Tsinghua Science and Technology
Doi: 10.26599/TST.2026.9010079

Research Article

Adaptive AI Agent Migration via Generative Diffusion-based Reinforcement Learning in Edge Intelligence Systems

Jie Gao[†], Xingdan Wang[†], Zhiqing Tang [(✉)], Jiahui Chen, Zhi Yao, Jiong Lou, Jianxiong Guo, and Weijia Jia

Abstract Deploying Large Language Model (LLM)-based AI agents at the network edge enables real-time task handling in 6G networks. However, resource heterogeneity and user mobility necessitate agent migration to maintain Quality of Service (QoS). Unlike stateless containers, AI agents encapsulate invocation histories, planning contexts, and memory stores, making their migration considerably more complex. To address this issue, we propose AMD, an adaptive AI Agent Migration framework that integrates conditional Diffusion models with reinforcement learning to jointly optimize latency and resource utilization under dynamic edge environments. Specifically, we introduce a diffusion model as a generative prior to produce high-quality global deployment plans, effectively avoiding the local-optima problem common in conventional reinforcement learning. A reinforcement learning-based module then performs online policy refinement to accommodate real-time environmental variations. We implement AMD on a distributed system built upon AgentScope and validate it across geographically distributed edge servers. Experimental results demonstrate that AMD reduces the average task latency by 3.9%–27.6% and improves resource utilization by up to 53.3% compared to baseline strategies.

Keywords Edge Intelligence, AI Agent, Migration, Reinforcement Learning.

• Jie Gao, Xingdan Wang, and Jiahui Chen are with the Faculty of Arts and Sciences, and also with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China (e-mail: {jiegao, wangxingdan}@mail.bnu.edu.cn).
• Zhiqing Tang is with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China (e-mail: zhiqingtang@bnu.edu.cn).
• Zhi Yao is with the School of Artificial Intelligence, Beijing Normal University, Beijing 100875, China; and also with the Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Zhuhai 519087, China.
• Jiong Lou is with Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China.
• Jianxiong Guo and Weijia Jia are with the Faculty of Arts and Sciences, Beijing Normal University, Zhuhai 519087, China; and also with the Guangdong Key Lab of AI and Multi-Modal Data Processing, Beijing Normal-Hong Kong Baptist University, Zhuhai 519087, China.
[†] Equal Contribution.
[(✉)] To whom correspondence should be addressed.
Manuscript received: 27-Mar-2026; revised: 09-Jun-2026; accepted: 10-Aug-2026

1 Introduction

The rapid advancement of Large Language Models (LLMs) has accelerated the deployment of AI agents for automated task handling. While cloud-based deployment offers abundant resources for AI agent applications such as autonomous driving, intelligent robotics, and voice assistants [1], the evolution toward 6G networks with full air–land–sea coverage and stringent latency requirements poses new challenges for managing massive data streams from both terrestrial and non-terrestrial nodes. In such multi-modal scenarios, transmitting large volumes of raw data (e.g., real-time video, point clouds, and sensor readings) to the cloud incurs prohibitive delay. Processing data and executing tasks at the network edge has therefore become imperative [2].

Deploying AI agents at the network edge enables localized task planning and execution [3]. Compared to centralized cloud architectures, this paradigm offers improved scalability and connectivity [4]. Its advantages are as follows. First, it reduces data transmission distance, thereby lowering end-to-end latency [5]. Second, local processing significantly reduces backhaul traffic, saving bandwidth and lowering operational costs [6]. Furthermore, keeping data at the edge reduces the exposure of sensitive information to the public network, thereby enhancing privacy [7]. Finally, edge-deployed agents can maintain service continuity even under intermittent connectivity [8].

Despite these advantages, the systematic deployment and dynamic management of AI agents in edge intelligence systems remain underexplored [9]. While the inherent mobility of edge users necessitates active migration to maintain Quality of Service (QoS) [8, 10], existing frameworks fail to accommodate the unique characteristics of agents along two critical dimensions: data statefulness and structural topology. Mechanically, from a data and state perspective, traditional container migration primarily revolves around replicating stateless images or synchronizing generic file systems [11, 12], while general LLM-serving migration focuses on massive model weight delivery or dynamic request rescheduling [13, 14]. In stark contrast, AI agent migration introduces a high degree of statefulness, centering on highly dynamic runtime contexts that encapsulate intricate invocation histories, multi-step

planning tracks, and persistent memory stores [1]. The technical consequence of this distinction is a strict requirement for preserving cross-node semantic and logical consistency during handoffs under tightly constrained edge bandwidth, rendering coarse-grained, network-level state-recovery methods obsolete [15]. From an architectural perspective, agents do not operate as isolated microservices or independent LLM endpoints, but as heavily interconnected networks. Because of the topological dependencies inherent in multi-agent systems, agents frequently interact; thus, a single-agent migration decision propagates a cascading impact throughout the entire dependency topology, dynamically shifting the communication latency and resource profiles of all interconnected peer agents [16, 17]. This structural coupling exponentially expands the operational decision space into an exceptionally high-dimensional online sequential decision problem. Under such a massive and volatile state space, conventional heuristics and standard reinforcement learning strategies invariably suffer from sparse rewards and poor convergence, causing them to easily trap in locally suboptimal layouts [18].

The first challenge lies in optimizing AI agent placement under heterogeneous edge resource constraints [19]. Computing capacity directly affects inference speed, while storage constrains the types of agents that can be deployed. For example, multi-modal agents require servers with sufficient storage to handle large data volumes [20]. As the system scales, this becomes a high-dimensional online sequential decision problem. Conventional heuristic methods often fail to explore this large solution space effectively, and frequently converge to locally suboptimal solutions. Additionally, communication resources dictate the interaction delay among agents; latency-sensitive agents should be co-located with their dependencies, whereas data-intensive agents require nodes with ample bandwidth.

The second challenge is managing agent migration under user mobility. Preserving runtime context is essential, as historical interactions directly influence LLM-based agent responses [7, 16]. Unlike traditional container migration, agent migration primarily involves synchronizing memory and configuration, rather than transferring the entire code base. The high volatility of edge networks demands a migration policy that is both responsive and stable. Unstable policy updates may cause performance degradation and excessive migration overhead, which is unacceptable for real-time applications.

To address these challenges, we propose AMD, a novel framework for adaptive AI agent placement and migration in edge intelligence systems that integrates conditional diffusion models with reinforcement learning. We formulate the migration process by jointly modeling latency and resource costs, covering transmission, initialization, migration, and computational overheads. We implement a distributed edge intelligence system based on AgentScope and validate it on geographically distributed edge servers [1]. The experimental results show that our algorithm reduces the average latency by 3.9%–27.6% and improves resource utilization by up to 53.3% compared to baseline strategies.

This paper extends our previous work [21]. In contrast to the conference version, which relied on separate ant-colony-based heuristic algorithms (ALP and ALM) for isolated placement and migration, this manuscript introduces a comprehensive architectural, methodological, and empirical overhaul to establish a unified framework, AMD. Compared with the preliminary version, the newly developed components make this extension substantial across multiple dimensions. First, we reformulate the optimization problem into a comprehensive Markov Decision Process (MDP) to maximize long-term cumulative reward, replacing the separate conference formulations. Second, we incorporate a Graph Convolutional Network (GCN) to encode the intricate topological dependencies among agents into a global condition vector [22]. This vector guides our novel two-stage decision-making architecture: an offline conditional diffusion model that learns high-quality deployment layouts to alleviate conventional local search limitations, coupled with an online PPO-based reinforcement learning layer that refines the policy through lightweight neural network inference to drastically reduce decision latency. By combining generative planning with reinforcement learning, the framework maximizes long-term cumulative reward to pursue optimal deployment decisions. The main contributions are summarized as follows:

(1) We propose AMD, an adaptive framework for AI agent migration in edge intelligence systems, which integrates conditional diffusion models with reinforcement learning. AMD captures the dynamic characteristics of user mobility and resource fluctuations, thereby maximizing resource utilization and reducing end-to-end latency while constraining migration overhead.

(2) We introduce a diffusion model as a generative prior to generate high-quality global initial plans, effectively alleviating the local-optima problem commonly encountered by conventional reinforcement learning in complex edge environments.

(3) We implement the distributed edge AI system using AgentScope and validate the effectiveness of the proposed framework through extensive experiments on globally distributed edge nodes.

The rest of this paper is organized as follows: Section 2 describes the related work. Section 3 presents the system model and problem formulation. Section 3.4 to 3.8 details our algorithm. Section 4 outlines the experimental setup and evaluations. Finally, Section 5 concludes this paper.

2 Related Work

2.1 Agent Deployment in Edge Systems

The rapid advancement of edge intelligence has made AI agent deployment a central research topic for next-generation networked services. Qi *et al.* propose a forecasting-driven framework that enables proactive deployment via spatio-temporal prediction and incentive-compatible contracting [23]. Xiao *et al.* present a semantic-aware framework for 6G, utilizing decentralized multi-objective optimization to achieve performance gains with reduced computational cost [24]. Kong *et al.* develop a recursive offloading framework for LLM serving, using dynamic strategies to optimize resource utilization and reduce communication overhead [25]. Qayyum *et al.* propose an LLM-driven multi-agent framework for intelligent network management, deploying specialized agents according to role-specific requirements [26]. Yao *et al.* propose a decentralized offloading scheme based on mixed MAPPO for distributed decision-making [27]. Wang et al. propose a mobile edge-enabled trust evaluation model to enhance security and prevent malicious attacks in Internet of Things (IoT) environments [28]. However, most existing works rely on idealized mathematical programming or heuristic methods. These approaches often treat dynamic edge environments as static, resulting in considerable latency penalties during environmental fluctuations and resource contention. Furthermore, they struggle to identify high-quality near-global optima in large-scale state spaces, frequently settling for local solutions.

2.2 Dynamic Migration and Mobility

Given the dynamic nature of edge environments, recent research has shifted from static placement to active migration management in order to maintain Quality of Service (QoS). Zheng *et al.* propose a cloud-edge collaborative framework that combines semantic constraint identification with topology-aware PPO to address dynamic service requirements [29]. Sun *et al.* design an LLM serving system that supports runtime request rescheduling through live migration for improved load balancing [15]. Zhang *et al.* propose a mobility-aware cooperative DRL approach with trajectory prediction for joint offloading and resource allocation [30]. Gao *et al.* apply multi-agent reinforcement learning to collaborative decision-making in dynamic environments, providing useful insights for agent migration [31]. Zhong *et al.* propose a contract-based method for vehicular agent migration combined with a generative diffusion algorithm [32]. Although these strategies improve mobility support, they typically model AI agents as stateless containers, overlooking the substantial overhead of context synchronization and memory transfer inherent in stateful agent migration.

2.3 Generative RL for Decision Making

Generative models are effective at capturing complex probability distributions, while reinforcement learning (RL) enables policy refinement through environmental interaction. A recent comprehensive survey by Xu *et al.* established a systematic taxonomy for diffusion-based RL, highlighting its effectiveness in trajectory-level planning and decision-making [33]. Zeng *et al.* propose an improved Double Deep Q Network (DQN) for task scheduling, utilizing a modified experience replay mechanism to enhance learning efficiency [34]. Chen *et al.* propose a federated DRL approach for collaborative vehicular task offloading [35]. Bi *et al.* present a method that integrates hybrid Q-networks with intent recognition for coordinated decision-making [36]. Yang *et al.* employ a diffusion-based algorithm to enhance solution space exploration in task offloading [37]. Zhai *et al.* develop a step-level Q-value model for LLM agents, leveraging Direct Preference Optimization (DPO) to guide action selection [38]. However, existing RL methods often suffer from sparse rewards and poor convergence in high-dimensional state spaces, leading to inefficient exploration. Conversely, while purely generative approaches can effectively capture global distributions, they lack the ability to perform real-time corrections in response to transient environmental variations.

3 System Model and Problem Formulation

3.1 System Model

As shown in Fig. 1, our system adopts a closed-loop architecture that spans from task input to physical execution, designed for real-time agent migration in edge intelligence environments. First, the User-Task Layer converts requests from edge users into input signals. The AMD Framework Layer continuously monitors the resource status of the Edge Infrastructure Layer and converts the collected data into contextual information. Based on this context, the algorithmic module generates an adaptive migration decision, which is then dispatched to the infrastructure layer for execution. The infrastructure layer executes the migration across distributed edge servers and feeds back status updates, forming a closed-loop optimization cycle. The key system components are defined as follows.

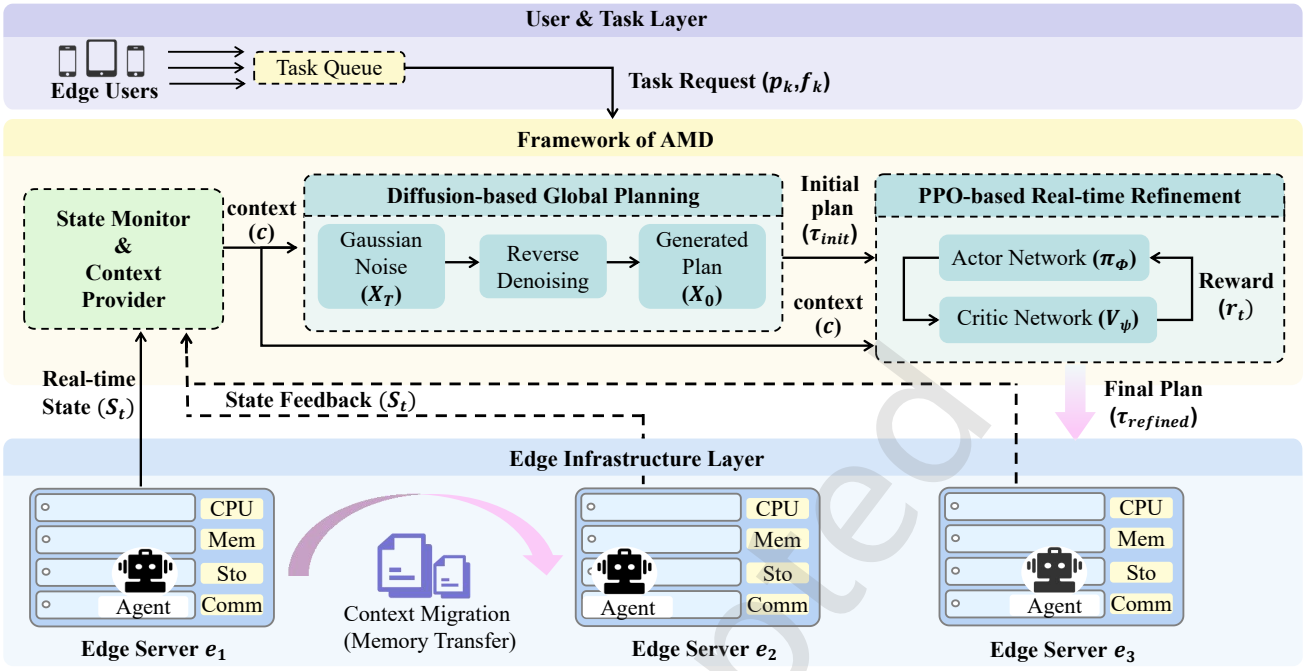


Fig. 1 An overview of our system model.

AI Agent: Let the AI agent set be $\mathbf{A} = \{a_1, a_2, \ldots, a_{|\mathbf{A}|}\}$, where $|\mathbf{A}|$ denotes the total number of agents. Each AI agent a is characterized by its computing resource requirement Q_a^{cpu} (CPU cycles), communication requirement Q_a^{comm} (Mbps), memory requirement Q_a^{mem} (GB), and storage requirement Q_a^{sto} (GB).

Each AI agent is defined not only by executable code but also by its evolving state. Unlike stateless microservices, agent migration requires encapsulating its reasoning process. As detailed in Table 1, the migratable state spans its persistent memory store and invocation/planning history, together with the agent configuration and decision metadata required to reconstruct the agent on the destination server.

Table 1 Migration management components of AI agents

Category	Component
Agent Configuration	System Prompt Model Config Memory Flag
Runtime State	Persistent Memory Store Invocation & Planning History
Deployment Info	Server Assignment Network Port
Decision Metadata	Dependency Matrix Resource Profile

Edge System: The edge server set is defined as $\mathbf{E} = \{e_1, e_2, \ldots, e_{|\mathbf{E}|}\}$. Each edge server $e \in \mathbf{E}$ has resource capacities denoted as Q_e^{cpu}, Q_e^{comm}, Q_e^{mem}, and Q_e^{sto}. Furthermore, let $\mathbf{A}_e$ denote the set of AI agents running on server e, and $e(a)$ denote the host server of agent a.

Task: The active task set is defined as $\mathcal{K} = \{k_1, k_2, \ldots, k_{|\mathcal{K}|}\}$. Each task k consists of a prompt p_k and a file f_k, and requires a set of AI agents $\mathbf{A}_k$. S_k^{sto} denotes the storage space required for task k.

Utilization: For any edge server $e \in \mathbf{E}$, $U_{cpu,e}$ and $U_{mem,e}$ denote its real-time CPU and memory utilization ratios, respectively, calculated as the ratio of allocated resources to the total capacities Q_e^{cpu} and Q_e^{mem}.

3.2 Cost Model

3.2.1 Latency Model

The end-to-end latency of a task comprises file transfer, agent orchestration (startup and migration), task processing, and inter-agent communication.

1) Transmission Latency: It includes file upload and prompt transmission, and is defined as:

$$T_k^{\text{tra}} = \frac{S_k^{\text{file}}}{\beta_e} + T_p\,, \tag{1}$$

where S_k^{file} represents the file size, β_e denotes the available bandwidth of the edge node, and T_p is the prompt transmission time.

2) Initiation Latency: The initiation latency accumulates the startup time of all required AI agents, which depends on

the current load of each host server. It is defined as:

$$T_k^{\text{init}} = \sum_{a \in \mathbf{A}_k} T_{\text{init}}(|\mathbf{A}_{e(a)}|)\,. \tag{2}$$

3) Migration Latency: Migration involves memory export, transfer, and reload. Let $e_s(a)$ and $e_d(a)$ denote the source and destination servers of agent a, respectively. The memory transfer time for agent a migrating from $e_s(a)$ to $e_d(a)$ is:

$$T_a^{\text{trans}} = \frac{Q_a^{\text{mem}}}{\beta_{e_s(a),e_d(a)}}\,, \tag{3}$$

where $\beta_{e_s(a),e_d(a)}$ denotes the link bandwidth between the two servers. The total migration latency for task k is:

$$T_k^{\text{mig}} = \sum_{a \in \mathbf{A}_k^{\text{mig}}} \left(T_{e_s(a)}^{\text{export}} + T_a^{\text{trans}} + T_{e_d(a)}^{\text{init}} + T_{e_d(a)}^{\text{load}}\right)\,, \tag{4}$$

where $\mathbf{A}_k^{\text{mig}}$ is the subset of agents requiring migration. T^{export}, T^{init}, and T^{load} represent the time for memory export, agent re-initialization at the destination server, and memory loading, respectively.

Orchestration Cost: We define the orchestration cost as the sum of initiation and migration latencies:

$$T_k^{\text{orch}} = T_k^{\text{init}} + T_k^{\text{mig}}\,. \tag{5}$$

This metric measures the total time from placement decision to agent readiness, covering startup queuing under current server load and cross-server migration. For static strategies without migration, $T_k^{\text{mig}} = 0$, but the orchestration cost may still be high when resource contention on the assigned servers delays agent initialization.

4) Processing Latency: This component covers file processing and LLM inference:

$$T_k^{\text{pro}} = T_{\text{file}} + T_{\text{LLM}}\,, \tag{6}$$

where T_{file} denotes the file processing time and T_{LLM} denotes the inference latency of the LLM.

5) Inter-Agent Communication Latency: When interdependent agents are placed on different servers, communication latency is incurred for coordinating their execution. This component captures the placement-sensitive portion of end-to-end delay:

$$T_k^{\text{comm}} = \sum_{a_i,a_j \in \mathbf{A}_k} \mathbf{M}_{ij} \cdot \mathbb{I}(\tau_{a_i} \neq \tau_{a_j}) \cdot \mathrm{L}(\tau_{a_i}, \tau_{a_j})\,, \tag{7}$$

where $\mathbf{M}_{ij}$ is the dependency indicator between agents a_i and a_j; τ_{a_i} denotes the server hosting agent a_i; and $\mathrm{L}(\cdot)$ is the network latency function between servers. The total task latency is:

$$T_k = T_k^{\text{tra}} + T_k^{\text{orch}} + T_k^{\text{pro}} + T_k^{\text{comm}}\,. \tag{8}$$

3.2.2 *Resource Cost*

The resource cost quantifies the consumption of computing, storage, and communication resources.

1) Computing Cost: This is defined as the server-side occupation time, i.e., the total duration during which the task occupies computing resources on the edge server. It excludes the network transmission latency T_k^{tra} (which consumes bandwidth rather than server CPU) and the aggregate memory transfer time $T_k^{\text{trans}} = \sum_{a \in \mathbf{A}_k^{\text{mig}}} T_a^{\text{trans}}$ (which is bounded by inter-server link bandwidth). Formally:

$$O_k^{\text{cpu}} = T_k - \left(T_k^{\text{tra}} + T_k^{\text{trans}}\right)\,. \tag{9}$$

2) Storage Cost: This accounts for the storage consumed by task files and agent memory:

$$O_k^{\text{sto}} = S_k^{\text{sto}} + \sum_{a \in \mathbf{A}_k} Q_a^{\text{mem}}\,. \tag{10}$$

3) Communication Cost: Interdependent agents incur bandwidth consumption for inter-agent communication:

$$O_k^{\text{comm}} = \sum_{a \in \mathbf{A}_k} W_a\,, \tag{11}$$

where W_a denotes the communication overhead of agent a. The total resource cost is:

$$O_k = O_k^{\text{cpu}} + O_k^{\text{sto}} + O_k^{\text{comm}}\,. \tag{12}$$

3.2.3 *Constraints*

For any edge server e, the aggregated resource demands of all hosted agents must not exceed its capacity:

$$\begin{cases} \sum_{a \in \mathbf{A}_e} Q_a^{\text{cpu}} \leqslant Q_e^{\text{cpu}}, & \forall e \in \mathbf{E} \\ \sum_{a \in \mathbf{A}_e} Q_a^{\text{mem}} \leqslant Q_e^{\text{mem}}, & \forall e \in \mathbf{E} \\ \sum_{a \in \mathbf{A}_e} Q_a^{\text{sto}} \leqslant Q_e^{\text{sto}}, & \forall e \in \mathbf{E} \\ \sum_{a \in \mathbf{A}_e} Q_a^{\text{comm}} \leqslant Q_e^{\text{comm}}, & \forall e \in \mathbf{E} \end{cases} \tag{13}$$

3.3 Problem Formulation

We formulate the optimization problem as a Markov Decision Process (MDP) by treating agent placement and migration as a sequential decision-making process. The objective is to find an optimal policy π that maximizes the long-term QoS. To fully characterize this MDP, we assume a discrete-time, fully observable environment where the global system state s_t is synchronized through network-wide telemetry at discrete decision intervals of $\Delta t = 60$ s. This multi-second granularity is strategically chosen to filter out transient resource fluctuations and suppress policy chattering, while maintaining operational agility against persistent environment dynamics. The state transition probability $P(s_{t+1}|s_t, a_t)$ is modeled as a hybrid process defined by a transition function $s_{t+1} = f(s_t, a_t, \xi_t)$. Here, the structural agent-to-server routing layout transitions deterministically in direct response to the executed migration action a_t, whereas the underlying server capacities and network states evolve stochastically, driven by exogenous

environmental entropy ξ_t such as dynamic task workloads and mobility-induced latency shifts.

Specifically, the cumulative discounted reward J is defined as:

$$J = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(t) \right], \tag{14}$$

where γ is the discount factor and $r(t)$ is the immediate reward at time step t.

Problem Analysis: The objective is to maximize the long-term cumulative reward J while balancing task efficiency and resource stability. Given the sequential and stochastic nature of the migration problem, deep reinforcement learning (DRL) provides a suitable framework for solving this MDP [18]. However, in large-scale edge systems, the high dimensionality of the state space results in sparse reward signals, often rendering conventional RL methods inefficient. Therefore, we introduce a conditional diffusion model as a generative prior. Unlike conventional search-based methods, the diffusion model operates from a global perspective. By learning the distribution of high-quality historical deployments, it transforms Gaussian noise into a structured deployment plan τ_{init}. This provides a feasible initial deployment. This initial plan is then refined online using a PPO-based policy layer. The clipping mechanism in PPO constrains policy update magnitudes between iterations, promoting stable convergence. This prevents policy collapse under environmental variations while preserving responsiveness.

3.4 Algorithm Settings

State: The state s provides a global view of the current system status, defined as:

$$s = \{\mathbf{V}_{server}, M_{adj}, \tau_{curr}\}, \tag{15}$$

where $\mathbf{V}_{server}$ is the server resource vector that captures real-time CPU, memory, storage, and bandwidth availability; $M_{adj} \in \{0,1\}^{|\mathbf{A}|\times|\mathbf{A}|}$ is the dependency adjacency matrix encoding the inter-agent dependencies; and τ_{curr} is the current deployment vector that maps each agent to its host server.

Action: To limit computational overhead, we restrict the action space to agents selected for migration. Let $\mathcal{A}_{mig} \subseteq \mathbf{A}$ denote the subset of agents designated for migration based on the resource availability ratio threshold η_{th}. The action at time step t assigns each migrating agent to a target server:

$$a(t) = \{\tau_i \mid i \in \mathcal{A}_{mig}, \tau_i \in \{1, \ldots, |\mathbf{E}|\}\}, \tag{16}$$

where τ_i denotes the target server index assigned to agent i.

To formalize the triggering mechanisms, the resource availability ratio η_e of server $e \in \mathbf{E}$ is quantified as:

$$\eta_e = \frac{1}{4} \sum_q \frac{Q_e^q - \mathcal{D}_{q,e}}{Q_e^q}, \tag{17}$$

where $q \in \{\text{cpu, mem, sto, comm}\}$ denotes the resource type, and $\mathcal{D}_{q,e}$ is the aggregated demand of type q on server e. A server is considered resource-constrained if its availability ratio falls below the threshold, i.e., $\eta_e < \eta_{\text{th}}$ (we set $\eta_{\text{th}} = 0.2$). Accordingly, the migration set $\mathcal{A}_{mig}$ is re-evaluated at each decision interval $\Delta t = 60$ s (consistent with the MDP decision interval). An agent a is appended to $\mathcal{A}_{mig}$ if its host server $e(a)$ satisfies $\eta_{e(a)} < \eta_{\text{th}}$, or if the user-to-server distance exceeds a threshold D_{th}. This evaluation period strategically balances migration responsiveness against communication overhead.

Reward: The immediate reward r_t is defined as the negated weighted sum of latency and resource costs:

$$r_t = -\omega \cdot (\mathcal{C}_{time} + \mathcal{C}_{res}), \tag{18}$$

where ω is a normalization factor. Note that the per-task latency model in Section 3 (Eq. (1)–(8)) characterizes the end-to-end delay of individual tasks, whereas the reward function operates at the system level to guide deployment decisions. Among the per-task latency components, transmission latency T_k^{tra} depends on the file size and the edge node's bandwidth, while processing latency T_k^{pro} varies with server capacity. Both are indirectly influenced by placement but are already accounted for through the resource overload penalty $\mathcal{P}_{over}$ and load balancing term $\mathcal{C}_{bal}$, which discourage placing agents on overloaded servers. In contrast, inter-agent communication delay is directly determined by the relative placement of dependent agents across servers. Therefore, $\mathcal{C}_{time}$ focuses on this cross-server communication cost, as defined below.

1) Time and Communication Cost ($\mathcal{C}_{time}$): This term captures the inter-agent communication latency incurred when interdependent agents are placed on different servers. It serves as a proxy for the placement-sensitive portion of end-to-end delay, since co-locating dependent agents reduces both communication overhead and migration necessity:

$$\mathcal{C}_{time} = \sum_{i=1}^{|\mathbf{A}|} \sum_{j=i+1}^{|\mathbf{A}|} \mathbf{M}_{ij} \cdot \mathbb{I}(\tau_i \neq \tau_j) \cdot \text{dist}(\tau_i, \tau_j), \tag{19}$$

where $\mathbf{M}_{ij}$ is the dependency indicator; τ_i denotes the server assignment of agent i; and $\text{dist}(\cdot)$ is the transmission cost function.

2) System Stability Cost ($\mathcal{C}_{res}$): This term penalizes resource overload and load imbalance:

$$\mathcal{C}_{res} = \mathcal{P}_{over} + \mathcal{C}_{bal}, \tag{20}$$

where $\mathcal{P}_{over}$ is the resource overload penalty. The total re-

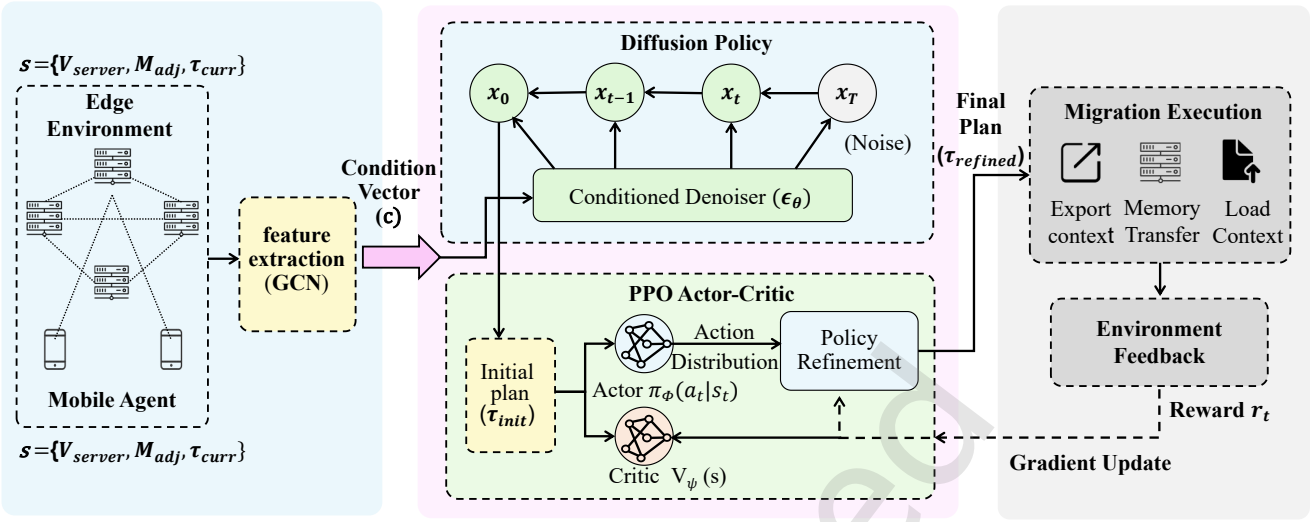


Fig. 2 AMD algorithm framework.

source demand on each server should not exceed its capacity. The penalty is defined as:

$$\mathcal{P}_{over} = \eta_{pen} \cdot \sum_{e \in \mathbf{E}} \sum_{q} \max\left(0, \mathcal{D}_{q,e} - Q_e^q\right), \tag{21}$$

where η_{pen} is the penalty coefficient, $q \in \{\text{cpu, mem, sto, comm}\}$ denotes the resource type (consistent with the four resource dimensions defined in Section 3.1), and $\mathcal{D}_{q,e} = \sum_{a \in \mathbf{A}_e} Q_a^q$ is the aggregated demand on server e.

$\mathcal{C}_{bal}$ is the load balancing cost, which quantifies the deviation of resource utilization across servers. We focus on CPU and memory as the primary balancing targets, since they are the dominant bottleneck resources for LLM inference workloads and exhibit the highest variance across servers in practice. Storage and communication utilization are already constrained by $\mathcal{P}_{over}$:

$$\mathcal{C}_{bal} = \sigma(\mathbf{U}_{cpu}) + \sigma(\mathbf{U}_{mem}), \tag{22}$$

where $\sigma(\cdot)$ denotes the standard deviation across all servers.

In summary, maximizing the system-level reward inherently minimizes the micro per-task latency through explicit proxy mappings. Specifically, minimizing $\mathcal{C}_{time}$ directly suppresses the cross-server communication latency T_k^{comm} by enforcing agent co-location. Meanwhile, the stability cost $\mathcal{C}_{res}$ acts as a bottleneck regularizer: the overload penalty $\mathcal{P}_{over}$ eliminates saturated node assignments to tightly bound the queuing and context-loading stalls that inflate the orchestration cost (T_k^{orch}), while the load-balancing term $\mathcal{C}_{bal}$ mitigates localized resource contention to reduce the LLM inference tail latency within T_k^{pro}. Therefore, optimizing this macro reward metric mathematically guarantees the systematic reduction of the comprehensive end-to-end task latency and implicitly curbs migration overhead, where $\mathcal{C}_{\text{time}}$ avoids redundant migration by reducing cross-server agent separation and $\mathcal{C}_{\text{res}}$ suppresses overload-caused migration.

3.5 Policy

The policy $\pi(a_t|s_t)$ maps the current state s_t to a distribution over actions. As detailed later for our refined PPO scheme, the actor network additionally takes the diffusion-generated plan τ_{init} as input, so the effective policy becomes $\pi(a_t|s_t, \tau_{\text{init}})$. For each agent i requiring migration, the network independently outputs a probability vector $p_i \in [0,1]^{|\mathbf{E}|}$ over candidate servers, and the target server is sampled from this distribution. Although agents are assigned independently, the shared state s_t (which includes global resource status $\mathbf{V}_{server}$ and the current deployment τ_{curr}) implicitly encodes inter-agent resource competition, enabling the policy to account for contention across agents. We adopt a hybrid architecture that combines offline generative modeling with online reinforcement learning. Specifically, a pre-trained conditional diffusion model (parameterized by θ) provides a high-quality policy prior, which is then refined by a PPO-based actor-critic network (parameterized by ϕ and ψ, respectively).

1) Diffusion-based Policy Prior Generation

We construct a training dataset $\mathcal{D} = \{(\mathbf{x}_0^i, \mathbf{c}^i)\}_{i=1}^{N_d}$ from heuristic deployment solutions, where N_d denotes the number of training samples. To ensure diversity, we collect solutions from multiple heuristic strategies (e.g., Greedy, Round-Robin, and randomized search) under varied environmental conditions, so that $\mathcal{D}$ covers a broad region of the feasible solution space rather than concentrating around a single local optimum. The diffusion model learns the joint distribution over

these diverse solutions; at inference time, the subsequent PPO refinement further corrects any residual sub-optimality in the generated initial plans. Each deployment decision $\mathbf{x}_0$ is represented as a one-hot encoding in a $|\mathbf{E}|$-dimensional space. For each agent, the target server label is embedded into a continuous vector. The environment condition vector $\mathbf{c} \in \mathbb{R}^{d_{cond}}$ is constructed by aggregating server resource features. We model the multi-agent interaction topology as an unweighted, undirected dependency graph, where the adjacency matrix $\mathbf{M}_{\text{adj}} \in \{0,1\}^{|\mathbf{A}|\times|\mathbf{A}|}$ defines the presence of co-execution or communication dependencies between agent pairs. To extract these structural features, we deploy a Graph Convolutional Network (GCN) layer [22]. Crucially, the GCN's parameter weight matrix $\mathbf{W}$ is not fixed; instead, it is optimized end-to-end via backpropagation alongside the conditional noise predictor ϵ_θ by minimizing the global MSE loss in Eq. (24). This ensures that topological agent dependencies are adaptively mapped into the continuous condition vector $\mathbf{c}$ without requiring an isolated training phase.

Algorithm 1 Training of AMD

Input: $\mathbf{A}, \mathbf{E}, \mathbf{c}, T, \{\bar{\alpha}_t\}_{t=1}^T$
Output: Trained θ

```
Initialize θ and variance schedule {ᾱ_t};
Construct D = {(x_0^(i), c^(i))}_{i=1}^{N_d} with one-hot labels;
while not converged do
    Sample batch (x_0, c) ~ D, t ~ [1, T], ε ~ N(0, I);
    Update θ by minimizing Eq. (24);
end while
return θ
```

During pre-training, the diffusion model learns the mapping from environmental conditions to deployment decisions. The core component is the conditional noise predictor ϵ_θ. The network processes inputs through three branches: deployment embedding, time-step embedding $\text{Emb}(t)$, and condition projection. We use an additive fusion strategy:

$$y = f_{\mathbf{x}}(\mathbf{x}_t) + f_t(t) + f_{\mathbf{c}}(\mathbf{c})\,, \tag{23}$$

where f denotes the embedding transformation for each branch.

Given random noise $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ and time step t, the model is trained by minimizing the mean squared error (MSE):

$$\mathcal{L}_{pre} = \mathbb{E}_{\mathbf{x}_0,\epsilon,t,\mathbf{c}}\left[\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t, \mathbf{c})\|^2\right]\,, \tag{24}$$

where $\mathbf{x}_t$ is the noisy sample obtained via the reparameterization trick:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon\,. \tag{25}$$

We pre-train the diffusion model on a dataset of $N_d = 1,000$ synthetic deployment scenarios to ensure robust generalization. Each training sample is constructed by pairing a multi-agent dependency graph containing 2–8 nodes with randomly generated unweighted, undirected edges along with 2–5 edge servers featuring heterogeneous resource capacities. To guarantee label quality and diversity, deployment targets $\mathbf{x}_0$ are generated by mixing three distinct heuristics with equal probability: a resource-aware greedy strategy that sorts agents by their footprint and sequentially maps them to minimize immediate costs; a cyclic Round-Robin strategy to maximize structural server exploration; and a randomized local search to capture non-trivial feasible regions. During training, these labels are converted to one-hot target vectors, and the condition $\mathbf{c}$ concatenates GCN topology embeddings with server resource metrics, which are subsequently jointly optimized by minimizing the mean squared error (MSE) loss.

At inference time, the model starts from Gaussian noise $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$ and performs T reverse denoising steps. The state transition at each step is:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t, \mathbf{c})\right) + \sigma_t\mathbf{z}\,, \tag{26}$$

where $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$ for $t > 1$ and $\mathbf{z} = 0$ for $t = 1$. The final output $\mathbf{x}_0$ is discretized via a per-agent argmax over the softmax outputs to produce the initial plan τ_{init}. This serves as a policy prior for the subsequent online refinement, constraining the exploration to feasible regions of the solution space.

While the diffusion model is trained on data derived from multiple heuristics, it is not a mere mimicry of individual strategies. By learning the joint distribution across diverse solutions under various environmental conditions, the model captures the underlying feasible solution manifold rather than a single suboptimal mapping. This generative approach offers two advantages. First, it performs distributional smoothing, effectively filtering out the structural noise and local-optima biases inherent in individual heuristic samples. Second, the diffusion model acts as a global strategist that interpolates between heuristic strengths, identifying high-quality deployment patterns that single heuristics might overlook in high-dimensional spaces. This provides a near-optimal prior that significantly narrows the search space for subsequent PPO refinement.

2) PPO-based Online Policy Refinement

To adapt to real-time environmental changes, we employ PPO for online policy refinement. The diffusion-generated plan τ_{init} is incorporated into the actor network as follows: τ_{init} is concatenated with the current state s_t to form the actor's input, so that the network can observe the diffusion-suggested placement while deciding the final action. This

design allows PPO to treat τ_{init} as an informed starting point and refine it based on real-time observations. Starting from this augmented input, the actor network explores actions to maximize the cumulative reward J. The probability ratio between the current and previous policies is defined as:

$$\rho_t(\phi) = \frac{\pi_\phi(a_t|s_t, \tau_{\text{init}})}{\pi_{\phi_{old}}(a_t|s_t, \tau_{\text{init}})}. \tag{27}$$

We employ Generalized Advantage Estimation (GAE) to compute the advantage $\hat{A}_t$. The Temporal Difference (TD) error δ_t is:

$$\delta_t = r_t + \gamma V_\psi(s_{t+1}) - V_\psi(s_t), \tag{28}$$

where V_ψ is the value network. The advantage $\hat{A}_t$ is:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \tag{29}$$

where γ is the discount factor and $\lambda \in [0, 1]$ is the GAE parameter that balances bias and variance. The actor is updated by maximizing the clipped surrogate objective:

$$L^{\text{clip}}(\phi) = \hat{\mathbb{E}}_t\Big[\min\big(\rho_t(\phi)\hat{A}_t, \text{clip}(\rho_t(\phi), 1-\epsilon_{\text{clip}}, 1+\epsilon_{\text{clip}})\hat{A}_t\big)\Big] \tag{30}$$

The value network is updated by minimizing the MSE loss:

$$\mathcal{L}(\psi) = \hat{\mathbb{E}}_t\left[(V_\psi(s_t) - \hat{R}_t)^2\right], \tag{31}$$

where $\hat{R}_t$ denotes the empirical discounted return. The clipping mechanism constrains the magnitude of policy updates, ensuring stable convergence and preventing the policy from deviating excessively from the prior τ_{init} due to transient environmental noise.

During diffusion sampling, we employ action masking, where infeasible server assignments that violate capacity constraints receive a score of $-\infty$ and are never selected. For RL training, we use a penalty-based approach: the reward function (Eq. (18)) includes $\mathcal{P}_{over}$ which imposes high costs on constraint violations, enabling the policy to learn feasible placements through gradient feedback. At execution time, the actor network outputs server assignments directly via argmax, relying on the learned policy to avoid infeasible actions.

3.6 Training of AMD

Algorithm 1 describes the training process. Firstly, the model parameters θ and the variance schedule $\bar{\alpha}_t$ are initialized. Then, a conditional dataset containing one-hot labels is constructed. Batch data $(\mathbf{x}_0, \mathbf{c})$ is sampled from the dataset, and a random time step $t \in [1, T]$ and noise $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ are randomly selected. The denoising network parameters θ are updated by minimizing the mean square error loss (Eq. (24)) until convergence.

3.7 AMD Execution

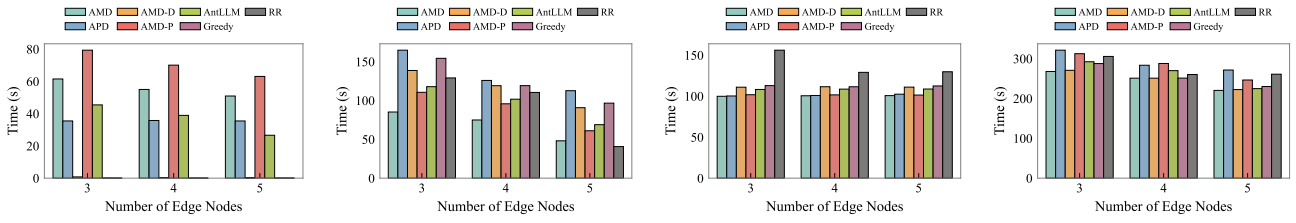
The end-to-end operational workflow of the AMD framework follows a logical sequence of perception, planning, refinement, and execution. First, the State Monitor continuously collects real-time resource metrics from edge servers and monitors inter-agent dependencies. Subsequently, GCN-based feature extraction is employed to encode the global environment and topological characteristics into a condition vector. Guided by this context, the conditional diffusion model generates a structured initial deployment plan τ_{init} through a reverse denoising process, effectively avoiding local optima from a global perspective. Then, the PPO-based refinement module integrates the initial plan with real-time observations to perform online policy fine-tuning, yielding the final decision $\tau_{refined}$. Finally, the system executes the agent migration, synchronizing runtime contexts via memory export, transfer, and reloading to ensure service continuity.

Algorithm 2 AMD Execution

Input: $\mathbf{A}, \mathbf{E}, s_t, \theta, \phi, W$
Output: τ_{refined}

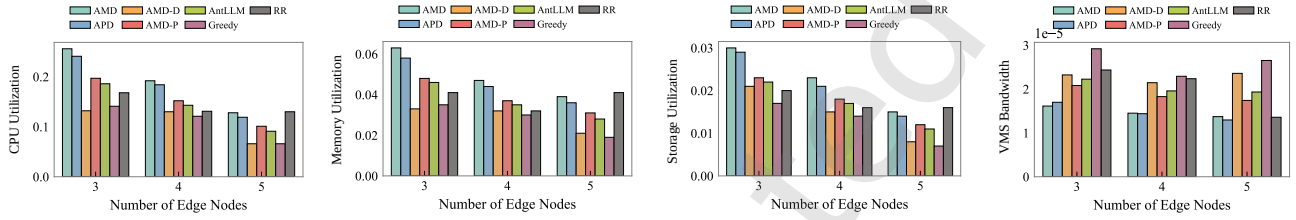
1: **while** system is running **do**
2: Monitor server resources $[Q_e^{\text{cpu}}, Q_e^{\text{mem}}, Q_e^{\text{sto}}, Q_e^{\text{comm}}]$ and compute resource availability ratio η_e per server;
3: **if** Initial placement OR $(\exists e : \eta_e < \eta_{\text{th}})$ OR user mobility triggers **then**
4: Generate X and $\tilde{M}$ from topology;
5: Compute embedding H via GCN (Eq. (32));
6: Form condition $\mathbf{c}$ by concatenating H and resource metrics;
7: Sample noise $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$;
8: **for** $t = T, \ldots, 1$ **do**
9: Denoise $\mathbf{x}_{t-1}$ using Eq. (26);
10: **end for**
11: Map $\mathbf{x}_0$ to initial plan: $\tau_{\text{init}} = \arg\max(\mathbf{x}_0)$ per agent;
12: Generate action distribution π via actor network with input $(s_t, \tau_{\text{init}})$;
13: Determine final plan τ_{refined} from π;
14: Execute migration and context transfer;
15: **end if**
16: **end while**
17: **return** τ_{refined}

Algorithm 2 describes the runtime execution procedure, which continuously monitors resource status. In Line 2, the system collects resource metrics (CPU, memory, storage, bandwidth) and evaluates the resource availability ratio η_e. When initial placement or migration is triggered, the system constructs the agent feature matrix $X \in \mathbb{R}^{|\mathbf{A}| \times d}$, where each row contains the resource requirements of an agent (i.e., $[Q_a^{\text{cpu}}, Q_a^{\text{comm}}, Q_a^{\text{mem}}, Q_a^{\text{sto}}]$), and the augmented adjacency matrix $\tilde{M} = M_{adj} + I$ (adding self-connections to the



(a) Decision Cost (b) Orchestration Cost (c) Execution Time (d) Total Time

Fig. 3 Delay performance with different numbers of servers.



(a) CPU Utilization (b) Memory Utilization (c) Storage Utilization (d) VMS Bandwidth

Fig. 4 Resource utilization and VMS bandwidth with different numbers of servers.

dependency graph). The system then computes topological features using the GCN (Line 5):

$$H = \sigma(\tilde{D}^{-1/2}\tilde{M}\tilde{D}^{-1/2}XW)\,, \tag{32}$$

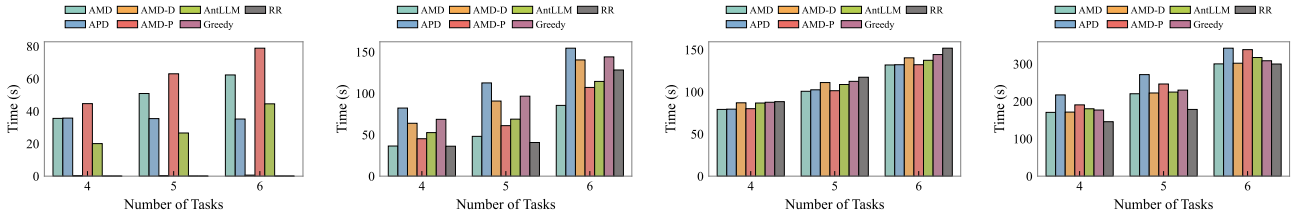
where $\tilde{D}$ is the degree matrix of $\tilde{M}$ (i.e., $\tilde{D}_{ii} = \sum_j \tilde{M}_{ij}$) and W is the learnable weight matrix. This embedding is concatenated with resource vectors to form the condition $\mathbf{c}$. Lines 7–10 perform reverse denoising to generate the initial plan τ_{init}. The actor network then refines this initial plan based on the current state, producing the action distribution π (Line 12). The refined plan τ_{refined} is then executed, and agent migration is carried out accordingly. The framework of the AMD algorithm is illustrated in Fig. 2.

3.8 Complexity Analysis

1) Diffusion Sampling Complexity: The diffusion model first computes the GCN embedding (Eq. (32)) with time complexity $O(N^2 d)$, where d is the feature dimension. In our implementation, we set the number of denoising steps $T = 1000$ and the denoiser hidden dimension $h = 256$. Each denoising step requires noise predictor inference with complexity $O(N \cdot h)$. The overall time complexity is therefore $O(N^2 d + T \cdot N \cdot h)$. The space complexity is $O(|\theta| + N \cdot d + \dim(\mathbf{c}))$, determined by parameter storage and intermediate feature maps.

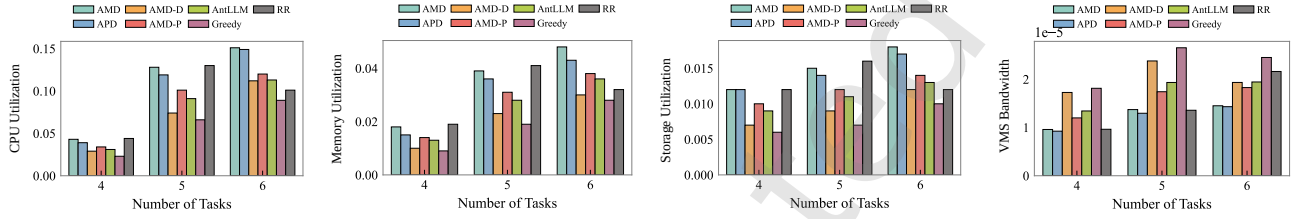
2) PPO Fine-Tuning Complexity: The PPO module updates after collecting $R = 256$ rollout steps during online execution to adapt to environmental changes. The actor-critic network has a hidden dimension of 128. Each PPO update epoch processes mini-batches of size $B = 128$ and runs for $K = 10$ epochs. Each step involves actor-critic network inference with complexity $O(N \cdot h)$ and reward computation over the adjacency matrix with complexity $O(N^2)$. The total time complexity per PPO update cycle is scaled as $O(R \cdot (N \cdot h + N^2) + K \cdot (R/B) \cdot (N \cdot h + N^2))$. The space complexity for storing trajectories is $O(R \cdot (d_s + d_a))$, where d_s is the state dimension and d_a is the action dimension.

3) Runtime Scaling with Server Count: The relationship between server count E and decision cost is twofold. On one hand, as E increases, the action space for agent placement expands as $O(E^{|\mathcal{A}_{\text{mig}}|})$, which would increase computational overhead. On the other hand, in the runtime system, more edge servers expand the cluster capacity and relax allocation constraints. Since AMD heavily relies on action masking to filter out resource-violating actions during both iterative denoising steps (T) and PPO update epochs (K), a larger E broadens the feasible solution space and drastically reduces the constraint-verification overhead. This provides an intuitive explanation for the decreasing decision cost observed in Fig. 3(a) as the server count grows to 5. In the runtime system, the constraint-relaxation benefit outweighs the polynomial growth of the action space.



(a) Decision Cost (b) Orchestration Cost (c) Execution Time (d) Total Time

Fig. 5 Delay performance with different numbers of tasks.



(a) CPU Utilization (b) Memory Utilization (c) Storage Utilization (d) VMS Bandwidth

Fig. 6 Resource utilization and VMS bandwidth with different numbers of tasks.

4 Evaluation

4.1 Experimental Settings

User trajectory data is obtained from the GeoLife Trajectories 1.3 dataset. The task workload consists of 100 independent document-processing tasks. Each task invokes 2–4 AI agents depending on the document type, and the total number of active agents ranges from 8 to 24 across experiments. Edge servers are deployed on Huawei Cloud in Beijing, Shanghai, Guiyang, and Singapore. To meet the computational requirements of AI agents, servers in Beijing and Shanghai are configured with 2 cores and 4 GB RAM, while those in Guiyang and Singapore have 2 cores and 2 GB RAM. Storage and bandwidth are configured as 50 GB / 30 Mbps for Beijing and Shanghai, and 40 GB / 20 Mbps for Guiyang and Singapore. Inter-server distances are calculated using Euclidean distance.

We implement the AMD framework using PyTorch and AgentScope. For the generative prior, the diffusion model uses a 3-layer MLP as the conditioned denoiser. For the online refinement stage, the PPO hyperparameters are summarized in Table 2. To prevent unstable migration decisions during online policy updates, AMD employs three mechanisms. First, the PPO clipping parameter $\epsilon = 0.2$ constrains the policy update magnitude, preventing drastic deviations from the diffusion-generated initial plan. Second, gradient clipping (max norm=0.5) mitigates gradient explosion during backpropagation. Third, advantage normalization stabilizes training by centering and scaling advantage estimates. Additionally, the policy is initialized from the diffusion model's high-quality deployment plan rather than random initialization, reducing exploration variance and accelerating convergence.

Table 2 PPO Online Refinement: Reproducible Hyperparameters

Item	Setting
Network Structure	State $\rightarrow$ 128 (hidden) $\rightarrow$ Actor/Critic
Optimizer	Adam
Learning Rate	1×10^{-4}
Discount Factor γ	0.99
GAE λ	0.95
Clip Parameter ϵ	0.2
Trajectory Length	256 steps
Update Interval	Per 256-step rollout
PPO Epochs	10
Batch Size	128

Baselines: We evaluate performance against the following baselines:

(1) APD: A static variant of our framework that disables dynamic migration, used to evaluate the benefit of mobility-aware migration.

(2) AMD-P: An RL-only variant based on policy gradient, which directly optimizes the policy network without diffusion-based initialization.

(3) AMD-D: A variant that uses the trained diffusion model to generate deployment plans through iterative denoising, without RL fine-tuning.

(4) AntLLM: Combines ant colony optimization with LLM-assisted post-optimization for placement and migration.
(5) Greedy: Selects the server with the highest score based on available capacity and resource requirements.
(6) RR (Round-Robin): Deploys agents to servers in a cyclic order among those with sufficient resources.

4.2 Experimental Results

1) Performance with different numbers of servers

Delay Performance: Fig. 3 illustrates the average latency as the number of edge servers increases. Overall, latency decreases as the number of servers increases, due to greater resource availability. In Figs. 3 and 5, we decompose the end-to-end delay into four components. (a) Decision cost is the time the algorithm spends computing a placement or migration decision. (b) Orchestration cost covers agent preparation, including startup queuing under current server load and cross-server migration. (c) Execution time is the task processing duration. (d) Total time sums all components, including transmission and inter-agent communication latencies not shown separately.

As shown in Fig. 3(a), the decision cost of AMD-D, Greedy, and RR remains negligible, as their decision-making processes incur minimal overhead. Since APD uses a static deployment strategy without iterative optimization, its decision cost remains constant regardless of server count. In contrast, AMD, AMD-P, and AntLLM show decreasing decision cost as the number of servers increases, because a larger server pool relaxes resource constraints and reduces the number of action-masking rejections during optimization. AMD-P has higher decision cost than AMD despite lacking the diffusion component, because the diffusion-generated initial plan gives PPO a warm start and speeds up convergence. Without this prior, AMD-P must explore the full action space from scratch and needs more gradient steps per decision epoch.

Fig. 3(b) shows the orchestration cost, which covers both initial agent startup and any subsequent migration. APD has the highest orchestration cost because its static deployment places agents on a fixed set of servers without later rebalancing. This causes resource contention and long queuing during agent initialization. AMD distributes agents across servers through diffusion-guided planning, which reduces initialization queuing and keeps orchestration cost low. Greedy and RR lack real-time resource awareness and show moderate contention. AMD-P and AntLLM have higher orchestration cost because their exploration or iterative optimization may place agents on suboptimal servers before converging.

As illustrated in Fig. 3(c), execution time for most algorithms remains stable, as task execution is largely independent of the number of servers. RR exhibits higher execution latency due to load imbalance caused by its resource-unaware scheduling.

Fig. 3(d) presents the total task latency. The ranking is: AMD $<$ AMD-D $<$ AntLLM $\approx$ Greedy $<$ RR $<$ AMD-P $<$ APD. Specifically, AMD reduces the average total latency by 3.9%, 6.5%, and 11.4% compared to AMD-D, AntLLM, and Greedy/RR, respectively. Overall, AMD reduces total latency by up to 27.6% compared to the worst-performing baseline (APD), and by 3.9% compared to the next-best variant (AMD-D).

Resource Utilization: As shown in Fig. 4, average per-server resource utilization decreases as more servers share a fixed workload, since each server handles fewer agents. The resource utilization ranking is: AMD $>$ APD $>$ AMD-P $>$ AntLLM $>$ RR $>$ AMD-D $>$ Greedy. Fig. 4(a)–(c) compare CPU, memory, and storage utilization. AMD achieves efficient resource utilization by actively sensing node loads and dynamically migrating tasks. In contrast, AMD-D and APD lack dynamic adjustment capabilities, leading to uneven distribution and resource waste. AMD-P wastes resources on ineffective exploration, while AntLLM suffers from heavy optimization overhead. RR and Greedy fail to achieve effective load balancing. Compared to the baselines, AMD improves resource utilization by up to 53.3%, with average CPU, memory, and storage utilization improving by approximately 27.1%, 27.1%, and 26.0%, respectively. Fig. 4(d) shows Virtual Machine Server (VMS) bandwidth usage. AMD-D, Greedy, and RR consume more VMS bandwidth due to suboptimal scheduling decisions. AMD keeps VMS bandwidth low by balancing agent placement across servers.

2) Performance with different numbers of tasks

Fig. 5 shows that latency increases with the number of tasks due to higher scheduling and execution overhead.

Delay Performance: Fig. 5(a)–(b) show the decision cost and orchestration cost, respectively. Although RR performs adequately under light load, it does not scale well as the number of tasks increases due to growing resource contention. AMD maintains stable overhead by dynamically rebalancing agent placements. Fig. 5(c) confirms that AMD achieves the lowest execution latency, whereas RR suffers from queueing delays due to load imbalance. Overall (Fig. 5(d)), AMD reduces total latency by approximately 5% compared to the next-best algorithm, confirming its scalability.

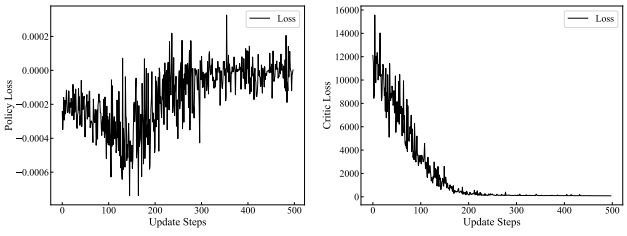
Resource Utilization: Fig. 6(a)–(c) compare CPU, memory, and storage utilization under varying task loads. RR exhibits

declining resource efficiency as task volume increases due to its static scheduling strategy. Among all algorithms, AMD consistently achieves the highest resource utilization. The resource utilization ranking is: AMD > APD > RR > AMD-P > AntLLM > AMD-D > Greedy. Specifically, compared to AMD-D, AntLLM, Greedy, RR, AMD-P, and APD, AMD achieves 23.8%, 26.0%, and 24.4% improvements in CPU, memory, and storage resource utilization, respectively, across varying task quantities. Fig. 6(d) shows that AMD maintains low VMS bandwidth usage, while Greedy and AMD-D consume higher bandwidth.

This performance edge, consistently reflected in the latency results (Figs. 3 and 5) and the resource utilization results (Figs. 4 and 6), stems from the cooperative action of the diffusion initialization and PPO refinement stages. On one hand, the diffusion-based initialization functions as a global structural planning module. Because high-dimensional edge environments with complex inter-agent topological dependencies cause traditional heuristics like Greedy and RR to easily trap in local optima, the conditional diffusion model addresses this by learning the global deployment distribution from diverse historical layouts. By encoding the agent dependency topology into a continuous condition vector via a GCN layer, it executes multi-step generative reverse denoising to yield a globally optimized initial plan τ_{init}, which strategically co-locates highly interdependent agents, fundamentally compressing the structural cross-server communication latencies and context stalls reflected in the lower total latency of Fig. 3(d). On the other hand, the PPO refinement layer operates as an agile runtime adaptation module. While the diffusion prior secures a high-quality global layout, it is an offline generative process that cannot adapt to transient, short-term environmental fluctuations such as user mobility or workload spikes. The superimposed PPO layer bridges this vulnerability by performing fine-grained, step-level online policy fine-tuning atop τ_{init}. Utilizing lightweight neural network inference based on real-time telemetry, PPO updates deployment decisions with minimal decision latency. Crucially, PPO's clipping mechanism constrains the policy update magnitude, absorbing live environmental variance without disrupting the globally optimized macro-topology established by the diffusion prior. Consequently, this distinct division of labor ensures that AMD maintains the tight latency bounds and high resource utilization observed across Figs. 3–6 under highly volatile edge states.

3) Convergence of AMD

Fig. 7 depicts the training convergence of AMD. The policy loss (Fig. 7(a)) gradually increases toward zero with fluctuations, indicating progressive policy improvement as the actor adjusts action probabilities to maximize cumulative reward. The critic loss (Fig. 7(b)) decreases rapidly and converges, indicating that the value network learns an accurate estimate of the state values. These results confirm the convergence and training stability of AMD.



(a) Policy Loss (b) Critic Loss

Fig. 7 Training loss of AMD.

4) Per-Agent Execution Latency

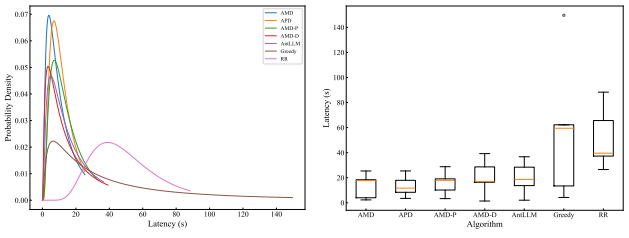
To complement the system-level evaluation in Figs. 3–5, we run a separate profiling experiment to measure the per-agent execution latency, i.e., the time a single agent spends on its sub-task after initialization. This profiling targets the task-processing phase at the individual agent level. Because the data are collected independently from the system-level measurements in Figs. 3–6, the absolute values are not directly comparable.

PDF: In Fig. 8(a), AMD shows a sharp peak in the low-latency region, indicating both low average per-agent execution latency and high predictability. AMD-P and AntLLM exhibit wider distributions, while Greedy and RR show long tails, reflecting poor adaptability to varying conditions.

Box-plot: Fig. 8(b) shows the per-agent execution latency distribution across all methods. AMD has a compact interquartile range and low per-agent execution latency, indicating stable performance at the individual agent level. APD also shows low per-agent latency because its static placement assigns agents to well-matched servers. After the startup and queuing phase, task processing runs efficiently with local resources. However, this comes at the cost of much higher system-level orchestration overhead (Figs. 3(b) and 3(d)). In contrast, Greedy and RR exhibit high medians and numerous outliers, reflecting large latency variability. These results show that AMD achieves both low system-level total latency and stable per-agent execution performance.

5 Conclusions

This paper addressed the challenge of migrating LLM-based AI agents in dynamic edge intelligence systems by proposing AMD, an adaptive migration framework that integrates conditional diffusion models with reinforcement learning.



(a) PDF (b) Box-plot

Fig. 8 Per-agent execution latency analysis.

The diffusion model was used to learn the distribution of high-quality deployments, producing well-structured global initialization plans that alleviated the local-optima problem commonly encountered in conventional reinforcement learning. PPO was employed for online policy refinement, maintaining service continuity under user mobility and resource variations. Experimental results on a geographically distributed testbed built upon AgentScope showed that AMD consistently outperformed all baseline strategies in the evaluated scenarios. Specifically, it reduced the average total task latency by 3.9%–27.6% across different baselines and improved resource utilization by up to 53.3%. The current evaluation uses 3–5 edge servers with up to 24 agents, reflecting a realistic small-scale edge deployment. Future work will explore larger-scale scenarios with more edge nodes and agent populations to further validate scalability. We also plan to investigate the impact of varying agent counts as an independent experimental variable and to evaluate AMD under heterogeneous network conditions.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant U25A20436, Grant 62302048, and Grant 62272050; in part by Guangdong Higher Education Association under Grant 24GQN97; in part by the Guangdong Provincial Higher Education Institutions under Grant 2024KTSCX219; in part by Beijing Normal University at Zhuhai Education Reform Project under Grant jx2025037; in part by the Research Fund of Guangxi Key Lab of Multi-Source Information Mining and Security under Grant MIMS24-07; in part by Institute of Artificial Intelligence and Future Networks and Engineering Center of AI and Future Education, Guangdong Provincial Department of Science and Technology, China; in part by Zhuhai Science-Tech Innovation Bureau under Grant 2320004002772; and in part by the Interdisciplinary Intelligence Super Computer Center of Beijing Normal University at Zhuhai.

Reference

[1] D. Gao, Z. Li, X. Pan, W. Kuang, Z. Ma, B. Qian, F. Wei, W. Zhang, Y. Xie, D. Chen, L. Yao, H. Peng, Z. Zhang, L. Zhu, C. Cheng, H. Shi, Y. Li, B. Ding, and J. Zhou, Agentscope: A flexible yet robust multi-agent platform, 2024.

[2] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, Edge intelligence: Paving the last mile of artificial intelligence with edge computing, *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[3] Z. Yao, Z. Tang, W. Yang, and W. Jia, Enhancing LLM QoS through Cloud-Edge Collaboration: A Diffusion-based Multi-Agent Reinforcement Learning Approach , *IEEE Transactions on Services Computing*, no. 01, pp. 1–17, Apr. 2025, doi: 10.1109/TSC.2025.3562362.

[4] X. Chi, H. Chen, G. Li, Z. Ni, N. Jiang, and F. Xia, Edsp-edge: Efficient dynamic edge service entity placement for mobile virtual reality systems, *IEEE Transactions on Wireless Communications*, vol. 23, no. 4, pp. 2771–2783, 2024.

[5] S. Greengard, *The internet of things*. MIT press, 2021.

[6] T. Bai, C. Pan, Y. Deng, M. Elkashlan, A. Nallanathan, and L. Hanzo, Latency minimization for intelligent reflecting surface aided mobile edge computing, *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 11, pp. 2666–2682, 2020.

[7] T. Wang, Y. Mei, W. Jia, X. Zheng, G. Wang, and M. Xie, Edge-based differential privacy computing for sensor–cloud systems, *Journal of Parallel and Distributed Computing*, vol. 136, pp. 75–85, 2020.

[8] Z. Tang, F. Mou, J. Lou, W. Jia, Y. Wu, and W. Zhao, Multi-user layer-aware online container migration in edge-assisted vehicular networks, *IEEE/ACM Transactions on Networking*, vol. 32, no. 2, pp. 1807–1822, 2023.

[9] G. Cai, R. Tian, L. Yang, Y. Jia, L. Li, and J. Wang, Efficient inference for edge large language models: A survey, *Tsinghua Science and Technology*, vol. 31, no. 3, pp. 1365–1380, 2026.

[10] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, Migration modeling and learning algorithms for containers in fog computing, *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 712–725, 2018.

[11] F. Mou, J. Lou, Z. Tang, Y. Wu, W. Jia, Y. Zhang, and W. Zhao, Adaptive digital twin migration in vehicular edge computing and networks, *IEEE Transactions on Vehicular Technology*, 2024.

[12] J. Lou, Z. Tang, and W. Jia, Energy-efficient joint task assignment and migration in data centers: A deep reinforcement learning approach, *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 961–973, 2022.

[13] L. Chen, C. Shen, P. Zhou, and J. Xu, Collaborative service placement for edge computing in dense small cell networks, *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 377–390, 2021.

[14] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, Edge com-

puting for autonomous driving: Opportunities and challenges, *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019.

[15] B. Sun, Z. Huang, H. Zhao, W. Xiao, X. Zhang, Y. Li, and W. Lin, Llumnix: Dynamic scheduling for large language model serving, in *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, 2024, pp. 173–191.

[16] S. Wang, C. Yuen, W. Ni, Y. L. Guan, and T. Lv, Multiagent deep reinforcement learning for cost-and delay-sensitive virtual network function placement and routing, *IEEE Transactions on Communications*, vol. 70, no. 8, pp. 5208–5224, 2022.

[17] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, Joint service placement and request routing in multi-cell mobile edge computing networks, in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 10–18.

[18] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications, *IEEE transactions on cybernetics*, vol. 50, no. 9, pp. 3826–3839, 2020.

[19] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, and T. Zhou, A survey on edge computing systems and tools, *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1537–1562, 2019.

[20] S. Tuli, F. Mirhakimi, S. Pallewatta, S. Zawad, G. Casale, B. Javadi, F. Yan, R. Buyya, and N. R. Jennings, Ai augmented edge and fog computing: Trends and challenges, *Journal of Network and Computer Applications*, vol. 216, p. 103648, 2023.

[21] X. Wang, J. He, Z. Tang, J. Guo, J. Lou, L. Qian, T. Wang, and W. Jia, Adaptive ai agent placement and migration in edge intelligence systems, in *2025 IEEE 25th International Conference on Communication Technology (ICCT)*. IEEE, 2025, pp. 1475–1480.

[22] T. Kipf, Semi-supervised classification with graph convolutional networks, *arXiv preprint arXiv:1609.02907*, 2016.

[23] H. Qi, M. Liwang, S. Hosseinalipour, L. Fu, S. Zou, X. Wang, W. Ni, and Y. Hong, Fusion: Forecast-embedded agent scheduling with service incentive optimization over distributed air-ground edge networks, *arXiv preprint arXiv:2512.14323*, 2025.

[24] Y. Xiao, X. Li, H. Zhou, Y. Li, Y. Gao, G. Shi, P. Zhang, and M. Krunz, Sanet: A semantic-aware agentic ai networking framework for cross-layer optimization in 6g, *arXiv preprint arXiv:2512.22579*, 2025.

[25] Z. Wu, S. Sun, Y. Wang, M. Liu, B. Gao, J. Lu, T. Wu, Z. Yang, and T. Wen, Recursive offloading for llm serving in multi-tier networks, *IEEE Transactions on Mobile Computing*, 2025.

[26] A. Qayyum, A. Albaseer, J. Qadir, A. Al-Fuqaha, and M. Abdallah, Llm-driven multi-agent architectures for intelligent self-organizing networks, *IEEE Network*, 2025.

[27] S. Yao, M. Wang, J. Ren, T. Xia, W. Wang, K. Xu, M. Xu, and H. Zhang, Multi-agent reinforcement learning for task offloading in crowd-edge computing, *IEEE Transactions on Mobile Computing*, 2025.

[28] T. Wang, P. Wang, S. Cai, X. Zheng, Y. Ma, W. Jia, and G. Wang, Mobile edge-enabled trust evaluation for the internet of things, *Information Fusion*, vol. 75, pp. 90–100, 2021.

[29] R. Zheng, Y. Zheng, Z. Cheng, L. Luo, H. Luo, G. Sun, H. Yu, and D. Niyato, Agentvne: Llm-augmented graph reinforcement learning for affinity-aware multi-agent placement in edge agentic ai, *arXiv preprint arXiv:2601.02021*, 2026.

[30] X. Zhang, C. Wang, Y. Zhu, J. Cao, and T. Liu, Multi-agent deep reinforcement learning with trajectory prediction for task migration-assisted computation offloading, *IEEE Transactions on Mobile Computing*, 2025.

[31] H. Gao, X. Wang, W. Wei, A. Al-Dulaimi, and Y. Xu, Comddpg: Task offloading based on multiagent reinforcement learning for information-communication-enhanced mobile edge computing in the internet of vehicles, *IEEE Transactions on Vehicular Technology*, vol. 73, no. 1, pp. 348–361, 2023.

[32] Y. Zhong, J. Kang, J. Wen, D. Ye, J. Nie, D. Niyato, X. Gao, and S. Xie, Generative diffusion-based contract design for efficient ai twin migration in vehicular embodied ai networks, *IEEE Transactions on Mobile Computing*, 2025.

[33] C. Xu, J. Guo, Y. Liang, H. Huang, H. Zou, X. Zheng, S. Yu, X. Chu, J. Cao, and T. Wang, Diffusion models for reinforcement learning: Foundations, taxonomy, and development, *arXiv preprint arXiv:2510.12253*, 2025.

[34] L. Zeng, Q. Liu, S. Shen, and X. Liu, Improved double deep q network-based task scheduling algorithm in edge computing for makespan optimization, *Tsinghua Science and Technology*, vol. 29, no. 3, pp. 806–817, 2024.

[35] X. Chen, B. Xiao, X. Lin, Z. Chen, and G. Min, Multi-agent collaboration for vehicular task offloading using federated deep reinforcement learning, *IEEE Transactions on Mobile Computing*, 2025.

[36] X. Bi, M. He, and Y. Sun, Mix q-learning for lane changing: a collaborative decision-making method in multi-agent deep reinforcement learning, *IEEE Transactions on Vehicular Technology*, 2025.

[37] Y. Yang, W. Ma, W. Sun, J. He, Y. Fu, C. Yuen, and Y. Zhang, Diffusion-based multi-agent reinforcement learning for semantic vehicular edge computing, *IEEE Transactions on Services Computing*, 2025.

[38] Y. Zhai, T. Yang, K. Xu, D. Feng, C. Yang, B. Ding, and H. Wang, Enhancing decision-making for llm agents via step-level q-value models, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 25, 2025, pp. 27 161–27 169.

Author biography

Jie Gao is currently pursuing her master's degree at Beijing Normal University, Zhuhai, China. Her research interests mainly include edge computing, reinforcement learning and their applications.

Xingdan Wang received the B.S. degree in Data Science and Big Data Technology from the College of Arts and Sciences, Beijing Normal University, China, in 2026. Her primary research interests lie in the fields of Edge Intelligence, AI Agent orchestration, and Deep Reinforcement Learning. Her recent work focuses on optimizing resource allocation and task migration in distributed systems.

Zhiqing Tang (Member, IEEE) received the B.S. degree from School of Communication and Information Engineering, University of Electronic Science and Technology of China, China, in 2015 and the Ph.D. degree from Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in 2022. He is currently an Associate Professor with Faculty of Arts and Sciences, Beijing Normal University, China. His current research interests include edge AI, diffusion models, multi-agent systems, and reinforcement learning.

Jiahui Chen is an undergraduate student at the Faculty of Arts and Sciences, Beijing Normal University at Zhuhai, China. His research focuses on multi-agent systems, computer vision, and their practical applications.

Zhi Yao received the B.S. degree from College of Electronic and Information Engineering, Shandong University of Science and Technology, China, in 2020, and the M.S. degree from South China Academy of Advanced Optoelectronics, South China Normal University, China, in 2023. He is currently pursuing the Ph.D. degree in School of Artificial Intelligence, Beijing Normal University, China. His current research interests include mobile edge computing, vector database, LLM request scheduling, and reinforcement learning.

Jiong Lou received the B.S. degree and Ph.D. degree in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in 2016 and 2023. Since 2023, he has held the position of Research Assistant Professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. He has published more than ten papers in leading journals and conferences (e.g., ToN, TMC and TSC). His current research interests include edge computing, task scheduling and container management.

Jianxiong Guo (Member, IEEE) received his Ph.D. degree from the Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA, in 2021, and his B.E. degree from the School of Chemistry and Chemical Engineering, South China University of Technology, Guangzhou, China, in 2015. He is currently an Associate Professor with the Faculty of Arts and Sciences, Beijing Normal University, and also with the Guangdong Key Lab of AI and Multi-Modal Data Processing, Beijing Normal-Hong Kong Baptist University, Zhuhai, China. He is a member of IEEE/ACM/CCF. He has published more than 90 peer-reviewed papers and has been a reviewer for many famous international journals and conferences. His research interests include social networks, wireless sensor networks, combinatorial optimization, and machine learning.

Weijia Jia (Fellow, IEEE) is currently the Director of the Institute of AI and Future Networks and the Director of the Super Intelligent Computer Center, Beijing Normal University (BNU), China, and also a Chair Professor at Beijing Normal-Hong Kong Baptist University (BNBU), China. He has served as the Vice President for Research at BNBU from 2020 to 2024. Prior joining BNU Zhuhai, he was the Deputy Director of the State Key Laboratory of IoT for Smart City, University of Macau, and a Zhiyuan Chair Professor at Shanghai Jiao Tong University. From 1995 to 2013, he was with the City University of Hong Kong as a Professor. He has over 700 publications in international journals/conferences. He has received the first Prize of Scientific Research Awards from the Ministry of Education of China in 2017 (list 2) and top 2% World Scientists in Stanford list (2020–2024). He is the Distinguished Member of CCF.