_{d,t}}{\delta f_d} \right\rfloor - 1,\ t + \tau_{d,t} - 1 \right\} \tag{6}$$

3.3.2 Task Offloading

Each edge server e maintains a computation queue for each terminal $d \in D$. Let $I_{d,e}(t)$ be the unique index of the task from d added to e's queue at the start of time slot t ($I_{d,e}(t) = 0$ if no such task exists). Let $\eta_{d,e}(t)$ (in bits) be the task's data size ($\eta_{d,e}(t) = 0$ if no task arrives).

Edge server computation queues follow First-In-First-Out (FIFO). Let $q_{d,e}(t)$ (in bits) be the length of the queue of d in e at the end of time slot t. Let $q_e(t)$ denote the set of non-empty queues in e at time slot t:

$$q_e(t) = \{ d \mid \eta_{d,e}(t) > 0 \vee q_{d,e}(t-1) > 0 \} \tag{7}$$

Let $Q_e(t) = |q_e(t)|$ be the count of non-empty queues in e in time slot t.

Let f_e (in CPU cycles per second) be computing capacity of edge server e. Let $drop_{d,e}(t)$ (in bits) be the data dropped from d's queue on e at the end of the time slot t. The queue length $q_{d,e}(t)$ updates as:

$$q_{d,e}(t) = \begin{cases} (q_{d,e}(t-1) + \eta_{d,e}(t)) - \dfrac{f_e \delta}{\rho_{d,t} Q_e(t)} \left(\mathbb{1}(d \in q_e(t)) - drop_{d,e}(t) \right), & \mathbb{1}(d \in q_e(t)) - drop_{d,e}(t) \geq 0, \\ 0, & \mathbb{1}(d \in q_e(t)) - drop_{d,e}(t) < 0, \end{cases} \tag{8}$$

where $\mathbb{1}(\cdot)$ is the indicator function (1 if the condition holds, 0 otherwise).

For a terminal task d added to edge server e's queue at the start of the time slot t, $T_{d,e}^{comp}(t)$ denotes the time slot when the task is processed or discarded; $T_{d,e}^{comp}(t) = 0$ if $I_{d,e}(t) = 0$.

Let $\hat{T}_{d,e}^{comp}(t)$ denote the time slot when processing starts:

$$\hat{T}_{d,e}^{comp}(t) = \max\left\{ t,\ \sup_{t' \in \{0,1,\ldots,t-1\}} T_{d,e}^{comp}(t') + 1 \right\} \tag{9}$$

The task size $\eta_{d,e}(t)$ satisfies: it is not larger than the total processing capacity of edge server e from $\hat{T}_{d,e}^{comp}(t)$ to $T_{d,e}^{comp}(t)$, but greater than the total capacity from $\hat{T}_{d,e}^{comp}(t)$ to $T_{d,e}^{comp}(t)-1$. Formally:

$$\sum_{t'=\hat{T}_{d,e}^{comp}(t)}^{T_{d,e}^{comp}(t)} \frac{\delta f_e}{\rho_{d,t} Q_e(t')} \mathbb{1}(d \in q_e(t')) \geq \eta_{d,e}(t) \tag{10}$$

$$\sum_{t'=\hat{T}_{d,e}^{comp}(t)}^{T_{d,e}^{comp}(t)-1} \frac{\delta f_e}{\rho_{d,t} Q_e(t')} \mathbb{1}(d \in q_e(t')) < \eta_{d,e}(t) \tag{11}$$

3.3.3 Task Computation Latency

Task computation latency is defined as the sum of waiting time and processing time. Let $delay_d(t)$ (in time slots) denote the latency of the task $I_d(t)$.

If processed locally:

$$delay_d(t) = T_d^{comp}(t) - t + 1 \tag{12}$$

If offloaded to an edge server:

$$delay_d(t) = \sum_{e \in E} \sum_{t'=t}^{T} \mathbb{1}\left(I_{d,e}(t) = I_d(t)\right) T_{d,e}^{comp}(t') - t + 1 \tag{13}$$

3.4 Problem Definition

The optimization objective is to minimize the task computation latency: given the current system state, determine the optimal edge server for offloading such that task $I_d(t)$'s latency $delay_d(t)$ satisfies $delay_d(t) \leq \tau_{d,t}$.

The MEC environment exhibits continuous time-varying characteristics, and traditional algorithms lack scalability in large-scale dynamic scenarios. Thus, we employ DRL to address this problem. Formally, the terminal device policy is a mapping from system states to actions, denoted as $\pi_d : \mathcal{S} \rightarrow \mathcal{A}$. Defining task computation latency as the offloading cost, minimizing this cost corresponds to minimizing latency. Let $\gamma \in (0,1]$ be the discount factor (weighting future costs). The goal is to find each terminal's optimal policy to minimize the expected long-term latency:

$$\pi_d^* = \arg\min_{\pi_d} \mathbb{E}\left[\sum_{t \in T} \gamma^{t-1} delay_d(t) \mid \pi_d\right] \tag{14}$$

4 Heterogeneous Computing Power Scheduling Methods

The MEC environment is dynamically changing, and the load uncertainty of numerous terminal devices and edge servers renders traditional single-agent reinforcement learning ineffective for task offloading decision-making, even failing to converge in extreme cases. Thus, this section proposes a multi-agent reinforcement learning algorithm, centralized replay buffer-based deep recurrent Q-network (CRB-DRQN), enabling terminal devices and edge servers to make offloading decisions that satisfy the requirements of delay-sensitive tasks under unknown load conditions. Fig. 2 illustrates the overall execution process of the algorithm, involving three types of actors: a third-party trusted authority responsible for the collection and storage of experience, edge servers for network training and task computation, and terminal devices with

offloading decision capabilities. The system architecture adopts a fusion mode that enables cloud-based training and edge-based execution, laying a theoretical foundation for large-scale deployment. A single edge server manages multiple terminal devices. Edge servers are responsible for training neural networks but do not directly participate in task offloading decision-making. Instead, offloading decisions are determined and executed by terminal devices. Compared with centralized algorithms, when edge servers have sufficient computing resources, the proposed architecture can support a larger number of terminal devices. When the number of terminal devices is enormous, the computing capacity can be linearly expanded by increasing the number of edge servers.

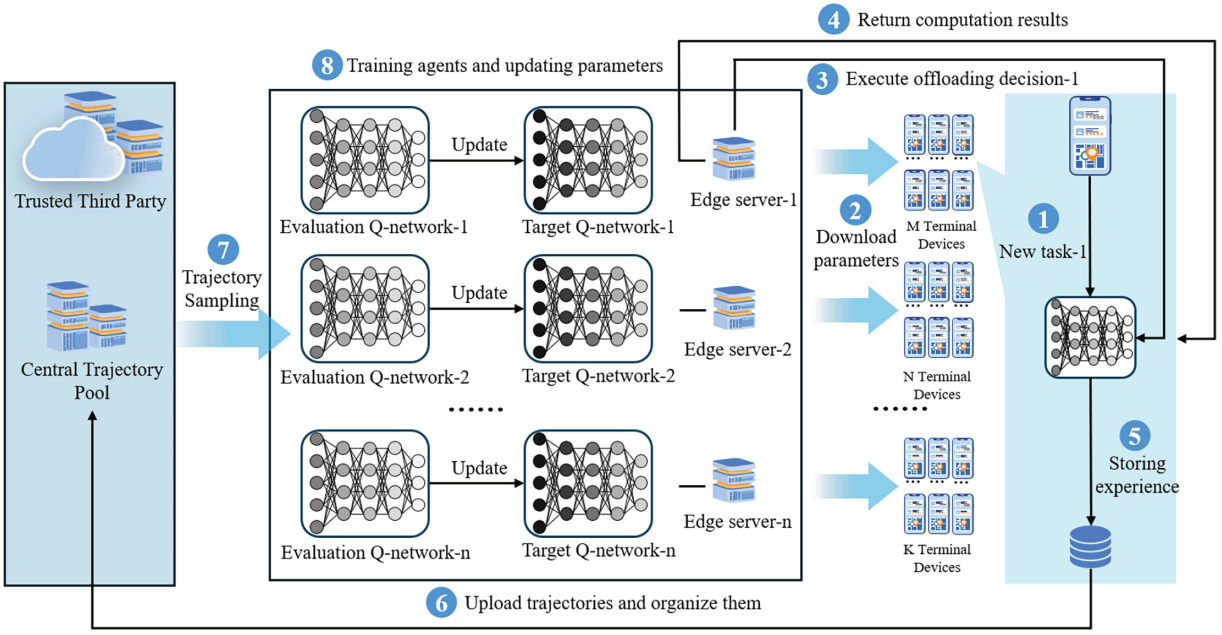


Figure 2: The overall execution process of CRB-DRQN.

To reduce the impact of changes in the MEC environment on agent training, this section uses an ensemble of all agents' experience trajectories to train the neural network of each agent. However, the agent's experience trajectory contains private data such as task information. Therefore, this section builds a third-party trusted authority in the cloud. Each agent uploads its own experience trajectory to the third-party trusted authority, which is responsible for integrating the experience trajectories to form a global experience trajectory, storing it in the central trajectory pool, and then training the corresponding neural network.

When a new task arrives, the terminal device requests the neural network weights of the Evaluation Q-network from the corresponding edge server. Based on the offloading decision generated by the neural network, the terminal device offloads the task to the designated edge server, and the edge server returns the task's calculation result to the terminal device. The terminal device collects experience samples and stores them in the experience replay buffer. When the number of experience samples stored in the experience replay buffer reaches a preset threshold, they form an experience trajectory. The terminal device then uploads this experience trajectory to the central trajectory pool of the third-party trusted authority, which integrates all uploaded experience trajectories. The edge server adopts similarity-based sampling to extract experience trajectories from the central trajectory pool, uses them to train the agent, and updates the network parameters accordingly.

4.1 POMDP Build

4.1.1 Status

This section assumes each edge server broadcasts its number of non-empty computation queues at the end of each time slot. The matrix $Load(t)$ represents each edge server's historical load levels over the previous C time slots. Let $Q_j(t)$ denote the load level of edge server j at time t; $Load(t)$ is a $C \times E$ matrix with elements defined as:

$$\{Load(t)\}_{i,j} = Q_j(t - C + i - 1) \tag{15}$$

At the beginning of time slot t, the terminal device d observes the state:

$$s_d(t) = \left(\eta_d(t), \mu_d^{comp}(t), \mu_d^{tran}(t), q_{d,e}(t-1), Load(t)\right) \tag{16}$$

4.1.2 Action

At the start of time slot t, if a new task arrives at terminal device d, d selects an action for this task. The action of terminal device d in time slot t is denoted as:

$$a_d(t) = (x_d(t), y_d(t)) \tag{17}$$

Let $Space$ denote the action space, where $Space = \{0,1\}^{1+E}$.

4.1.3 Rewards

This section uses negative rewards associated with task delays. If task $I_d(t)$ has been processed, the reward is:

$$R_d(t) = -delay_d\left(s_d(t), a_d(t)\right) \tag{18}$$

On the other hand, if task $I_d(t)$ has been discarded, then:

$$R_d(t) = Const \tag{19}$$

where $Const < 0$ is a negative constant.

4.2 Neural Network Structure

As shown in Fig. 2, the Target and Evaluation Q-networks share the same structure on edge servers. The Target Q-network stabilizes learning and computes target Q-values; the Evaluation Q-network handles policy evaluation and parameter updates.

The Evaluation Q-network calculates Q-values for all actions from the current state and selects the highest-value action for execution. This guides the agent's decision-making, enabling near-optimal action selection, interaction with the environment, and gradual learning of the optimal policy.

The structure of the Evaluation Q-network on terminal devices matches that on edge servers. The core goal of the neural network is to map each state to a set of Q-values corresponding to possible actions. This mapping process involves the collaborative work of multiple network layers.

State information is passed through the input layer to the neural network. The state information includes $\eta_d(t), \mu_d^{comp}(t), \mu_d^{tran}(t), q_{d,e}(t-1), Load(t)$, where $\eta_d(t), \mu_d^{comp}(t), \mu_d^{tran}(t), q_{d,e}(t-1)$ are passed to

the fully connected layer, and $Load(t)$ (for edge server load level prediction) is passed to the long short-term memory (LSTM) layer.

The LSTM layer learns the dynamics of edge server load levels and predicts near-future load levels using a LSTM unit. Hidden neurons in the LSTM are sequentially connected to matrix rows to track sequence changes (e.g., load level variations between time slots). Outputs from the last LSTM cell (indicating future load dynamics) are passed to the fully connected layer as $Load(t)_c$. Two such layers use ReLU-activated neurons, with connections between anterior/posterior layers for information transmission.

The Advantage-Value layer includes Advantage Network (A fully connected network learning action advantage values $A(s_d(t), a; \theta_a)$, where θ_a denotes network weights) and Value Network (A fully connected network learning state values $V(s_d(t); \theta_v)$, where θ_v denotes network weights).

These values are computed through neural network layers (input → Advantage-Value) and trained on the edge server. The output layer calculates the Q-value of state-action pairs via:

$$Q(s_d(t), a) = V(s_d(t); \theta_v) + \left[A(s_d(t), a; \theta_a) - \frac{1}{|Space|} \sum_{a' \in Space} A(s_d(t), a'; \theta_a) \right] \tag{20}$$

Here, the second term adjusts the advantage value relative to the average advantage of all actions in the state.

4.3 Familiarity Trajectory Sampling

4.3.1 Trajectory

This section defines an experience sample obtained by the terminal device at time t as: $(s_d(t), a_d(t), r_d(t), s_d(t+1))$. This tuple includes: Current state $s_d(t)$, Action $a_d(t)$ taken,Reward $r_d(t)$ received, Next state $s_d(t+1)$. It is stored in the replay buffer local to the terminal device.

Traditional DQN stores single-step experiences in a replay buffer. This section proposes a familiarity trajectory sampling algorithm that uses temporal sequences of multiple experiences as the agent's trajectory. When a terminal device stores an experience sample, it shares the trajectory starting at time t to a central trajectory pool (managed by a third-party entity). A trajectory is defined as:

$$Tra = \begin{Bmatrix} (s_d(t), a_d(t), r_d(t), s_d(t+1)), \\ (s_d(t+1), a_d(t+1), r_d(t+1), s_d(t+2)), \\ \vdots \\ (s_d(t+n), a_d(t+n), r_d(t+n), s_d(t+n+1)) \end{Bmatrix} \tag{21}$$

where n is the number of multi-step experiences (a constant).

Let $\mathcal{T} = \{Tra_1, Tra_2, \dots\}$ denote the set of all trajectories in the central pool. The central pool organizes trajectories uploaded by terminal devices into a replay buffer, from which edge servers sample during training. The size of $\mathcal{T}$ is denoted $|\mathcal{T}|$.

4.3.2 Familiarity Trajectory Sampling

Based on prior random sampling methods, this section proposes a familiarity trajectory sampling strategy. The core idea is: trajectories with fewer historical samples should have a higher re-sampling probability. Define N_i as the familiarity of trajectory Tra_i (increases with each sampling). As N_i grows, the probability of re-sampling decreases.

The sampling probability of trajectory Tra_i is:

$$P(i) = \frac{\left(\frac{\ln(N_i)}{N_i}\right)^{\frac{\alpha}{2}}}{\sum_{p=1}^{|\mathcal{T}|}\left(\frac{\ln(N_p)}{N_p}\right)^{\frac{\alpha}{2}}} \tag{22}$$

where $|\mathcal{T}|$ denotes the number of trajectories in the central pool, α is a hyperparameter that balances the importance of priority, N_i represents the familiarity (sampling count) of Tra_i, and $P(i)$ is the sampling probability of Tra_i.

4.4 Task Offloading Algorithm

First, initialize the Target Q-network, Evaluation Q-network, and counter $count_e$ of the edge server e; for the terminal device d, initialize its initial state $s_d(1)$ and trajectory counter n_d, and download the network weights θ_d of the Evaluation Q-network corresponding to the edge server e. When the time slot $t \in T$ starts, if a new task $I_d(t)$ arrives at the terminal device d, the terminal device d will select an action $a_d(t)$ according to the ϵ-greedy strategy and observe the state information $s_d(t+1)$ of the next time slot $t+1$. Meanwhile, it judges according to the trajectory storage condition: if the trajectory counter n_d satisfies $n_d \mod n = 0$ (where n is the trajectory length), the current trajectory batch $Tra_{d,i}$ is sent to the third-party trusted authority and the trajectory index i is incremented; otherwise, the state transition experience $(s_d(t), a_d(t), r_d(t), s_d(t+1))$ is appended to the local trajectory and the trajectory counter n_d is updated. In addition, if the current time slot t meets the periodic network weight download condition $(\mathrm{mod}\ (t, tag) == 0)$, the terminal device d will download the updated Evaluation Q-network weights θ_e from the associated edge server e. As shown in Fig. 3a and Algorithm 1.

Meanwhile, the edge server e extracts the trajectory set I from the central trajectory pool based on the familiarity sampling mechanism, updates the familiarity of each trajectory. For each experience sample $I_{j,k}$ (the k-th experience sample of the j-th trajectory in the trajectory set I) in each trajectory $Tra_j \in I$, the target Q-value $\hat{Q}_{e,I_{j,k}}^{Target}$ is calculated using Eq. (23). As shown in Fig. 3b and Algorithm 2.

$$\hat{Q}_{e,I_{j,k}}^{Target} = R_{j,k} + \gamma Q_e\left(s_{j,k+1}, a_{j,k}^{Next}, \theta_e^-\right) \tag{23}$$

The target Q-value $\hat{Q}_{n,I_{j,k}}^{Target}$ reveals the expected long-term reward of the action $a_{I_{j,k}}^{Next}$ given the state $s_{I_{j,k+1}}$, i.e., the sum of the reward in the experience sample $I_{j,k}$ and the Q-value of the action that may be selected under the given conditions. Among them, $a_{I_{j,k}}^{Next}$ can be calculated using Eq. (24).

$$a_{j,k}^{Next} = \arg\min_{a \in Space} Q_e\left(s_{j,k+1}, a, \theta_e\right) \tag{24}$$

Subsequently, the gradient descent method is used to minimize the loss function $L(\theta_e, \hat{Q}_e^{Target})$ and update the Evaluation Q-network parameters θ_e via Eq. (25), while incrementing its training counter $count_e$. When $count_e$ satisfies $count_e \mod update == 0$, the edge server synchronizes the Evaluation Q-network parameters θ_e to the Target Q-network parameters θ_e^-, completing the delayed update of the network parameters.

$$L(\theta_e, \hat{Q}_e^{Target}) = \frac{1}{n|I|}\sum_{I_{j,k} \in I}\left(\hat{Q}_{e,I_{j,k}}^{Target} - Q_e\left(s_{I_{j,k+1}}, a_{I_{j,k}}, \theta_e\right)\right)^2 \tag{25}$$

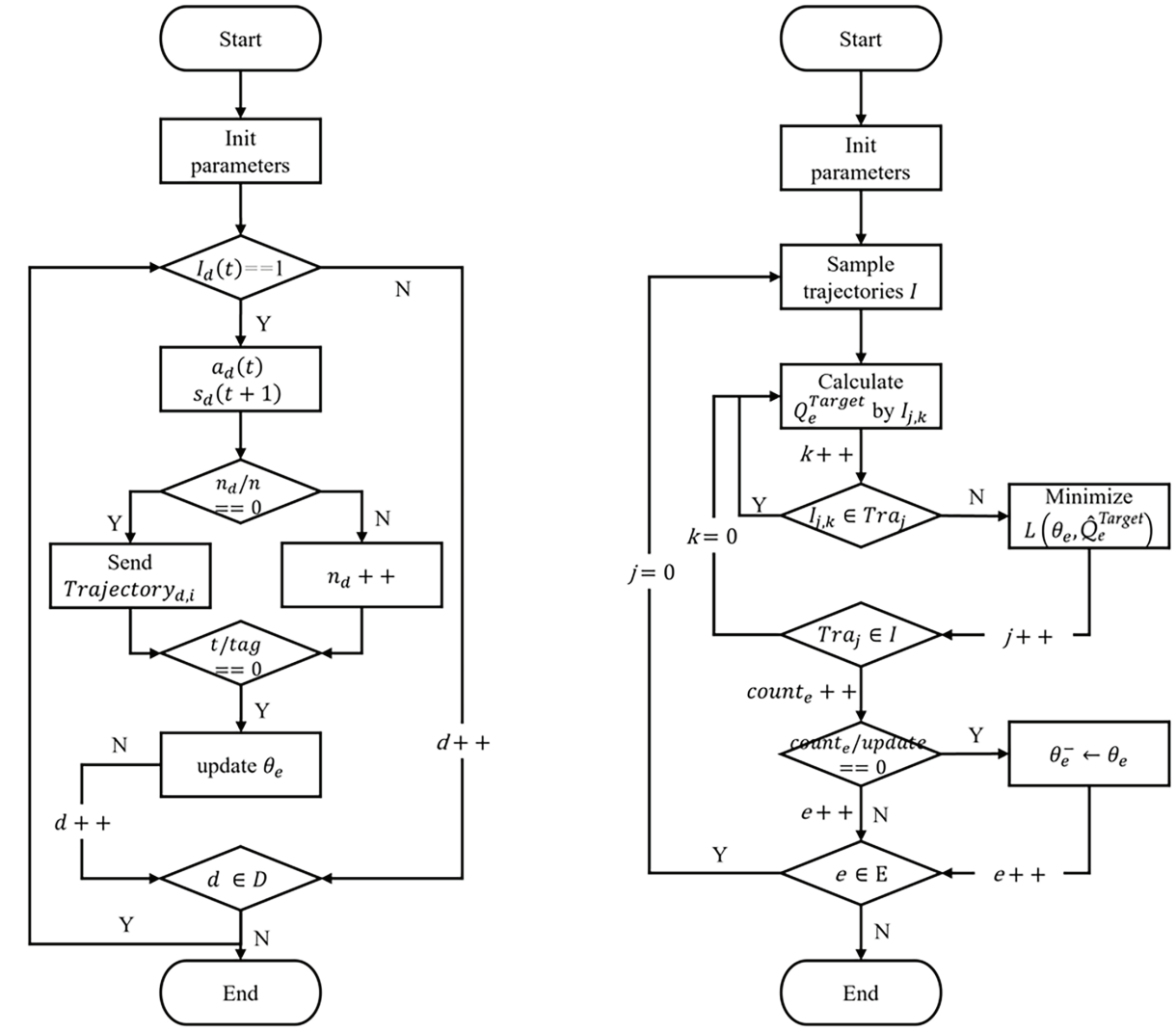


Figure 3: Flowchart of the CRB-DRQN Algorithm. Figure (**a**) illustrates the execution flow of the CRB-DRQN algorithm on terminal devices. Figure (**b**) illustrates the execution flow of the CRB-DRQN algorithm on edge devices.

Algorithm 1: Algorithm for terminal device d

Input: Task $I_d(t)$, state $s_d(t-1)$
Output: State $s_d(t+1)$

1: Terminal device downloads Evaluation Q-network weights, initializes $s_d(1)$, $n_d \leftarrow 0$
2: **if** $I_d(t)$==1 **then**
3: Terminal device selects action $a_d(t)$ via ϵ-greedy policy by $s_d(t-1)$
4: Observe state $s_d(t+1)$
5: **if** $mod(n_d, n) == 0$ **then**
6: Send to third-party trusted authority: $Tra_{d,i}$
7: **else**
8: $Tra_{d,i} \leftarrow Tra_{d,i} + (s_d(t), a_d(t), r_d(t), s_d(t))$

(Continued)

Algorithm 1 (continued)

$n_d \leftarrow n_d + 1$
end if
if $mod(t, tag) == 0$ **then**
  Terminal device downloads updated Evaluation Q-network weights: θ_e
end if
return State $s_d(t+1)$
end if

The time complexity of this algorithm is $O(T \times (|D| + |E| \times I \times K))$, where T denotes the total number of time steps, $|D|$ and $|E|$ represent the number of terminal devices and edge servers, respectively, I is the number of trajectories sampled in a single iteration, and K denotes the number of experience samples per trajectory. Its core consists of an initialization phase and a time-step main loop: The initialization requires traversing all edge servers $O(|E|)$ and terminal devices $O(|D|)$ to complete network parameter loading; in the main loop, each time step involves processing task action selection and trajectory recording for all terminal devices $O(|D|)$, as well as trajectory sampling and experience calculation $O(I \times K)$ for all edge servers $O(|E|)$. The space complexity is $O(|E| + |D|)$, which stems from the independent storage of Q-network parameters and counters by edge servers and terminal devices, with no dependence on task scale. The algorithm enables parallel processing of terminal devices and edge servers, reducing synchronization overhead and adapting to the asynchronous decision-making requirements of end-edge collaboration.

Algorithm 2: Algorithm for edge server e

Input: Trajectories I
Output: Network weights θ_e^-
for $e \in E$ **do**
  Initialization: Target Q-network, Evaluation Q-network, $count_e \leftarrow 0$
end for
for $Tra_j \in I$ **do**
  for $I_{j,k} \in Tra_j$ **do**
    Calculate Q_e^{Target} according to Eq. (23)
  end for
  Update to minimize $L(\theta_e, \hat{Q}_e^{Target})$ according to Eq. (25)
end for
$count_e \leftarrow count_e + 1$
if $mod(count_e, update) == 0$ **then**
  $\theta_e^- \leftarrow \theta_e$
end if
return Network weights θ_e^-

4.5 Convergence Analysis

Based on the theory of Bellman operator, this section conducts a rigorous theoretical analysis on the convergence of the CRB-DRQN algorithm in the heterogeneous cloud-edge-end environment. By transforming the POMDP into a belief MDP, defining the corresponding Bellman operator and proving its contractivity, the convergence characteristics of the algorithm are finally derived based on the Banach fixed-point theorem.

Let the true state of the system be $s \in \mathcal{S}$, the observation sequence of terminal device d at time slot t be $o_{1:t}$, and the action sequence be $a_{1:t-1}$. The belief state $b \in \mathcal{B}$ is defined as the probability distribution of the true state:

$$b(s) = \mathbb{P}(s|o_{1:t}, a_{1:t-1}) \tag{26}$$

The belief state space $\mathcal{B}$ is the set of all possible probability distributions. Based on the belief state, a belief MDP can be constructed as a 5-tuple $(\mathcal{B}, \mathcal{A}_d, \mathcal{T}_b, \mathcal{R}_b, \gamma)$, where $\mathcal{A}_d$ denotes the action space, $\mathcal{T}_b$ is the state transition function, $\mathcal{R}_b$ represents the reward function, and discount factor γ is 0.9.

Under the framework of belief MDP, the policy of terminal device d is defined as π_d:$\mathcal{B} \to \mathcal{A}_d$. The state-value function and action-value function corresponding to policy π_d are respectively given by:

$$V^{\pi_d}(b) = \mathbb{E}_{\pi_d}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | b_t = b\right] \tag{27}$$

$$Q^{\pi_d}(b, a) = \mathbb{E}_{\pi_d}[r_t + \gamma V^{\pi_d}(b_{t+1}) | b_t = b, a_t = a] \tag{28}$$

For the double Q-network architecture of the CRB-DRQN algorithm, define the Bellman expectation operator for the Evaluation Q-network as:

$$T^{\pi_d} Q(b, a) = \mathcal{R}_b(b, a) + \gamma \mathbb{E}_{o', b' \sim \mathcal{T}_b}[Q(b', \pi_d(b'))] \tag{29}$$

Define the Bellman optimality operator for the Target Q-network as follows:

$$T^* Q(b, a) = \mathcal{R}_b(b, a) + \gamma \mathbb{E}_{o', b' \sim \mathcal{T}_b}\left[\max_{a' \in \mathcal{A}_d} Q(b', a')\right] \tag{30}$$

To prove the convergence of the algorithm, it is necessary to verify that the aforementioned Bellman operator is a contraction mapping. Define the action-value function space Q as the set of all bounded functions, and its infinity norm is given by:

$$\| Q_1 - Q_2 \|_\infty = \max_{b \in \mathcal{B}, a \in \mathcal{A}_d} |Q_1(b, a) - Q_2(b, a)| \tag{31}$$

Theorem 1: *The Bellman expectation operator $\mathcal{T}^{\pi_d}$ is a γ-contraction mapping.*

Proof: For any $Q_1, Q_2 \in \mathcal{Q}$, the following holds:

$$\begin{aligned} |\mathcal{T}^{\pi_d} Q_1(b, a) - \mathcal{T}^{\pi_d} Q_2(b, a)| &= \gamma |\mathbb{E}[Q_1(b', \pi_d(b')) - Q_2(b', \pi_d(b'))]| \\ &\le \gamma \mathbb{E}[|Q_1(b', \pi_d(b')) - Q_2(b', \pi_d(b'))|] \\ &\le \gamma \| Q_1 - Q_2 \|_\infty \end{aligned} \tag{32}$$

Taking the supremum yields $\| \mathcal{T}^{\pi_d} Q_1 - \mathcal{T}^{\pi_d} Q_2 \|_\infty \le \gamma \| Q_1 - Q_2 \|_\infty$, and the proof is completed. □

Theorem 2: *The Bellman optimality operator $\mathcal{T}^*$ is a γ-contraction mapping.*

Proof: For any $Q_1, Q_2 \in \mathcal{Q}$, the following holds:

$$\begin{aligned} |\mathcal{T}^* Q_1(b,a) - \mathcal{T}^* Q_2(b,a)| &= \gamma \left| \mathbb{E}\left[\max_{a'} Q_1(b',a') - \max_{a'} Q_2(b',a') \right] \right| \\ &\leq \gamma \mathbb{E}\left[\max_{a'} |Q_1(b',a') - Q_2(b',a')| \right] \\ &\leq \gamma \parallel Q_1 - Q_2 \parallel_\infty \end{aligned} \tag{33}$$

Thus, $\mathcal{T}^*$ is also a γ-contraction mapping.

According to the Banach fixed-point theorem, a contraction mapping has a unique fixed point in a complete metric space. Therefore, the Bellman expectation operator $\mathcal{T}^{\pi_d}$ and the optimality operator $\mathcal{T}^*$ have unique fixed points Q^{π_d} and Q^*, respectively, which satisfy:

$$T^{\pi_d} Q^{\pi_d} = Q^{\pi_d}, \quad T^* Q^* = Q^* \tag{34}$$

Let the Q-function at the k-th iteration be Q_k, then the iteration error satisfies:

$$\parallel Q_k - Q^* \parallel_\infty \leq \gamma^k \parallel Q_0 - Q^* \parallel_\infty + \frac{\epsilon}{1-\gamma} \tag{35}$$

where ϵ denotes the function approximation error. As $k \to \infty$, $\gamma^k \to 0$, and the algorithm converges. □

4.6 Discussion on Practical Issues

First, in the task offloading problem, traditional methods often rely on global state information, resulting in high communication costs. This paper adopts a partially observable modeling approach, enabling terminal devices to make decisions based solely on local observations. This design reduces the frequency of state transmission.

Second, the terminal devices in this study only need to obtain minimal local state information, reducing the risk of privacy leakage from the source.

Finally, the system architecture of this paper adopts a "centralized training-distributed execution" mode, which provides a theoretical basis for large-scale deployment. This modular design allows the system to linearly expand its processing capacity by adding edge nodes.

5 Performance Evaluation

This section details the experimental setup and results of the algorithm proposed in this section in a simulation environment. The evaluation metrics include average task delay, task failure rate under varying terminal device counts and task arrival rates, as well as the reward curve of each algorithm. Simulations are conducted on a desktop equipped with an AMD Ryzen 7 5700X 8-Core Processor, an NVIDIA GeForce RTX 3070Ti GPU, and 32 GB RAM.

5.1 Experimental Parameter Settings

A simulation scenario with four edge servers is built. Specific parameters are in Table 1. Neural network settings: batch size = 4, learning rate = 0.001, discount factor = 0.9, random exploration probability decays from 1 to 0.01. The root mean square propagation (RMSPROP) optimizer is used. The configuration of experimental parameters is based on the typical characteristics of real-world edge computing environments and research requirements. First, the time slot duration δ is set to 100 milliseconds to match the fine-grained time unit for task scheduling in edge computing, thereby avoiding delay inaccuracies caused by excessive

discretization. The CPU frequency of terminal devices (f_d = 2.5 GHz) and the frequency of edge servers (f_e = 41.8 GHz) refer to the common configurations in [27], which reflects the heterogeneity of resources. The transmission rate $O_{d,e}^{tran}$ = 14 Mbps corresponds to the bandwidth of a typical wireless network, while the processing density $\rho_{d,t}$ = 500 is determined based on the average number of CPU cycles required per unit of data, so as to balance the computational load. The delay constraint $\tau_{d,t}$ = 12 time slots is tailored for delay-sensitive applications, ensuring that tasks are completed within a reasonable time. Overall, the selection of parameters aims to simulate dynamic multi-access scenarios, verify the robustness of the algorithm under resource constraints, and provide a consistent benchmark for comparative experiments.

Table 1: Parameter settings.

Parameters	Value
δ	100 ms
f_d	2.5 GHz [27]
f_e	41.8 GHz [27]
$O_{d,e}^{tran}$	14 Mbps
$\rho_{d,t}$	500
$\tau_{d,t}$	12 time slots
n	4
$Const$	$-3\tau_{d,t}$

5.2 Introduction to Experimental Comparison Methods

The proposed task offloading algorithm (CRB-DRQN) is compared with the following benchmark algorithms:

(1) Decentralized algorithm: A decentralized actor-critic algorithm which contrasts with the centralized training/distributed execution framework of the proposed algorithm. Agents update networks independently using local observations, with no inter-agent information sharing.

(2) Local-Only strategy: Forces all agents to process computational tasks locally, omitting any offloading decision-making.

(3) Q-value Mixing (QMIX) algorithm [28]: A value-learning-based multi-agent reinforcement learning algorithm for solving cooperative task interaction/reward distribution challenges. It uses cross-entropy optimization for action selection, which is adapted to continuous action spaces. Implementation follows literature [29].

(4) Multi-Agent Proximal Policy Optimization (MAPPO) algorithm [30]: An actor-critic algorithm for multi-agent scenarios, optimized for continuous action spaces and massively parallel computing environments.

(5) Joint Optimal DRL Algorithm with Privacy Preservation (JODRL-PP) [5]: An actor-critic-based task offloading algorithm. It addresses incomplete multi-user information sharing via a third-party centralized training network, enabling effective offloading strategy learning.

(6) Multi-Tier offloading model [31] based on game theory principles. It addresses the limitations of edge server computational resources and inefficient resource allocation in vehicular networks by dividing the offloading process into two stages.

5.3 Evaluation of Experimental Results

5.3.1 Convergence of the Algorithm

Fig. 4 intuitively demonstrates the fast convergence property of the CRB-DRQN algorithm by comparing the average reward evolution curves of different algorithms. Experimental data show that the CRB-DRQN algorithm reaches steady-state convergence after approximately 170 training episodes, with the final average reward stabilizing in the −0.5 range-significantly outperforming the other benchmark algorithms (e.g., the JODRL-PP algorithm requires 410 episodes to converge and exhibits reward jump phenomena). This efficient convergence stems from the core design of the algorithm: first, the Double DQN and Target network mechanism effectively suppress the overestimation of Q-values by decoupling action selection and value evaluation; second, the centralized training architecture integrates global experience trajectories through a third-party trusted authority, enabling agents to obtain an approximate global perspective in partially observable environments and avoiding policy oscillations caused by environmental non-stationarity in decentralized algorithms. In addition, the similarity-based trajectory sampling mechanism further optimizes exploration efficiency by dynamically adjusting the sampling probability of high-value experiences, promoting the smooth approximation of the Q-function to the optimal solution.

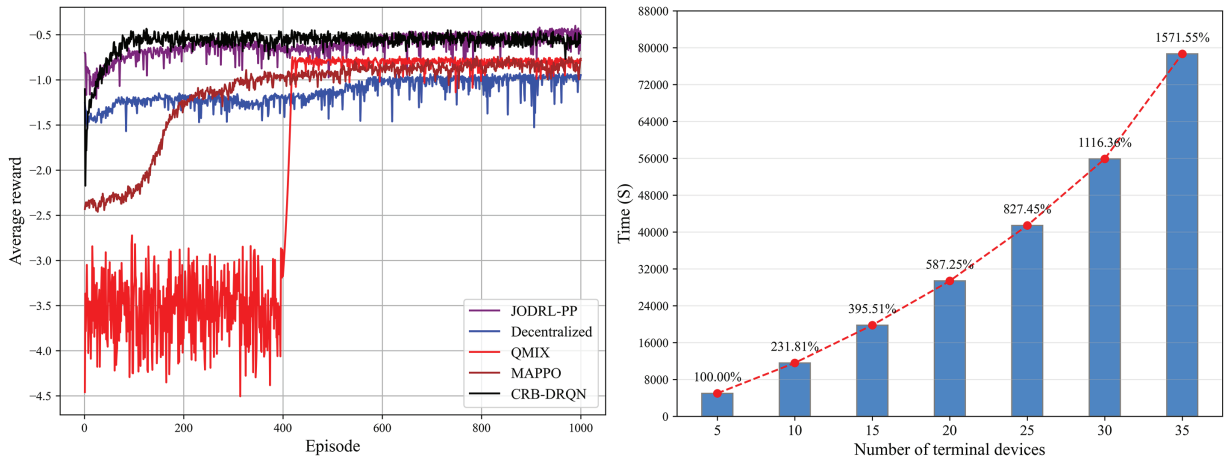
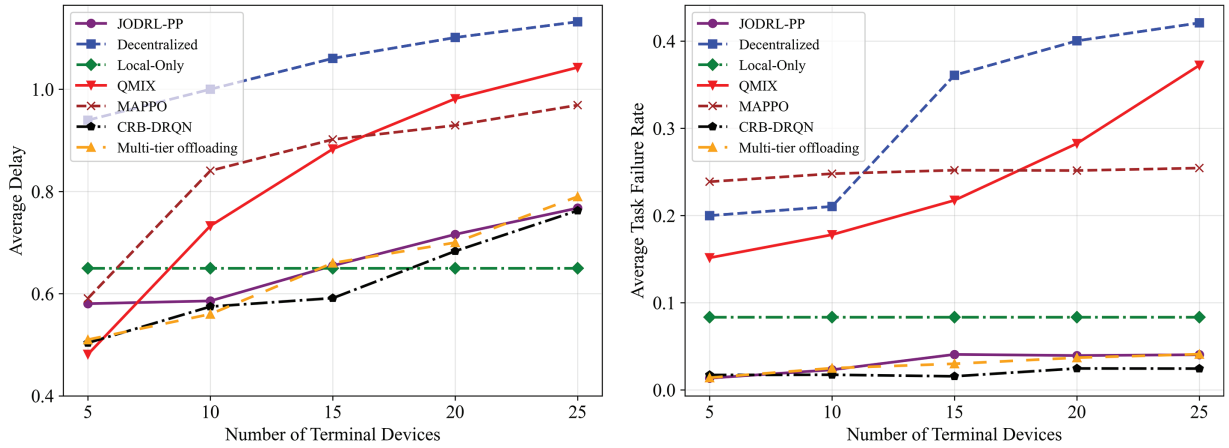


Figure 4: Average rewards of different algorithms and Convergence time of CRB-DRQN algorithm.

Fig. 4 also illustrates the variation trend of the convergence time of single agent under different terminal device scales, and its linear growth pattern verifies the algorithm's stability in dynamic scaling scenarios. When the CRB-DRQN algorithm faces the expansion of large-scale terminal devices and edge server clusters, its distributed execution architecture ensures system scalability through a three-layer mechanism. First, the "centralized training-distributed execution" framework enables terminal devices to make independent real-time decisions based on local observed states, while edge servers achieve dynamic allocation of computational loads through asynchronous training and parameter updates, effectively avoiding single-point bottlenecks. Specifically, when the number of terminal devices increases from 5 to 35, the convergence time only rises approximately linearly, indicating that the algorithm has strong robustness to scale expansion. When the algorithm is scaled to larger-scale scenarios, the computational pressure can be dispersed by adding edge nodes, and the processing capacity can be expanded linearly.

5.3.2 Comparison of Task Delay and Failure Rate under Different Terminal Device Scales

Fig. 5a shows the comparison results of the average latency of each algorithm task at different terminal device scales. Although the average latency of all algorithms is controlled within the latency constraints, it should be noted that this metric only counts successfully processed tasks and does not include discarded tasks due to insufficient resources. Except for the Local-Only algorithm, the average computing latency of the other algorithms increases monotonically with the expansion of the terminal device. Compute latency is related to the computing resources and load of the device. As the number of terminal devices increases, the number of tasks that each edge server needs to process increase, which directly leads to an increase in queue wait times, resulting in an increase in the average latency of computing. The CRB-DRQN algorithm maintains a low latency level under different terminal device scales. When the number of terminal devices is 5, the latency (0.50 s) is 46.8%, 15.3%, and 13.8% lower than that of the Decentralized algorithm (0.94 s), the MAPPO algorithm (0.59 s), and the JODRL-PP algorithm (0.58 s), respectively, slightly higher than the QMIX algorithm (0.48 s). It is worth noting that the latency gap between the JODRL-PP algorithm and the CRB-DRQN algorithm gradually decreases with the increase of device scale, and when the number of terminal devices reaches 25, the latency value of the two algorithms tends to be the same level (CRB-DRQN algorithm: 0.762 s, JODRL-PP algorithm: 0.767 s). This phenomenon can be attributed to the performance optimization of the CRB-DRQN algorithm through the following mechanisms: firstly, by improving the neural network, it has better load awareness ability in dynamically changing scenarios, and secondly, through the global experience sharing mechanism realized by the central trajectory pool, so that each agent can learn the optimal offloading strategy collaboratively, effectively reducing the local decision bias.



(a) Comparison of Average Delay with Different Terminal Numbers

(b) Comparison of Task Failure Rate with Different Terminal Numbers

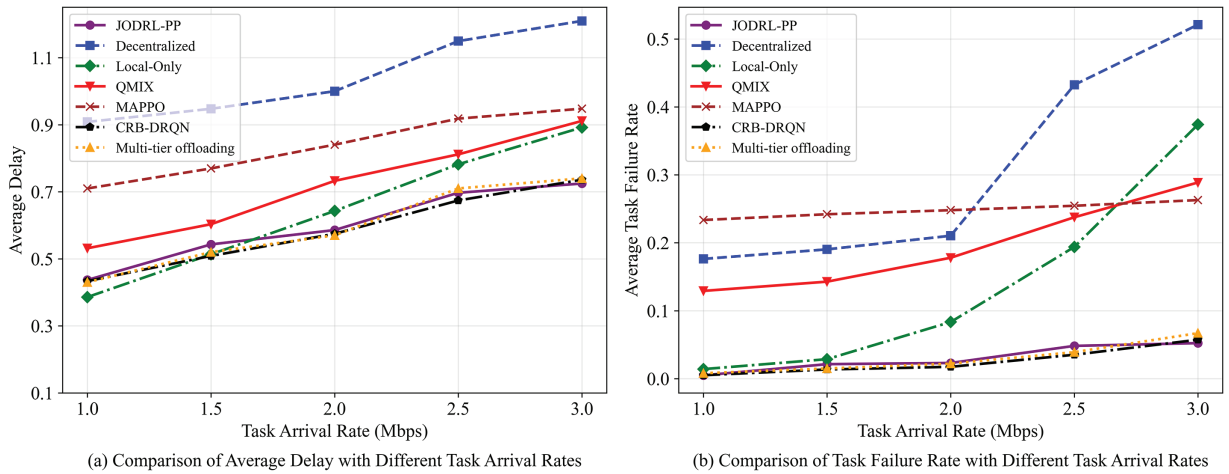
Figure 5: Task delay and failure rate under varying scales of terminal devices. Figure (**a**) shows the comparison results of the average latency of each algorithm task at different terminal device scales. Figure (**b**) presents how terminal device scale affects task failure rates.

Fig. 5b presents how terminal device scale affects task failure rates. Experimental data indicate that at 25 devices, task failure rates diverge significantly: Local-Only fails to handle high concurrency due to limited local resources. The Decentralized and QMIX algorithms see linear failure rate growth with scale expansion, reaching 0.42 and 0.37 respectively at 25 devices. MAPPO shows a conservative strategy, with its failure rate stably within 0.24 ± 0.02. Conversely, JODRL-PP and Multi-tier offloading control the failure rate at 0.04 for 25 devices. The proposed CRB-DRQN algorithm achieves global state awareness via the central trajectory pool, stabilizing the final task failure rate at 0.017%–57.5% lower than JODRL-PP and the Multi-tier offloading strategy. This validates the empirical playback mechanism's role in enhancing system reliability.

5.3.3 Comparison of Task Delay and Failure Rate under Different Task Scales

This section compares the convergence of the algorithm, the task delay and failure rate at different terminal device scales, and the task delay and failure rate at different task scales.

Fig. 6a presents the average latency across different task sizes. As the task arrival rate rises from 1.0 to 3.0 Mbps, all algorithms' average latencies increase. At 1.0 Mbps, our CRB-DRQN algorithm (0.43 s) outperforms Decentralized (0.90 s), QMIX (0.53 s), and MAPPO (0.71 s), matching JODRL-PP (0.43 s) and Multi-tier offloading (0.42 s) but slightly higher than Local-Only (0.38 s). When the rate hits 3.0 Mbps, Local-Only latency spikes to 0.89 s (and keeps rising) due to terminal devices' local computing resource constraints. Notably, Local-Only still outperforms Decentralized, QMIX, and MAPPO, whose longer waiting times create efficiency bottlenecks in their offloading strategies. CRB-DRQN maintains the lowest latency between 1.5–2.5 Mbps. At 3.0 Mbps, its 0.73 s latency is marginally higher than JODRL-PP's 0.72 s and Multi-tier offloading 0.72 s, revealing CRB-DRQN's optimization boundary under high load.



(a) Comparison of Average Delay with Different Task Arrival Rates

(b) Comparison of Task Failure Rate with Different Task Arrival Rates

Figure 6: Task delay and failure rate under varying task scales. Figure (**a**) presents the average latency across different task sizes. Figure (**b**) compares task failure rates under different task sizes.

Fig. 6b compares task failure rates under different task sizes. Experimental data indicate that as task arrival rate rises from 1.0 to 3.0 Mbps, failure rates of the Decentralized and Local-Only algorithms soar from 0.18, 0.014 to 0.52, 0.37, respectively. The former suffers from environmental instability, hampering effective strategy execution; the latter is limited by insufficient local computing power to meet demands. The JODRL-PP, Multi-tier offloading, and CRB-DRQN algorithms exhibit strong reliability and robustness, with maximum failure rates of 0.052 (JODRL-PP, 3.0 Mbps), 0.051 (Multi-tier offloading, 3.0 Mbps) and 0.057 (CRB-DRQN, 3.0 Mbps). Within 1.0–2.5 Mbps task arrival rate, CRB-DRQN outperforms others; yet at 3.0 Mbps, its failure rate (0.057) marginally exceeds JODRL-PP's 0.052 and Multi-tier offloading's 0.056. Fig. 6 suggests CRB-DRQN still has optimization potential under high load, possibly due to mismatches between neural network-perceived and actual load levels.

6 Conclusion

To address core challenges in MEC environments-including a large number of terminal devices and frequent dynamic changes in system states-this section proposes a end-edge collaborative task offloading method based on POMDP. A three-layer offloading framework encompassing edge computing resources, terminal devices, and third-party trusted authority is constructed to solve the policy mismatch problem

caused by environmental dynamics in traditional algorithms. At the modeling level, a slotted transmission and computation model is innovatively introduced to quantify task processing delay and queue dynamics, and a partially observable state space is built based on edge server load history to reduce system information interaction overhead.

The proposed CRB-DRQN algorithm adopts a hybrid architecture of centralized experience sharing and distributed policy execution: through a central trajectory pool constructed by third-party trusted authority, it achieves safe integration and efficient reuse of cross-agent experience trajectories; a familiarity trajectory sampling mechanism is designed to optimize the distribution of training samples, effectively alleviating the policy lag problem of traditional experience replay mechanisms in dynamic environments; the neural network structure designed based on DDQN and LSTM enhances the prediction ability of edge server load fluctuations, enabling agents to make accurate offloading decisions under partially observable conditions.

Experimental results show that CRB-DRQN exhibits significant advantages in scenarios with expanded terminal device scales and dynamic changes in task loads, providing a new solution for task offloading in MEC environments and effectively improving system reliability and offloading efficiency.

In real-world environments, the dynamic changes of tasks and resources are often more frequent and complex. The performance of the algorithm can be evaluated in a demonstration system, thereby identifying and resolving more potential issues. Additionally, game theory can be combined with multi-agent reinforcement learning techniques to further explore the mechanism of strategic interaction among multiple devices.

Acknowledgement: Not applicable.

Funding Statement: This work was funded by the State Grid Corporation Science and Technology Project "Research and Application of Key Technologies for Integrated Sensing and Computing for Intelligent Operation of Power Grid"(Grant No. 5700-202318596A-3-2-ZN).

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Wang Luo and Bao Feng; methodology, Wang Luo and Jinwei Mao; software, Jinwei Mao, Shuang Yang, Daohua Zhu and Wei Liang; validation, Shuang Yang, Jiangtao Xu and Zhechen Huang; formal analysis, Jinwei Mao and Bao Feng; investigation, Shuang Yang and Jiangtao Xu; resources, Daohua Zhu and Wang Luo; data curation, Jiangtao Xu, Zhechen Huang and Wei Liang; writing—original draft preparation, Jinwei Mao, Wang Luo, Shuang Yang, Daohua Zhu and Zhechen Huang; writing—review and editing, Bao Feng, Jiangtao Xu, Zhechen Huang and Wei Liang; visualization, Jinwei Mao, Bao Feng, Zhechen Huang and Wei Liang; supervision, Wang Luo and Daohua Zhu; project administration, Wang Luo, Jiangtao Xu and Daohua Zhu; funding acquisition, Wang Luo and Daohua Zhu. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data and materials are not publicly available.

Ethics Approval: Not applicable.

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

References

1. Zhang C, Yang J. Multi-tree genetic programming for adaptive dynamic fault-tolerant task scheduling of satellite edge computing. Future Gener Comput Syst. 2026;175:108099. doi:10.1016/j.future.2025.108099.
2. Ranaweera P, Jurcut AD, Liyanage M. Survey on multi-access edge computing security and privacy. IEEE Commun Surv Tutor. 2021;23(2):1078–124. doi:10.1109/comst.2021.3062546.

3. Tang X, Li X, Yu R, Wu Y, Ye J, Tang F, et al. Digital-twin-assisted task assignment in multi-UAV systems: a deep reinforcement learning approach. IEEE Internet Things J. 2023;10(17):15362–75. doi:10.1109/jiot.2023.3263574.
4. Lou J, Tang Z, Jia W, Zhao W, Li J. Startup-aware dependent task scheduling with bandwidth constraints in edge computing. IEEE Trans Mob Comput. 2023;23(2):1586–600. doi:10.1109/tmc.2023.3238868.
5. Wu G, Chen X, Gao Z, Zhang H, Yu S, Shen S. Privacy-preserving offloading scheme in multi-access mobile edge computing based on MADRL. J Parallel Distrib Comput. 2024;183:104775. doi:10.1016/j.jpdc.2023.104775.
6. Sun F, Zhang Z, Chang X, Zhu K. Toward heterogeneous environment: lyapunov-orientated imphetero reinforcement learning for task offloading. IEEE Trans Netw Serv Manag. 2023;20(2):1572–86. doi:10.1109/tnsm.2023.3266779.
7. Jiang X, Yu FR, Song T, Leung VC. A survey on multi-access edge computing applied to video streaming: some research issues and challenges. IEEE Commun Surv Tutor. 2021;23(2):871–903. doi:10.1109/comst.2021.3065237.
8. Xu X, Liu K, Dai P, Jin F, Ren H, Zhan C, et al. Joint task offloading and resource optimization in NOMA-based vehicular edge computing: a game-theoretic DRL approach. J Syst Archit. 2023;134(1):102780. doi:10.1016/j.sysarc.2022.102780.
9. Zhang C, Yang J. An energy-efficient collaborative offloading scheme with heterogeneous tasks for satellite edge computing. IEEE Trans Netw Sci Eng. 2024;11(6):6396–407. doi:10.1109/tnse.2024.3476968.
10. Huang X, Huang G. Joint optimization of energy and task scheduling in wireless-powered irs-assisted mobile-edge computing systems. IEEE Internet Things J. 2023;10(12):10997–13. doi:10.1109/jiot.2023.3242951.
11. Deng Y, Chen X, Zhu G, Fang Y, Chen Z, Deng X. Actions at the edge: jointly optimizing the resources in multi-access edge computing. IEEE Wirel Commun. 2022;29(2):192–8. doi:10.1109/mwc.006.2100699.
12. Porambage P, Okwuibe J, Liyanage M, Ylianttila M, Taleb T. Survey on multi-access edge computing for internet of things realization. IEEE Commun Surv Tutor. 2018;20(4):2961–91. doi:10.1109/comst.2018.2849509.
13. Liu J, Yang P, Chen C. Intelligent energy-efficient scheduling with ant colony techniques for heterogeneous edge computing. J Parall Distrib Comput. 2023;172:84–96. doi:10.1016/j.jpdc.2022.10.003.
14. Deng X, Yin J, Guan P, Xiong NN, Zhang L, Mumtaz S. Intelligent delay-aware partial computing task offloading for multiuser industrial Internet of Things through edge computing. IEEE Internet Things J. 2021;10(4):2954–66. doi:10.1109/jiot.2021.3123406.
15. Li Y, Liu B, Wu E, Li J, Zhou Z, Zhang W. DRA-MQoS: an MQoS scheduling algorithm based on resource feature matching in federated edge cloud. Concurr Comput Pract Exp. 2023;35(2):e7478. doi:10.1002/cpe.7478.
16. Liu Q, Zeng L, Bilal M, Song H, Liu X, Zhang Y, et al. A multi-swarm PSO approach to large-scale task scheduling in a sustainable supply chain datacenter. IEEE Trans Green Commun Netw. 2023;7(4):1667–77. doi:10.21203/rs.3.rs-1844078/v1.
17. Gao A, Zhang S, Zhang Q, Hu Y, Liu S, Liang W, et al. Task offloading and energy optimization in hybrid UAV-assisted mobile edge computing systems. IEEE Trans Veh Technol. 2024;73(8):12052–66. doi:10.1109/tvt.2024.3380003.
18. Cai J, Liu W, Huang Z, Yu FR. Task decomposition and hierarchical scheduling for collaborative cloud-edge-end computing. IEEE Trans Serv Comput. 2024;17(6):4368–82. doi:10.1109/tsc.2024.3402169.
19. Chen W, Wei X, Chi K, Yu K, Tolba A, Mumtaz S, et al. Computation time minimized offloading in NOMA-enabled wireless powered mobile edge computing. IEEE Trans Commun. 2024;72(11):7182–97. doi:10.1109/tcomm.2024.3405316.
20. Fan W, Zhang Y, Zhou G, Liu Y. Deep reinforcement learning-based task offloading for vehicular edge computing with flexible RSU-RSU cooperation. IEEE Trans Intell Transp Syst. 2024;25(7):7712–25. doi:10.1109/tits.2024.3349546.
21. Guo H, Chen X, Zhou X, Liu J. Trusted and efficient task offloading in vehicular edge computing networks. IEEE Trans Cogn Commun Netw. 2024;10(6):2370–82. doi:10.1109/globecom48099.2022.10000816.
22. Li K, Wang X, He Q, Wang J, Li J, Zhan S, et al. Computation offloading in resource-constrained multi-access edge computing. IEEE Trans Mob Comput. 2024;23(11):10665–77. doi:10.1109/tmc.2024.3383041.

23. Li L, Qiu Q, Xiao Z, Lin Q, Gu J, Ming Z. A two-stage hybrid multi-objective optimization evolutionary algorithm for computing offloading in sustainable edge computing. IEEE Trans Consum Electron. 2024;70(1):735–46. doi:10.1109/tce.2024.3376930.
24. Lin H, Yang L, Guo H, Cao J. Decentralized task offloading in edge computing: an offline-to-online reinforcement learning approach. IEEE Trans Comput. 2024;73(6):1603–15. doi:10.1109/tc.2024.3377912.
25. Liu Q, Zhang H, Zhang X, Yuan D. Improved DDPG based two-timescale multi-dimensional resource allocation for multi-access edge computing networks. IEEE Trans Veh Technol. 2024;73(6):9153–8. doi:10.1109/tvt.2024.3360943.
26. Liu P, An K, Lei J, Sun Y, Liu W, Chatzinotas S. Computation rate maximization for SCMA-aided edge computing in IoT networks: a multi-agent reinforcement learning approach. IEEE Trans Wirel Commun. 2024;23(8):10414–29. doi:10.1109/twc.2024.3371791.
27. Neto JLD, Yu SY, Macedo DF, Nogueira JMS, Langar R, Secci S. ULOOF: a user level online offloading framework for mobile edge computing. IEEE Trans Mob Comput. 2018;17(11):2660–74. doi:10.1109/tmc.2018.2815015.
28. Han C, Yao H, Mai T, Zhang N, Guizani M. QMIX aided routing in social-based delay-tolerant networks. IEEE Trans Veh Technol. 2021;71(2):1952–63. doi:10.1109/tvt.2021.3133449.
29. Peng B, Rashid T, Schroeder de Witt C, Kamienny PA, Torr P, Boehmer W, et al. FACMAC: factored multi-agent centralised policy gradients. Adv Neural Inf Process Syst. 2021;34:12208–21.
30. Yu C, Velu A, Vinitsky E, Gao J, Wang Y, Bayen A, et al. The surprising effectiveness of PPO in cooperative multi-agent games. Adv Neural Inf Process Syst. 2022;35:24611–24.
31. Lin H, Xiao B, Zhou X, Zhang Y, Liu X. A multi-tier offloading optimization strategy for consumer electronics in vehicular edge computing. IEEE Trans Consum Electron. 2025;71(1):2118–30. doi:10.1109/tce.2025.3527043.