EAI Endorsed Transactions
on Scalable Information Systems
Research Article EAI.EU

Intelligent Collaborative Resource Allocation for Mechanical Manufacturing Edge Networks: A Deep Reinforcement Learning Approach

Kaile Xiao[1], Xiajing Wang[2], Jing Wang[1,*]

[1]School of Applied Science and Technology, Beijing Union University, Beijing, China
[2]The Open University of China, Beijing, China

Abstract

Industry 4.0 is transforming mechanical manufacturing systems into edge-enabled, networked, and intelligent environments, where concurrent task execution, heterogeneous resource coordination, and dependency-aware scheduling have become critical requirements. In such scenarios, resource allocation must jointly consider computational demand, storage demand, business priority, deadline urgency, and inter-task dependencies, while enabling coordinated decisions among distributed edge agents. However, existing single-agent reinforcement learning methods have limited capability to model complex dependency relationships and heterogeneous resource collaboration under concurrent workloads, whereas conventional multi-agent systems often rely on coarse-grained task modeling and simplified cooperation mechanisms. To address these limitations, this paper proposes MIRA, a multi-agent deep reinforcement learning-based method for resource allocation in mechanical manufacturing edge networks. MIRA first decomposes tasks into fine-grained dependent subtasks, constructs a deadline-aware multi-metric priority function, and introduces a dynamic weight adjustment mechanism to balance computational demand, storage demand, normalized business priority, and deadline urgency. It then employs an adjacency matrix to characterize topology-aware agent interactions, enabling coordinated decision-making between computing agents and storage agents. Furthermore, MIRA incorporates an event-triggered state-exchange mechanism that updates subtask priorities and agent policies under changing workload, deadline, resource, and topology conditions. Experimental results on a PCB-derived simulated scheduling workload show that MIRA outperforms the selected baselines across multiple scheduling metrics.

Received on 03 June 2026; accepted on 02 July 2026; published on 07 July 2026

Keywords: Multi-agent deep reinforcement learning, edge intelligence, resource allocation, intelligent manufacturing

doi: 10.4108/eetsis.13306

1. Introduction

Driven by Industry 4.0, mechanical manufacturing systems are increasingly being transformed into edge-enabled, networked, and intelligent production environments, where edge controllers process production, inspection, and equipment-state data close to the shop floor. Edge computing enables production-site data to be processed locally, reducing transmission delay and cloud-side computing pressure [1]. However, manufacturing tasks usually execute concurrently, such as equipment status monitoring, production process optimization, and quality inspection, and their computing and storage requirements change over time.

Since such resource allocation requires sequential decisions under changing task and resource states, reinforcement learning has been increasingly adopted for task offloading and resource scheduling in edge-assisted manufacturing networks. Early studies often used single-agent reinforcement learning or centralized optimization. For example, Zhou et al. [2] proposed a Q-Learning and DDQN-based resource allocation method for task scheduling, Sadia et al. [3] addressed energy-efficiency optimization in IoT networks through

*Corresponding author. Email: 20230032@buu.edu.cn

intelligent reflecting surfaces and phase cooperation, and the LiMPO framework [4] considered user mobility offloading from a single-agent perspective. Related studies by Wu et al. [5], Baker et al. [6], Bhandari et al. [7], and Zhu et al. [8] also use transfer learning-based offloading or alternating optimization strategies. These methods are effective for their respective edge-computing scenarios, but concurrent manufacturing workloads require a scheduler that can represent interactions among tasks, resources, priorities, and time constraints.

As this research area has advanced, scholars have introduced multi-agent systems and distributed optimization to handle more complex scenarios. Shamim et al. [9] utilized position-specific scoring mechanisms for anomaly detection in IoT environments. Gebrekidan et al. [10] considered resource constraints across devices and servers, and Al Aghbari et al. [11] proposed federated mountain gazelle optimization for energy-aware task offloading in edge networks. Studies by Huang et al. [12], Zhou et al. [13], and Akyildiz et al. [14] further emphasize multi-agent collaboration. Building on this direction, fast response to resource fluctuations and dynamic task priorities remains an important issue for mechanical manufacturing edge scheduling.

Two limitations remain in the setting considered in this work. First, fine-grained multi-task features in mechanical manufacturing, such as differentiated computing/storage requirements and task dependencies, need to be represented more explicitly. Second, collaboration and competition mechanisms between agents need to better support joint resource allocation under concurrent tasks. To address these challenges, this study proposes the MIRA algorithm, a collaborative resource allocation strategy for mechanical manufacturing edge networks based on multi-agent deep reinforcement learning (MARL). MIRA is designed to improve adaptive scheduling under concurrent manufacturing workloads and heterogeneous edge-resource constraints.

Specifically, our contributions in this paper are as follows:

- Fine-grained task modeling and multi-metric balanced scheduling: MIRA decomposes manufacturing tasks into dependency-aware subtasks and constructs a deadline-aware multi-metric priority function that integrates computational demand, storage demand, normalized business priority, and deadline urgency. A dynamic weight adjustment mechanism is used to balance resource demand, task importance, and latency constraints.
- Resource collaborative allocation mechanism: MIRA designs a "priority preemption + remaining-resource compensation" strategy and uses adjacency matrices to model communication relationships among computing and storage agents. By aggregating topology-aware interaction information, agents can coordinate resource allocation under concurrent workloads.
- Dynamic adaptation and real-time information interaction: MIRA introduces an event-triggered state synchronization mechanism to recalculate subtask priorities and update allocation policies when workload urgency, resource availability, or agent topology changes.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 presents the architecture. Section 4 describes the model and algorithm. Section 5 reports experiments. Section 6 concludes the paper.

2. Related Work

Existing studies can be roughly grouped into three streams: resource-allocation modeling for manufacturing systems, DRL-based edge offloading, and cooperative resource scheduling. In manufacturing-oriented resource allocation, Lei et al. [15] formulated remanufacturing service resource allocation with rough fuzzy numbers, structural entropy weighting, and NSGA-II, while Yu et al. [16] decomposed mechanical products into meta actions for reliability allocation. These works show the importance of decomposing manufacturing activities and modeling resource constraints. Wu et al. [17] further reflects the broader role of feature modeling in distributed intelligent systems through frequency-domain feature screening for federated learning privacy protection. For edge-resource management, Peng and Shen [18] developed a DRL-based resource-management method for vehicular MEC, and Kong et al. [19] combined computing and caching decisions with DDPG to reduce mobile-network energy consumption. This stream provides efficient online decision models, but tasks are usually treated as coarse offloading requests or mobile-network service demands.

Resource coordination has also been studied in broader edge and IoT scenarios. Hussien et al. [20] used XGBoost and LSTM for energy-consumption prediction, which is relevant to resource-state estimation. Deng et al. [21] optimized latency, fairness, and load balancing for cloud gaming via DRL-based edge resource allocation, and Lin et al. [22] designed an online learning method for energy-aware resource management in industrial IoT VR streaming. Zhang et al. [23] studied DRL-based partial offloading for MEC networks, while Li et al. [24] and Hazarika et al. [25] introduced multi-agent or neighborhood-information-assisted offloading for dynamic MEC

and IoV networks. Xiao et al. [26] approached edge-server pre-allocation through a double-auction mechanism. Together, these studies improve latency, energy, fairness, or trading efficiency, but they are mostly designed around communication or computing-resource allocation rather than the joint coordination of computing demand, storage demand, priority, deadline urgency, and subtask dependency in manufacturing edge networks.

Dynamic response under changing workloads is another important direction. Chai et al. [27] modeled multi-task offloading as DAG-structured subtasks and introduced A-PPO for joint offloading and resource allocation. Waqar et al. [28] used decentralized value iteration for MEC-supported offloading and resource allocation, Xu et al. [29] combined deep double Q-learning with DDPG for mixed discrete-continuous resource decisions, and Yu et al. [30] studied attention-based continuous RL for digital-twin-enabled edge computing. These methods demonstrate the value of decentralized learning, hybrid decision modeling, and asynchronous environment adaptation. The manufacturing setting considered here further requires priority recalculation and topology-aware agent updates when deadlines, resource availability, or agent participation change during scheduling. Some cooperative MARL methods, including MAPPO, QMIX, and VDN, provide centralized-training/decentralized-execution and value-factorization mechanisms for multi-agent coordination [31–33]. These methods provide useful methodological background for multi-agent coordination. Dependency-aware offloading studies further model applications as DAG-structured subtasks so that predecessor-successor constraints can be considered during scheduling [34, 35]. These studies support the necessity of explicit dependency modeling, but they mainly focus on offloading decisions or secondary resource allocation rather than joint priority evolution and heterogeneous computing/storage-agent coordination in manufacturing edge networks.

Overall, existing studies have advanced edge-resource allocation from the perspectives of resource modeling, edge offloading, and multi-agent collaboration. However, in mechanical manufacturing scenarios, tasks are often jointly constrained by resource heterogeneity, execution deadlines, and process dependencies. Existing methods still have difficulty supporting subtask-level modeling, coordinated computing-storage allocation, and dynamic interaction among agents within a unified scheduling process. Therefore, this paper proposes MIRA to enable adaptive resource allocation for concurrent manufacturing tasks.

3. Architecture Design

We design a three-layer MARL-based resource-allocation framework for concurrent tasks and heterogeneous edge resources, as shown in Figure 1. This framework enables collaborative resource allocation across task access and processing, agent interaction, and resource allocation.

The bottom layer is the task perception and preprocessing layer. In the edge network for mechanical manufacturing, tasks such as equipment status monitoring, production process optimization, and quality inspection are continuously generated and submitted to the system. This layer extracts task attributes in real time, including task type, computing resource requirements, storage requirements, and time constraints. We adopt an application partitioning strategy to decompose tasks into fine-grained subtasks and construct a fine-grained task model. By disassembling and classifying tasks, complex tasks are broken down into multiple subtasks with clear resource requirements. Based on a deadline-aware multi-metric balanced scheduling strategy, these subtasks are prioritized by jointly considering computational demand, storage demand, normalized scheduling priority, and deadline urgency, providing clear and ordered task inputs for subsequent resource allocation.

The middle layer is the agent interaction and collaboration layer. This layer is the core of the entire framework, where each agent corresponds to a resource-management unit that manages either computing capacity or storage capacity in an edge node. Agents are modeled based on deep neural networks and possess learning and decision-making capabilities. Real-time information interaction is performed among agents, including resource-status information maintained by each agent and task-processing progress. According to the designed collaboration and competition mechanisms, when facing the demand for multi-task resource allocation, agents dynamically adjust resource allocation strategies through priority preemption, remaining-resource compensation, and topology-aware state exchange.

The top layer is the resource dynamic allocation and execution layer. In this layer, resources are allocated to tasks according to the strategy determined by the agent interaction and collaboration layer. Through real-time state exchange between agents, this layer captures task changes and resource fluctuations, such as sudden changes in task priorities or resource loss caused by faults. The algorithm then adjusts the allocation scheme so that resource assignments remain adaptive to current task requirements.

Overall, the three-layer framework improves adaptive resource allocation under concurrent manufacturing workloads.

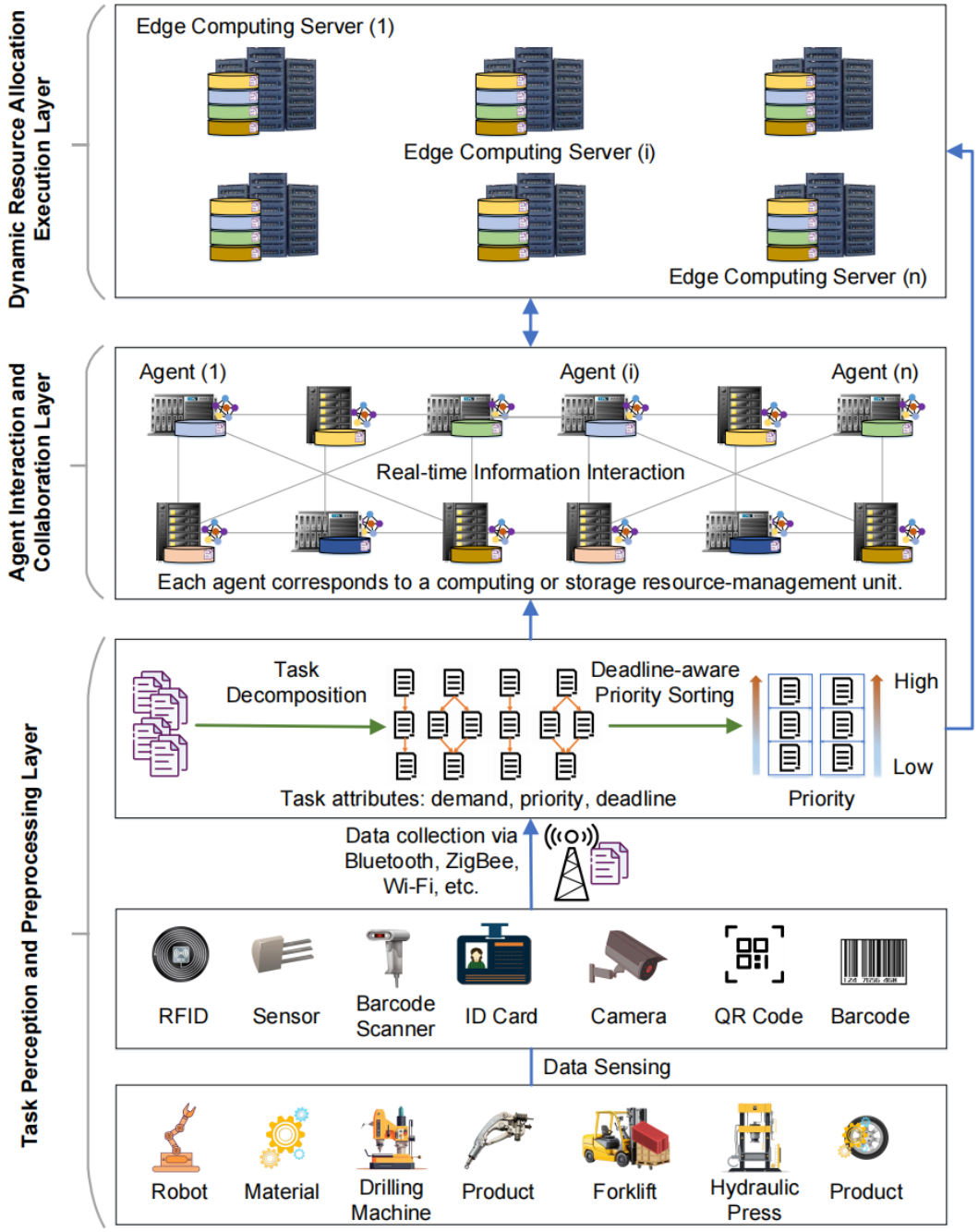

Figure 1. Three-Layer Architecture for Intelligent Mechanical Manufacturing

4. Model and Algorithm Design

Manufacturing edge networks process concurrent monitoring, optimization, and inspection tasks with heterogeneous computing, storage, and deadline requirements. Efficient collaborative allocation of heterogeneous resources has become a key challenge in ensuring the performance of manufacturing systems. To this end, we propose a Mechanical Intelligence Resource Allocation (MIRA) algorithm based on multi-agent deep reinforcement learning, with the algorithm flow shown in Figure 2.

Figure 2 summarizes the workflow of MIRA. The input task set is first transformed into fine-grained subtasks and ranked by the deadline-aware priority function. The agents then encode local resource states and topology-aware interaction information to select subtasks and allocate feasible computing or storage resources. Finally, rewards are calculated from utilization improvement, priority satisfaction, timeout

penalty, and collaborative completion, and the Actor-Critic networks are updated accordingly.

4.1. System Model

For clarity, calligraphic letters denote sets, bold uppercase letters denote matrices, and lowercase indexed symbols denote agent-, task-, or subtask-level variables. The task set is denoted as $\mathcal{T} = \{T_i\}_{i=1}^{n}$, and the agent set is denoted as $\mathcal{A} = \{a_j\}_{j=1}^{m}$. The subsets $\mathcal{A}_C$ and $\mathcal{A}_S$ denote computing-resource agents and storage-resource agents, respectively. The adjacency matrix among agents is denoted as $\mathbf{A}$, and $\mathbf{A}_{jk} = 1$ indicates that agents a_j and a_k have a direct communication or data-interaction relationship.

We model tasks and resources as follows. Define the task set $\mathcal{T} = \{T_i\}_{i=1}^{n}$, where each task T_i is described by the attribute vector $[c_i, s_i, t_{\text{lim}}^i, p_i^0]$. Here, p_i^0 denotes the raw business priority of task T_i, which is preset according to manufacturing requirements such as defect criticality or order urgency on a 0–1 scale. In each scheduling batch, the normalized business priority used by the scheduler is denoted as $p_i(t)$, obtained from p_i^0 after batch-level normalization and possible updates of raw business priority, such as newly inserted urgent tasks. c_i represents the computing resource requirement (which must satisfy $\sum_{i=1}^{n} c_i \le C_{\text{total}}$); for example, image-based quality inspection generally requires higher computational demand when processing complex parts. s_i denotes the storage resource requirement (which must satisfy $\sum_{i=1}^{n} s_i \le S_{\text{total}}$), such as a certain amount of storage resources required to store historical data of equipment status monitoring; t_{lim}^i is the time limit for task execution, such as production tasks for some urgent orders with strict time constraints. Each subtask T_{il} inherits the time limit of the parent task, i.e., $t_{\text{lim}}^{il} = t_{\text{lim}}^i$, and inherits the normalized scheduling priority of the parent task, i.e., $p_{il}(t) = p_i(t)$. The agent set $\mathcal{A} = \{a_j\}_{j=1}^{m}$, where each agent a_j corresponds to one resource-management unit managing either computing capacity or storage capacity of an edge node, and its static attribute vector is $[R_j]$ (where R_j represents the initial capacity of a single resource, a general term for computing resource capacity C_j or storage resource capacity S_j). The dynamic remaining resources are updated in real time as state variables (the remaining computing resources are C_j^{remain}, and the remaining storage resources are S_j^{remain}). The total system resources are accumulated according to the agent management type:

$$C_{\text{total}} = \sum_{j \in \mathcal{A}_C} C_j, \quad S_{\text{total}} = \sum_{j \in \mathcal{A}_S} S_j \tag{1}$$

To obtain fine-grained task representations, task T_i is decomposed into k_i subtasks $\{T_{i1}, T_{i2}, \cdots, T_{ik_i}\}$ according to the dependencies and resource requirement relationships between tasks. For example, the processing task of a large mechanical part can be decomposed into subtasks such as rough machining, finish machining, and surface treatment. The resource requirements of subtasks satisfy $\sum_{l=1}^{k_i} c_{il} = c_i$ and $\sum_{l=1}^{k_i} s_{il} = s_i$, and ensure that the resource requirements of subtasks $c_{il} \ge 0$, $s_{il} \ge 0$. After constructing the fine-grained task model, a multi-metric balanced scheduling strategy is designed to determine the priority of subtasks. To reflect the deadline constraints of manufacturing tasks, the subtask priority function is defined by four factors:

$$P_{il}(t) = \alpha_t \hat{c}_{il} + \beta_t \hat{s}_{il} + \gamma_t p_{il}(t) + \lambda_t d_{il}(t) \tag{2}$$

where $\hat{c}_{il} = c_{il} / \max_{T_{pq}} c_{pq}$ and $\hat{s}_{il} = s_{il} / \max_{T_{pq}} s_{pq}$ are the normalized computational and storage demands of subtask T_{il}, respectively; $p_{il}(t)$ is the normalized scheduling priority inherited from its parent task in scheduling cycle t; and $d_{il}(t)$ is the deadline urgency term:

$$d_{il}(t) = \text{clip}\left(\frac{e_i(t) + \hat{\tau}_{il}^{\text{rem}}}{t_{\text{lim}}^i + \varepsilon}, 0, 1 \right) \tag{3}$$

Here, $e_i(t)$ denotes the elapsed time from the arrival of task T_i to scheduling cycle t, $\hat{\tau}_{il}^{\text{rem}}$ is the estimated remaining processing time of subtask T_{il}, t_{lim}^i is the task deadline, and ε is a small constant that avoids division by zero. A larger $d_{il}(t)$ indicates that the task is closer to its deadline and should receive higher scheduling urgency. The dynamic weights α_t, β_t, γ_t, and λ_t correspond to computational demand, storage demand, normalized scheduling priority, and deadline urgency, respectively, and satisfy $\alpha_t + \beta_t + \gamma_t + \lambda_t = 1$.

The dynamic weight update is designed so that high resource utilization increases the corresponding resource-demand weight, while high deadline urgency increases the deadline-aware weight:

$$\begin{aligned}
\tilde{\alpha}_t &= \alpha_t + \eta_{\text{weight}} \max(0, U_C(t) - U_C^{\text{th}}), \\
\tilde{\beta}_t &= \beta_t + \eta_{\text{weight}} \max(0, U_S(t) - U_S^{\text{th}}), \\
\tilde{\gamma}_t &= \gamma_t, \\
\tilde{\lambda}_t &= \lambda_t + \eta_{\text{weight}} \frac{1}{|\mathcal{B}_t|} \sum_{T_{il} \in \mathcal{B}_t} d_{il}(t), \\
Z_t &= \tilde{\alpha}_t + \tilde{\beta}_t + \tilde{\gamma}_t + \tilde{\lambda}_t, \\
\alpha_{t+1} &= \tilde{\alpha}_t / Z_t, \quad \beta_{t+1} = \tilde{\beta}_t / Z_t, \\
\gamma_{t+1} &= \tilde{\gamma}_t / Z_t, \quad \lambda_{t+1} = \tilde{\lambda}_t / Z_t.
\end{aligned} \tag{4}$$

In this formulation, $U_C(t)$ and $U_S(t)$ are the computational and storage resource utilization rates at scheduling cycle t, U_C^{th} and U_S^{th} are the utilization thresholds

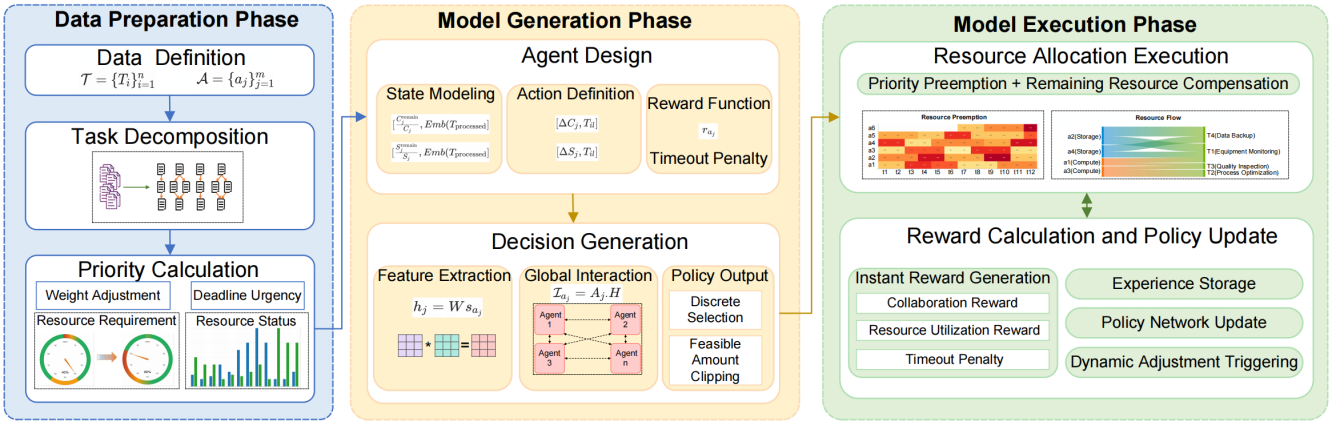

Figure 2. Algorithm flow

used to detect resource pressure, η_{weight} is the weight learning rate, and $\mathcal{B}_t$ is the set of pending subtasks in the current scheduling cycle. Thus, when computational or storage utilization exceeds its threshold, the corresponding demand weight is increased; when the pending task batch becomes deadline-critical, the deadline-aware weight λ_t is increased. The normalization term Z_t keeps the priority scale bounded and preserves the role of normalized business priority through γ_t.

Based on the task and resource models, we further define the agent state and action spaces. Each agent a_j in the algorithm is a resource-management unit for either computing capacity or storage capacity. The agent state s_{a_j} consists of remaining resources and task processing information, i.e.:

- If $a_j \in \mathcal{A}_C$, then $s_{a_j} = \left[\frac{C_j^{\text{remain}}}{C_j}, Emb(T_{\text{context}})\right]$, containing only the remaining proportion of computing resources and task features.
- If $a_j \in \mathcal{A}_S$, then $s_{a_j} = \left[\frac{S_j^{\text{remain}}}{S_j}, Emb(T_{\text{context}})\right]$, containing only the remaining proportion of storage resources and task features.

Here, $Emb(T_{\text{context}})$ denotes the task-context embedding of pending subtasks and recent processing states. It is a 64-dimensional vector including task delay, resource requirements, normalized scheduling priority $p_{il}(t)$, and deadline-related features such as deadline urgency $d_{il}(t)$, generated through a task embedding layer composed of a two-layer fully connected neural network.

The agent action a_{a_j} represents a resource allocation decision, formalized as:

- If $a_j \in \mathcal{A}_C$, then $a_{a_j} = [\Delta C_j, T_{il}]$, allocating only computing resources ΔC_j to subtask T_{il}.
- If $a_j \in \mathcal{A}_S$, then $a_{a_j} = [\Delta S_j, T_{il}]$, allocating only storage resources ΔS_j to subtask T_{il}.

The selected subtask must satisfy the following feasibility constraints:

- Computing resource allocation: $0 \le \Delta C_j \le \min(c_{il}, C_j^{\text{remain}})$.
- Storage resource allocation: $0 \le \Delta S_j \le \min(s_{il}, S_j^{\text{remain}})$.

To evaluate the quality of resource allocation decisions, the agent reward function r_{a_j} is designed:

$$\begin{aligned} r_{a_j} =& \omega_1 \cdot \Delta U_R^j + \omega_2 \cdot p_{il}(t) \\ & - \omega_3 \cdot I_{\text{timeout}}^{il}(t) \cdot \delta_{aj,T_i} \cdot I_{\text{unfinished}}^{aj,il}(t) + \omega_4 \cdot I_{\text{collab}} \end{aligned} \tag{5}$$

Here, δ_{aj,T_i} is the participation marker of agent a_j and task T_i (takes 1 when agent a_j has allocated resources to any subtask T_{il} of task T_i, otherwise 0, clarifying the association relationship with subtasks); $I_{\text{unfinished}}^{aj,il}(t)$ is the agent-subtask completion status marker at scheduling cycle t (for subtask T_{il} to which agent a_j has allocated resources, takes 1 when the resource allocation for the subtask is incomplete, and 0 when completed); I_{collab} is the collaborative completion marker (takes 1 when the agent and other agents jointly complete the allocation of the same subtask, otherwise 0).

Here, $\Delta U_R^j = U_R^j(t) - U_R^j(t-1)$, $U_R^j = \frac{R_j - R_j^{\text{remain}}}{R_j}$ is the local resource utilization rate; $I_{\text{timeout}}^{il}(t)$ is the timeout indicator for subtask T_{il}, taking 1 when the parent task of subtask T_{il} exceeds its deadline at scheduling cycle t and 0 otherwise; $\omega_1, \omega_2, \omega_3, \omega_4$ are reward weight coefficients. This function guides agents to prioritize high-priority tasks, optimize resource allocation strategies,

maximize system resource utilization efficiency, and avoid timeouts for tasks they are involved in.

Meanwhile, for agent modeling, s_{a_j} is a d_s-dimensional state vector. The system predefines a transformation matrix $W \in \mathbb{R}^{d_h \times d_s}$. Through the matrix multiplication operation $h_j = W s_{a_j}$, the input state is mapped into a d_h-dimensional feature vector h_j, completing the mapping from the original input to the feature space. Then, the adjacency matrix A is defined as a binary matrix, where $A_{jk} = 1$ if and only if agent j and k are directly connected in the physical network or have data interaction needs (such as computing agents and storage agents needing to collaborate to complete task allocation). $H \in \mathbb{R}^{m \times d_h}$ is the matrix composed of all agent feature vectors, where m denotes the number of agents. The global interaction information $\mathcal{I}_{a_j}$ of agent a_j is calculated by $A_{j\cdot} \cdot H$, where $A_{j\cdot}$ is the j-th row of the adjacency matrix A. This operation aggregates the feature information of other agents with direct communication relationships with agent a_j. Then, the local feature h_j is concatenated with the global interaction information $\mathcal{I}_{a_j}$ to obtain a new input vector $[h_j; \mathcal{I}_{a_j}]$, which is input into an action-specific linear function $g_{\theta_{j,a}}$. This function performs a linear transformation for each candidate action based on the corresponding parameter vector $\theta_{j,a}$, i.e.,

$$z_{j,a} = g_{\theta_{j,a}}([h_j; \mathcal{I}_{a_j}]) = \theta_{j,a}^T [h_j; \mathcal{I}_{a_j}] \tag{6}$$

By adjusting the action-specific parameter $\theta_{j,a}$, the transformation can assign different logits to different candidate actions and extract useful decision information from the input vector.

For each candidate action $a \in \mathcal{A}_{a_j}$, the Actor network computes an action-specific logit $z_{j,a} = g_{\theta_{j,a}}([h_j; \mathcal{I}_{a_j}])$. These logits are normalized using the softmax function to obtain the action probability distribution:

$$\pi_{\theta_j}(a \mid s_{a_j}, \mathcal{I}_{a_j}) = \frac{\exp(z_{j,a})}{\sum_{a' \in \mathcal{A}_{a_j}} \exp(z_{j,a'})} \tag{7}$$

For continuous resource allocation amounts (such as ΔC_j, ΔS_j), the discrete task-selection probability is used for subtask selection, while the continuous allocation amount is generated by a rule-based feasible allocation head and clipped to the feasible interval. Specifically, after subtask T_{il} is selected, let $r_{il}^R(t)$ denote its remaining demand for the resource type managed by agent a_j. The allocation head sets $\Delta R_j = \min(r_{il}^R(t), R_j^{\text{remain}})$ and clips it to $[0, R_j^{\text{remain}}]$. Therefore, the policy probability is defined for the discrete subtask-selection component:

$$\pi_{\theta_j}(T_{il} \mid s_{a_j}, \mathcal{I}_{a_j}) = \frac{\exp(z_{j,T_{il}})}{\sum_{T' \in \mathcal{T}_{a_j}^{\text{cand}}} \exp(z_{j,T'})} \tag{8}$$

Here, $\mathcal{T}_{a_j}^{\text{cand}}$ denotes the feasible candidate subtasks for agent a_j. The complete executed action is $a_{a_j} = [\Delta R_j, T_{il}]$, where T_{il} is selected according to the discrete policy and ΔR_j is produced deterministically by the feasible allocation head after clipping. The continuous allocation amount is not treated as an additional stochastic decision variable; the learning component focuses on subtask selection and uses the Critic feedback for the selected discrete action.

4.2. Edge Resource Allocation Strategy for Mechanical Manufacturing

At initialization, the task set $\mathcal{T}$ and agent set $\mathcal{A}$ are loaded into the scheduling system.

During scheduling, computing and storage agents select feasible subtasks according to local resource states and topology-aware interaction information. When a subtask requires resources from multiple agents, MIRA applies the "priority preemption + remaining-resource compensation" strategy. When a selected subtask cannot be fully served by one resource-management agent, its residual demand is compensated by redundant resources from other agents of the same type according to the priority order. If multiple agents select the same subtask in one scheduling cycle, resource allocation is executed according to the priority order and the remaining unmet demand of that subtask. Once the demand of the selected subtask is fully satisfied, subsequent allocations to the same subtask are masked out within the current scheduling cycle.

In each scheduling cycle t, all agents operate in parallel: agent a_j obtains its current state $s_{a_j}^t$, which contains information related to the managed resources, such as remaining resource amount and task processing progress; at the same time, through the communication relationship defined by the adjacency matrix A, it interacts with other agents to obtain global interaction information $\mathcal{I}_{a_j}^t$, which aggregates the feature information of other agents with direct communication relationships with agent a_j (including the resource status of agents of the same type and the task allocation progress of agents of cross types).

Based on the above information, agent a_j generates an action probability distribution through its own neural network and samples an action $a_{a_j}^t$ from it. The joint actions of all agents update the system state to $s_{a_j}^{t+1}$ $(j = 1, 2, \cdots, m)$, which specifically includes:

- For a computing agent, update $C_j^{\text{remain}} = C_j^{\text{remain}} - \Delta C_j$.
- For a storage agent, update $S_j^{\text{remain}} = S_j^{\text{remain}} - \Delta S_j$.

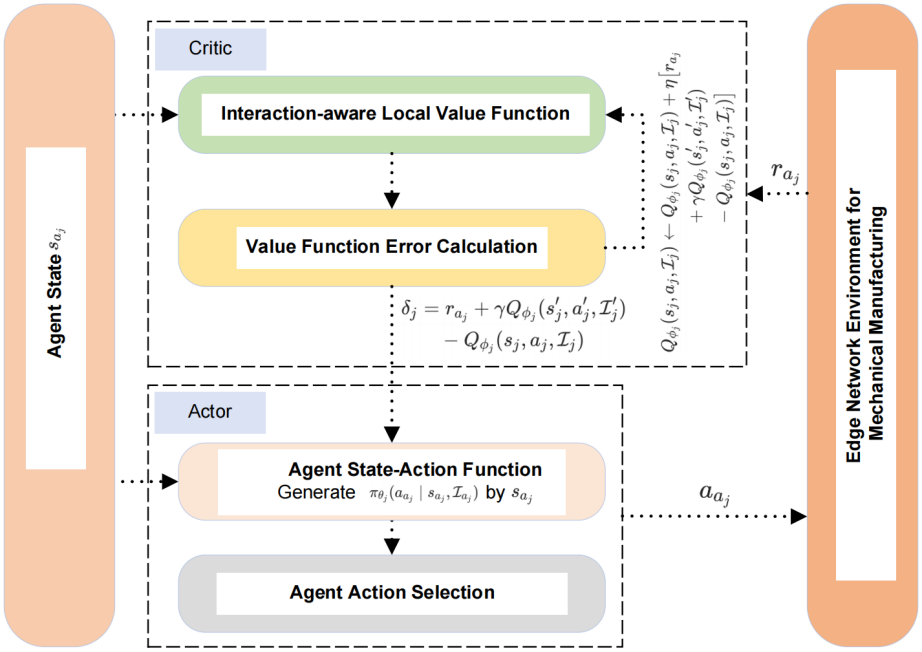

Figure 3. Actor-Critic architecture

- Mark the associated computing/storage agent of subtask T_{il} as a_j, and record the allocated resource amount and completion status.

After the action is executed, each agent a_j obtains $r_{a_j}^t$ according to the reward function. The design of the reward function comprehensively considers the resource allocation effect of the agent itself and its impact on the entire multi-agent system, including:

- When computing agent a_j and storage agent a_k jointly complete the allocation of a subtask, both parties receive a collaborative reward ω_4 (triggered by $I_{\text{collab}} = 1$).

- If a subtask times out due to delayed resource allocation by an agent, that agent receives the timeout penalty, while the other agent does not bear the penalty if it has completed the allocation (judge whether the agent has not completed the subtask allocation and bears the penalty through $I_{\text{unfinished}}^{aj,il}(t)$) .

Each agent stores its own experience $(s_{a_j}^t, a_{a_j}^t, r_{a_j}^t, s_{a_j}^{t+1})$ in the experience pool, which is used to store the experiences of all agents at different time steps, providing data support for subsequent model optimization.

In the learning process, the Critic network and Actor network are used to collaboratively optimize the agent strategy, as shown in Figure 3.

The Critic network adopts a distributed architecture. The Critic of each agent receives the local state-action pair (s_{a_j}, a_{a_j}) and global interaction information $\mathcal{I}_{a_j}$, and estimates an interaction-aware local action-value function $Q_j(s_j, a_j, \mathcal{I}_j)$, which measures the impact of the agent's actions under local interaction information.

The Actor network updates its own parameters θ_j through policy gradients:

$$\theta_j \leftarrow \theta_j + \eta_{\text{actor}} \cdot E\Big[\nabla_{\theta_j} \log \pi_{\theta_j}(T_{il} \mid s_{a_j}, \mathcal{I}_{a_j}) \cdot \Big(r_{a_j} + \rho Q_{\phi_j}(s'_j, a'_j, \mathcal{I}'_j) - Q_{\phi_j}(s_j, a_j, \mathcal{I}_j)\Big)\Big] \tag{9}$$

where η_{actor} is the policy gradient learning rate, and $\rho \in [0, 1]$ is the discount factor, which balances immediate and future rewards; s'_j represents the next state of the agent, i.e., $s_{a_j}^{t+1}$ after the action is executed. The logarithmic policy term corresponds to the learned discrete subtask-selection component, while the feasible allocation head supplies the clipped amount ΔR_j for the executed action. Each agent independently maintains a local Critic network $Q_{\phi_j}(s_j, a_j, \mathcal{I}_j)$ to estimate this interaction-aware local action-value function. The Actor is updated using the TD-error term estimated by the local Critic.

In addition, the weight update is executed once after each scheduling cycle to reflect changes in task priorities and resource states. The real-time interaction mechanism is triggered under four conditions, as shown in Figure 4. First, the normalized priority

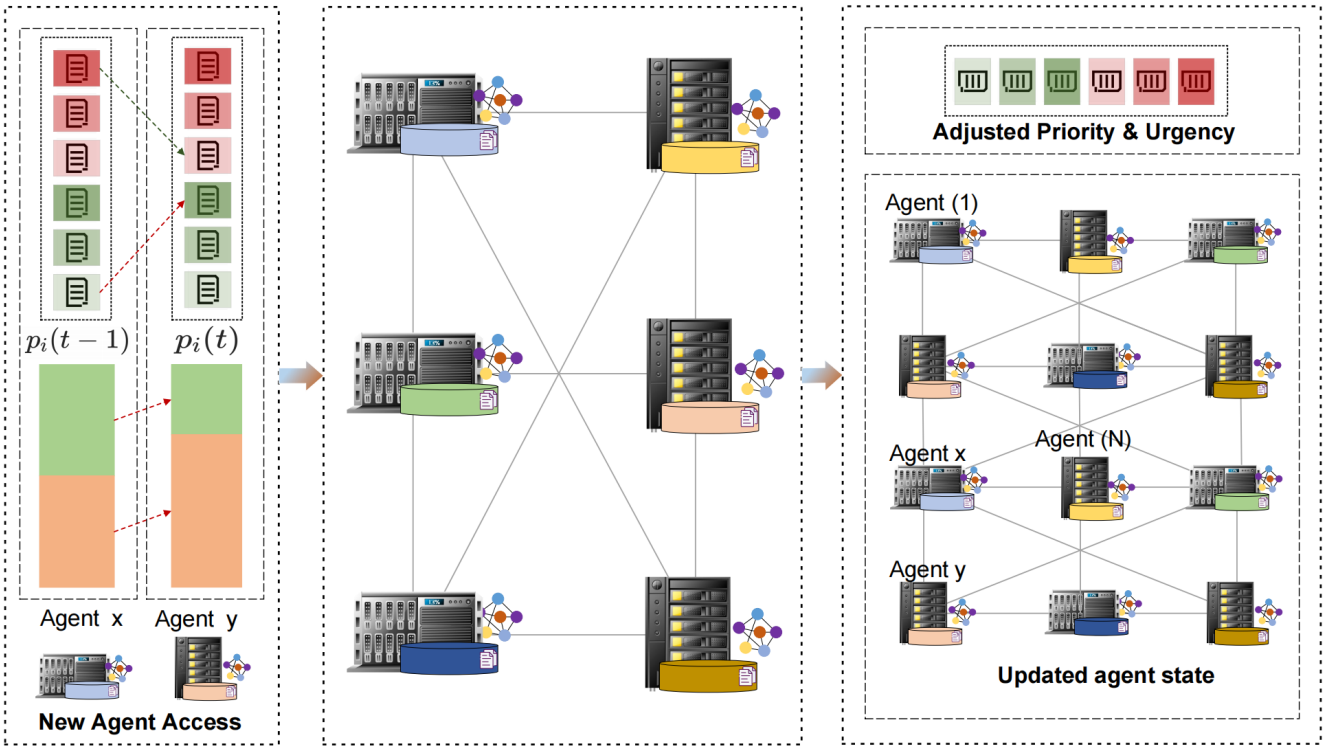

Figure 4. Dynamic Adjustment Mechanism

changes sharply, i.e., $|p_i(t) - p_i(t-1)| > \epsilon_p$. Second, the average deadline urgency of pending subtasks exceeds d^{th}, i.e., $\text{mean}_{T_{il} \in B_t} d_{il}(t) > d^{\text{th}}$. Third, the remaining computing or storage capacity changes by more than 20% between two adjacent cycles, i.e., $|C_j^{\text{remain}}(t) - C_j^{\text{remain}}(t-1)| > 0.2C_j$ or $|S_j^{\text{remain}}(t) - S_j^{\text{remain}}(t-1)| > 0.2S_j$. Fourth, a new computing or storage agent joins the system. Here, $p_i(t)$ denotes the normalized scheduling priority after considering raw business priority, batch-level normalization, and newly inserted urgent tasks, rather than the immutable raw priority p_i^0. Under this mechanism, agents exchange updated state information, and the algorithm recalculates the subtask priority $P_{il}(t)$ accordingly. These dynamic adjustments form a closed loop through which agents adapt to environmental changes and improve the resource allocation policy over repeated scheduling cycles.

5. Experiments

5.1. Experimental Setup

The experiments use a PCB surface-defect dataset [36] to construct a PCB-derived simulated scheduling workload. Based on the PCB surface-defect dataset, each image is converted into a scheduling task described by computational demand c_i, storage demand s_i, raw priority p_i^0, and deadline t_{lim}^i. The computational demand is determined by image resolution and the number of annotated defects, while the storage demand is determined by the image file size and XML annotation complexity. The raw priority is assigned according to defect severity, with solder-joint missing treated as the most critical defect, followed by bridging/offset and scratches. The priority is further adjusted using the <difficult> tag and normalized within each scheduling batch. The task deadlines are set to 100 ms, 300 ms, and 500 ms for solder-joint missing, bridging/offset, and scratches, respectively, to represent different urgency levels in PCB inspection.

The simulated edge-resource environment contains 10 resource-management agents, including six computing agents and four storage agents. Computing capacity is represented by normalized CPU-cycle units, and storage capacity is measured in MB. Computing and storage capacities are randomly generated within predefined heterogeneous ranges, and the initial remaining resources are initialized with random proportions between 0.5 and 0.8. In the simulation, one scheduling cycle is used as the basic time unit, and subtask processing time is estimated from the remaining normalized computational demand and the computing capacity allocated in the current cycle.

In the main experiment, the adjacency matrix is generated as a sparse interaction graph among resource-management agents. The communication graph uses a

ring backbone with additional cross-resource interaction edges. In the 10-agent main setting, the sparse adjacency matrix contains 14 directed active state-exchange messages per scheduling cycle, corresponding to an average directed out-degree of 1.4. If in-degree and out-degree are both counted, the average directed degree is 2.8. The communication estimate in Section 5 counts directed active state-exchange messages after sparse adjacency filtering. The topology generated under each random seed is kept fixed for all scheduling batches under that seed. In the scalability experiment, the same sparse-edge rule is maintained when the number of agents increases from 5 to 50. The computing-to-storage agent ratio is kept approximately at 3:2, consistent with the 6:4 setting in the main experiment; for example, the 50-agent setting contains 30 computing-resource agents and 20 storage-resource agents.

To reduce the influence of stochastic initialization and task sampling, all experiments are repeated under five independent random seeds. For the main comparison in Fig. 5–Fig. 10, MIRA is trained offline under the configuration in Table 1 for each random seed, and the trained policy is then evaluated on task batches sampled from the 1444 PCB-derived task records. A 260-record subset is additionally used for reporting training-convergence behavior. The generated task batches provide a controlled scheduling workload in which all compared algorithms are evaluated under identical workload samples and initial resource states. The training-convergence experiment uses 220 training episodes and 64 sampled tasks per training episode; the network update mini-batch size is 64 transitions. Task batches are generated by randomly sampling PCB images from the dataset and converting them into scheduling parameters according to the mapping rules described above. All algorithms use the same generated task batches and initial resource states under each random seed. The implementation executes each scheduling batch cycle by cycle until the sampled tasks are completed or their simulated deadlines are reached. Table 1 reports the agent configuration and training settings used for implementing MIRA.

5.2. Baselines, Parameters, and Metrics

We compare MIRA with two representative baselines:

1. Multi-constraint Resource Allocation Heuristic (MRAH) [37]: MRAH allocates production resources under multiple technological, temporal, and equipment constraints using a heuristic scheduling strategy;

2. Advantage Actor-Critic-based Resource Allocation (A2C-RA) [8]: A2C-RA uses an Advantage Actor-Critic policy to select task execution nodes with the objective of reducing service delay;

Regarding the algorithm-related parameters in the experiment, the initialization range of the deep neural network parameter θ_j of the MIRA algorithm is $[-0.1, 0.1]$, the discount factor ρ is 0.9, and the actor learning rate η_{actor} is 0.001. The deadline-aware weight parameters are set as follows: $\alpha_0 = 0.25$, $\beta_0 = 0.25$, $\gamma_0 = 0.30$, $\lambda_0 = 0.20$, $\eta_{\text{weight}} = 0.05$, $U_C^{\text{th}} = 0.80$, $U_S^{\text{th}} = 0.80$, $d^{\text{th}} = 0.75$, $\epsilon = 10^{-6}$, and $\epsilon_p = 0.05$. The reward weights are set to $\omega_1 = 0.30$, $\omega_2 = 0.20$, $\omega_3 = 0.30$, and $\omega_4 = 0.20$. The heuristic weights of MRAH are selected by grid search on validation batches and then fixed for all reported test batches. The learning rate of the Advantage Actor-Critic algorithm in the A2C-RA algorithm is set to 0.002, and the discount factor is 0.9, the same as that of the MIRA algorithm. A2C-RA is trained under the same number of episodes, random seeds, task batches, and initial resource states as MIRA, while using its own reward objective listed in Table 2. Since A2C-RA is an on-policy Actor-Critic baseline, the replay buffer is not used for its policy update.

All compared methods are adapted to the same computing/storage scheduling setting and evaluated under identical task batches, initial resource states, and random seeds. Therefore, the selected baselines are used to evaluate both heuristic resource allocation and single-policy Actor-Critic scheduling under the same feasible-allocation constraints.

The main evaluation metrics include resource utilization, task execution time, priority-task satisfaction ratio, and system stability. The evaluation also includes training convergence, communication overhead, scalability, and ablation analyses to examine model stability, deployment feasibility, and component contribution.

5.3. Resource Utilization

Resource utilization is evaluated by computing resource utilization $U_C = \frac{C_{\text{total}} - \sum_{j \in \mathcal{A}_C} C_j^{\text{remain}}}{C_{\text{total}}}$ and storage resource utilization $U_S = \frac{S_{\text{total}} - \sum_{j \in \mathcal{A}_S} S_j^{\text{remain}}}{S_{\text{total}}}$. In the simulated heterogeneous edge-resource setting, the abscissa is set as task load level, representing the proportional gradient between total task resource demand and total system resources (1 corresponds to 20% load, 50 corresponds to 100% load), and the ordinate is resource utilization.

Computational Resource Utilization. As shown in Figure 5, MIRA achieves higher computational resource utilization under medium and high task-load levels after introducing the deadline-aware priority update. Its utilization increases rapidly at low loads, reaches approximately 91.5% around load level 36, and remains above 86% at the highest load. This trend suggests that the deadline-aware priority update is associated

Table 1. Training and inference configuration of MIRA

Item	Setting
Number of agents	10 in the main experiment; 5–50 in the scalability experiment
Training episodes	220 episodes with 5 random seeds
Mini-batch size for network update	64 transitions
Replay buffer size	10000 transitions
Actor/Critic network structure	Two hidden layers with 128 and 64 units

Table 2. Baseline adaptation in the simulated scheduling setting

Algorithm	State/Input	Action	Reward/Objective
MRAH	$[c_i, s_i, p_i(t), t^i_{\text{lim}}]$ and remaining computing/storage resources	Assign a task to a feasible resource node	Multi-constraint heuristic score
A2C-RA	Same task/resource state as MIRA without adjacency aggregation	Select execution node and feasible resource allocation	Delay-oriented reward with timeout penalty
MIRA	Task embedding, local resource state, and adjacency-based interaction	Subtask selection and feasible resource allocation	Utilization, priority, timeout, and collaboration reward

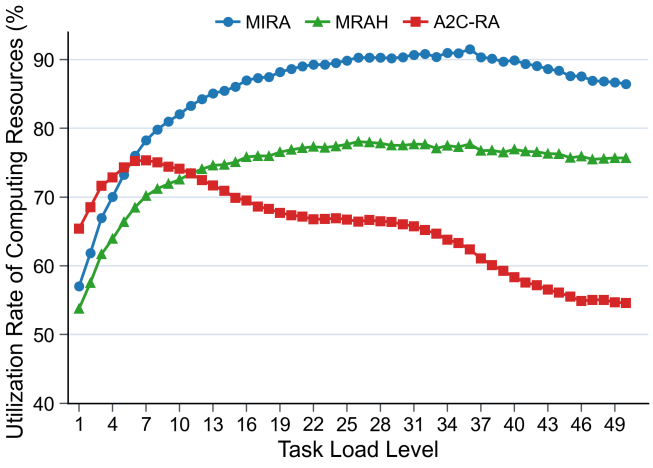

Figure 5. Computing resource utilization under different task-load levels

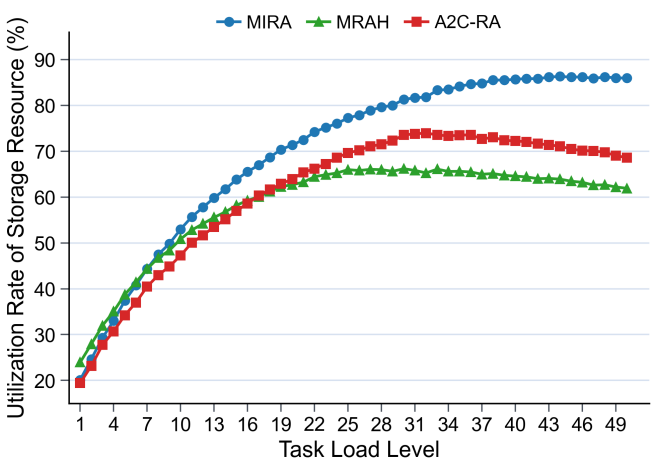

Figure 6. Storage resource utilization under different task-load levels

with earlier allocation of computing resources to high-urgency subtasks under the simulated PCB-derived scheduling workload. MRAH improves steadily but saturates earlier because its preset heuristic rules provide limited dynamic priority adaptation. A2C-RA achieves competitive utilization at very low load levels due to rapid delay-oriented node selection, but its utilization decreases under high loads because it lacks heterogeneous-resource coordination with deadline-aware priority preemption.

Storage Resource Utilization. As shown in Figure 6, in terms of storage resource utilization, MIRA gradually optimizes storage resource allocation through agent collaboration and deadline-aware task ordering. The utilization rate increases with task load and reaches about 86.3% in the high-load region, after which it remains stable with only a slight decline as the system approaches saturation. This pattern suggests that remaining-resource compensation is associated with lower fragmented residual storage. MRAH allocates storage resources quickly in the initial stage, but the curve saturates near the medium-load region and then decreases because static rules provide limited support for changing storage fragments and task urgency. A2C-RA also improves storage utilization at medium loads, but its delay-minimization objective gives limited attention to long-term storage occupation, so the utilization drops more clearly under high loads.

5.4. Execution Time and Priority Satisfaction

Average Task Execution Time. Average task execution time denotes the mean completion time of tasks within each normalized priority interval, measured in milliseconds (ms).

As shown in Figure 7, MIRA reduces average task execution time for medium- and high-priority tasks by using normalized priority and deadline urgency in the scheduling score. The execution-time curve

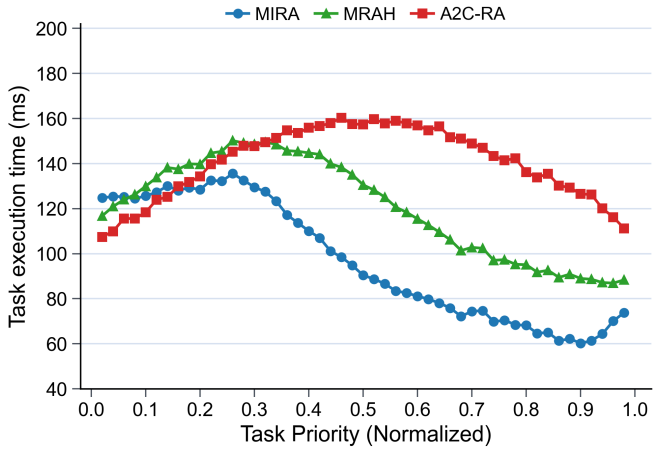

Figure 7. Average task execution time under different task-priority levels

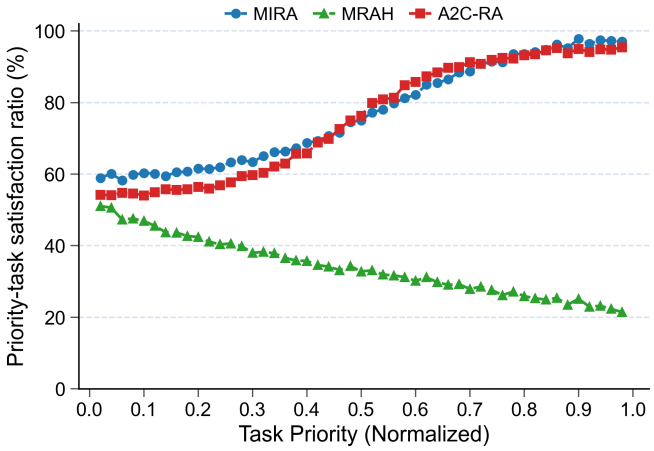

Figure 8. Priority-task satisfaction ratio under different task-priority levels

generally decreases in the medium- and high-priority regions, indicating that high-priority subtasks are scheduled earlier. A slight rebound at the highest priority level is caused by intensified competition among deadline-critical subtasks under limited edge resources. MRAH is affected b y fi xed he uristic rules in the high-priority region. A2C-RA focuses on delay reduction, but without explicit priority preemption and heterogeneous-resource coordination, its execution time remains higher than MIRA in the high-priority region.

Priority-Task Satisfaction Ratio. The priority-task satisfaction ratio measures on-time completion within each normalized priority interval. In Figure 8, tasks are grouped according to normalized priority intervals. For each interval g, the interval-level priority-task satisfaction ratio is computed as

$$\mathrm{PSR}(g) = \frac{N_{\text{on-time}}(g)}{N_{\text{total}}(g)}, \tag{10}$$

where $N_{\text{on-time}}(g)$ is the number of tasks completed before their deadlines in interval g, and $N_{\text{total}}(g)$ is the total number of tasks in that interval. For Figure 8, $\mathrm{PSR}(g)$ is multiplied by 100 and reported as a percentage. For the aggregate high-priority evaluation used in the ablation study, high-priority tasks are defined as the top 30% tasks ranked by normalized priority in each scheduling batch.

Figure 8 shows that MRAH has a relatively high priority-task satisfaction ratio when the task priority is low, but the ratio decreases as task priority increases. Its fixed allocation rules limit the number of high-priority tasks completed on time when deadline pressure increases. A2C-RA performs competitively in the medium and high task-priority regions, confirming its advantage in delay-oriented scheduling. However, MIRA maintains the highest satisfaction ratio in the high-priority region, reaching about 97%, because it explicitly couples normalized priority with deadline urgency in the scheduling score.

5.5. System Stability

System stability is measured through the fluctuation of remaining computing and storage resources during task execution. Lower variance indicates smoother resource states across scheduling cycles.

Remaining Computational Resource Variance. As shown in Figure 9, the remaining-computing-resource variance of MIRA presents stable dynamic adaptation. Across the task processing cycles, the MIRA curve remains in a narrow range and is overall lower than the baselines, suggesting that adjacency-based state exchange is associated with smoother computing-resource fluctuations. The MRAH curve increases significantly in later cycles because fixed heuristic rules provide limited compensation for changing resource pressure. A2C-RA shows stronger fluctuations, reflecting the limitations of single-policy delay optimization when heterogeneous computing resources must be coordinated over multiple cycles.

Remaining Storage Resource Variance. Figure 10 shows that MIRA achieves lower storage-resource variance through adjacency matrix interaction. Its remaining-storage-resource variance stays at a low level over the task processing cycles, suggesting that adjacency-based state exchange is associated with smoother storage-resource fluctuations. M RAH's v ariance shows an upward trend in later cycles because its heuristic strategy has limited dynamic adjustment to real-time resource requirements. A2C-RA shows advantages for low-latency-sensitive tasks, but since it focuses more

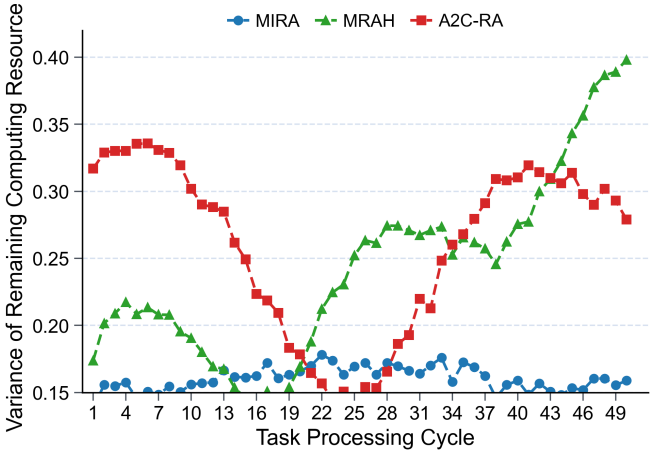

Figure 9. Variance of remaining computing resources over task processing cycles

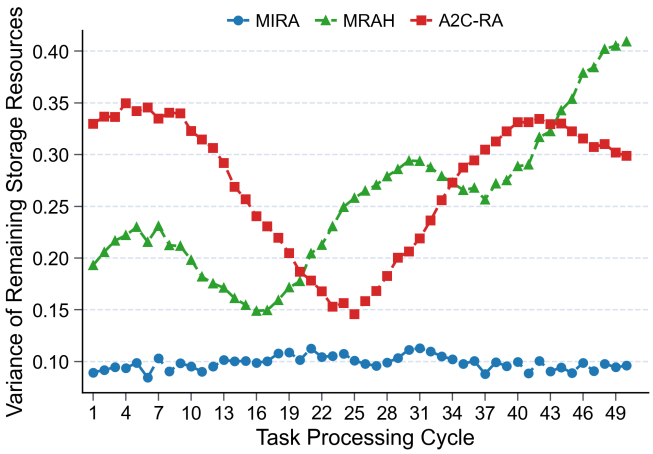

Figure 10. Variance of remaining storage resources over task processing cycles

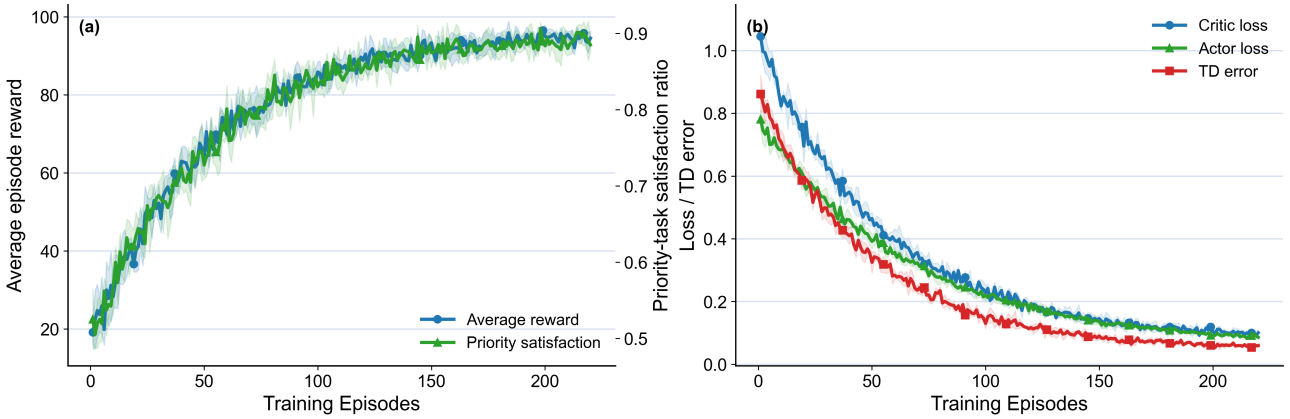

Figure 11. Training convergence of MIRA. (a) Average episode reward and priority-task satisfaction ratio during training. (b) Actor loss, Critic loss, and TD error during training

on minimizing service delay than on balanced storage allocation, local peaks still appear in the variance curve.

5.6. Training Convergence Analysis

To verify whether the MARL model can be trained stably, we record the average episode reward, priority-task satisfaction ratio, Actor loss, Critic loss, and TD error over 220 training episodes. Five random seeds are used, and each curve reports the mean value with a shaded standard deviation region. The same PCB-derived task generation process and edge-resource node generation process are used as in the main experiment.

As shown in Figure 11, the average episode reward increases rapidly in the early stage and then gradually stabilizes after approximately 160 episodes. The Actor loss, Critic loss, and TD error decrease and remain within a low range in the later stage. The shaded regions remain narrow after approximately 160 episodes, indicating limited sensitivity to random initialization and task sampling. The remaining small oscillations in the later training stage are mainly caused by dynamic resource states and exploration, rather than by divergence of the learning process.

5.7. Communication Overhead and Scalability Analysis

MIRA uses an adjacency matrix to aggregate information from directly connected agents. If each active edge transmits the compact local state vector in a scheduling cycle, the communication traffic can be estimated as

$$B_{\text{cycle}} = \sum_{(j,k)\in E_t} \text{size}(s_j), \tag{11}$$

where E_t is the set of directed active state-exchange messages after sparse adjacency filtering in scheduling cycle t. Communication overhead is estimated according to the active directed state-exchange messages in the sparse adjacency graph. Each transmitted state

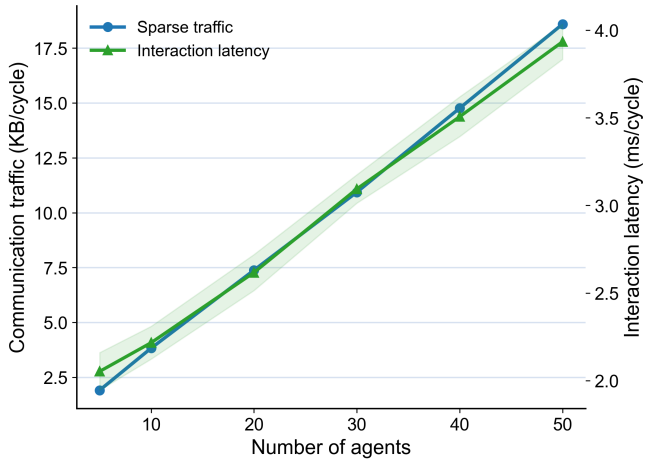

Figure 12. Communication overhead under different numbers of agents

vector contains one remaining-resource ratio, a 64-dimensional task embedding, and five task-progress fields, resulting in 280 bytes under float32 representation. Thus, the 10-agent setting with 14 active directed messages incurs $14 \times 280/1024 = 3.83$ KB/cycle. We vary the number of agents from 5 to 50 and report the measured traffic and interaction latency. Across these scalability settings, the computing-to-storage ratio is kept approximately 3:2, and the same sparse-edge rule described in the experimental setup is used for the adjacency matrix.

Figure 12 shows that the communication overhead increases with the number of agents. However, the adjacency-based sparse interaction keeps the growth close to $O(|E|d)$, where $|E|$ is the number of valid directed active messages, and d is the exchanged state-feature dimension. For example, the measured sparse traffic increases from 3.83 KB/cycle with 10 agents to 18.59 KB/cycle with 50 agents, while the interaction latency increases from 2.22 ms/cycle to 3.93 ms/cycle.

Table 3. Communication complexity and overhead comparison. Communication overhead for A2C-RA and MRAH is reported in the 10-agent main setting, and MIRA is further evaluated under 5–50 agents for scalability analysis

Method	Communication overhead description	Measured overhead
MIRA	Adjacency-based interaction; $O(\|E\|d)$ sparse and $O(m^2d)$ dense	3.83–18.59 KB/cycle
A2C-RA	No explicit multi-agent graph; lower communication	0.84 KB/cycle
MRAH	Heuristic rule exchange; minimal communication	0.42 KB/cycle

5.8. Ablation on Deadline-Aware Weighting and Interaction

To evaluate the contribution of the deadline-aware weight update and the interaction mechanisms, an ablation study is conducted under the same PCB-derived task distribution. We compare the full model with three variants: removing deadline-aware weight updating, removing adjacency-based interaction, and removing remaining-resource compensation. Table 4 reports the ablation results of the main MIRA components. The mean and standard deviation are computed over the same five random seeds used in the main experiment.

Table 4. Ablation results of the main MIRA components

Variant	Avg. time (ms)	Miss rate (%)	PSR (%)
MIRA-full	84.52 ± 0.74	4.92 ± 0.81	91.6 ± 1.1
MIRA-w/o deadline	91.58 ± 0.87	8.64 ± 0.67	87.1 ± 0.5
MIRA-w/o adjacency	94.47 ± 1.30	7.51 ± 0.54	85.5 ± 1.2
MIRA-w/o comp.	89.14 ± 1.07	7.00 ± 0.71	87.9 ± 0.7

The full MIRA model obtains the lowest average execution time and deadline miss rate while maintaining the highest priority-task satisfaction ratio. Removing the deadline-aware weight update substantially increases the deadline miss rate from 4.92% to 8.64%, which indicates that deadline urgency is an important factor in the simulated PCB-derived scheduling workload with tight latency constraints. Removing adjacency-based interaction increases execution time and lowers priority satisfaction, indicating that topology-aware agent interaction contributes to cross-resource coordination.

6. Conclusions

In this study, motivated by Industry 4.0-driven edge-enabled manufacturing, we established a resource scheduling framework for concurrent multi-tasking in mechanical manufacturing edge networks and evaluated it under a PCB-derived simulated scheduling workload. The proposed MIRA method integrates MARL, fine-grained task modeling, deadline-aware dynamic priority weighting, topology-aware agent interaction, and the "priority preemption + remaining resource compensation" strategy. Compared with MRAH and A2C-RA in this simulation setting, MIRA achieves higher computational/storage resource utilization under medium- and high-load conditions, lower average execution time for medium- and high-priority tasks, and a higher priority-task satisfaction ratio. The convergence and scalability experiments further indicate that MIRA can be trained stably.

These results suggest that MIRA is effective under the PCB-derived simulated workload and heterogeneous

edge-resource settings considered in this study. The evaluation in this study is conducted on a PCB-derived simulated scheduling workload; validation with real production traces and physical edge devices remains an important direction for future work. Future work will also further compress exchanged state messages and study privacy-preserving cross-regional manufacturing resource collaboration.

Acknowledgements This research was supported by the R&D Program of Beijing Municipal Education Commission (Grant No. KM202311417003), the Beijing Natural Science Foundation (No. 3254045) and the National Science and Technology Major Project of China (No. 2025ZD1008205).

References

[1] Ahmed N, Girija S, Baker T, Al Aghbari Z. MAESTRO: A Multilayer Architecture Based on Fog Computing and SDN for Real-Time Emergency Routing in Urban Settings. *IEEE Internet Things J.* 2026;13(4):6370–6386.

[2] Zhou H, Jiang K, Liu X, Li X, Leung VCM. Deep Reinforcement Learning for Energy-Efficient Computation Offloading in Mobile-Edge Computing. *IEEE Internet Things J.* 2022;9(2):1517–1530.

[3] Sadia H, Kamal Hassan A, Haq Abbas Z, Abbas G, Baker T, Saeed N. Maximizing Energy Efficiency in IRS-Assisted Phase Cooperative PS-SWIPT-Based Self-Sustainable IoT Network. *IEEE Open J Commun Soc.* 2025;6:4311–4327.

[4] Zaman SKu, Jehangiri AI, Maqsood T, et al. LiMPO: lightweight mobility prediction and offloading framework using machine learning for mobile edge computing. *Cluster Comput.* 2023;26:99–117.

[5] 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.

[6] Baker T, Al Aghbari Z, Khedr AM, Ahmed N, Girija S. EDITORS: Energy-aware Dynamic Task Offloading using Deep Reinforcement Transfer Learning in SDN-enabled Edge Nodes. *Internet of Things*. 2024;25:101118.

[7] Bhandari S, Vu TX, Chatzinotas S. LEO-Based Edge Computing Service Platform for Challenging Geographical Terrain. *IEEE Open J Commun Soc.* 2025;6:9991–10009.

[8] Zhu D, Li T, Tian H, Yang Y, Liu Y, Liu H, Geng L, Sun J. Speed-Aware and Customized Task Offloading and Resource Allocation in Mobile Edge Computing. *IEEE Commun Lett.* 2021;25(8):2683–2687.

[9] Shamim N, Asim M, Baker T, Pervez Z, Awad AI, Zomaya AY. Integrating system calls and position-specific scoring for enhanced anomaly detection in Internet of Things environments. *Comput Secur.* 2025;158:104613.

[10] Gebrekidan TZ, Stein S, Norman TJ. Combinatorial Client-Master Multiagent Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing. In: *AAMAS '24: Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*. 2024. p. 2273–2275.

[11] Al Aghbari Z, Khedr AM, Ahmed N, Girija S, Baker T. RedTops: real-time energy-aware dynamic task offloading via federated mountain gazelle optimisation in SDN-enhanced edge computing. *Neural Comput Applic.* 2025;37(19):13795–13833.

[12] Huang Z, Li D, Cai J, Lu H. Collective reinforcement learning based resource allocation for digital twin service in 6G networks. *J Netw Comput Appl.* 2023;217:103697.

[13] Zhou H, Elsayed M, Erol-Kantarci M. RAN Resource Slicing in 5G Using Multi-Agent Correlated Q-Learning. In: *2021 IEEE 32nd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. 2021. p. 1179–1184.

[14] Akyıldız HA, Gemici F, Hökelek I, Çırpan HA. Hierarchical Reinforcement Learning Based Resource Allocation for RAN Slicing. *IEEE Access*. 2024;12:75818–75831.

[15] Lei W, Zhao H, Liu X, Zhang Z, Xia XH, Evans S. Optimal Remanufacturing Service Resource Allocation for Generalized Growth of Retired Mechanical Products: Maximizing Matching Efficiency. *IEEE Access*. 2021;9:89655–89674.

[16] Yu H, Zhang G, Ran Y, Li M, Jiang D, Chen Y. A Reliability Allocation Method for Mechanical Product Based on Meta-Action. *IEEE Trans Reliab.* 2020;69(1):373–381.

[17] Wu SB, Baker T, Wang YJ, Tan YA, Li YZ, Ren MX. FRISC: Mitigating Privacy Leakage in Federated Learning through Frequency-domain Feature Screening. *Concurrency Comput Pract Exper.* 2026;38(6):e70640.

[18] Peng H, Shen X. Deep Reinforcement Learning Based Resource Management for Multi-Access Edge Computing in Vehicular Networks. *IEEE Trans Netw Sci Eng.* 2020;7(4):2416–2428.

[19] Kong X, Duan G, Hou M, Shen G, Wang H, Yan X, Collotta M. Deep Reinforcement Learning-Based Energy-Efficient Edge Computing for Internet of Vehicles. *IEEE Trans Ind Inform.* 2022;18(9):6308–6316.

[20] Hussien A, Maksoud A, Al-Dahhan A, Abdeen A, Baker T. Machine learning model for predicting long-term energy consumption in buildings. *Discover Internet of Things*. 2025;5(1):18.

[21] Deng X, Zhang J, Zhang H, Jiang P. Deep-Reinforcement-Learning-Based Resource Allocation for Cloud Gaming via Edge Computing. *IEEE Internet Things J.* 2023;10(6):5364–5377.

[22] Lin P, Song Q, Wang D, Yu FR, Guo L, Leung VCM. Resource Management for Pervasive-Edge-Computing-Assisted Wireless VR Streaming in Industrial Internet of Things. *IEEE Trans Ind Inform.* 2021;17(11):7607–7617.

[23] Zhang S, Gu H, Chi K, Huang L, Yu K, Mumtaz S. DRL-Based Partial Offloading for Maximizing Sum Computation Rate of Wireless Powered Mobile Edge Computing Network. *IEEE Trans Wireless Commun.* 2022;21(12):10934–10948.

[24] Li X, Qin Y, Huo J, Huangfu W. Heuristically Assisted Multiagent RL-Based Framework for Computation Offloading and Resource Allocation of Mobile-Edge Computing. *IEEE Internet Things J.* 2023;10(17):15477–15487.

[25] Hazarika B, Singh K, Biswas S, Li CP. DRL-Based Resource Allocation for Computation Offloading in IoV

Networks. *IEEE Trans Ind Inform.* 2022;18(11):8027–8038.
[26] Xiao K, Shi W, Gao Z, Yao C, Qiu X. DAER: A Resource Preallocation Algorithm of Edge Computing Server by Using Blockchain in Intelligent Driving. *IEEE Internet Things J.* 2020;7(10):9291–9292.
[27] Chai F, Zhang Q, Yao H, Xin X, Gao R, Guizani M. Joint Multi-Task Offloading and Resource Allocation for Mobile Edge Computing Systems in Satellite IoT. *IEEE Trans Veh Technol.* 2023;72(6):7783–7795.
[28] Waqar N, Hassan SA, Mahmood A, Dev K, Do DT, Gidlund M. Computation Offloading and Resource Allocation in MEC-Enabled Integrated Aerial-Terrestrial Vehicular Networks: A Reinforcement Learning Approach. *IEEE Trans Intell Transp Syst.* 2022;23(11):21478–21491.
[29] Xu J, Ai B, Chen L, Cui Y, Wang N. Deep Reinforcement Learning for Computation and Communication Resource Allocation in Multiaccess MEC Assisted Railway IoT Networks. *IEEE Trans Intell Transp Syst.* 2022;23(12):23797–23808.
[30] Yu J, Alhilal AY, Zhou T, Hui P, Tsang DHK. Attention-Based QoE-Aware Digital Twin Empowered Edge Computing for Immersive Virtual Reality. *IEEE Trans Wireless Commun.* 2024;23(9):11276–11290.
[31] Yu C, Velu A, Vinitsky E, Gao J, Wang Y, Bayen AM, Wu Y. The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022. Vol. 35.
[32] Rashid T, Samvelyan M, Schroeder C, Farquhar G, Foerster J, Whiteson S. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2018. p. 4295–4304.
[33] Sunehag P, Lever G, Gruslys A, Czarnecki WM, Zambaldi V, Jaderberg M, Lanctot M, Sonnerat N, Leibo JZ, Tuyls K, Graepel T. Value-Decomposition Networks for Cooperative Multi-Agent Learning Based on Team Reward. In: *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2018. p. 2085–2087.
[34] Liu C, Wang H, Zhao M, Liu J, Zhao X, Yuan P. Dependency-aware online task offloading based on deep reinforcement learning for IoV. *J Cloud Comput.* 2024;13:136.
[35] Chen Y, Luo X, Liang P, Han J, Xu Z. Priority-based DAG task offloading and secondary resource allocation in IoT edge computing environments. *Computing.* 2024;106:3229–3254.
[36] China Telecom, Raisecom Technology. The PCB-AoI Public Dataset [Internet]. 2022 [cited 2026 Jul 6]. Available from: https://www.kaggle.com/datasets/kubeedgeianvs/pcb-aoi
[37] Aloui A, Hadj-Hamou K. A heuristic approach for a scheduling problem in additive manufacturing under technological constraints. *Comput Ind Eng.* 2021;154:107115.