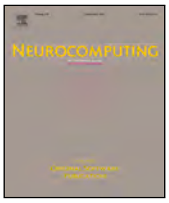
Contents lists available at ScienceDirect

Neurocomputing

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



Dynamic weight reinforcement learning method considering multiple factors in mobile edge computing system

Shihua Li [a], Yanjie Zhou [b,*], Xiangqian Liu [a], Ning Wang [a], Junqi Wang [c], Bing Zhou [a], Zongmin Wang [a]

[a] *School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou, 450001, China*
[b] *School of Management, Zhengzhou University, Zhengzhou, 450001, China*
[c] *School of Information Management, Zhengzhou University of Aeronautics, Zhengzhou, 450016, China*

A R T I C L E I N F O

Communicated by J. Xu

Keywords:
Dynamic weight
Multi-objective deep reinforcement learning
Computation offloading

A B S T R A C T

The accelerated advancement of Mobile Edge Computing (MEC) has facilitated significant progressions in digital medical diagnostic services. However, the multi-region, multi-user, and multi-priority characteristics in medical diagnostic services within MEC systems make the task of offloading problematic. On the one hand, the task offloading problem involves multiple conflicting objectives, such as minimizing delay costs, server load balancing, and user fairness. Some researchers have explored these issues and proposed reinforcement learning-based methods to tackle such problems. However, existing work often characterizes objectives as linear scalarizations of multiple objectives, overlooking their conflicts. On the other hand, the preferences for various objectives in different regions cannot be predicted in advance and may indeed differ, making it challenging to handle in existing research. To address these challenges, we propose a multi-objective, multi-agent reinforcement learning approach. In this approach, the reward at each step is a vector, with each scalar corresponding to an objective. Furthermore, we propose a multi-agent tournament selection method to identify historically significant preferences. This mechanism considers the strategies of other agents while preserving the policies previously learned by the current agent. The objective is to achieve cooperative scheduling, allowing agents to synchronize their decisions based on their historical preferences and those of other agents. Simulation results demonstrate that the proposed algorithm outperforms several baseline algorithms across various performance metrics.

1. Introduction

In recent years, the swift progress of communication technologies and the rapid expansion of mobile applications have led to the widespread adoption of mobile devices, such as smartphones, smart bracelets, and wearable medical devices [1]. The expansion of mobile devices has resulted in a substantial increase in the demand for computing and storage resources. However, the limited computational resources and battery capacity inherently restrict portable mobile devices, rendering them inadequate for supporting computationally intensive tasks [2]. Cloud computing supports computationally intensive tasks, but the sheer volume of tasks can overwhelm cloud computing infrastructure, hindering it from delivering optimal services [3]. Mobile edge computing (MEC) is emerging as a promising solution by providing edge servers with increased computational and storage resources in the proximity of mobile devices [4]. Offloading computational tasks to edge servers mitigates the computational burden on cloud servers and mobile devices and meets user quality of experience requirements [5].

With the continuous advancement of artificial intelligence technology, health diagnosis algorithms have made significant progress [6–8]. This breakthrough has made it possible for users to undergo health monitoring without having to visit a hospital [9]. However, the influx of a large number of users exerts considerable strain on MEC systems. Additionally, task offloading in MEC systems typically needs to consider multiple competing factors, such as task delay cost, server load balancing, and user fairness. This necessitates the development of scheduling algorithms for MEC systems that consider multiple conflicting objectives to offload task requests generated by a large number of mobile devices to edge servers concurrently.

Due to the computational complexity of task scheduling problems, most research efforts on multi-objective problems are based on heuristic

* Corresponding author (Yanjie Zhou, ieyjzhou@zzu.edu.cn).
E-mail addresses: lishihua@gs.zzu.edu.cn (S. Li), ieyjzhou@zzu.edu.cn (Y. Zhou), liuxiangqian@gs.zzu.edu.cn (X. Liu), WNing@ha.edu.cn (N. Wang), wangjunqi@zua.edu.cn (J. Wang), iebzhou@zzu.edu.cn (B. Zhou), zmwang@zzu.edu.cn (Z. Wang).

https://doi.org/10.1016/j.neucom.2024.129194
Received 26 April 2024; Received in revised form 21 October 2024; Accepted 14 December 2024
Available online 21 December 2024

or meta-heuristic algorithms [10]. However, these approaches are effective for static task scheduling, such as in manufacturing environments where all task order information is available in advance. They are not suitable for dynamically generated tasks in MEC systems.

Deep reinforcement learning (DRL) algorithms have shown promising performance in dynamically scheduling tasks in MEC scenarios [11]. DRL can establish a Markov decision process based on the MEC system scenario. The Markov model can adjust its decision mechanism in response to the dynamic nature of tasks and the environment [12]. Additionally, DRL can pre-train and save models, enabling the rapid generation of scheduling plans. This capability enhances the quality of experience for users [13]. These considerations form the basis for designing a reinforcement learning-based algorithm to address task scheduling issues in MEC scenarios.

In fact, there have been numerous multi-objective reinforcement learning-based solutions for offloading dynamic tasks in MEC scenarios [14]. These studies mainly focus on predefined weights for multiple objectives, transforming the weighted sum of multiple objectives into a single objective, or optimizing the weights of multiple objectives to find Pareto-optimal solutions. However, in most real-world applications of MEC systems, several issues arise: (1) Multiple objective preferences are often unpredictable and may change over time. (2) There are conflicts among multiple objectives, the minimum delay cost, the minimum range of server occupancy time, and the minimum range of user waiting time are three important criteria for evaluating MEC systems, and improving one of these objectives may worsen the others. For example, a smaller variance in user waiting time means a fairer user experience, but this can increase the waiting time for some users and increase the cost of delays. (3) Different regions focus on different factors at the same time [15]. These factors have prompted us to design a multi-objective reinforcement learning approach to address the specific problem considered in this paper.

Based on the aforementioned issues, this paper introduces a multi-objective, multi-agent scheduling algorithm. The algorithm partitions the areas covered by the MEC system into multiple blocks, with each edge server responsible for one block. Each intelligent agent is designated to a block and holds its own dynamic preferences. The agent takes preferences and system states as inputs to make scheduling decisions, and employs the multi-agent tournament (MT) method to select a preference from its own set and those of other agents as the historical preference for updating the network. The main contributions of this paper are summarized as follows.

- We define a framework for multi-objective scheduling of multi-priority tasks in MEC systems. The model simultaneously considers three conflicting objectives including total delay cost, server occupancy time range, and user waiting time range.
- We propose a real-time scheduling algorithm based on multi-agent reinforcement learning, considering dynamic weights across multiple regions and multiple objectives, termed multi-objective reinforcement learning with multiple objectives (MORL-MOT). We also develop a multi-agent tournament (MT) method. MT enables each agent to consider local historical preferences and those of other agents comprehensively. It selects the most suitable historical preference to optimize the Q-network while mitigating the risk of converging to local optima.
- We conduct extensive experiments. Simulation results demonstrate that the MORL-MOT algorithm achieves a favorable balance among the three objectives and exceeds the current state-of-the-art approaches on various evaluation metrics.

The remaining sections of the paper are organized as follows: Section 2 discusses related work. In Section 3, we present the system model and problem definition. Section 4 provides a detailed explanation of the MORL-MOT algorithm. Section 5 showcases the experimental setup and results. Finally, Section 6 concludes the paper.

2. Related work

The multi-objective scheduling problem is common across various industries, and many notable researchers have employed different approaches to address this issue. In this section, we examine the multi-objective scheduling problem, focusing on solutions based on heuristic algorithms and those based on reinforcement learning algorithms.

2.1. *Multi-objective scheduling problem*

The multi-objective scheduling problem has attracted significant attention in computer science, as efficient task scheduling is essential for system performance and resource utilization. To tackle this problem, researchers have developed various algorithms aimed at optimizing different objectives. [16] presents an emergency scheduling algorithm based on the combination of deep reinforcement learning and Lamarckian local search, which effectively dispatches water valves and safety plugs to isolate contaminated water, thereby reducing the residual concentration of pollutants and the scheduling cost. [17] introduces a low-complexity stepwise optimization algorithm designed to minimize user energy consumption while adhering to constraints on latency and transmission power. [18] devises an improved multi-objective optimization algorithm based on the immune algorithm, balancing the relationship between response time and energy consumption. [19] introduces a joint cost-benefit and resource-aware Mobile Edge Computing offloading algorithm. This algorithm balances processing speed, memory, energy, and performance on mobile devices in the workplace through intelligent resource awareness while minimizing the cost of remote resource utilization.

Most of these studies focus on scenarios where the weights are unknown, aiming to find the Pareto front of the multi-objective problem or to identify the Pareto optimal solution of the multi-objective weights through specific heuristics. However, for MEC scenarios with dynamically changing weights and randomly generated tasks, it is difficult to devise such rules.

2.2. *Heuristic algorithm-based solutions*

The solutions based on heuristic algorithms are commonly used early approaches to tackle multi-objective scheduling problems. [20] combines computation offloading with Dynamic Voltage and Frequency Scaling and proposes a multi-objective evolutionary algorithm based on Non-dominated Sorting Genetic Algorithm III (NSGA-III). This approach effectively balances energy consumption and communication costs. [21] introduces a Lagrangian dual decomposition method and proposes an algorithm that combines task offloading, resource allocation, and security assurance, enhancing task processing efficiency and service quality. [22] introduces a task scheduling algorithm for vehicular networks based on ant colony optimization, effectively reducing system latency, energy consumption, and load imbalance. [23] models the network as a straightforward, undirected, unweighted graph and designs a hybrid algorithm (GA-PSO) based on genetic algorithms and particle swarm optimization. This approach optimizes load balancing and task latency for edge servers. [24] considers the temporal logic of tasks and proposes a task-scheduling strategy based on genetic algorithms combined with multi-connection techniques. This strategy aims to balance various performance metrics. [25] introduces the smart patient-device connectivity matrix and proposes a multi-objective optimization algorithm based on a non-dominated sorting genetic algorithm, simultaneously optimizing the total task offloading cost and server load variance. [26] conceptualizes computing tasks in MEC as a directed acyclic graph and introduces an enhanced NSGA-II algorithm to optimize offloading scheduling strategies, focusing on reducing energy consumption and latency.

Heuristic-based solutions perform well in optimizing scheduling strategies in static MEC environments. However, retraining scheduling strategies becomes necessary when the MEC environment changes, leading to increased time overhead. Therefore, solutions based on heuristic algorithms cannot guarantee optimal performance in dynamic MEC systems.

2.3. DRL algorithm-based solutions

Reinforcement learning develops strategies through continuous interaction with the environment, and multi-objective reinforcement learning algorithms have been applied in various domains, including but not limited to dynamic MEC systems. In [27], a dual Q-learning approach is used to optimize the offloading of MEC tasks, aiming to minimize energy consumption and latency. [28] introduces an interactive multi-objective reinforcement learning algorithm based on a preference structure, employing weighted coefficients to scalarize the reward function. By adjusting these weighted coefficients, it is possible to search for solutions that match the preferences of the decision maker. [29] proposes an enhanced Soft Actor-Critic algorithm and a novel experience replay method to achieve multi-objective optimization for electric vehicle fuel, emission reduction, and battery life preservation. [30] extends the classical fuzzy Q-learning algorithm by using non-dominated Q-values to represent action values. By iteratively applying the Bellman equation, it considers the global Q-function and simultaneously identifies multiple optimal non-dominated strategies. [31] utilizes multi-objective optimization for scheduling valves and fire hydrants, seeking to balance the efficiency of sewage evacuation with scheduling costs. [32] introduces a task offloading solution capable of incremental learning and online learning based on real-time data, aiming to provide users with high quality of experience. [33] formulates a two-stage deep reinforcement learning strategy. The first stage employs a deep Q-network algorithm to generate scheduling policies, while the second stage utilizes a deep deterministic policy gradient algorithm to determine vehicle transmission power policies. This approach effectively balances execution latency, processing accuracy, and energy consumption. [34] models the optimization problem of MEC networks as a multi-objective optimization problem, which is transformed into a single-objective optimization problem through linear weighting. This method efficiently reduces latency and energy consumption. [35] designed a graph convolutional network to extract dependency information among tasks and proposed a task scheduling algorithm based on DRL to minimize transmission and computation costs. In addition, some multi-intelligence based approaches also consider the optimization problem under uncertainty of the target state model. [36] employs a partitioned-interval design approach, where time is divided into different intervals and a different matrix function is designed within each interval to adapt to the switching of the system state. This approach allows the Lyapunov function to be adjusted at the instant of switching to accommodate changes in the system dynamics. [37] provides a new methodology and theoretical support for solving the observer design problem for networked systems under actuator saturation and state constraints through mathematical modeling and theoretical analysis. [38] employs the theory of nonlinear planning and optimization by constructing an appropriate Lyapunov function to analyze the stability of the system and proved that the proposed algorithm can guarantee the convergence of the global objective function.

Most of the mentioned studies apply linear weighting and scalarization of multi-objective rewards, introducing reinforcement learning algorithms for optimization. This solution is suitable when the weights of multiple objectives are known in advance and remain constant over time. However, in dynamic Mobile Edge Computing (MEC) systems, the weights of various objectives cannot be predetermined and may change over time, which prompts our investigation into the dynamic weight multi-objective optimization problem.

3. Problem

In this section, we delineate the motivation of this work and define the problem of our edge server deployment.

3.1. Motivation

Heart disease claims millions of lives annually and remains one of the leading causes of death worldwide. In recent years, advances in wearable ECG monitoring devices and ECG diagnostic algorithms have made continuous ECG monitoring possible. However, the large-scale online ECG monitoring of millions of individuals has placed significant pressure on communication networks. To ensure efficient real-time scheduling of these tasks while optimizing multiple objectives, this paper develops a dynamic multi-objective reinforcement learning algorithm for MEC, specifically tailored to ECG detection and diagnosis. Moreover, task offloading in MEC systems must account for multiple conflicting objectives, with preferences that are region-specific, unpredictable, and subject to change over time.

The scheduling algorithms addressing multiple objectives have been extensively studied with significant advancements. Previous studies have often modeled the problem based on predefined information and employed heuristic algorithms to obtain Pareto optimal solutions [20,22]. However, the long solution times make them impractical for real-time solving of dynamically changing service requests. Some studies utilizing reinforcement learning algorithms have achieved real-time scheduling [33,34]. However, most of these studies combine multiple objectives into a single one by weighted summation, ignoring conflicts between multiple objectives. While some works have considered dynamically changing preferences [15,39,40], they fail to address variations in preferences across different regions. Therefore, this paper proposes a scheduling scheme based on multi-objective reinforcement learning. This approach accommodates multiple objectives based on the preferences of different regions and proactively adjusts resource allocation in response to the real-time distribution of service requests.

3.2. Problem definition

As illustrated in Fig. 1, a comprehensive MEC system consists of multiple base stations, edge servers, and mobile devices. For simplicity, we consider that the edge servers are located near the base stations. The communication time between edge servers and base stations could be omitted, which is an assumption in many previous studies [41]. Let $\mathcal{N} = \{1, \dots, N\}$ represent the set of edge servers, where N is the total number of edge servers. Let $\mathcal{M} = \{1, \dots, M\}$ represent the set of mobile devices, where M is the total number of mobile devices. Mobile devices can transmit electrocardiogram diagnostic tasks to servers at any time and from any location.

Each task generated by a mobile device is expressed as a tuple $k =< t_k, d_k, c_k, p_k >$, where t_k represents the time when the mobile device initiates task transmission, and d_k represents the size of the data packet for task k. c_k represents the CPU cycles required for the computation of task k and p_k represents the priority of task k. When a task k is generated, it is first offloaded to its nearest edge server, which is called the local server. Each task can either be computed on its corresponding local server or transferred from the local server to other edge servers for computation based on the decision of its corresponding local server. Fig. 2 illustrates the complete process of task k being transmitted to another server for processing. Here, T_k^m represents the time taken for task k to transfer from the mobile device to the local server, $T_k^{w_1}$ denotes the time spent waiting for scheduling locally, T_k^t signifies the time for task k to transmit from the local server to another server, $T_k^{w_2}$ indicates the time spent waiting for processing on the other server, and T_k^n refers to the processing time for task k on the other server.

Table 1 summarizes the notations used in this paper.

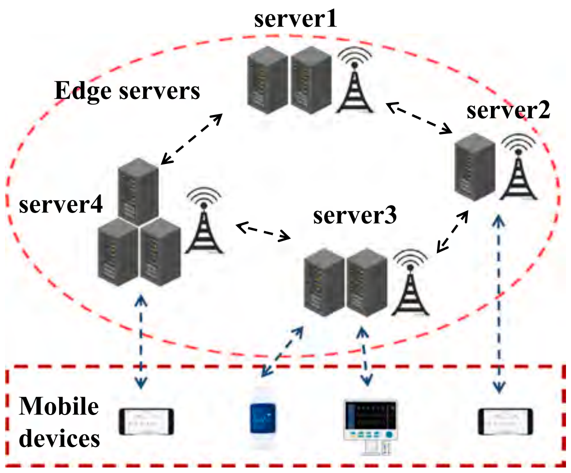


Fig. 1. The diagram of the edge computing architecture.

Table 1
Notations.

Notation	Definition
$\mathcal{N}$	The set of edge servers.
N	The number of edge servers.
C_n	The Compution power of edge server n.
$\mathcal{M}$	The set of mobile devices.
M	The number of mobile devices.
$\mathcal{K}$	The set of tasks.
K	The number of tasks.
t_k	The time at which task k is generated.
d_K	The size of the data packet for task k.
c_k	The CPU cycles required for computing task k.
p_k	The priority of task k.
$\mathcal{P}$	The transmission power.
$\mathcal{G}$	The channel gains between different devices.
B	The spectrum bandwidth.
σ^2	The gaussian white noise.
g	The path loss constant.
ϵ	The path loss exponent.
$\mathcal{L}$	The distance between communication devices.

3.3. Local server computation

In an ECG monitoring system, the data transmitted in the uplink is considerably much larger than the diagnostic results transmitted in the downlink. Consequently, the uplink latency is a dominant factor [42]. If a task is not processed by its corresponding local server, the computation performed by the local server is solely used to transfer the task from the mobile device to another server, where it will await execution.

The signal-to-noise ratio (SNR) e_{mn} for the uplink transmission of the mobile device m is given by Eq. (1).

$$e_{mn} = \frac{\mathcal{P}_m \cdot \mathcal{G}_m^n}{\sigma^2 + \sum_{m' \in \mathcal{M}, m' \neq m} \sum_{n' \in \mathcal{N}, n' \neq n} \mathcal{P}_{m'} \cdot \mathcal{G}_{m'}^{n'}}, \tag{1}$$

where $\mathcal{P}_m$ and $\mathcal{P}_{m'}$ represent the transmission power of mobile devices m and m', respectively. σ^2 represents Gaussian white noise. $\mathcal{G}_m^n$ denotes the channel gains between the local server n and the mobile device m. $\mathcal{G}_{m'}^{n'}$ denotes the channel gains between the local server n' and the mobile device m'. The channel gain for communication links is denoted by $\mathcal{G} = g \cdot \mathcal{L}^{-\epsilon}$, where g and ϵ represent the path loss constant and the exponent of the communication link. $\mathcal{L}$ denotes the distance between communication devices. As a result, the transmission rate, represented by r_{mn}, from the mobile device m to the local server n is defined in Eq. (2).

$$r_{mn} = B \cdot log_2(1 + e_{mn}), \tag{2}$$

where B indicates the available spectrum bandwidth. Therefore, the transmission time from mobile device m to the local server n of the task k can be expressed as Eq. (3).

$$T_k^m = d_k / r_{mn}. \tag{3}$$

The computational time of the task k on the server n is T_k^n, which is defined as follows.

$$T_k^n = c_k / C_n. \tag{4}$$

3.4. Other server computation

If the local server has a substantial number of tasks, it can offload some tasks to other edge servers for execution and calculate the delay time incurred by the transfer. For example, server n decides to transfer task k to server n'. We consider that the server and the base station are deployed at the same location, and the transmission between servers is also the transmission between base stations, so we use Eq. (1) and Eq. (2) to compute the transmission rate $r_{nn'}$. The transmission time for task k from local server n to other edge server n' is represented by $T_k^t = d_k / r_{nn'}$.

It is important to note that regardless of whether task k is executed on the local server or transferred to another edge server if the server is currently processing tasks, task k must to wait for the server to become available. This waiting time cannot be predicted in advance. We define the total waiting time for task k as T_k^w and the completion time of task k as T_k^e.

3.5. Multi-objective problem definition

The aim of this paper is to determine the optimal solution that minimizes the total delay costs for all tasks, as well as the ranges of server occupancy time and user wait time.

The total cost of all task delays D is:

$$D = \sum_{k=1}^{K} (T_k^m + T_k^t + T_k^w + T_k^n) \cdot \beta_k, \tag{5}$$

where $\beta_k = p_k / a_k$ is the delay coefficient of task k. When task k is computed on its corresponding local server, $T_k^t = 0$.

The server usage time range E indicates the balance of the server load. This balance is contingent upon the tasks assigned to each server and the computation time of those tasks.

$$E = max\{\sum_{k=1}^{K} T_k^n | n \in \mathcal{N}\} - min\{\sum_{k=1}^{K} T_k^n | n \in \mathcal{N}\}. \tag{6}$$

The user waiting time is defined as the task completion time minus the time when the task was generated. Consequently, the calculation of the range of user waiting time U is as follows:

$$U = max\{T_k^e - t_k | k \in \mathcal{K}\} - min\{T_k^e - t_k | k \in \mathcal{K}\}. \tag{7}$$

We define the studied problem as a multi-objective problem that simultaneously minimizes three objectives: D, E, and U. The objectives in this multi-objective problem are interdependent tradeoffs. For example, if the scheduling policy results in lower-priority tasks experiencing longer wait times, the total delay cost may be reduced. However, tasks that arrive earlier may encounter increased waiting times, which exacerbates the extreme difference in user waiting time. Additionally, the preferences between these objectives vary from the perspectives of the operator and users, necessitating dynamic adjustments. When there are an increasing number of higher-priority tasks, the focus should be on minimizing the delay cost. Conversely, in the context of routine monitoring, prioritizing the balance of server loads and minimizing discrepancies in user waiting times is of paramount importance. This approach ensures greater user satisfaction.

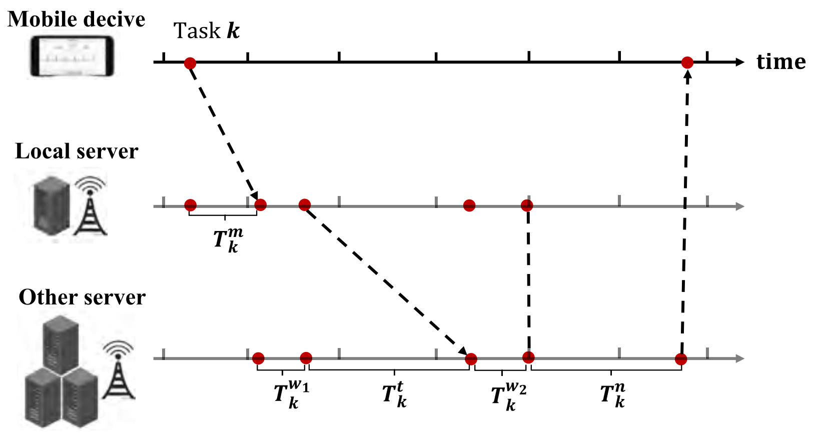


Fig. 2. The diagram of the task offloading process.

4. MORL-MOT

This section describes the proposed MORL-MOT algorithm. The method models the task offloading problem in MEC systems as a Markov model of multi-agent collaborative scheduling, utilizing Deep Double Q-Network (DDQN) as the core algorithm. Additionally, a multi-agent tournament approach is proposed to address the dynamic changes in preferences across different regions. Each agent stores the encountered preferences locally, and when the Q-network requires updating, the multi-agent tournament approach is employed to select fundamental preferences. These preferences are then trained with the current system preferences, taking into full account the stochastic and individualized nature of different regional preferences.

4.1. Markov model

The state spaces $S_n = \{S_n^L, S_n^F\}$ are defined as follows. **State space:**

$$S_n^L = \{C_n, \mathbf{t}_n, \mathbf{c}_n, \mathbf{p}_n, \aleph_n\}, \tag{8}$$

$$S_n^F = \{C_n, \mathbf{t}_1, \mathbf{c}_1, \mathbf{p}_1, \dots, \mathbf{t}_n, \mathbf{c}_n, \mathbf{p}_n, \aleph_n\}, \tag{9}$$

where S_n^L denotes the local state of edge server n. The fusion state of edge server n is a composite of the operational states of its other servers. C_n is the computational capacity of server n. $\mathbf{t}_n = \mathcal{R}^{triple}(t_k, \mathcal{K}_n)$ is the feature representing the arrival times of all tasks with server n as the local server. $\mathbf{c}_n = \mathcal{R}^{triple}(c_k, \mathcal{K}_n)$ is the feature representing the required CPU cycles required by all tasks with server n as the local server. $\mathbf{p}_n = \mathcal{R}^{triple}(p_k, \mathcal{K}_n)$ is the feature representing the delay coefficients of all tasks with server n as the local server. $\mathcal{R}^{triple}(x, y)$ is calculated by Eq. (10). $\mathcal{K}_n$ is the set of all tasks with server n as the local server. $\aleph_n$ is the number of all tasks with server n as the local server. The local state is employed to select the most urgently needed task k to be processed from $\mathcal{K}_n$. The fusion state selects the server n from $\mathcal{N}$ that is currently best suited to process task k.

$$\mathcal{R}^{triple}(x, y) = (Min\{x, x \in y\}, \frac{1}{|y|}\sum_{x \in y} x, Max\{x, x \in y\}). \tag{10}$$

Action space:

$$\mathcal{A}_n = \{\mathcal{A}_n^L, \mathcal{A}_n^F\}, \tag{11}$$

$$\mathcal{A}_n^L = \{Min\{t_k\}, Min\{c_k\}, Max\{p_k\} | k \in \mathcal{K}_n\}, \tag{12}$$

$$\mathcal{A}_n^F = \{1, 2, \dots, N\}, \tag{13}$$

where $\mathcal{A}_n^L$ is the type of the selected task. $\mathcal{A}_n^F$ is the index of the chosen server. $Min\{t_k\}$, $Min\{c_k\}$ and $Max\{p_k\}$ correspond to three task types, representing the earliest arrival time, the minimum required CPU cycles, and the maximum delay coefficient, respectively.

Reward: The MORL-MOT algorithm prioritizes three objectives: total delay cost, server load balancing, and user fairness, which are determined by Eqs. (5), (6), and (7), respectively. The reward is formulated based on the following considerations: (1) Conflicts exist between multiple objectives; therefore, we structure the reward as a vector corresponding to these multiple objectives. (2) The magnitudes of multiple objectives are disproportionate; thus, the three reward values are normalized to the range of [−1, 0] using the min–max normalization method. (3) The optimization goal is to minimize the three objectives; consequently, the negative values of the three objectives are utilized.

4.2. The MORL-MOT algorithm

The architecture of the MORL-MOT algorithm is shown in Fig. 3, where each server functions as an independent decision-making agent. The process begins with the decision-making phase, during which each agent generates scheduling policies based on the current local state, fusion state, and preferences. Next, the network update phase follows, during which each agent samples a batch of experiences from $\mathcal{B}_n$ and selects historical preferences using the multi-agent contest method. Subsequently, the parameters of the Q-network and target Q-network are updated using experiences and both preferences (current and historical). The specific details of the proposed algorithm are outlined in Algorithm 1.

Each server's responsibility for preferences at the same time may be different. We use the preference weight generation method from Ref. [39] to initialize the preference space $\mathcal{W}_n$ for each agent. Then, we initialize each agent's Q-network, target Q-network, experience buffer $\mathcal{B}_n$, and preference history buffer W_n.

At each time step t, all agents first synchronize the states of various servers to generate the fusion state S_n^F. Then, each agent randomly selects a preference $\mathbf{w}_{nt}$ from the preference space $\mathcal{W}_n$ as the current preference. If $\mathbf{w}_{nt}$ does not exist in W_n, it is added to W_n along with the current time step t. Otherwise, the time step corresponding to $\mathbf{w}_{nt}$ is updated to the latest t.

Then each agent makes task selection action A_{nt}^L with ε-greedy strategy based on local state S_{nt}^L and current preference $\mathbf{w}_{nt}$, and server selection action A_{nt}^F with ε-greedy strategy based on fusion state S_{nt}^F and current preference $\mathbf{w}_{nt}$. The ε-greedy strategy is defined as:

$$A_{nt} = \begin{cases} \text{Randomly select an action from the action space } \mathcal{A}_{nt} & \text{with probability } \varepsilon \\ argmax_{a \in \mathcal{A}_{nt}} \mathbf{Q}(S_{nt}, A_{nt}; \mathbf{w}_n^t) & \text{with probability } 1-\varepsilon \end{cases} \tag{14}$$

After performing the action $< A_{nt}^L, A_{nt}^F >$, the agent observes the reward $\mathbf{r}_t$ and the local state S_{nt+1}^L of the next step, and stores $< S_{nt}^L, A_{nt}^F, \mathbf{r}_t, S_{nt+1}^L >$ into the experience buffer $\mathcal{B}_n$. We randomly sample

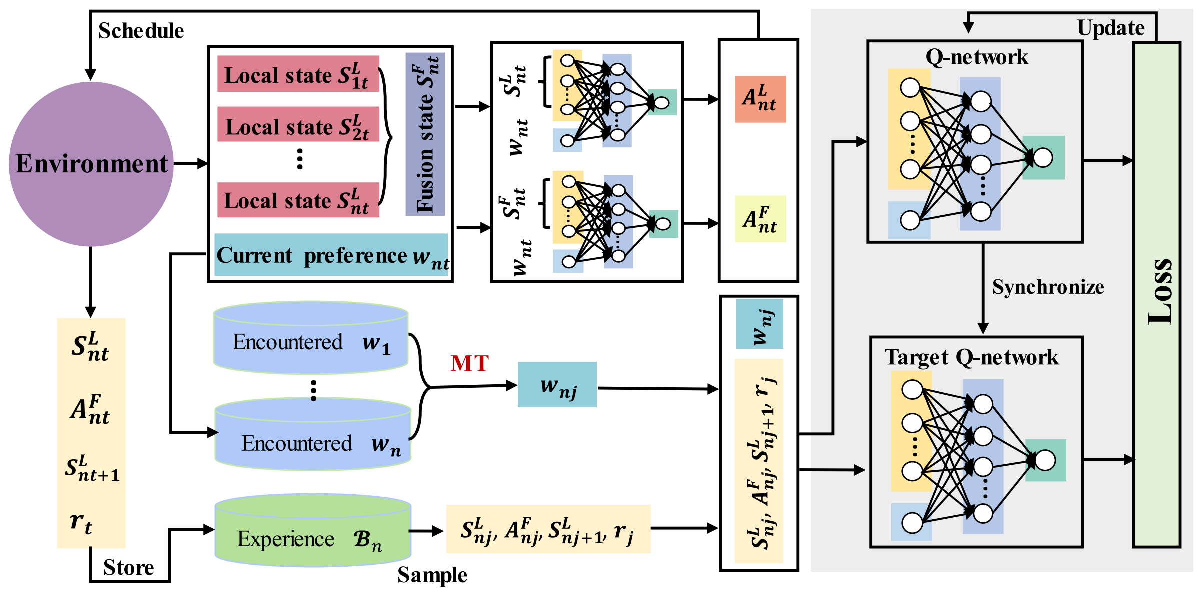


Fig. 3. The diagram of the MORL-MOT.

Algorithm 1 MORL-MOT algorithm

```
1: Initialize the Q-network $Q_n(s_n, a_n, \theta_n)$, the target Q-network $\overline{Q_n}(s_n, a_n, \theta_n)$,
   the preference weight space $\mathcal{W}_n$, the experience buffer $\mathcal{B}_n$ and the weight
   history buffer $W_n$ for each agent $n$.
2: Initialize the location state $S^L_n$ for each agent $n$.
3: for $t \in \{0, 1, ..., T\}$ do
4:     Initialize the fusion state $S^F_{nt}$ for each agent $n$.
5:     for $n \in N$ do
6:         Randomly select a weight $\mathbf{w}_{nt}$ from $\mathcal{W}_n$.
7:         if $\mathbf{w}_{nt}$ is not in $W_n$ then
8:             Add $\mathbf{w}_{nt}$ and $t$ to $W_n$.
9:         else
10:            Update t of $\mathbf{w}_{nt}$ in $W_n$.
11:        end if
12:        Select action $A^L_{nt}$ using Eq. (14).
13:        Select action $A^F_{nt}$ using Eq. (14).
14:        Take actions $< A^L_{nt}, A^F_{nt} >$
15:        Observe $\mathbf{r}_t$ and next location state $S^L_{nt+1}$.
16:        Store transition $< S^L_{nt}, A^F_{nt}, \mathbf{r}_t, S^L_{nt+1} >$ in $\mathcal{B}_n$.
17:        Sample a mini-batch $b_n$ of transitions from $\mathcal{B}_n$.
18:        for each sampled transition $< S^L_{nj}, A^F_{nj}, \mathbf{r}_j, S^L_{nj+1} >$ do
19:            Sample a weight $\mathbf{w}_{nj}$ from $W_n$ using the MT method.
20:            if all tasks are completed then
21:                $\mathbf{y}_j = \mathbf{y}'_j = \mathbf{r}_j$
22:            else
23:                $\mathbf{y}_j = \mathbf{r}_j + \gamma \overline{\mathbf{Q}_n}(S^L_{nj+1}, \underset{a \in \mathcal{A}}{argmax}\, \mathbf{Q}_n(S^L_{nj+1}, a, \mathbf{w}_{nt})\mathbf{w}_{nt}, \mathbf{w}_{nt})$
24:                $\mathbf{y}'_j = \mathbf{r}_j + \gamma \overline{\mathbf{Q}_n}(S^L_{nj+1}, \underset{a \in \mathcal{A}}{argmax}\, \mathbf{Q}_n(S^L_{nj+1}, a, \mathbf{w}_{nj})\mathbf{w}_{nj}, \mathbf{w}_{nj})$
25:            end if
26:        end for
27:    end for
28:    Each agent calculates the average loss of the sample.
29:    $Loss_n = \frac{1}{2}(|\mathbf{y}_j - \mathbf{Q}_n(S^L_{nj}, A^F_{nj}, \mathbf{w}_{nt})| + |\mathbf{y}'_j - \mathbf{Q}_n(S^L_{nj}, A^F_{nj}\mathbf{w}_{nj})|)$
30:    Each agent updates Q-network $Q_n(s_n, a_n, \mathbf{w}_n)$.
31:    Each agent synchronizes target Q-network $\overline{Q_n}(s_n, a_n, \mathbf{w}_n)$ every $Z$ steps.
32:    Each agent updates $\varepsilon$.
33:    if all tasks are completed then
34:        Initialize the location state $S^L_n$ for each agent $n$.
35:        Clear $\mathcal{B}_n$.
36:    end if
37: end for
```

a small batch from $\mathcal{B}_n$ to train the Q-network. We use the multi-agent tournament method for each sample to select a historical preference to maintain previously learned strategies. As mentioned earlier, we store each current preference $\mathbf{w}_{nt}$ along with the time step in W_n. Preferences for smaller time steps indicate that the Q-network has not been trained with this preference as input for a long time, which complicates the network's ability to leverage previously learned strategies. By introducing the historical preference w_{nj} with a more minor time step along with the current preference $\mathbf{w}_{nt}$, we can better sustain the previously learned strategies. Additionally, preferences for different servers are distinct. Therefore, we propose an MT method to select historical preferences. This approach involves selecting H preferences from each agent's preference set W_n, where H is the tournament size. Subsequently, the time step sizes of the H preferences are transformed into probabilities, where smaller time steps correspond to higher probabilities. A preference is then chosen from the H preferences based on these probabilities. The selected preference is shared among all agents, and the average of the preferences is computed to serve as the historical preference. As outlined in Algorithm 1, the weights and the sample are input into the Q-network and target Q-network. The loss of Q-network is calculated according to line 29, and the Q-network is updated by gradient descent method. The target Q-network is synchronized every Z time step. The parameter ε is updated at each time step.

4.3. Q-network architecture

Fig. 4 depicts the Q-network architecture used by the MORL-MOT algorithm. The network consists of an input layer, an output layer, and three fully connected layers (FC). The number of neurons in the input layer is $3{\times}3{\times}n{+}2{+}1$, where n represents the number of servers, $3{\times}3{\times}n$ represents the $\{\mathbf{t}_n, \mathbf{c}_n, \mathbf{p}_n\}$ of all n servers, 2 denotes C_n and $\aleph_n$ of server n, and 1 represents the current preference. The output layer consists of $3 \times |\mathcal{A}|$ neurons, and the output is reshaped into a $3 \times |\mathcal{A}|$ matrix, where each row represents the multi-objective Q-value of an action. The Q-values are multiplied by the current preference and argmax is computed to obtain the action.

The MORL-MOT algorithm trains sampled samples on the current preference and the historical preference selected using the MT method. The loss calculation formula is as follows:

$$Loss_n = \frac{1}{2}(|\mathbf{y}_j - \mathbf{Q}_n(S^L_{nj}, A^F_{nj}, \mathbf{w}_{nt})| + |\mathbf{y}'_j - \mathbf{Q}_n(S^L_{nj}, A^F_{nj}\mathbf{w}_{nj})|), \quad (15)$$

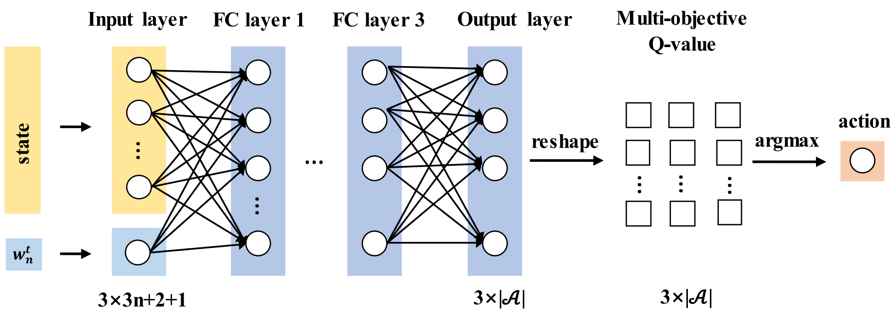


Fig. 4. The architecture of the Q-network.

$$\mathbf{y}_j = \mathbf{r}_j + \gamma \overline{\mathbf{Q}_n}(S^L_{nj+1}, \underset{a\in\mathcal{A}}{argmax}\, \mathbf{Q}_n(S^L_{nj+1}, a, \mathbf{w}_{nt})\mathbf{w}_{nt}, \mathbf{w}_{nt}), \tag{16}$$

$$\mathbf{y}'_j = \mathbf{r}_j + \gamma \overline{\mathbf{Q}_n}(S^L_{nj+1}, \underset{a\in\mathcal{A}}{argmax}\, \mathbf{Q}_n(S^L_{nj+1}, a, \mathbf{w}_{nj})\mathbf{w}_{nj}, \mathbf{w}_{nj}), \tag{17}$$

where $\mathbf{w}_{nt}$ and $\mathbf{w}_{nj}$ represent the current preference and historical preference of agent n, respectively. $\mathbf{Q}_n(S^L_{nj}, A^F_{nj}, \mathbf{w}_{nt})$ denotes the network Q-value vector for action A^F_{nj} in state S^L_{nj} and current preference $\mathbf{w}_{nt}$. It is worth noting that we have two-stage action selection: task selection action A^L_{nj} made under the local state S^L_{nj} and server selection action A^F_{nj} made under the fusion state S^F_{nj}. To simplify the update of the Q network and maintain end-to-end mapping capability, we calculate the loss based on state S^L_{nj} and action A^F_{nj}.

4.4. Multi-agent tournament

The MT algorithm is utilized to select the historical preference for updating the Q-network. Specifically, for the t-th training iteration, each agent has a current preference $\mathbf{w}_{nt}$. If $\mathbf{w}_{nt}$ does not exist in the historical preference pool W_n, then the pair $(t, \mathbf{w}_{nt})$ is stored in W_n; if $\mathbf{w}_{nt}$ already exists in W_n, it is replaced with the new $(t, \mathbf{w}_{nt})$. When computing the historical preference w_{nj} required for network updates, each agent employs a tournament selection algorithm to choose a preference. The tournament selection algorithm works by randomly selecting H preferences from its preference pool W_n, then comparing their t-values and returning the preference with the smallest t-value. This selected preference is then shared with other agents, and each agent obtains n preferences $\{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_n\}$. Finally, the historical preference $\mathbf{w}_{nj}$ is computed as the average of these n preferences.

4.5. Complexity analysis

The time complexity of the MORL-MOT algorithm primarily includes two components: the training complexity of the Q-network during the training phase and the computational complexity of the Q-network during the testing phase. First, let us analyze the time complexity of the training process. As shown in Fig. 4, the Q-network mainly consists of an input layer, an output layer, and three fully connected layers. The number of neurons in the input layer is given by $3\times 3\times n+2+1$, where n is the number of servers, and the number of neurons in the output layer is $3\times|\mathcal{A}|$. Let f_i represent the number of neurons in the ith layer of the Q-network, then $f_0 = 3\times 3\times n+2+1$ and $f_4 = 3\times|\mathcal{A}|$. Let T_{max} and V represent the maximum number of iterations and one time step of an iteration, respectively. Let $|B_n|$ denote the batch size of experience replay. Since each server is an agent and each agent contains a Q-network, the time complexity of the training process is $O(n\times T_{max}\times V\times \sum_{i=1}^{4} f_{i-1}\times f_i)$.

Once training is complete, the trained Q-network makes scheduling decisions for each task at every time step during the testing phase. Consequently, the time complexity of the MORL-MOT testing phase is $O(n\times V\times \sum_{i=1}^{4} f_{i-1}\times f_i)$.

5. Simulation results and discussion

To verify the proposed method, this section discussed the experimental results by simulation.

5.1. Environment and benchmark data set

The proposed algorithm is programmed in the Python programming language with libraries including TensorFlow and Keras. The Python-implemented algorithms were executed on a server with 32 GB of memory and an Intel Core i7-12700KF CPU.

The problem addressed in this paper is the multi-objective, priority-based task scheduling problem with dynamically changing weights in mobile edge computing. The dynamic variation of multi-objective weights and the randomness of task attributes both influence the performance of scheduling algorithms. To verify the proposed algorithm, various instances must be tested. In this paper, we utilized nine instances with varying numbers of servers and tasks using the following parameter settings to validate the algorithm's scalability, as shown in Table 2. Since the focus of this paper is not on server deployment, the computing capacity of servers C_n is randomly generated from [20, 30, 50] GHz. t_k represents the arrival time of task k, which is randomly generated and follows a Poisson distribution as the scheduling time progresses. a_k represents the arrival position of tasks, which depends on which server acts as the local server and is also randomly generated. d_k represents the size of tasks, which are randomly generated from [300, 500] Kb. c_k represents the CPU cycles required for tasks, randomly generated from [500, 1000] cycles. Parameter p_k represents the priority of tasks, randomly generated. Other parameters are shown in Table 3.

5.2. Performance evaluation metrics

We design the following evaluation metrics, based on [39], to assess the performance of the proposed MORL-MOT.

(1) Regret: we use regret to assess the performance of the strategy, with regret defined as the difference between the optimal value and the value achieved by the proposed algorithm. A smaller regret value indicates that the proposed algorithm is closer to the optimal solution, signifying indicates better performance. The *regret* function is defined as Eq. (18).

$$\begin{aligned} regret &= \sum_{n=1}^{N}(\mathbf{w}_n\cdot \mathbf{O}^n_{\mathbf{w}_n} - \mathbf{w}_n\cdot \mathbf{R}^n) \\ &= \sum_{n=1}^{N}(\mathbf{w}_n\cdot \sum_{i=1}^{I}\gamma^{i-1}\cdot \mathbf{r}^n_i), \end{aligned} \tag{18}$$

where n is the index of server and $\mathbf{O}^n_{\mathbf{w}}$ represents the optimal solution with weight $\mathbf{w}_n$. $\mathbf{R}^n$ denotes the accumulated actual reward during the training process, and $\mathbf{r}^n_i$ is the actual reward value for the ith episode.

(2) Adaptation Error(AE): AE is a metric used to assess the adaptability of a policy to dynamic weight scenarios. It is defined as the average ratio of the absolute difference between the actual return and

Table 2
Benchmark data set.

Instances	N	K	C_n	t_k	a_k	d_k	c_k	p_k
Instance-1	3	50	[20,30,50]	Poisson distribution	Random	[300,500]	[500,1000]	[1,2,3]
Instance-2	3	100	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -
Instance-3	3	300	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -
Instance-4	4	50	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -
Instance-5	4	100	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -
Instance-6	4	300	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -
Instance-7	6	50	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -
Instance-8	6	100	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -
Instance-9	6	300	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -	- - - - - -

- means that this cell uses the same value as the above cell.

Table 3
Parameter setting.

Parameter	Value
The transmission power of mobile device ($\mathcal{P}$)	1 W
The spectrum bandwidth B).	10 MHz
The path loss constant (g).	0.01
The path loss exponent (ϵ).	2
The gaussian white noise (σ^2).	−100 dBm
The learning rate (l_r).	0.001
The sample buffer size ($\mathcal{Q}$).	32
The memory buffer size ($\mathcal{Y}$).	6000
The tournment size ($\mathcal{T}$).	2
The preference weight space size ($\|\mathcal{W}\|$).	153

the optimal solution to the optimal solution for all weight scenarios $\mathbf{w}_n$. A smaller AE indicates greater adaptability to dynamic weight scenarios. The calculation of AE is expressed in Eq. (19).

$$AE = \frac{1}{N}\sum_{n=1}^{N}(\frac{1}{|\mathcal{W}_n|}\sum_{\mathbf{w}_n\in\mathcal{W}_n}|\frac{\mathbf{w}_n\cdot\mathbf{R}^n-\mathbf{w}_n\cdot\mathbf{O}^n_{\mathbf{w}_n}}{\mathbf{w}_n\cdot\mathbf{O}^n_{\mathbf{w}_n}}|). \tag{19}$$

(3) Comprehensive Objective Metrics: this paper considers three optimization objectives including the minimum delay cost D, the minimum range of server occupancy time E, and the minimum range of user waiting time U. Given a set of multi-objective weights $\mathbf{w} = (w_1, w_2, w_3)$, we scalarize the three objectives into a composite indicator $COI = w_1 \cdot D + w_2 \cdot E + w_3 \cdot U$. To quantify the performance of the algorithm, we design the average minimum delay cost (ADC), average server occupancy time difference ($ASOT$), average user waiting time difference ($AUDT$), and average composite evaluation indicator ($ACOI$), which are formulated as Eqs. (20), (21), (22), and (23), respectively.

$$ADC = \frac{1}{N}\sum_{n=1}^{N}(\frac{1}{|\mathcal{W}_n|}\sum_{\mathbf{w}_n\in\mathcal{W}_n}D^n_{\mathbf{w}_n}), \tag{20}$$

$$ASOT = \frac{1}{N}\sum_{n=1}^{N}(\frac{1}{|\mathcal{W}_n|}\sum_{\mathbf{w}_n\in\mathcal{W}_n}E^n_{\mathbf{w}_n}), \tag{21}$$

$$AUDT = \frac{1}{N}\sum_{n=1}^{N}(\frac{1}{|\mathcal{W}_n|}\sum_{\mathbf{w}_n\in\mathcal{W}_n}U^n_{\mathbf{w}_n}), \tag{22}$$

$$ACOI = \frac{1}{N}\sum_{n=1}^{N}(\frac{1}{|\mathcal{W}_n|}\sum_{\mathbf{w}_n\in\mathcal{W}_n}COI^n_{\mathbf{w}_n}), \tag{23}$$

where $D^n_{\mathbf{w}_n}$, $E^n_{\mathbf{w}_n}$, $U^n_{\mathbf{w}_n}$, $COI^n_{\mathbf{w}_n}$ are the best D, E, U, COI for $\mathbf{w}_n$, respectively.

5.3. Baselines

In order to demonstrate the performance of the proposed MORL-MOT algorithm, we compare it with four state-of-the-art multi-objective reinforcement learning algorithms with dynamic weights: Naive [43], MORL-DWS [15], MORL-ODT [39], and MORL-COP [40], which are summarized as follows.

- **Naive** [43]: This approach formulates the overall user utility by aggregating multiple single-objective Q-values into a composite function. The composite function is then employed to select actions.
- **MORL-DWS** [15]: This approach utilizes a dynamic weight setting mechanism to achieve a weight-based multi-objective Q-value output. This is achieved by introducing a weight vector as input into the Q-network.
- **MORL-ODT** [39]: This approach employs a tournament method to select significant preferences from the preference set for training based on the MORL-DWS algorithm.
- **MORL-COP** [40]: This approach utilizes a non-dominated sorting method to select important preferences from the preference set for training based on the MORL-ODT algorithm.

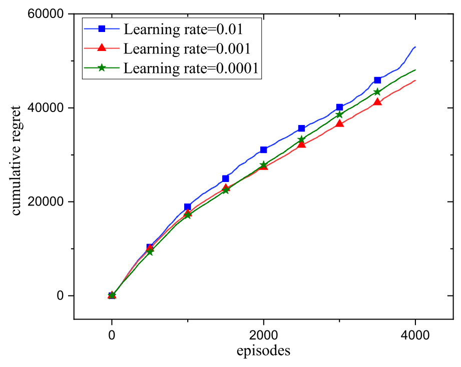
5.4. Experimental result

Based on the experimental setup described above, we conducted extensive experiments to validate the performance of the proposed MORL-MOT algorithm. We performed 10 repeated runs for each experiment, using different seeds for each run. The final experimental results were obtained by averaging the results across multiple seeds. Each experimental training is 4000 steps, the total time is about 3215 s, and each test is about 0.8 s. The experimental results are summarized as follows.

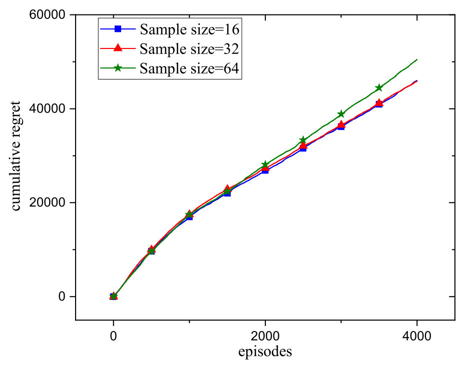
5.4.1. Comparison of cumulative regret with different parameters

This paper assesses the algorithm's performance using regret values, with smaller regret values indicating better performance. Since our method is based on multi-agent reinforcement learning, each agent possesses its own regret value, in contrast to other methods that involve only a single agent. Therefore, for comparison purposes, we aggregate the regret values of all agents and compare them with those of other algorithms. To highlight differences after convergence, we employ accumulated regret values, obtained by summing up the regret values at each step of the learning process. Fig. 5 presents the experiments based on Instance-1 to evaluate the impact of parameters such as learning rate, sample batch size, memory buffer size, and tournament size on the performance of MORL-MOT.

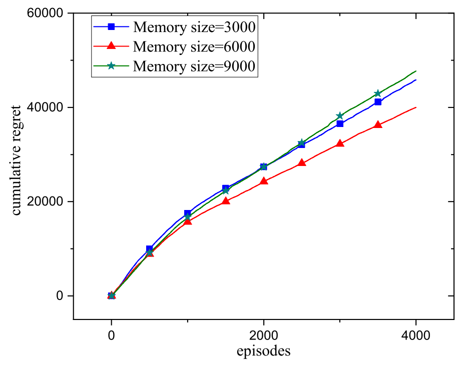
From Fig. 5(a), it can be seen that larger and smaller learning rates contribute to increased accumulated regret. Consequently, a fixed learning rate of 0.001 was adopted in subsequent experiments. Fig. 5(b) demonstrates that larger sampling batch sizes result in significantly elevated cumulative regret. Each agent conducting sampled learning aims to continue learning from past experiences and converge faster. There are several drawbacks to large batch sampling. Firstly, it leads to repeated samples in local collections, which deteriorates sample quality. Secondly, it results in cooperative agents making decisions based on a shared network state. Although local states may differ, experiences stored in the experience pool after collaborative decisions are made are shared among agents. Thirdly, large batch sampling causes redundant learning, slowing down the convergence rate. Smaller sampling batches



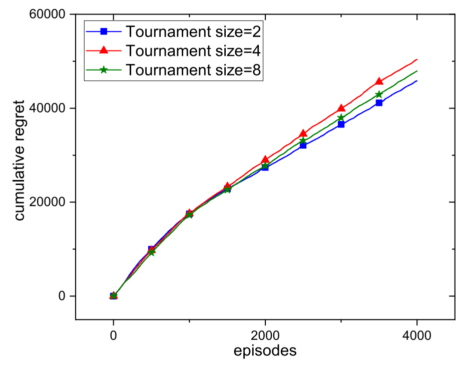
(a) Different learning rates.



(b) Different batch sizes.



(c) Different memory sizes.



(d) Different tourment sizes.

Fig. 5. Performance analysis of learning rate, sample batch size, memory buffer size, and tournament size.

do not result in a significant increase in cumulative regret, but they limit the number of experiences that agents can consider, which in turn leads to an upward trend in cumulative regret in the later stages of training. Therefore, we adopted a fixed sampling batch size of 32 in subsequent experiments. Fig. 5(c) shows that both larger and smaller memory buffer sizes lead to increased cumulative regret. A larger memory buffer size delays updates, causing agents to sample poorer experiences learned during the early stages of network training. Conversely, a smaller buffer limits the storage of experiences, restricting the algorithm's performance. Therefore, we adopted a fixed buffer size of 6000 in the following experiments. Fig. 5(d) demonstrates that a larger tournament size results in increased cumulative regret. However, if the tournament size is set to 1, it becomes a random selection method, which contradicts our original intention. We conducted experiments that confirmed its inferior performance. Therefore, we used a fixed tournament size of 2 in subsequent experiments.

5.4.2. Comparison of cumulative regret for baseline algorithms

Fig. 6 compares the cumulative regret performance of several baseline algorithms using 3 servers, as the basis for the experiment. Figs. 6(a)–6(c) correspond to Instance-1 - Instance-3, respectively. The Naive algorithm exhibits the highest cumulative regret because its Q-values are updated independently for each target without considering the weights of individual targets. Consequently, it struggles to balance multiple objectives, resulting in a significant deviation between the optimal solution and the actual reward. MORL-DWS outperforms Naive by incorporating dynamic preferences for various objectives as input, allowing each agent to consider the changing preferences of multiple objectives when making scheduling decisions. In addition, it introduces a DER experience replay mechanism to prevent overfitting. Building upon MORL-DWS, the MORL-ODT algorithm stores historical weights and episode values in a weight history pool. It employs a tournament selection method to choose the historical weight with the highest episode value for training the Q-network, which helps maintain previously learned policies and enhancing performance. In most instances, MORL-COP outperforms MORL-ODT. It uses a non-explicit sorting method to select preferences, contributing to keeping policies learned in the Q-network. The non-explicit sorting method avoids focusing too much on a particular goal, which prevents local optima. Although MORL-DWS, MORL-ODT, and MORL-COP incorporate multi-objective weights into the decision-making process, they only select historical weights from their respective history pools when updating the Q-network, neglecting the preferences of other agents. This is despite the fact that each agent's local tasks may be sent to other agents for processing. In contrast, the proposed MORL-MOT algorithm employs a collaborative tournament of all agents to jointly select weights, thereby fully considering the preferences of each agent. Furthermore, it converts episode values into probabilities for tournament selection, ensuring that the algorithm does not get stuck in local optima. As a result, MORL-MOT exhibits superior performance.

5.4.3. Comparison of cumulative regret for different instances

Fig. 7 illustrates the cumulative regret of each algorithm across all instances. The number of servers or tasks in each instance varies, and the attributes of tasks are randomly generated to simulate diverse Mobile Edge Computing (MEC) systems.

As can be seen in Fig. 7, the MORL-MOT algorithm consistently achieves lower cumulative regret in all instances compared to the other benchmark algorithms. Compared to the Naive algorithm, the MORL-MOT algorithm considers dynamic preferences for multiple goals. Compared to the MORL-DWS algorithm, the MORL-MOT algorithm uses a tournament approach to select historical weights that are more helpful than the randomized approach in making correct decisions based on prior experience. Compared to the MORL-ODT algorithm

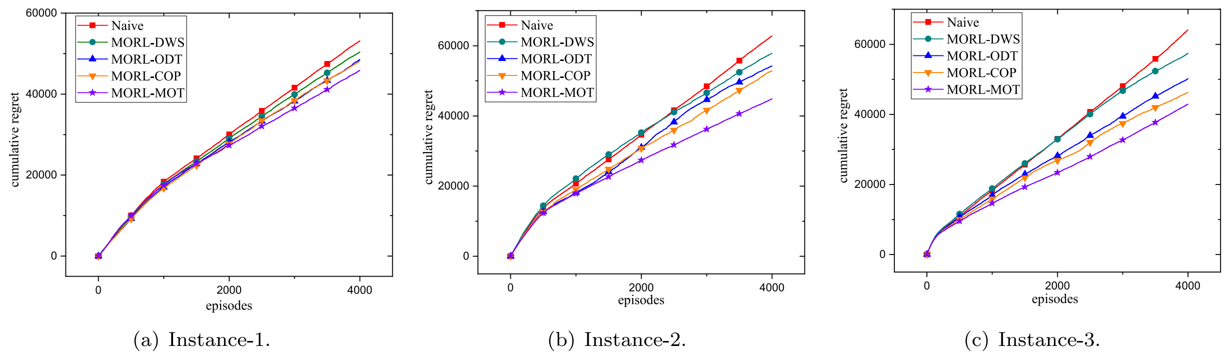


Fig. 6. Comparison of cumulative regret for baseline algorithms.

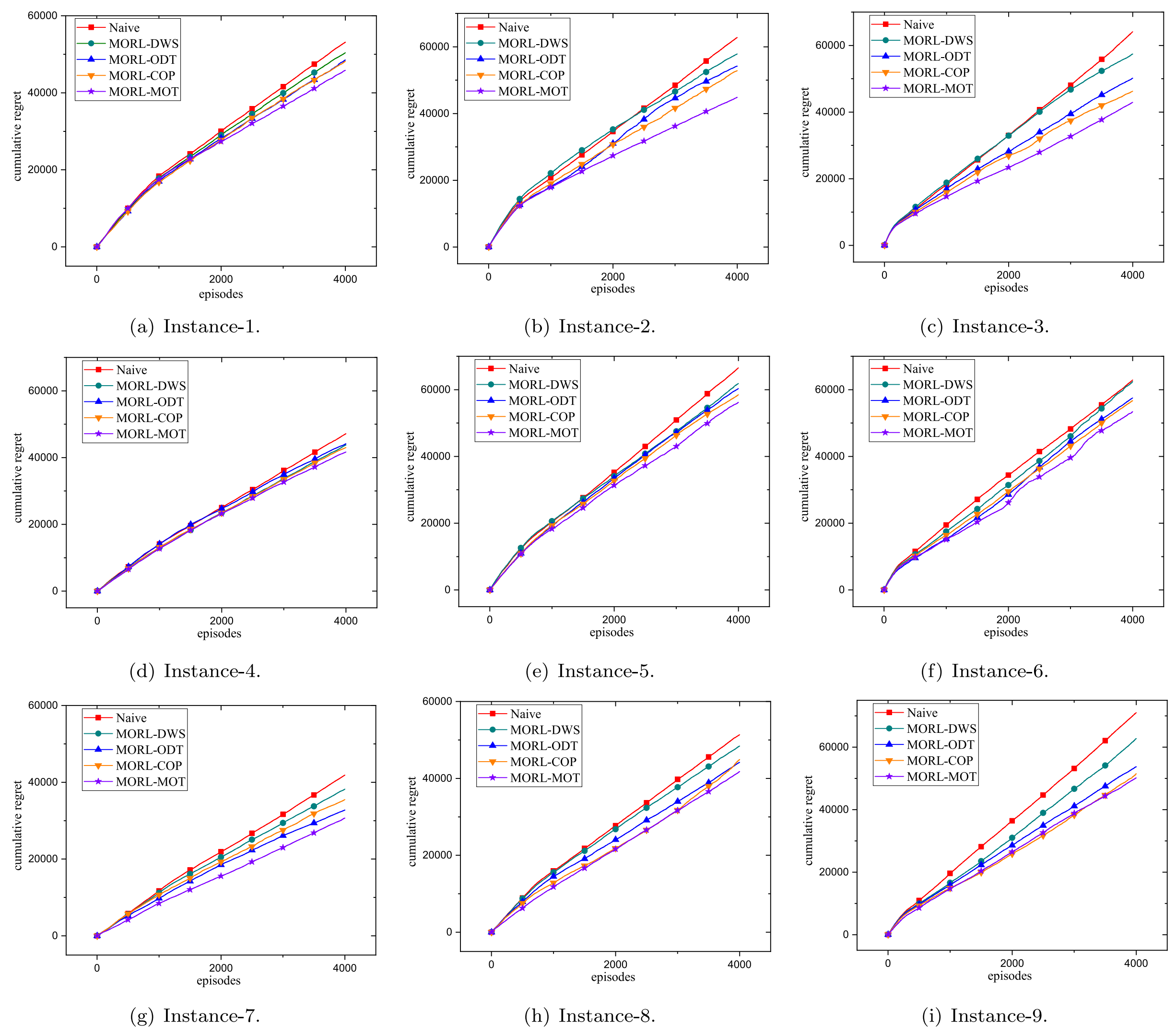
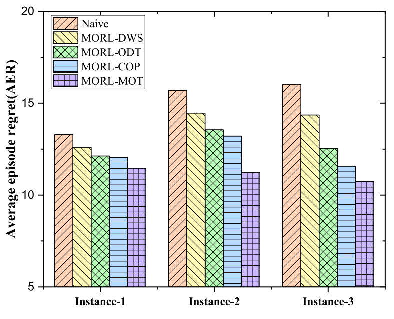


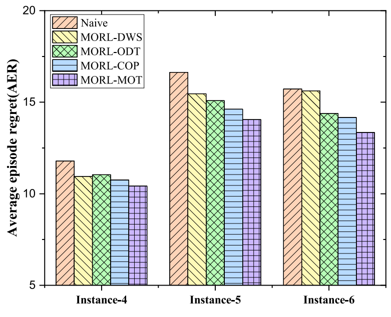
Fig. 7. Comparison of cumulative regret for different instances.

and the MORL-COP algorithm, the MORL-MOT algorithm uses a multi-intelligence tournament approach, which takes into account the preferences of other intelligences and helps to reduce the impact of differing preferences in different regions and to achieve collaborative scheduling between multiple intelligences.

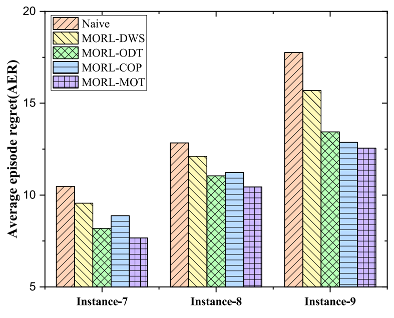
To more clearly illustrate the effectiveness of the MORL-MOT algorithm, we present the AER of each algorithm across all instances in Fig. 8. From the figure, it is evident that the MORL-MOT algorithm consistently achieves lower AER in all instances compared to other baseline algorithms. This indicates that MORL-MOT exhibits the smallest difference between the optimal value and the actual reward in the majority of episodes, reaffirming its ability to adapt to dynamic changes in weights and facilitate better collaboration among multiple agents to reach optimality. Furthermore, all algorithms employ the $\varepsilon - greedy$



(a)

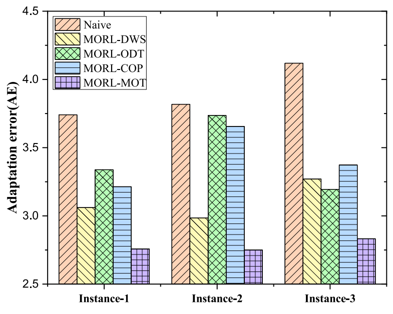


(b)

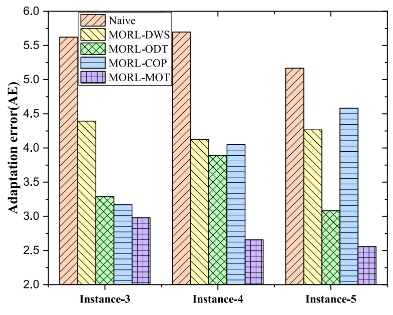


(c)

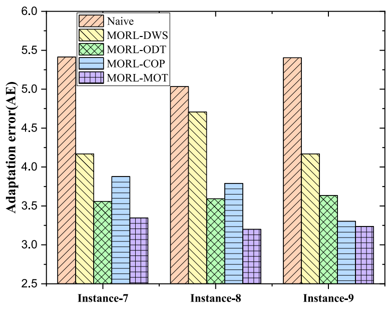
Fig. 8. Comparison of average episode regret.



(a)



(b)



(c)

Fig. 9. Comparison of adaptation error.

strategy for action selection. Consequently, during the initial stages, where the value of ϵ is less than the predetermined value, all algorithms select actions probabilistically. The relatively small difference between the optimal value and the actual reward during this period suggests that each algorithm selects actions based on probabilities. The lower AER for MORL-MOT indicates that this algorithm learns better strategies quickly and converges more effectively.

5.4.4. Comparison of adaptive error for different instances

Fig. 9 illustrates the adaptive error of each algorithm across all instances, and it is evident that the MORL-MOT algorithm performs the most effectively. In the Naive, MORL-DWS, MORL-ODT, and MORL-COP algorithms, each server functions as an intelligent agent, inputting its preferences and state into the Q-network. These algorithms take into account the dynamic preferences of different regions and users during the decision-making phase. However, during the network updating phase, all four algorithms select historical preferences from their local preferences to update the Q network, neglecting the influence of preferences from other intelligent agents. In contrast, the MORL-MOT algorithm not only considers the preferences of each intelligent agent during the decision phase, but also employs the MT algorithm during the network update phase to incorporate the historical preferences of other intelligent agents. This approach maximizes the utilization of previously learned strategies and prevents the network from overfitting to the current preference region.

5.4.5. Comparison of optimization objectives for different instances

Tables 4–7 present the ADC, ASOT, AUDT, and ACOI for each algorithm across all instances. The best-performing results are highlighted in bold.

Table 4 clearly shows that the ADC of the MORL-MOT algorithm significantly surpasses that of other baseline algorithms, indicating its effectiveness in minimizing delay costs.

Table 5 illustrates that the ASOT of the MORL-MOT algorithm outperforms that of other baseline algorithms. This indicates that the MORL-MOT algorithm effectively minimizes substantial discrepancies in server occupation time, thereby enhancing scheduling efficiency.

In Table 6, the MORL-MOT algorithm exhibits significantly improved AUDT compared to other baseline algorithms. This indicates that the MORL-MOT algorithm is effective in minimizing the maximum differences in user waiting times, which ensures a relatively fair delay for each user, thereby enhancing the overall user experience.

Finally, from Table 7, the MORL-MOT algorithm shows a significantly superior ACOI compared to other baseline algorithms, indicating its optimal balance among various objectives. In summary, the MORL-MOT algorithm, with its proposed MT method, is well-suited for dynamic-weight MEC systems. It coordinates scheduling across multiple servers, contributing to efficient task processing.

6. Conclusions

The objective of this paper is to tackle the task offloading problem for multi-region, multi-priority tasks in a mobile edge computing environment. We formulated the problem as an optimization task with three dynamic objectives: delay cost, the range of server occupancy time, and the range of user wait time. To tackle this challenge, we proposed a multi-objective, multi-agent solution along with a multi-agent tournament method. The multi-agent tournament method ensures each agent can select locally relevant historical preferences. The proposed algorithm enables agents to select a historical preference for training the Q network. This effectively maintains previously learned policies while promoting collaboration among agents to accomplish task scheduling under multiple objectives. Extensive simulation experiments were conducted, and the results demonstrate that the proposed MORL-MOT algorithm outperforms state-of-the-art methods across various metrics.

Table 4
Comparison of the performance evaluation metrics ADC with the state-of-the-art.

Instances	Naive [43]	MORL-DWS [15]	MORL-ODT [39]	MORL-COP [40]	This paper
Instance-1	2317.31	2123.27	1949.45	1710.53	**1257.30**
Instance-2	8206.47	8518.95	7086.13	6539.82	**6099.98**
Instance-3	72 249.38	70 295.13	66 262.72	61 495.34	**59 265.32**
Instance-4	1386.06	1267.06	1095.82	1009.39	**895.88**
Instance-5	5824.17	5651.32	4998.28	4441.66	**3766.36**
Instance-6	60 599.14	57 925.69	48 430.29	46 478.91	**44 965.78**
Instance-7	1272.22	1245.78	1216.11	1169.03	**1056.15**
Instance-8	4940.22	4311.37	4037.57	3937.93	**3575.77**
Instance-9	44 442.38	35 516.49	358 694.28	31 820.75	**27626.12**

Table 5
Comparison of the performance evaluation metrics ASOT with the state-of-the-art.

Instances	Naive [43]	MORL-DWS [15]	MORL-ODT [39]	MORL-COP [40]	This paper
Instance-1	10.35	10.30	17.77	21.03	**9.27**
Instance-2	50.81	38.26	27.31	32.87	**22.18**
Instance-3	101.96	74.88	72.42	69.26	**63.28**
Instance-4	27.77	26.72	21.79	26.37	**19.69**
Instance-5	43.95	47.81	46.41	60.47	**37.62**
Instance-6	109.34	138.52	151.88	132.89	**103.71**
Instance-7	16.87	16.17	20.04	18.98	**14.76**
Instance-8	37.26	33.75	46.75	45.70	**25.66**
Instance-9	111.09	107.23	99.14	108.63	**86.83**

Table 6
Comparison of the performance evaluation metrics AUDT with the state-of-the-art.

Instances	Naive [43]	MORL-DWS [15]	MORL-ODT [39]	MORL-COP [40]	This paper
Instance-1	60.89	52.69	41.43	46.58	**37.00**
Instance-2	108.89	108.94	94.54	74.06	**69.93**
Instance-3	265.31	277.37	241.85	203.06	**176.01**
Instance-4	56.39	42.37	41.07	37.81	**26.40**
Instance-5	62.58	111.14	62.91	59.32	**54.11**
Instance-6	234.02	237.93	205.34	194.91	**192.63**
Instance-7	29.33	47.58	33.89	33.57	**28.35**
Instance-8	92.24	33.75	47.58	48.24	**43.68**
Instance-9	298.23	167.53	149.27	147.32	**107.23**

Table 7
Comparison of the performance evaluation metrics ACOI with the state-of-the-art.

Instances	Naive [43]	MORL-DWS [15]	MORL-ODT [39]	MORL-COP [40]	This paper
Instance-1	2388.55	2186.27	2008.65	1778.14	**1303.58**
Instance-2	8366.17	8666.15	7207.97	6646.75	**6192.09**
Instance-3	72 249.38	70 647.39	66 576.98	61 767.66	**59 265.32**
Instance-4	1470.22	1336.16	1158.69	1073.57	**941.97**
Instance-5	5930.70	5810.28	5107.59	4561.45	**3858.09**
Instance-6	60 942.50	58 302.15	48 787.51	46 806.71	**45 262.13**
Instance-7	1318.433	1309.54	1270.04	1221.58	**1099.28**
Instance-8	5069.73	4402.48	4131.92	4031.87	**3645.11**
Instance-9	44 851.71	35 791.25	358 942.71	32 076.71	**27820.19**

CRediT authorship contribution statement

Shihua Li: Writing – original draft, Visualization, Conceptualization. **Yanjie Zhou:** Writing – original draft, Validation, Supervision, Methodology, Funding acquisition. **Xiangqian Liu:** Visualization, Data curation. **Ning Wang:** Methodology, Data curation, Conceptualization. **Junqi Wang:** Methodology, Investigation, Conceptualization. **Bing Zhou:** Writing – review & editing, Resources, Project administration, Conceptualization. **Zongmin Wang:** Writing – review & editing, Validation, Project administration, Methodology.

Declaration of competing interest

The author(s) declared no potential conflicts of interest concerning this article's research, authorship, and/or publication.

Acknowledgments

The work described in this paper was financially supported by the National Key R&D Program of China (No. 2022YFC3803203), National Natural Science Foundation of China (72201252), and Henan Zhongyuan Medical Science and Technology innovation and Development Foundation (24YCG2006).

Data availability

Data will be made available on request.

References

[1] X. Zhu, Y. Xiao, Adaptive offloading and scheduling algorithm for big data based mobile edge computing, Neurocomputing 485 (2022) 285–296, http://dx.doi.org/10.1016/j.neucom.2021.03.141.

[2] I. Ridhawi, S. Otoum, M. Aloqaily, Y. Jararweh, T. Baker, Providing secure and reliable communication for next generation networks in smart cities, Sustainable Cities Soc. 56 (2020) 102080, http://dx.doi.org/10.1016/j.scs.2020.102080.

[3] G. Selvan, R. Ganeshan, I. Diana, J. Ananth, FACVO-DNFN: deep learning-based feature fusion and distributed denial of service attack detection in cloud computing, Knowl.-Based Syst. 261 (2023) 110132, http://dx.doi.org/10.1016/j.knosys.2022.110132.

[4] L. Wang, K. Wang, C. Pan, W. Xu, N. Aslam, L. Hanzo, Multi-agent deep reinforcement learning-based trajectory planning for multi-UAV assisted mobile edge computing, IEEE Trans. Cognit. Commun. Netw. 7 (2021) 73–84, http://dx.doi.org/10.1109/TCCN.2020.3027695.

[5] S. Nath, J. Wu, Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems, Intell. Converged Netw. 1 (2020) 181–198, http://dx.doi.org/10.23919/ICN.2020.0014.

[6] R. López-Blanco, S. Alonso, S. Rodríguez-González, J. Prieto, J.M. Corchado, Trustworthy artificial intelligence -based federated architecture for symptomatic disease detection, Neurocomputing 579 (2024) 127415, http://dx.doi.org/10.1016/j.neucom.2024.127415.

[7] P. Feng, J. Fu, N. Wang, Y. Zhou, B. Zhou, Z. Wang, Semantic-aware alignment and label propagation for cross-domain arrhythmia classification, Knowl.-Based Syst. 264 (2023) 110323, http://dx.doi.org/10.1016/j.knosys.2023.110323.

[8] H. Wang, Y. Zhou, B. Zhou, Z. Wang, A novel method for detection of ECG with deep learning, in: International Conference on Computer and Communications, 2021, pp. 631–635, http://dx.doi.org/10.1109/ICCC54389.2021.9674506.

[9] S. Tuli, N. Basumatary, S. Gill, M. Kahani, C. Arya, S. Wander, R. Buyya, HealthFog: An ensemble deep learning based Smart Healthcare System for Automatic Diagnosis of Heart Diseases in integrated IoT and fog computing environments, Future Gener. Comput. Syst.- Int. J. Esci. 104 (2020) 187–200, http://dx.doi.org/10.1016/j.future.2019.10.043.

[10] Y. Zhou, G.M. Lee, A bi-objective medical relief shelter location problem considering coverage ratios, Int. J. Ind. Eng.-Theory Appl. Pract. 27 (2020) 971–988, http://dx.doi.org/10.23055/ijietap.2020.27.6.7603.

[11] G. Liu, G. Chen, V. Huang, Policy ensemble gradient for continuous control problems in deep reinforcement learning, Neurocomputing 548 (2023) 126381, http://dx.doi.org/10.1016/j.neucom.2023.126381.

[12] W.-C. Jiang, V. Narayanan, J.-S. Li, Model learning and knowledge sharing for cooperative multiagent systems in stochastic environment, IEEE Trans. Cybern. 51 (2021) 5717–5727, http://dx.doi.org/10.1109/TCYB.2019.2958912.

[13] W. Zhan, C. Luo, J. Wang, C. Wang, G. Min, H. Duan, Q. Zhu, Deep-reinforcement-learning-based offloading scheduling for vehicular edge computing, IEEE Internet Things J. 7 (2020) 5449–5465, http://dx.doi.org/10.1109/JIOT.2020.2978830.

[14] Z. Wang, J. Zhang, Y. Li, Q. Gong, W. Luo, J. Zhao, Automated reinforcement learning based on parameter sharing network architecture search, in: 2021 6th International Conference on Robotics and Automation Engineering, 2021, pp. 358–363, http://dx.doi.org/10.1109/ICRAE53653.2021.9657793.

[15] A. Abels, D.M. Roijers, T. Lenaerts, A. Nowe, D. Steckelmacher, Dynamic weights in multi-objective deep reinforcement learning, in: K. Chaudhuri, R. Salakhutdinov (Eds.), in: International Conference on Machine Learning, vol. 97, 2019, pp. 09–15, http://dx.doi.org/10.48550/arXiv.1809.07803.

[16] C. Hu, Q. Wang, W. Gong, X. Yan, Multi-objective deep reinforcement learning for emergency scheduling in a water distribution network, Memet. Comput. 14 (2022) 211–223, http://dx.doi.org/10.1007/s12293-022-00366-9.

[17] Z. Ding, D. Xu, R. Schober, H.V. Poor, Hybrid NOMA offloading in multi-user MEC networks, IEEE Trans. Wireless Commun. 21 (2022) 5377–5391, http://dx.doi.org/10.1109/TWC.2021.3139932.

[18] S.-f. Zhu, E.-l. Sun, Q.-h. Zhang, J.-h. Cai, Computing offloading decision based on multi-objective immune algorithm in mobile edge computing scenario, Wirel. Pers. Commun. 130 (2023) 1025–1043, http://dx.doi.org/10.1007/s11277-023-10318-2.

[19] H. Tout, A. Mourad, N. Kara, C. Talhi, Multi-persona mobility: joint cost-effective and resource-aware mobile-edge computation offloading, IEEE-ACM Trans. Netw. 29 (2021) 1408–1421, http://dx.doi.org/10.1109/TNET.2021.3066558.

[20] Q. Zhu, A. Lu, Y. Hou, Energy- and cost-aware scheduling for task- dependency applications in mobile edge computing, in: 2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design, 2022, pp. 1016–1021, http://dx.doi.org/10.1109/CSCWD54268.2022.9776171.

[21] Y. He, D. Zhai, F. Huang, D. Wang, X. Tang, R. Zhang, Joint task offloading, resource allocation, and security assurance for mobile edge computing-enabled UAV-assisted VANETs, Remote Sens. 13 (2021) 1547, http://dx.doi.org/10.3390/rs13081547.

[22] Y. Sun, Z. Wu, K. Meng, Y. Zheng, Vehicular task offloading and job scheduling method based on cloud-edge computing, IEEE Trans. Intell. Transp. Syst. 24 (2023) 14651–14662, http://dx.doi.org/10.1109/TITS.2023.3300437.

[23] W. Li, X. Sun, B. Wan, H. Liu, J. Fang, Z. Wen, A hybrid GA-PSO strategy for computing task offloading towards MES scenarios, Peerj Comput. Sci. 9 (2023) 1273, http://dx.doi.org/10.7717/peerj-cs.1273.

[24] Y. Sun, H. Li, T. Wei, Y. Zhang, Z. Wang, W. Wu, C. Fang, Dependency-aware flexible computation offloading and task scheduling for multi-access edge computing networks, in: 24th International Symposium on Wireless Personal Multimedia Communications, 2021, pp. 1–6, http://dx.doi.org/10.1109/WPMC52694.2021.9700432.

[25] C. Zhu, J. Ren, H. Wan, T. Qin, Wireless body area networks task offloading method combined with multiple communication and computing resources supported by MEC, Iet Commun. 17 (2023) 1188–1198, http://dx.doi.org/10.1049/cmu2.12606.

[26] Y. Cui, D. Zhang, T. Zhang, P. Yang, H. Zhu, A new approach on task offloading scheduling for application of mobile edge computing, in: 2021 IEEE Wireless Communications and Networking Conference, 2021, pp. 1–6, http://dx.doi.org/10.1109/WCNC49053.2021.9417286.

[27] S. Almelu, S. Veenadhari, Task offloading strategy using double Q-learning based optimization in MEC, in: 2022 IEEE International Conference on Current Development in Engineering and Technology, 2022, pp. 1–5, http://dx.doi.org/10.1109/CCET56606.2022.10079954.

[28] H. Yamamoto, T. Hayashida, I. Nishizaki, S. Sekizaki, Development of interactive multi-objective reinforcement learning considering preference structure of a decision maker, IWCIA, in: 2017 IEEE 10th International Workshop on Computational Intelligence and Applications, vol. 10, 2017, pp. 165–169, http://dx.doi.org/10.1109/IWCIA.2017.8203579.

[29] R. Huang, H. He, Naturalistic data-driven and emission reduction-conscious energy management for hybrid electric vehicle based on improved soft actor-critic algorithm, J. Power Sources 559 (2023) 232648, http://dx.doi.org/10.1016/j.jpowsour.2023.232648.

[30] A. Asgharnia, H. Schwartz, M. Atia, Multi-objective fuzzy Q-learning to solve continuous state-action problems, Neurocomputing 516 (2023) 115–132, http://dx.doi.org/10.1016/j.neucom.2022.10.035.

[31] C. Hu, Q. Wang, W. Gong, X. Yan, Multi-objective deep reinforcement learning for emergency scheduling in a water distribution network, Memet. Comput. 14 (2022) 211–223, http://dx.doi.org/10.1007/s12293-022-00366-9.

[32] R. Wang, Y. Cao, A. Noor, T. Alamoudi, R. Nour, Agent-enabled task offloading in UAV-aided mobile edge computing, Comput. Commun. 149 (2020) 324–331, http://dx.doi.org/10.1016/j.comcom.2019.10.021.

[33] H. Yang, Z. Wei, Z. Feng, X. Chen, Y. Li, P. Zhang, Intelligent computation offloading for MEC-based cooperative vehicle infrastructure system: a deep reinforcement learning approach, IEEE Trans. Veh. Technol. 71 (2022) 7665–7679, http://dx.doi.org/10.1109/TVT.2022.3171817.

[34] S. Lai, R. Zhao, S. Tang, J. Xia, F. Zhou, L. Fan, Intelligent secure mobile edge computing for beyond 5G wireless networks, Phys. Commun. 45 (2021) 101283, http://dx.doi.org/10.1016/j.phycom.2021.101283.

[35] T.T. Nguyen, N.D. Nguyen, P. Vamplew, S. Nahavandi, R. Dazeley, C.P. Lim, A multi-objective deep reinforcement learning framework, Eng. Appl. Artif. Intell. 96 (2020) 103915, http://dx.doi.org/10.1016/j.engappai.2020.103915.

[36] Y. Sun, X. Yang, P. Shi, H. Su, Consensus tracking of switched heterogeneous nonlinear systems with uncertain target, IEEE Trans. Circuits Syst. I. Regul. Pap. (2024) 1–11, http://dx.doi.org/10.1109/TCSI.2024.3376531.

[37] Y. Sun, X. Yang, Y. Zhao, H. Su, Non-negative scaled edge-consensus of saturated networked systems via adaptive output-feedback control, Neurocomputing 586 (2024) 127632, http://dx.doi.org/10.1016/j.neucom.2024.127632.

[38] Y. Sun, X. Yang, H. Su, Fully distributed observer-based scaled consensus of multi-agent systems with actuator saturation and edge-based event-triggered communication, Neurocomputing 600 (2024) 128134, http://dx.doi.org/10.1016/j.neucom.2024.128134.

[39] F. Song, H. Xing, X. Wang, S. Luo, P. Dai, K. Li, Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach, Future Gener. Comput. Syst.- Int. J. Esci. 128 (2022) 333–348, http://dx.doi.org/10.1016/j.future.2021.10.013.

[40] X. Liu, Z.-Y. Chai, Y.-L. Li, Y.-Y. Cheng, Y. Zeng, Multi-objective deep reinforcement learning for computation offloading in UAV-assisted multi-access edge computing, Inform. Sci. 642 (2023) 119154, http://dx.doi.org/10.1016/j.ins.2023.119154.

[41] F. Qi, L. Zhuo, C. Xin, Deep reinforcement learning based task scheduling in edge computing networks, in: 2020 IEEE/CIC International Conference on Communications in China, 2020, pp. 835–840, http://dx.doi.org/10.1109/ICCC49849.2020.9238937.

[42] C. Wang, C. Liang, F.R. Yu, Q. Chen, L. Tang, Computation offloading and resource allocation in wireless cellular networks with mobile edge computing, IEEE Trans. Wireless Commun. 16 (2017) 4924–4938, http://dx.doi.org/10.1109/TWC.2017.2703901.

[43] C. Liu, X. Xu, D. Hu, Multiobjective reinforcement learning: a comprehensive overview, IEEE Trans. Syst. Man Cybern.-Syst. 45 (2015) 385–398, http://dx.doi.org/10.1109/TSMC.2014.2358639.

Shihua Li received the B.S. and the M.S. degree in electronic and communication engineering in Zhengzhou University, China, in 2018 and 2021, respectively. He is currently pursuing the Ph.D. degree with the School of Computer and Artificial Intelligence, Zhengzhou University, China. His research areas include optimization problem in industrial engineering, reinforcement learning and intelligence healthcare.

Yanjie Zhou received Ph.D. degree from the Department of Industrial Engineering at Pusan National University in 2020 and received B.S. Degree and M.S. Degree in Computer Science and Computer Applied Technology from Zhengzhou University in 2012 and 2015, respectively. He is currently an associate professor with the School of Management at Zhengzhou University. His research areas include optimization problems in industrial engineering, game theory, and intelligence healthcare.

Xiangqian Liu received the M.S. degree in software engineering from Henan University, Kaifeng, China, in 2022, where he is currently working toward the Ph.D. degree with the College of Computer Science and Artificial Intelligence, Zhengzhou University. His research interests include multimedia signal and image processing.

Ning Wang received the B.S. and the M.S. degree in computer science from Henan University, China, in 2017 and 2019, respectively. He is currently pursuing the Ph.D. degree with the School of Computer and Artificial Intelligence, Zhengzhou University, China. He is also a research with the Cooperative Innovation Center for Internet Healthcare of Zhengzhou University. His research interest is mainly in representation learning and physiological signal processing.

Junqi Wang, Doctor of Management, graduated from Ocean University of China. He is currently working at the School of Information Management of Zhengzhou University of Aeronautics. His main research interests are big data management and applications and digital transformation of enterprises.

Bing Zhou received the B.S. and M.S. degrees in computer science from Xi'an Jiaotong University, Xi'an, China, in 1986 and 1989, respectively, and the Ph.D. degree in computer science from Beihang University, Beijing, China, in 2003. He is currently a Professor with the School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou, China. His current research interests include intelligent diagnosis of ECG, computer vision, internet medical and multimedia applications.

Zongmin Wang received the Ph.D. degree in engineering from Tsinghua University. From 1996 to 1997, he went to the university of Hong Kong for collaborative scientific research. From 1997 to 2017, he served as a professor and doctoral tutor at Zhengzhou University. He is currently the director of the Cooperative Innovation Center for Internet Healthcare of Zhengzhou University. His main research interests are intelligent diagnosis of ECG, internet medical, computer network and intelligence healthcare.