A Load-Aware Task Offloading Method for Mobile Edge Computing Under Eligibility Constraints

YaRong Liu , ZiJian Che , XiaoLan Xie *

Posted Date: 11 May 2026

doi: 10.20944/preprints202605.0671.v1

Keywords: mobile edge computing; task offloading; deep reinforcement learning; eligibility constraints; heterogeneous tasks; load-aware coordination

 doi:10.20944/preprints202605.0671.v1

Article

A Load-Aware Task Offloading Method for Mobile Edge Computing Under Eligibility Constraints

YaRong Liu [1,2], ZiJian Che [1] and XiaoLan Xie [1,2,*]

[1] School of Computer Science and Engineering, Guilin University of Technology, Guilin 541006, China

[2] Guangxi Key Laboratory of Embedded Technology and Intelligent System, Guilin University of Technology, Guilin 541006, China

* Correspondence: 237290696@qq.com

Abstract

Mobile edge computing (MEC) enables computation-intensive and latency-sensitive tasks to be offloaded from mobile devices to nearby edge servers. Most existing MEC task offloading studies formulate offloading as a selection problem over a fixed or fully available set of candidate servers, which is restrictive in heterogeneous MEC scenarios with task-node eligibility constraints. Under such constraints, a task can be processed by an edge server only when task attributes, service requirements, link conditions, and node states jointly satisfy the corresponding eligibility conditions. The feasible action set therefore varies over time, while offloading decisions are further coupled with local queueing competition and long-term load evolution. To address this problem, this paper proposes RoSCo, a load-aware task offloading method with scheduling and constraint coordination for eligibility-constrained MEC systems. RoSCo constructs a dynamic feasible action set, applies eligibility-aware action masking to exclude infeasible offloading actions, introduces priority-driven local coordination to characterize service competition among heterogeneous tasks, and designs a load-responsive reward to guide congestion mitigation and load balancing. The offloading policy is learned using a dueling double deep Q-network (D3QN). Simulation results show that RoSCo reduces task drop rate and edge-node load imbalance while maintaining competitive task completion delay and energy consumption, especially under high-load and sparse-eligibility conditions.

Keywords: mobile edge computing; task offloading; deep reinforcement learning; eligibility constraints; heterogeneous tasks; load-aware coordination

1. Introduction

With the rapid development of the Internet of Things, mobile intelligent applications, and edge-assisted services, a growing number of mobile computing tasks are generated by resource-constrained mobile devices. These tasks are often computation-intensive and latency-sensitive, such as real-time inference, online decision-making, interactive sensing, and environment perception. Processing all tasks locally is usually inefficient because mobile devices are limited in computing capability, battery capacity, and thermal dissipation. Mobile edge computing (MEC) addresses this limitation by deploying computing resources at edge servers close to end users, so that computation tasks can be offloaded from mobile devices to nearby edge servers for low-latency execution and improved service quality [1,2].

Task offloading is a fundamental problem in MEC systems. Existing studies have mainly formulated MEC task offloading as an optimization or learning-based decision problem, where each task selects a local or edge execution mode according to delay, energy consumption, resource allocation, and system utility objectives [3,4]. With the increase in task arrival dynamics, channel variation, and resource competition, deep reinforcement learning has been widely introduced to learn online offloading policies under time-varying MEC environments [5,6]. These methods have

 doi:10.20944/preprints202605.0671.v1

improved the adaptability of offloading decisions, but many of them still rely on a simplified assumption that tasks can be assigned over a fixed or fully available candidate edge-server set.

This assumption is restrictive in heterogeneous MEC scenarios. In practical edge-service environments, whether a task can be processed by a specific edge server is not determined only by computation capacity or transmission rate. It may also depend on task attributes, service requirements, software or model availability, link conditions, node state, and current queue status. Therefore, different tasks may correspond to different eligible edge servers. A node that provides a low execution cost for one task may be infeasible for another task if the required service capability or execution condition is not satisfied. If such task-node eligibility constraints are ignored, the offloading policy may be trained over an action space that contains infeasible offloading actions, which weakens policy learning and leads to decisions that cannot be executed in the actual MEC system [7,8].

The difficulty is not limited to action feasibility. Under continuous arrivals of heterogeneous tasks, eligibility constraints reshape both the candidate-node set and the subsequent service competition at edge servers. When multiple tasks are concentrated on a limited subset of eligible nodes, local queueing competition becomes more pronounced. The execution delay of a task is then affected not only by its own offloading decision, but also by the service order, priority relationship, and workload accumulation at the selected edge node. Meanwhile, the current offloading decision changes node queues and affects future load distribution. Therefore, eligibility-constrained MEC offloading is coupled with dynamic feasible action sets, local queueing competition, and long-term load evolution [9,10].

Existing reinforcement learning methods do not fully resolve this coupling. Some studies focus on learning offloading policies under dynamic network states, but the feasible action set is often treated as fixed or only weakly constrained. Others consider resource allocation or queue-aware scheduling, but they usually pay limited attention to task-node eligibility and the resulting variation of the candidate action space. As a result, policy learning may still waste exploration on infeasible actions, underestimate local service competition among heterogeneous tasks, or optimize short-term execution cost without sufficiently accounting for load accumulation over time [11,12]. These limitations indicate that eligibility-constrained MEC offloading requires a policy design that jointly considers action feasibility, node-side service coordination, and long-term load regulation.

To address this problem, this paper investigates task offloading in a heterogeneous MEC system under dynamic task-node eligibility constraints. Unlike conventional task offloading studies that mainly optimize task assignment over a common candidate-node set, this paper explicitly models the eligibility relationship between task attributes and edge-server capabilities. On this basis, a coordinated task offloading method, termed RoSCo, is proposed to integrate eligibility-aware action control, priority-driven local coordination, and load-responsive reward design into a unified deep reinforcement learning framework. The purpose is not only to avoid infeasible offloading actions, but also to improve the coordination between current offloading decisions, local service competition, and long-term load balance.

The main contributions of this paper are summarized as follows:

(1) An eligibility-constrained MEC task offloading model is constructed for heterogeneous tasks. The model describes task-node eligibility relationships according to task attributes, service requirements, link conditions, and node states, so that the dynamic feasible action set can be explicitly represented during offloading decision-making.

(2) A coordinated task offloading method, RoSCo, is proposed. The method combines eligibility-aware action masking, priority-driven local coordination, and load-responsive reward design. The action masking mechanism excludes infeasible offloading actions during policy learning; the local coordination mechanism characterizes service competition among heterogeneous tasks at edge servers; and the load-responsive reward guides the policy to suppress persistent congestion and improve load balance.

(3) A dueling double deep Q-network (D3QN)-based learning procedure is developed for eligibility-constrained task offloading. The feasible action set is considered in both action selection

 doi:10.20944/preprints202605.0671.v1

and target-action evaluation, so that policy learning remains consistent with the dynamic action constraints of the MEC environment.

(4) Simulation experiments are conducted under varying task arrival rates and mobile-device scales. The results show that RoSCo achieves more stable overall performance than representative baseline methods, especially in terms of task drop rate and edge-node load imbalance. Under sparse-eligibility and high-load conditions, RoSCo also maintains competitive task completion delay and energy consumption.

The remainder of this paper is organized as follows. Section 2 reviews related studies on MEC task offloading, heterogeneous-task coordination, and deep reinforcement learning-based offloading methods. Section 3 presents the system model and problem formulation. Section 4 introduces the proposed RoSCo method. Section 5 reports the experimental settings and result analysis. Section 6 concludes this paper and discusses future work.

2. Related Work

2.1. Task Offloading in Mobile Edge Computing

Task offloading is one of the core problems in mobile edge computing. Existing studies have investigated how computation tasks can be allocated between local devices and edge servers to reduce task completion delay, energy consumption, or a weighted system cost. Early MEC offloading studies mainly formulated the problem as deterministic optimization, where task assignment, transmission power, bandwidth allocation, and computing-resource allocation were jointly optimized under latency and resource constraints [13,14]. These methods provide clear optimization structures, but they usually rely on relatively stable system states and predefined feasible execution options.

With the increasing dynamics of mobile networks, later studies further considered time-varying channel conditions, stochastic task arrivals, user mobility, and resource competition among multiple mobile devices [15,16]. In these scenarios, the offloading decision is no longer a static selection problem, but a sequential decision process affected by both current system states and future resource availability. However, most existing studies still assume that the candidate edge servers are either fixed or universally available once communication and computing resources are sufficient. The feasibility of an offloading action is therefore often determined by resource capacity or link quality alone.

This assumption is not sufficient for heterogeneous MEC services. In practical edge-service systems, different edge servers may support different service types, software environments, models, or processing capabilities. A task may be executable only on a subset of edge servers that satisfy its service requirements. Therefore, the offloading action space should not be treated as a fixed candidate-server set. Instead, it should be dynamically determined by the eligibility relationship between task attributes and node states. This distinction is important because infeasible offloading actions are not merely high-cost actions; they are actions that cannot be executed in the current MEC environment.

2.2. Coordination for Heterogeneous Tasks and Resource Competition

In addition to offloading decisions, resource competition and service coordination have been widely studied in edge computing systems. Existing works have investigated cooperative service caching, task scheduling, load balancing, and multi-user resource allocation to improve system efficiency under limited edge resources [17,18]. These studies show that node-side resource sharing and service-order arrangement can significantly affect the latency and reliability of MEC systems, especially when many users compete for nearby edge resources.

For heterogeneous tasks, service coordination becomes more complex. Different tasks may have different data sizes, required CPU cycles, delay budgets, priority levels, and service requirements. When these tasks are assigned to the same edge server, their completion delay is affected not only by

 doi:10.20944/preprints202605.0671.v1

transmission and computation resources, but also by the local queueing process and the service order at the edge node. Some studies have incorporated priority queues, task scheduling, or workload-aware resource allocation to reduce local congestion and improve service quality [19,20]. These methods provide useful mechanisms for handling resource competition after tasks enter edge servers.

However, in eligibility-constrained MEC systems, local competition is coupled with the feasibility of task-node matching. Tasks are not freely distributed over all edge servers; instead, they are concentrated on eligible nodes determined by task attributes and node states. As a result, the local queueing pressure at a node is shaped by the eligibility structure before the task is actually offloaded. If this relationship is ignored, the offloading policy may optimize only the current execution cost while overlooking how eligibility-induced concentration affects future queues and load distribution. Therefore, local coordination should be considered together with feasible-action construction rather than as an isolated post-offloading scheduling problem.

2.3. Deep Reinforcement Learning-Based MEC Task Offloading

Deep reinforcement learning (DRL) has been widely applied to MEC task offloading because of its ability to learn sequential decision policies under dynamic environments. By modeling the offloading process as a Markov decision process, DRL-based methods can map system states, such as channel conditions, task arrivals, queue lengths, and computing resources, to offloading actions through reward feedback [21,22]. Compared with conventional optimization methods, DRL-based approaches are more suitable for online decision-making when the system state changes frequently and the exact future environment is unknown.

Different DRL variants have been used for MEC offloading and resource management. D eep Q-network (DQN) has been adopted to learn discrete offloading decisions, while double DQN is commonly used to alleviate action-value overestimation. Dueling network structures f urther separate state-value estimation from action-advantage estimation, which can improve le arning stability when different actions have similar values under certain states [23,24]. In addi tion, multi-agent reinforcement learning has been introduced to distributed MEC systems whe re multiple mobile devices or edge nodes make decisions simultaneously [25,26]. These studie s demonstrate the potential of DRL in dynamic MEC optimization.

Nevertheless, most DRL-based offloading studies still define a fixed action space and rely on the agent to learn the relative value of each candidate action. This treatment is problematic when the action space itself changes with task attributes and node states. If infeasible actions remain in the action space, the agent may waste exploration on invalid decisions, and the reward signal may become less informative. Some constrained reinforcement learning methods introduce penalties for invalid decisions, but penalty-based handling does not fully remove infeasible actions from policy learning. For eligibility-constrained MEC offloading, a more direct approach is to explicitly construct the feasible action set and enforce it during both action selection and target-action evaluation.

2.4. Positioning of This Work

The above studies provide useful foundations for MEC task offloading, edge-resource coordination, and DRL-based decision-making. However, the problem considered in this paper differs from existing work in three aspects. First, the task offloading action space is not assumed to be fixed. Instead, the feasible edge-server set changes dynamically according to task-node eligibility relationships. Second, the local competition among heterogeneous tasks is not treated only as a node-side scheduling issue after offloading. It is incorporated into the offloading decision process because eligibility constraints may concentrate tasks on a limited set of eligible edge servers. Third, the objective is not restricted to minimizing the instantaneous delay or energy cost of the current task. The offloading policy is also guided to suppress persistent congestion and improve long-term load balance.

 doi:10.20944/preprints202605.0671.v1

Accordingly, this paper proposes RoSCo, a load-aware task offloading and coordination method for eligibility-constrained MEC systems. The method integrates eligibility-aware action masking, priority-driven local coordination, and load-responsive reward design within a D3QN-based learning framework. Compared with conventional DRL-based offloading methods, RoSCo explicitly excludes infeasible offloading actions, characterizes local service competition among heterogeneous tasks, and incorporates long-term load feedback into policy learning. This design aims to improve the consistency between the decision space used for learning and the actual execution conditions of heterogeneous MEC systems.

3. System Model and Problem Formulation

3.1. System Scenario and Network Model

This paper considers an MEC system composed of mobile terminals, a wireless access network, and multiple edge nodes. In this system, terminals generate tasks to be processed over consecutive time slots. Each task can either be executed locally or offloaded to the edge side. Different from conventional MEC offloading settings, this paper further considers the matching relationship between task attributes and node service capabilities. That is, whether a task can be accepted by a node depends not only on the current link condition and resource occupancy, but also on the service eligibility of that node. As a result, before entering the decision stage, each task first goes through a screening process over candidate service nodes. The contraction of the eligible node set further changes the aggregation pattern of tasks on the edge side and the resulting local competition structure. Figure 1 illustrates the overall relationship among task attributes, eligibility filtering, local fallback execution, and edge-side competition coordination in the scenario considered in this paper.

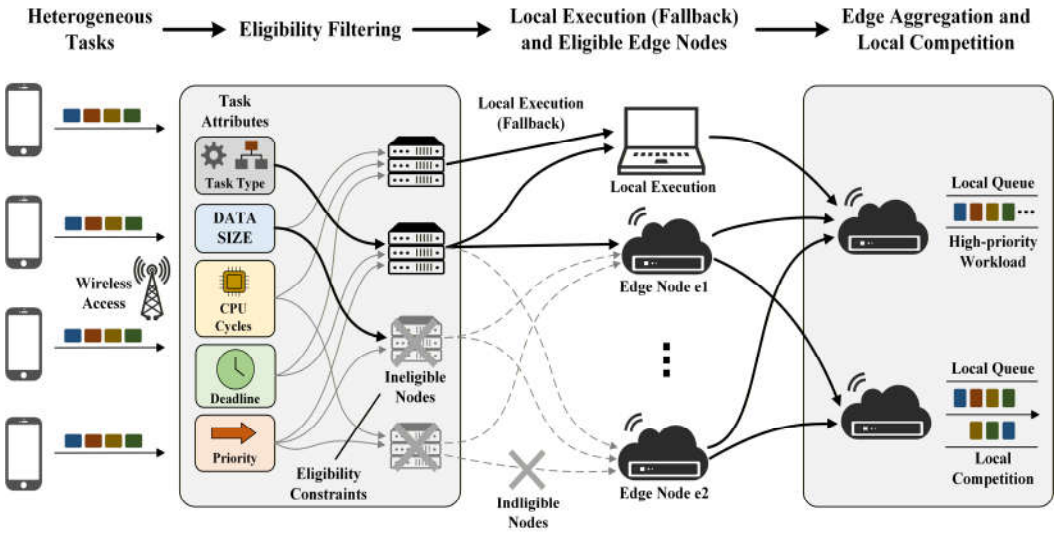


Figure 1. Illustration of the relationships among task attributes, eligibility filtering, and local competition.

As shown in Figure 1, once heterogeneous tasks generated by different mobile terminals enter the offloading decision stage, they are first matched against node service capabilities according to their task attributes. For tasks satisfying the eligibility conditions, the candidate execution locations are restricted to the set of eligible service nodes, which yields a contraction of the feasible set induced by eligibility constraints. For tasks that are not selected for offloading, or cannot satisfy the acceptance conditions of edge nodes, local execution is adopted as a fallback option. Meanwhile, after eligibility filtering, tasks from different terminals may converge to the same eligible edge nodes, thereby forming local queueing and competition at the node side. The priority queue shown in the figure is used to indicate that eligibility constraints not only change the range of candidate nodes for a task,

 doi:10.20944/preprints202605.0671.v1

but also further affect the task aggregation pattern and service order at the node side, which in turn has a persistent influence on subsequent delay, drop rate, and load evolution.

Let the set of terminals be $U = \{1,2,\dots,U\}$, and let the set of edge nodes be $E = \{1,2,\dots,E\}$.To unify the decision space, local execution is treated as a special node, denoted by $e = 0$. The extended set of candidate execution nodes is therefore defined as

$$\mathcal{E}_0 = \{0\} \cup \mathcal{E}, \tag{1}$$

and the task arriving from terminal u at time slot t is denoted by

$$J_u(t) = (d_u(t), c_u(t), \tau_u(t), \delta_u(t)), \tag{2}$$

Where $d_u(t)$ denotes the task input data size, $c_u(t)$ denotes the required number of CPU cycles, $\tau_u(t)$ denotes the task attribute label, and, $\delta_u(t)$ denotes the maximum tolerable completion delay of the task.

On the communication side, orthogonal access is assumed for terminals within each time slot, and bandwidth allocation is determined by the underlying network mechanism. This paper does not further optimize spectrum resources jointly. Accordingly, the uplink transmission rate between terminal u and edge node e at time slot t is given by

$$r_{u,e}(t) = B \log_2\left(1 + \frac{p_u h_{u,e}(t)}{\sigma^2}\right), \tag{3}$$

Where B is the system bandwidth, p_u is the terminal transmit power, $h_{u,e}(t)$ is the channel gain, and σ^2 is the noise power. On the computation side, the local computing capability of terminal u is denoted by f_u^{loc}, and the maximum computing capability of edge node e is denoted by f_e^{max}.

To indicate the execution location of a task, the binary offloading decision variable $x_{u,e}(t) \in \{0,1\}$ is defined. When $x_{u,e}(t) = 1$, task $J_u(t)$ is executed at node e; otherwise, it is not. For a newly arrived task, the execution location is unique. Therefore,

$$\sum_{e \in \mathcal{E}0} x_{u,e}(t) = 1, \quad \forall u, t, \tag{4}$$

3.2. Modeling of Eligibility Constraints and Local Competition

In this paper, eligibility constraints are modeled as a matching relationship between task attributes and node service capabilities. Unlike conventional settings that characterize node differences only by computing capability or link condition, this work further considers differences in task requirements with respect to runtime environment, service components, or processing capability. Therefore, whether a task can be offloaded to a node depends not only on the current resource state, but also on whether the node has the corresponding service eligibility. It should be emphasized that when the eligibility condition is not satisfied, execution at that node is not merely associated with a higher execution cost; rather, the task cannot be accepted and processed by that node at all.

Let $\mathcal{K}$ denote the set of task attributes, and let $\tau_u(t) \in \mathcal{K}$ denote the attribute category of task $J_u(t)$. For each edge node e, let $\mathcal{K}_e \subseteq K$ denote the set of service types supported by that node. Accordingly, the eligibility feasibility between task u and node e can be written as

$$g_{u,e}(t) = \begin{cases} 1, & \tau_u(t) \in \mathcal{K}_e, \\ 0, & \tau_u(t) \notin \mathcal{K}_e, \end{cases} \tag{5}$$

For the local node $e = 0$, it is treated as a fallback option that is always executable.

On the edge side, the node state is influenced not only by computing capability, but also by queue workload. Let $q_e(t)$ denote the total pending workload of node e at time slot t, measured uniformly in CPU cycles. Let q_e^{max} denote the maximum queue workload that node e can accommodate. Then, the dynamic feasible action set of task $J_u(t)$ is defined as

$$\mathcal{A}_u(t) = \{e \in \mathcal{E}_0 \mid g_{u,e}(t) = 1,\ q_e(t) + c_u(t) \le q_e^{max}\}, \tag{6}$$

This definition indicates that the problem considered in this paper no longer compares execution costs over a fixed candidate set. Instead, decisions are made within a feasible action set jointly determined by task attributes, node eligibility, and the current queue state.

Once eligibility constraints are introduced, task competition no longer spreads evenly over all nodes. Instead, it becomes concentrated on a subset of eligible service nodes. To characterize the

 doi:10.20944/preprints202605.0671.v1

service differences of heterogeneous tasks under local competition, a priority score is assigned to each task as

$$\pi_u(t) = \alpha\hat{\tau}_u(t) + \beta\hat{d}_u(t) + \gamma\hat{\zeta}_u(t), \quad (7)$$

Where $\hat{\tau}_u(t)$,$\hat{d}_u(t)$,and $\hat{\zeta}_u(t)$denote the normalized representations of the task attribute level, data size, and urgency, respectively. Here,α,β,and $\gamma$are weighting coefficients satisfying $\alpha + \beta + \gamma = 1$. The urgency term is defined by urgency level rather than by the maximum tolerable delay itself, so that tasks with shorter tolerable delay can be assigned higher priority.

At the execution side, tasks are assumed to be processed in descending order of priority. If two tasks have the same priority, they are served according to their arrival order [22,23]. To capture this mechanism without substantially increasing the state dimension, the paper further defines the amount of higher-priority pending workload that task $J_u(t)$ will face at node e as

$$q_{u,e}^{hp}(t) = \sum_{j \in \mathcal{Q}_e(t)} b_j(t), 1(\pi_j(t) \geq \pi_u(t)), \quad (8)$$

Where $\mathcal{Q}_e(t)$ denotes the set of queued tasks at node e at time slot t, $b_j(t)$ denotes the remaining CPU-cycle workload of queued task j at the current time slot, and $1(\cdot)$is the indicator function. This quantity describes the cumulative remaining workload that must be completed before the current task can begin service. It should be noted that $q_{u,e}^{hp}$ is directly observable from the environment at the current time slot and is incorporated as part of the task decision state, rather than being inferred solely from the total queue workload $q_e(t)$.

Since current decisions continue to affect future load, the node queue evolves over time according to the mechanism of processed workload release and newly arrived task accumulation. Let the duration of a time slot be Δt. Then, the queue evolution of node e is given by

$$q_e(t+1) = \max\{q_e(t) - f_e^{max}\Delta t, 0\} + \sum_{u \in U_t} x_{u,e}(t)c_u(t), \quad (9)$$

Where $\mathcal{U}_t$ denotes the set of terminals with newly arrived tasks at time slot t. This update relation indicates that the current offloading decision further influences the service feasibility and competitive environment of subsequent tasks by changing node load.

3.3. Performance Metrics and Optimization Objective

Based on the above model, this paper defines performance metrics from three aspects: task delay, energy consumption, and system load, and further constructs the long-term optimization objective accordingly. The paper adopts a commonly used energy modeling approach in MEC task offloading studies, while jointly considering the energy cost of local execution at the terminal side and collaborative processing at the edge side [24,25].

This paper assumes that terminal-side local tasks are processed immediately upon arrival and that no extra local queue is maintained. Therefore, if task $J_u(t)$ is executed locally, its completion delay and local energy consumption are given by

$$T_u^{loc}(t) = \frac{c_u(t)}{f_u^{loc}}, \quad E_u^{loc}(t) = \kappa_u(f_u^{loc})^2 c_u(t), \quad (10)$$

Where κ_u is the terminal-side energy coefficient. If the task is offloaded to edge node e, its completion delay consists of transmission delay, queue waiting delay, and edge execution delay. Specifically,

$$T_{u,e}^{tx}(t) = \frac{d_u(t)}{r_{u,e}(t)}, \quad T_{u,e}^{wait}(t) = \frac{q_{u,e}^{hp}(t)}{f_e^{max}}, \quad T_{u,e}^{cp}(t) = \frac{c_u(t)}{f_e^{max}}, \quad (11)$$

Hence, the total completion delay after offloading is

$$T_u^{off}(t) = T_{u,e}^{tx}(t) + T_{u,e}^{wait}(t) + T_{u,e}^{cp}(t), \quad (12)$$

For energy consumption, a system-level energy accounting model is adopted. That is, the terminal uplink transmission energy and the edge-node computation energy are jointly included in the total task energy consumption. Accordingly,

$$E_{u,e}^{tx}(t) = p_u T_{u,e}^{tx}(t), \quad E_{u,e}^{cp}(t) = \kappa_e(f_e^{max})^2 c_u(t), \quad (13)$$

 doi:10.20944/preprints202605.0671.v1

Where κ_e is the edge-side energy coefficient. Therefore, the total energy consumption after offloading is

$$E_u^{off}(t) = E_{u,e}^{tx}(t) + E_{u,e}^{cp}(t), \tag{14}$$

By combining local execution and edge execution, the unified task delay and energy consumption are written as

$$T_u(t)=x_{u,0}(t)T_u^{loc}(t)+\sum_{e\in E} x_{u,e}(t)T_u^{off}(t), \tag{15}$$

$$E_u(t)=x_{u,0}(t)E_u^{loc}(t)+\sum_{e\in E} x_{u,e}(t)E_u^{off}(t), \tag{16}$$

To indicate whether a task is completed within its service deadline, the task completion indicator is defined as

$$S_u(t) = \begin{cases} 1, & T_u(t) \le \delta_u(t), \\ 0, & T_u(t) > \delta_u(t), \end{cases} \tag{17}$$

It should be noted that infeasible actions and insufficient capacity have already been excluded during the construction of the feasible action set. Therefore, $S_u(t)$ is used only to indicate whether an executed task is completed before its deadline. The task drop-rate metric reported in the experiments follows this definition of task incompletion.

To account for the different scales of delay, energy consumption, and load, delay and energy consumption are normalized. Specifically, delay is normalized by the task delay budget, whereas energy consumption is normalized by the corresponding local execution energy, defined as

$$\bar{T}_u(t) = \frac{T_u(t)}{\delta_u(t)}, \quad \bar{E}_u(t) = \frac{E_u(t)}{E_u^{loc}(t)}, \tag{18}$$

To reflect the requirement of long-term load coordination, the load ratio of node e at time slot t is defined as

$$\rho_e(t) = \frac{q_e(t)}{q_e^{max}}, \tag{19}$$

and the system-level load penalty is defined as

$$L(t) = \frac{1}{|\mathcal{E}|}\sum_{e\in\mathcal{E}} \rho_e^2(t), \tag{20}$$

where the quadratic term increases the model sensitivity to heavily loaded nodes, so that the policy can suppress persistent backlog accumulation on a small number of nodes while optimizing average performance.

Based on the above definitions, the instantaneous system cost at time slot t is defined as

$$C(t) = \sum_{u\in\mathcal{U}_t} (\omega_1 \bar{T}u(t) + \omega_2 \bar{E}u(t) - \omega_4 S_u(t)) + \omega_3 L(t), \tag{21}$$

Where $\omega_1, \omega_2, \omega_3, \omega_4 \ge 0$ are weighting coefficients used to balance delay, energy consumption, system-level load penalty, and the benefit of task completion. Accordingly, the long-term optimization objective of this paper is

$$\min_{\pi} \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} C(t)\right], \tag{22}$$

subject to

$$\sum_{e\in\mathcal{E}_0} x_{u,e}(t) = 1, \quad \forall u, t, \tag{23}$$

$$x_{u,e}(t) \le g_{u,e}(t), \quad \forall u, e, t, \tag{24}$$

$$q_e(t) + x_{u,e}(t)c_u(t) \le q_e^{max}, \quad \forall u, e \in \mathcal{E}, t, \tag{25}$$

$$x_{u,e}(t) \in \{0,1\}, \quad \forall u, e, t, \tag{26}$$

Here, π denotes the offloading policy, and $\gamma \in (0,1)$ is the discount factor. These constraints correspond to the unique assignment constraint for each task, the eligibility feasibility constraint, the

 doi:10.20944/preprints202605.0671.v1

edge-node capacity constraint, and the discreteness constraint of the decision variable, respectively. Therefore, the problem can be viewed as a long-horizon sequential decision problem with a dynamic feasible action set.

3.4. Analysis of Problem Characteristics

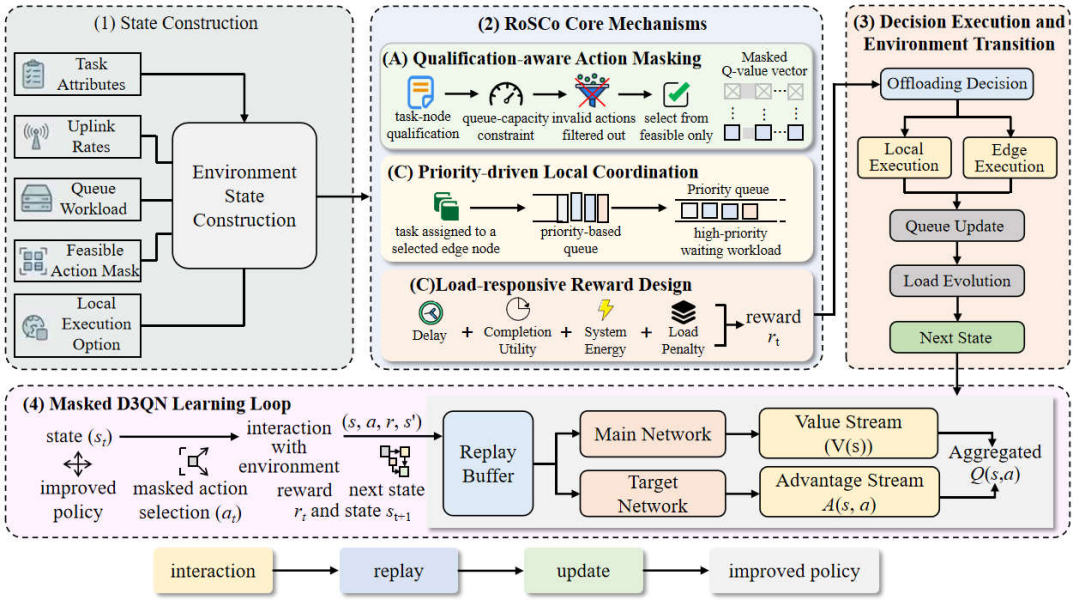
The difficulty of the problem does not lie simply in summing multiple cost terms such as delay, energy consumption, and load. Instead, it lies in the fact that the decision boundary and state evolution are both simultaneously shaped by eligibility constraints. For different tasks, the set of candidate execution nodes is not fixed in advance. It is jointly determined by task attributes, node eligibility, and the current queue state. Therefore, the feasible action set is highly dynamic.

Once a task enters a node, its waiting cost depends not only on node load, but also on the relative service order of the current task within the local queue. In this sense, offloading decisions become directly coupled with node-side local competition. On the other hand, the task allocation result at the current time slot not only determines the completion delay and energy consumption of the current task, but also continues to affect node load, feasible action space, and competition intensity in subsequent time slots through queue updates. Once local congestion is formed, its influence does not stop at a single decision epoch, but keeps accumulating in future states. Therefore, the problem studied in this paper should not be treated as a static optimization process solved independently at each time slot. If only the instantaneous cost is minimized greedily, tasks may remain concentrated on a small number of eligible service nodes under eligibility constraints, thereby aggravating hotspot backlog and degrading the completion quality of subsequent tasks. Hence, the problem is essentially a long-term coordination problem jointly driven by dynamic feasible action sets, heterogeneous competition, and accumulated load. The next section presents the method design on this basis.

4. Method Design

4.1. Overall Framework

The overall framework of RoSCo is shown in Figure 2. The method jointly addresses eligibility constraints, node-side local competition, and load variation within a unified decision process. After the current task arrives, the environment constructs the state representation according to the task attributes, uplink transmission rate, node queue workload, feasible-action information, and the local execution option. The policy then selects an execution location from the feasible action set. After the task is processed either locally or at the edge side, the environment updates the node queues and system load and further generates the state for the next time step.



 doi:10.20944/preprints202605.0671.v1

Figure 2. Overall framework of the RoSCo method.

In this process, the eligibility-aware action constraint, priority-driven local coordination, and load-responsive reward design correspond respectively to action screening, the characterization of node-side service order, and long-term load regulation. To implement the above decision process, this paper adopts D3QN as the policy learning framework [26–30]. In particular, the action mask is used in both current action selection and next-state target action computation, so that value updating is restricted to the feasible action sets associated with both the current state and the next state. The following subsections describe the action-space constraint, local coordination mechanism, reward construction, and policy learning procedure in turn.

4.2. Eligibility-Aware Action-Space Constraint

Under eligibility-constrained conditions, the set of eligible service nodes changes with both task attributes and node states. Accordingly, RoSCo directly incorporates the dynamic feasible action set into the decision process and restricts policy search to the currently feasible actions through an action-masking mechanism during action selection.

Let the action mask vector for the current task at time slot$t$be $m_u(t) = \{m_{u,e}(t)\}_{e\in\mathcal{E}0}$, where

$$m_{u,e}(t) = \begin{cases} 1, & e \in \mathcal{A}_u(t), \\ 0, & e \notin \mathcal{A}_u(t), \end{cases} \tag{27}$$

For the action-value output $Q(s_t, e; \theta)$ dynamic feasible action setof the main network, the current action is selected only from the feasible action set as

$$a_t = \arg \max_{e\in\mathcal{A}_u(t)} Q(s_t, e; \theta), \tag{28}$$

This treatment ensures that policy updating always focuses on the return differences among feasible actions, rather than relying on a large number of invalid interactions to identify infeasible nodes. For the problem considered in this paper, the action space is no longer a predetermined full action set, but a dynamic feasible action set jointly determined by task attributes, node eligibility, and queue state.

4.3. Priority-Driven Local Coordination Mechanism

The action-feasibility constraint determines which nodes a task can enter, but the waiting cost inside a node still depends on the service order in the local queue. Under continuous arrivals of heterogeneous tasks and shared edge resources, if decisions are made only according to total node load, the actual waiting pressure faced by the task at the node side cannot be adequately reflected.

To address this issue, RoSCo introduces a priority-driven local coordination mechanism at the node execution side. After a task is assigned to node e, the node sorts the queue according to the priority score $\pi_u(t)$ defined in Section 3 and serves tasks sequentially in descending order of priority. The local competition intensity relevant to the current task is characterized by $q_{u,e}^{hp}(t)$, namely the higher-priority pending workload. This quantity is explicitly computed by the environment for each candidate node at each time slot and is provided to the agent as a state feature. Therefore, the policy can perceive not only the total node load, but also the local waiting pressure that the current task may face at different nodes.

4.4. Load-Responsive Reward Design

If the reward is defined only by the delay and energy consumption of the current task, the policy may repeatedly choose a small number of nodes with high instantaneous return under eligibility constraints, thereby aggravating local backlog. To avoid this issue, RoSCo incorporates both current task performance and system load into the reward design under a task-level decision framework.

For the current decision task $J_u(t)$, the immediate reward at time slot t is defined as

$$r_t = -(\omega_1 \bar{T}_u(t) + \omega_2 \bar{E}_u(t) - \omega_4 S_u(t)) - \omega_3 L(t), \tag{29}$$

Here, the task-level terms ensure that policy learning directly targets the completion quality of the current task, whereas the system-level load term $L(t)$ reflects the influence of the current action

 doi:10.20944/preprints202605.0671.v1

on future system states. In this way, the state, action, and reward remain aligned at the same decision granularity, while the requirement of long-term coordination is incorporated into the learning process through the system-level load penalty.

Since infeasible actions have already been removed by the masking mechanism at the action-selection stage, this paper does not introduce an additional penalty term for eligibility violation. Accordingly, the focus of policy learning shifts from identifying invalid actions to comparing the long-term returns of feasible actions.

4.5. D3QN-Based Policy Learning

After defining the action constraint, local coordination mechanism, and reward design, this paper adopts D3QN to learn the long-term offloading policy. Its double-network estimation can alleviate action-value overestimation to a certain extent, while the dueling value decomposition is helpful for more stably distinguishing state value from action advantage. It is therefore suitable for the dynamically constrained decision environment considered in this work.

For the current task $J_u(t)$, the state vector at time slot t is represented as

$$s_t = [\phi_u(t), r_u(t), q(t), q_u^{hp}(t), m_u(t)], \tag{30}$$

Where $\phi_u(t) = [d_u(t), c_u(t), \tau_u(t), \delta_u(t), \pi_u(t)]$ denotes the current task-attribute features; $r_u(t) = \{r_{u,e}(t)\}_{e\in\mathcal{E}}$ denotes the uplink transmission rates to all edge nodes; $q(t) = \{q_e(t)\}_{e\in\mathcal{E}}$ denotes the total queue workload of all nodes;$q_u^{hp}(t) = \{q_{u,e}^{hp}(t)\}_{e\in\mathcal{E}}$ denotes the higher-priority pending workload faced by the current task at each node; and $m_u(t)$ denotes the feasible-action mask. In particular, categorical variables are encoded in discrete form before being input into the network.The priority score is retained together with the original task attributes as an aggregated description of task urgency and service level. For the local node $e = 0$,the edge-side feature components are uniformly set to zero, and a local-execution indicator is additionally introduced. At the same time, this action is always retained as feasible in the mask.

On this basis, action a_t denotes the execution-location selection of the current task within the feasible action set, the reward is defined as in the previous subsection, and the state transition is jointly determined by node queue updating and task arrival at the next time slot. The problem can therefore be formulated as a Markov decision process with action constraints.

In terms of network structure, this paper adopts a dueling value decomposition mechanism, in which state value and action advantage are estimated separately and then aggregated to obtain the action-value function. Specifically, the output of the main network is written as

$$Q(s_t, e; \theta) = V(s_t; \theta) + A(s_t, e; \theta) - \frac{1}{|\mathcal{A}(s_t)|}\sum_{e'\in\mathcal{A}(s_t)} A(s_t, e'; \theta), \tag{31}$$

Where $V(s_t;\theta)$ denotes the state-value function, $A(s_t, e; \theta)$ denotes the action-advantage function, and $\mathcal{A}(s_t)$ denotes the feasible action set under state s_t. Through this structure, the network can more stably distinguish the benefit of the state itself from the return differences among feasible actions under a dynamically constrained feasible action set.

On this basis, a double-network estimation mechanism is further adopted to achieve stable training. Let θ denote the parameters of the main network and θ^- denote those of the target network. For an experience sample (s_t, a_t, r_t, s_{t+1}), the target action of the next state is selected from its feasible action set as

$$a_{t+1} = \arg\max_{e\in\mathcal{A}(s_{t+1})} Q(s_{t+1}, e; \theta), \tag{32}$$

and the target value is then computed as

$$y_t = r_t + \gamma Q(s_{t+1}, a_{t+1}; \theta^-), \tag{33}$$

Where $\mathcal{A}(s_{t+1})$ denotes the feasible action set jointly determined by the task attributes and node states in the next state. The corresponding loss function is written as

$$\mathcal{L}(\theta) = \mathbb{E}[(y_t - Q(s_t, a_t; \theta))^2]. \tag{34}$$

 doi:10.20944/preprints202605.0671.v1

Algorithm 1. Training procedure of RoSCo

Input: initial main-network parameters θ, initial target-network parameters θ^{-}, and replay buffer D
Output: trained main-network parameters θ

1: Initialize the main-network parameters θ, target-network parameters θ^{-}, a nd replay buffer D.
2: **for** each training episode **do**
3: Reset the environment state.
4: **for** each decision epoch t **do**
5: Construct the state s_t according to the current task attributes, upli nk transmission rates, total node queue workload, node-level higher-priority pending workload, and the action mask.
6: Generate the current feasible action set $A(s_t)$ according to task e ligibility and node-capacity constraints.
7: Select action a_t in $A(s_t)$ according to the $\varepsilon - greedy$ rule.
8: Execute action a_t and update the local node queue according to t he priority-driven rule
9: Compute the immediate reward according to current-task delay, e nergy consumption, task completion gain, and system-level load penalty, and observe the next state s_{t+1}.
10: Store the transition sample (s_t, a_t, r_t, s_{t+1}) in replay buffer D.
11: Randomly sample a mini-batch from D.
12: For each sampled transition, generate the feasible action set $A(s_{t+1})$ of the next state.
13: Use the main network to select the target action in $A(s_{t+1})$

$$a_{t+1}^{*} = arg \max_{a \in A(s_{t+1})} Q(s_{t+1}, e; \theta)$$

14: Use the target network to compute the target value:

$$y_t = r_t + \gamma Q(s_{t+1}, a_{t+1}^{*}; \theta^{-})$$

15: Update the main-network parameters θ according to the loss function.
16: Periodically update the target-network parameters $\theta^{-} \leftarrow \theta$
17: **end for**
18: **end for**
19: Return θ

Algorithm 1 summarizes the training procedure of RoSCo. At each decision epoch, the environment constructs the feasible action set according to the current task and node states; the policy completes action selection under mask constraints; the environment returns the reward and next state after execution; the transition sample is stored in the replay buffer and used for subsequent network updating; and the target network is synchronized periodically to stabilize the training process

5. Experimental Results and Discussion

5.1. Experimental Settings

To evaluate the effectiveness of RoSCo in eligibility-constrained MEC scenarios with heter ogeneous tasks, this paper constructs a multi-terminal, multi-edge-node collaborative task offlo ading system in a discrete-time-slot simulation environment. In this system, terminals continu ously generate computing tasks with heterogeneous attributes and deadline constraints. Edge nodes are heterogeneous in computing capability, current load, and service eligibility. The tas

 doi:10.20944/preprints202605.0671.v1

k input size, computational demand, delay constraint, node eligibility distribution, and link st ate all vary dynamically with the scenario, so as to reflect the decision characteristics of MEC environments under eligibility constraints.

The comparison methods include Greedy, DQN, DDQN, CA-RL, PS-RL, and RoSCo. Gree dy selects the execution location according to the instantaneous cost at the current decision ep och and is used as a non-learning baseline. DQN uses a standard deep Q-network structure f or task offloading decisions. DDQN adopts a double-network structure to alleviate action-valu e overestimation. CA-RL introduces eligibility-constraint handling into the decision process. PS -RL further incorporates a priority-service mechanism into the reinforcement learning framewo rk.

All reinforcement learning methods use the same settings for the number of training epis odes, learning rate, batch size, discount factor, target-network update interval, and replay buff er. Unless otherwise specified, all reported results are obtained by independent training and t esting under five random seeds, and the shaded regions or error bars in the figures denote st andard deviation. The main simulation and training parameters are listed in Table 1. Some pa rameter settings follow common configurations in existing MEC task offloading studies and ar e further adjusted according to the scenario considered in this paper [31,32]. Two groups of p rimary experiments are conducted, with task arrival rate and terminal scale as the control var iables, respectively, in order to examine performance variations under increasing load and exp anding system scale.

All experiments are conducted on a local workstation equipped with an Intel Core i5-126 00KF processor, 32 GB DDR4 memory, and an NVIDIA GeForce RTX 4060 Ti (8 GB) GPU. Th e operating system is Windows 10 Professional 64-bit. The experimental code is implemented in Python and PyTorch, and both model training and testing are performed on a single GPU.

Table 1. Main simulation and training parameter settings.

Category	Parameter	Value
System scale	Number of terminals (terminal-scale experiment)	20–100
	Fixed number of terminals (arrival-rate experiment)	50
	Number of edge nodes	8
	Number of local execution options	1
Task load	Task arrival rate (arrival-rate experiment)	80–420
	Fixed task arrival rate (terminal-scale experiment)	250
Task constraints	Eligibility ratio ρ	0.2–1.0
Training	Training episodes	1000
	Time slots per episode	200
	D3QN learning rate	8e-4
	DQN/DDQN learning rate	1e-3
	Batch size	64
	Replay buffer size	20000
	Discount factor γ	0.95–0.96
Statistics	Number of random seeds	5

5.2. Results and Discussion

Figure 3 shows the overall performance of different methods under different task arrival rates. When the number of mobile terminals is fixed at 50, the system load gradually increases as the task arrival rate grows. Accordingly, the task drop rate, average delay, and node-load dispersion all increase, indicating that node-side competition under eligibility constraints becomes more pronounced in the high-load regime. Compared with the baseline methods, RoSCo maintains a more stable overall performance over the full task-arrival-rate range. Averaged over this range, RoSCo reduces drop rate, average delay, energy per task, and Load CV by 4.67%, 3.22%, 1.12%, and 8.95%,

 doi:10.20944/preprints202605.0671.v1

respectively, compared with CA-RL. When the task arrival rate increases to 420 tasks/slot, RoSCo still outperforms CA-RL by 0.86%, 1.04%, 0.75%, and 7.14% on the above four metrics, respectively. These results indicate that the advantage of RoSCo is not mainly reflected in the low-load regime, but in its ability to control local congestion and system-level load dispersion after the load increases. It should also be noted that RoSCo does not achieve the lowest energy per task at every sampled point, but it maintains a competitive energy-performance level overall.

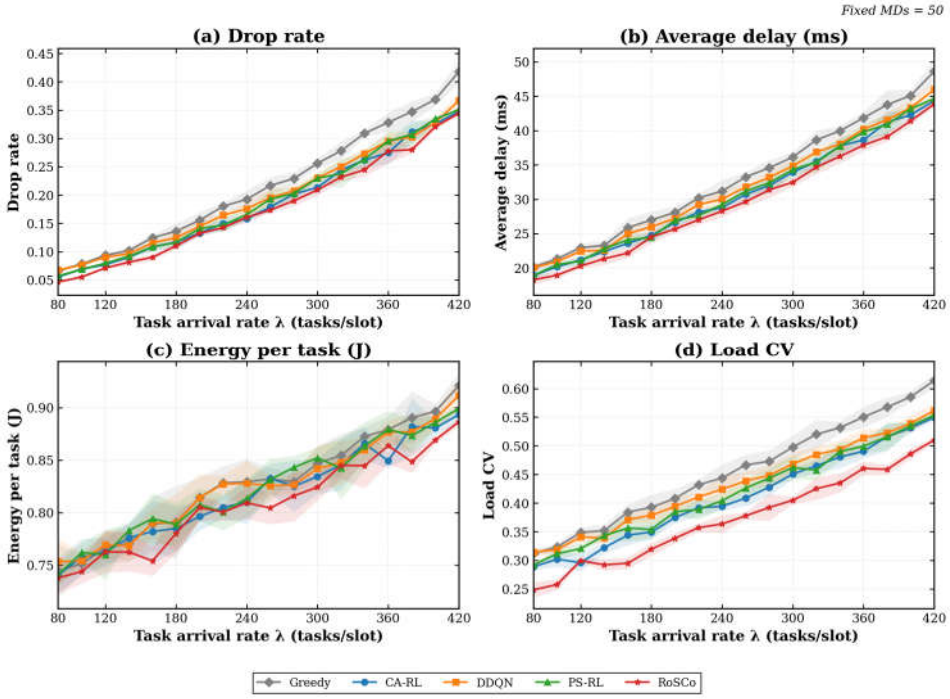


Figure 3. Performance comparison of different methods under different task arrival rates.

Figure 4 further compares the performance of different methods under different numbers of mobile terminals. When the task arrival rate is fixed at 250 tasks/slot, increasing the number of terminals leads to more concurrent task requests. As a result, node-side queueing and the local aggregation effect induced by eligibility constraints become more evident. Therefore, all methods exhibit performance degradation to different degrees in terms of drop rate, average delay, and load balancing. Averaged over the full range of terminal numbers, RoSCo reduces drop rate, average delay, energy per task, and Load CV by 5.00%, 3.69%, 1.16%, and 9.62%, respectively, relative to the best-performing baseline for each metric. In the medium- and large-scale regions, the advantage of RoSCo becomes more evident. For example, when the number of terminals is 80, RoSCo reduces drop rate, average delay, energy per task, and Load CV by 2.60%, 4.72%, 1.04%, and 7.22%, respectively, compared with PS-RL. At the largest system scale, the main strengths of RoSCo remain concentrated in drop rate and Load CV, whereas its performance on average delay and energy per task remains close to that of the best-performing baseline. Combining the trends in Figures 3 and 4, the gains of RoSCo mainly come from its joint treatment of task-destination selection and node-side local competition under eligibility constraints, rather than from the optimization of a single metric alone.

 doi:10.20944/preprints202605.0671.v1

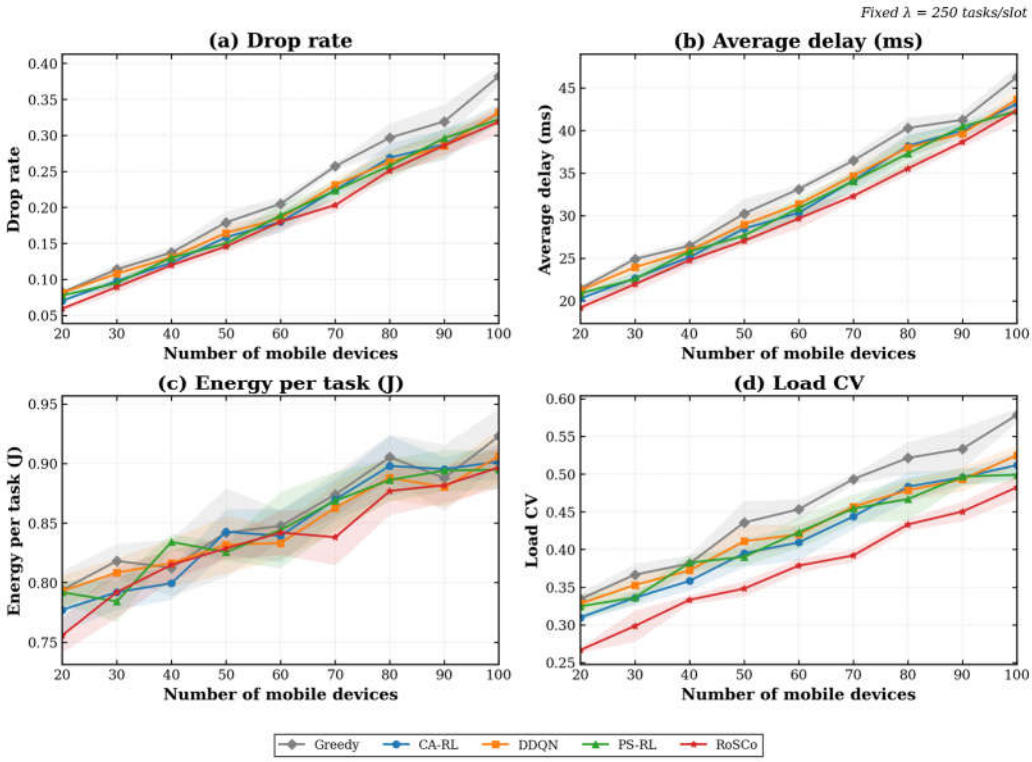


Figure 4. Performance comparison of different methods under different terminal scales.

Figure 5 presents the variation of normalized return during training for different reinforcement learning methods. Since Greedy does not involve parameter updating, this figure compares only learning-based methods with iterative training processes. Overall, the returns of all methods gradually increase as training proceeds, but clear differences exist in the growth rate and late-stage fluctuation behavior. DQN converges relatively slowly and reaches a lower return in the later training stage. After the double-network structure is introduced, DDQN shows improved training stability. PS-RL maintains relatively smooth growth in the middle and late training stages. Owing to its explicit handling of action feasibility, CA-RL exhibits a more stable training process than standard DQN and DDQN. By contrast, RoSCo maintains a higher return level in the middle and late stages. Over the last 50 training episodes, its average normalized return is 6.40% higher than that of CA-RL. This result indicates that, when dynamic feasible action sets and node-side local competition coexist, incorporating eligibility constraints, local coordination, and load-responsive rewards into the learning process is bene

ficial for improving training effectiveness.

 doi:10.20944/preprints202605.0671.v1

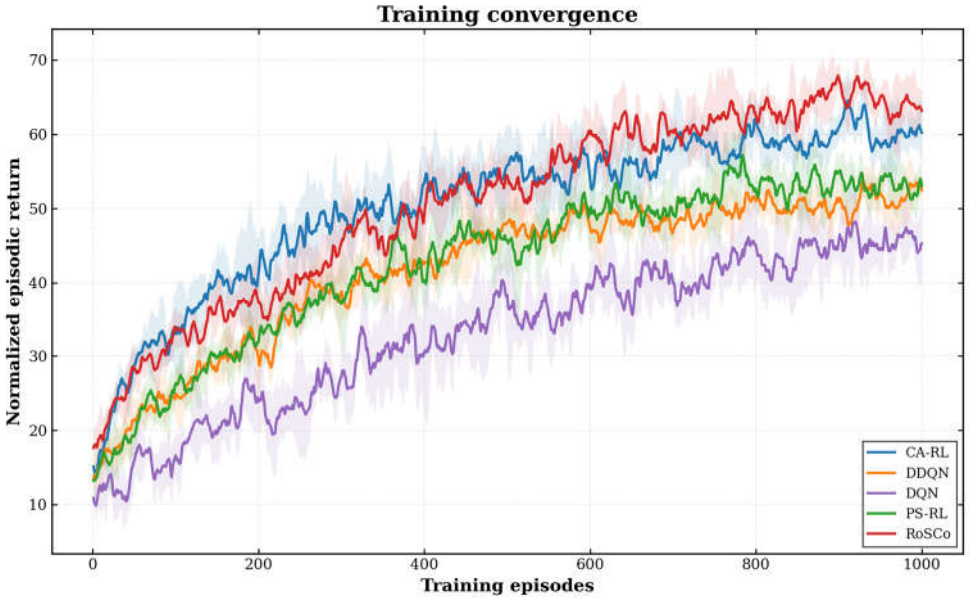


Figure 5. Training convergence curves of different learning-based methods.

Figure 6 reports the mechanism-validation results. Figure 6(a) mainly examines the impact of eligibility-constraint handling on the original action preference. For this reason, DDQN, CA-RL, and RoSCo are compared. As the eligibility ratio decreases, the feasible action set continues to shrink. Methods that do not explicitly adapt to a dynamic feasible action set are therefore more likely to direct their original action preferences toward infeasible nodes. Over the full eligibility-ratio range, the Pre-mask Invalid-Action Preference of RoSCo is reduced by 93.80% and 52.39%, respectively, compared with DDQN and CA-RL. Under the strongest constraint condition, namely $\varrho = 0.2$, the corresponding reductions still reach 94.40% and 53.15%. This result indicates that the eligibility-aware action constraint not only reduces invalid choices at the execution stage, but also substantially suppresses the original preference for infeasible actions at the policy-output level.

Figure 6(b) focuses on the influence of the local coordination mechanism on the service quality of high-priority tasks. Accordingly, CA-RL, PS-RL, and RoSCo are selected for comparison. The results show that RoSCo achieves the lowest average delay for high-priority tasks, with reductions of 8.17% and 3.20%, respectively, compared with CA-RL and PS-RL. This suggests that handling eligibility constraints only at the action-selection level is not sufficient to reduce the local waiting cost of critical tasks. Once tasks enter a node, the service order itself continues to affect the resulting delay. By incorporating the priority-driven local coordination mechanism into both state modeling and the decision process, RoSCo enables the policy to distinguish the local competition pressure across different candidate nodes more effectively, thereby improving the service quality of high-priority tasks.

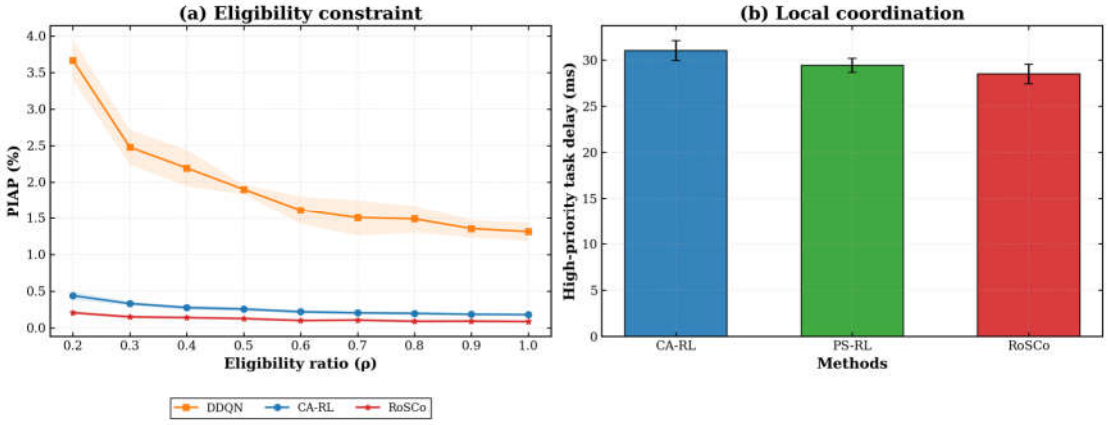


Figure 6. Mechanism-validation results.

 doi:10.20944/preprints202605.0671.v1

To further analyze the contribution of each constituent mechanism, an ablation study is conducted under the same settings. The results are summarized in Tables 2-a and 2-b. Compared with the full model, removing the action constraint increases drop rate, average delay, and Load CV by 14.87%, 12.95%, and 26.27%, respectively. This indicates that the dynamic feasible-action constraint is a key component for maintaining decision validity and controlling system-level load dispersion. Removing the priority-driven coordination mechanism increases average delay, energy per task, and high-priority task delay by 8.51%, 5.20%, and 7.98%, respectively, showing that node-side service order affects not only the delay of critical tasks but also the overall system performance. Removing the load-responsive rewards increases drop rate, average delay, and Load CV by 9.69%, 6.10%, and 20.72%, respectively. This indicates that decision making based only on task-level return is insufficient to suppress backlog accumulation at heavily loaded nodes and maintain long-term system stability. Overall, action-feasibility handling, priority-driven local coordination, and load-responsive rewards operate at different decision levels, and removing any one of them leads to performance degradation to varying degrees.

Table 2-a. Ablation results.

Variant	Drop Rate	Std.	Average Delay (ms)	Std.	Energy per Task (J)	Std.
RoSCo	0.158	0.007	27.48	0.54	0.812	0.011
Without priority mechanism	0.179	0.006	29.82	0.45	0.854	0.009
Without action constraint	0.182	0.008	31.044	0.70	0.841	0.006
Without load response	0.174	0.008	29.16	0.82	0.839	0.005

Table 2-b. Ablation results.

Variant	Load CV	Std.	High-Priority Task Delay (ms)	Std.
RoSCo	0.333	0.014	28.86	0.23
Without priority mechanism	0.386	0.005	31.16	0.71
Without action constraint	0.421	0.004	30.64	0.55
Without load response	0.402	0.006	29.29	0.52

Taken together, the trends shown in Figures 3–6 are largely consistent. The advantage of RoSCo is mainly reflected in its stable control of task drop rate, load dispersion, and high-priority task delay under eligibility constraints, rather than in absolute superiority on every metric under every scenario.

6. Conclusions

This paper investigated task offloading in heterogeneous mobile edge computing systems under task-node eligibility constraints. Different from conventional MEC offloading studies that assume a fixed or fully available candidate-server set, the considered scenario requires each task to be assigned only to execution locations satisfying its service requirements, link conditions, and node-state constraints. Under this setting, offloading decisions are coupled with dynamic feasible action sets, local queueing competition, and long-term edge-server load evolution.

To address this problem, RoSCo was proposed as a load-aware task offloading and coordination method for eligibility-constrained MEC systems. The method constructs a dynamic feasible action set for each arriving task and applies eligibility-aware action masking to exclude infeasible offloading actions during policy learning. A priority-driven local coordination mechanism was introduced to represent service competition among heterogeneous tasks at edge servers. In addition, a load-responsive reward was designed to guide the learning process toward congestion mitigation and

 doi:10.20944/preprints202605.0671.v1

load balancing. The offloading policy was learned using a dueling double deep Q-network, with the feasible action set considered in both current action selection and target-action evaluation.

Simulation results showed that RoSCo achieved a more stable trade-off among task drop rate, task completion delay, energy consumption, and edge-server load balance than representative baseline methods. The advantage was more evident under high-load and sparse-eligibility conditions, where feasible execution locations were limited and local queueing competition became more pronounced. The ablation results further indicated that eligibility-aware action masking, priority-driven local coordination, and load-responsive reward design contributed to different aspects of the overall performance, namely action feasibility control, local service coordination, and long-term load regulation.

Several limitations remain for future work. First, the current study was evaluated in a simulated MEC environment, and the system parameters were abstracted from typical edge-computing settings. Future work can further validate the proposed method using real edge traces or testbed-based experiments. Second, the current model focuses on single-task offloading decisions under dynamic eligibility constraints, while more complex dependency structures among tasks, such as workflow tasks or collaborative multi-service tasks, have not been considered. Third, the proposed framework can be extended to distributed or multi-agent MEC systems, where multiple mobile devices and edge servers learn coordinated policies under incomplete information and decentralized control.

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

Funding: This research was funded by the Guangxi Key Research and Development Program, Research on Cloud Native and Service Mesh-Based Container Cloud Platform Development and Application Demonstration, grant number GuiKe AB25069377, and the Guangxi Key Research and Development Program, Research and Industrialization of Cloud Intelligence Technology in Trusted Environments, grant number GuiKe AB25069300.

Data Availability Statement: Due to project privacy restrictions, the code is not publicly available. Researchers who are interested in the implementation details may contact the corresponding author by email for further information.

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

References

1. Dong, S.; Tang, J.; Abbas, K.; Hou, R.; Kamruzzaman, J.; Rutkowski, L.; Buyya, R. Task offloading strategies for mobile edge computing: A survey. Comput. Netw. 2024, 254, 110791. https://doi.org/10.1016/j.comnet.2024.110791
2. Luo, Z.; Dai, X. Reinforcement learning-based computation offloading in edge computing: Principles, methods, challenges. Alex. Eng. J. 2024, 108, 89–107. https://doi.org/10.1016/j.aej.2024.07.049
3. Gao, X.; Ang, M.C.; Althubiti, S.A. Deep reinforcement learning and Markov decision problem for task offloading in mobile edge computing. J. Grid Comput. 2023, 21, 78. https://doi.org/10.1007/s10723-023-09708-4
4. Peng, Z.; Wang, G.; Nong, W.; Qiu, Y.; Huang, S. Task offloading in multiple-services mobile edge computing: A deep reinforcement learning algorithm. Comput. Commun. 2023, 202, 1–12. https://doi.org/10.1016/j.comcom.2023.02.001
5. Wu, Z.; Jia, Z.; Pang, X.; Zhao, S. Deep reinforcement learning-based task offloading and load balancing for vehicular edge computing. Electronics 2024, 13, 1511. https://doi.org/10.3390/electronics13081511

 doi:10.20944/preprints202605.0671.v1

6. Qin, Y.; Chen, J.; Jin, L.; Yao, R.; Gong, Z. Task offloading optimization in mobile edge computing based on a deep reinforcement learning algorithm using density clustering and ensemble learning. Sci. Rep. 2025, 15, 211. https://doi.org/10.1038/s41598-024-84038-3
7. Zhang, X.; Fang, C.; Bai, Z.; Zhang, L.; Wang, P.; Cao, Z. Temporal dependency task offloading via deep reinforcement learning for mobile edge computing. Peer-to-Peer Netw. Appl. 2025, 18, 288. https://doi.org/10.1007/s12083-025-02101-w
8. Min, M.; Liu, Z.; Duan, J.; Zhang, P.; Li, S. Safe-learning-based location-privacy-preserved task offloading in mobile edge computing. Electronics 2024, 13, 89. https://doi.org/10.3390/electronics13010089
9. Li, Y.; Zhang, Z.; Chao, H.C. Service caching with multi-agent reinforcement learning in cloud-edge collaboration computing. Peer-to-Peer Netw. Appl. 2025, 18, 93. https://doi.org/10.1007/s12083-025-01915-y
10. Guo, R.; Zhou, L.; Li, L.; Song, Y.; Xie, X. Dependent task graph offloading model based on deep reinforcement learning in mobile edge computing. Electronics 2025, 14, 3184. https://doi.org/10.3390/electronics14163184
11. Peng, P.; Lin, W.; Wu, W.; Zhang, H.; Peng, S.; Wu, Q.; Li, K. A survey on computation offloading in edge systems: From the perspective of deep reinforcement learning approaches. Comput. Sci. Rev. 2024, 53, 100656. https://doi.org/10.1016/j.cosrev.2024.100656
12. Zheng, X.; Tahir, M.; Frnda, J.; Anwar, M.S.; Choi, A.; Kumar, R. Computation offloading based on incomplete information in edge computing networks. Cluster Comput. 2025, 28, 908. https://doi.org/10.1007/s10586-025-05560-1
13. Yao, Z.; Xia, S.; Li, Y.; Wu, G. Cooperative task offloading and service caching for digital twin edge networks: A graph attention multi-agent reinforcement learning approach. IEEE J. Sel. Areas Commun. 2023, 41, 3401–3413. https://doi.org/10.1109/JSAC.2023.3310080
14. Xie, M.; Ye, J.; Zhang, G.; Ni, X. Deep reinforcement learning-based computation offloading and distributed edge service caching for mobile edge computing. Comput. Netw. 2024, 250, 110564. https://doi.org/10.1016/j.comnet.2024.110564
15. Zheng, L.; Tan, L. A decentralized scheme for multi-user edge computing task offloading based on dynamic pricing. Peer-to-Peer Netw. Appl. 2025, 18, 91. https://doi.org/10.1007/s12083-025-01904-1
16. Pang, S.; Wang, T.; Gui, H.; He, X.; Hou, L. An intelligent task offloading method based on multi-agent deep reinforcement learning in ultra-dense heterogeneous network with mobile edge computing. Comput. Netw. 2024, 250, 110555. https://doi.org/10.1016/j.comnet.2024.110555
17. Chen, T.; Ai, J.; Xiong, X.; Hu, G. Cooperative service caching and task offloading in mobile edge computing: A novel hierarchical reinforcement learning approach. Electronics 2025, 14, 380. https://doi.org/10.3390/electronics14020380
18. Xiong, J.; Guo, P.; Wang, Y.; Meng, X.; Zhang, J.; Qian, L.; Yu, Z. Multi-agent deep reinforcement learning for task offloading in group distributed manufacturing systems. Eng. Appl. Artif. Intell. 2023, 118, 105710. https://doi.org/10.1016/j.engappai.2022.105710
19. Wu, H.; Geng, J.; Bai, X.; Jin, S. Deep reinforcement learning-based online task offloading in mobile edge computing networks. Inf. Sci. 2024, 654, 119849. https://doi.org/10.1016/j.ins.2023.119849
20. Li, N.; Zhai, L.; Ma, Z.; Zhu, X.; Li, Y. Lyapunov-guided deep reinforcement learning for service caching and task offloading in mobile edge computing. Comput. Netw. 2024, 250, 110593. https://doi.org/10.1016/j.comnet.2024.110593
21. Deng, T.; Yu, Z.; Yuan, D. Task offloading optimization in mobile edge computing under uncertain processing cycles and intermittent communications. Comput. Netw. 2024, 245, 110359. https://doi.org/10.1016/j.comnet.2024.110359
22. Segal, M. A multiserver system with preemptive priorities. Oper. Res. 1970, 18, 316–323. https://doi.org/10.1287/opre.18.2.316
23. Kella, O.; Yechiali, U. Waiting times in the non-preemptive priority M/M/c queue. Stoch. Models 1985, 1, 257–262. https://doi.org/10.1080/15326348508807014

 doi:10.20944/preprints202605.0671.v1

24. Wang, S.; Zhao, S.; Gui, H.; He, X.; Lu, Z.; Chen, B.; Fan, Z.; Pang, S. Energy-efficient collaborative task offloading in multi-access edge computing based on deep reinforcement learning. Ad Hoc Netw. 2025, 169, 103743. https://doi.org/10.1016/j.adhoc.2024.103743
25. You, C.; Huang, K.; Chae, H.; Kim, B.H. Energy-efficient resource allocation for mobile-edge computation offloading. IEEE Trans. Wirel. Commun. 2017, 16, 1397–1411. https://doi.org/10.1109/TWC.2016.2633522
26. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. Nature 2015, 518, 529–533. https://doi.org/10.1038/nature14236
27. van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double Q-learning. Proc. AAAI Conf. Artif. Intell. 2016, 30, 2094–2100. https://doi.org/10.1609/aaai.v30i1.10295
28. Wang, Z.; Schaul, T.; Hessel, M.; van Hasselt, H.; Lanctot, M.; de Freitas, N. Dueling network architectures for deep reinforcement learning. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; Volume 48, pp. 1995–2003. https://doi.org/10.5555/3045390.3045601
29. Pang, H.; Wang, Z. Dueling double deep Q network strategy in MEC for smart Internet of Vehicles edge computing networks. J. Grid Comput. 2024, 22, 37. https://doi.org/10.1007/s10723-024-09752-8
30. Ling, C.; Peng, K.; Wang, S.; Xu, X.; Leung, V.C.M. A multi-agent DRL-based computation offloading and resource allocation method with attention mechanism in MEC-enabled IIoT. IEEE Trans. Serv. Comput. 2024, 17, 3037–3051. https://doi.org/10.1109/TSC.2024.3466852
31. Ma, Y.; Zhao, Y.; Hu, Y.; He, X.; Feng, S. Multi-agent deep reinforcement learning for joint task offloading and resource allocation in IIoT with dynamic priorities. Sensors 2025, 25, 6160. https://doi.org/10.3390/s25196160
32. Jiang, G.; Huang, R.; Bao, Z.; Wang, G. A task offloading and resource allocation strategy based on multi-agent reinforcement learning in mobile edge computing. Future Internet 2024, 16, 333. https://doi.org/10.3390/fi16090333