Hybrid Actor-Critic Based Low-Overhead Scheduling Using MDP for Large-Scale Edge Computing Networks

Qian Zhang
Henan Institute of Economics and Trade, School of Artificial Intelligence, Zhengzhou 450046, China
E-mail: zhangqian@igit.ac.cn

Keywords: edge computing, low-cost scheduling, computing power resources, markov decision, hybrid actor critic algorithm

Received: August 14, 2025

The inconsistency of application requirements in large-scale edge computing networks leads to the need for edge computing scheduling to optimize multiple objectives at the same time. There are significant inherent conflicts between these objectives, and optimizing a single objective often comes at the expense of sacrificing the performance of other objectives, making it difficult to achieve global optimality in large-scale dynamic edge environments. Therefore, a hybrid Actor-Critic Based Low-Overhead Scheduling Using MDP for Large-Scale is proposed. Build a three-layer architecture of "end edge cloud", including terminal collection, edge processing upload, and cloud storage analysis. Enable data flow and processing at different levels, comprehensively analyze the performance and interrelationships of various targets at different levels, and reduce bandwidth and cloud load. Set targets for transmission efficiency, computation delay, transmission delay, and energy consumption, and comprehensively evaluate scheduling performance through cumulative weighted overhead. Considering the impact of various objectives on a global scale, scheduling strategies need to comprehensively consider all objectives when formulating. To achieve this goal, a Markov decision process is used to model the system dynamics, balancing unloading and resource allocation through state space, action space, and reward function, accurately capturing the characteristics of large-scale dynamic edge environments. Considering the changes in system state over time and the impact of different actions on system state, long-term weighted overhead is minimized; At the same time, the hybrid actor critic algorithm is introduced to process different types of actions through discrete and continuous actor networks, and the experience playback mechanism is used to solve the problem of real-time reward acquisition in asynchronous environments. In addition, the critical network is combined to optimize resource allocation, effectively handle complex action space, further balance the relationship between multiple targets, and achieve efficient and low-cost scheduling of large-scale edge computing network computing resources. Experimental results show that the proposed method performs well in large-scale edge computing network resource scheduling. In terms of resource utilization, the average is 93.2%, and the response time is always below 4ms. In terms of energy efficiency, the average task energy consumption is the lowest, saving more than one-third of energy compared to the comparative method. In scalability evaluation, when facing large-scale tasks, the cumulative weighted overhead increases almost linearly, with a growth rate far lower than the comparative method, demonstrating excellent scalability.

Povzetek: Predlagani pristop omogoča učinkovito več-ciljno razporejanje v velikih robnih omrežjih, saj uravnoteži prenos, zakasnitev, energijo in porabo virov v dinamičnem okolju. Rezultati kažejo visoko izkoriščenost virov, nizko zakasnitev ter dobro energetsko in skalabilnostno učinkovitost.

1 Introduction

As a key infrastructure supporting low latency and high reliability applications, edge computing effectively alleviates transmission delay and bandwidth pressure under the traditional cloud computing mode by sinking computing, storage and network resources to the network edge close to the data source, and provides real-time data processing capabilities for smart city, industrial Internet, automatic driving and other scenarios [1]. However, with the increasing number of edge devices and the growing complexity of application demands, the scheduling of computing power resources is facing enormous challenges. Traditional computing resource scheduling methods often have high overhead, requiring a large amount of network bandwidth and computing resources for resource status monitoring and task allocation, and may also lead to untimely scheduling decisions, affecting system performance [2-3]. In large-scale edge computing networks, this high overhead scheduling mode will further aggravate the resource tension situation and reduce the overall efficiency and reliability of the system. Therefore, researching a low-cost computing resource scheduling method has important practical significance. The low-cost scheduling method aims to achieve efficient allocation of computing power resources with minimal resource consumption, improve resource utilization, and reduce system operating costs [4]. Focus on large-scale edge computing networks, deeply analyze the characteristics and requirements of its computing resource scheduling, explore an innovative low overhead scheduling method, in order to provide strong technical support for the development of edge computing networks, and promote the wide application of edge computing in more fields [5].

Sheng et al. [6] proposed a multi-objective task scheduling algorithm based on an improved competitive deep double-Q network. Firstly, the deep double-Q network decomposes the maximum operation in the target into action selection and evaluation to eliminate overestimation. It uses an immediate reward experience sample classification method to store and prioritize important samples, improving sample utilization and training speed; Then, introduce competitive network structure optimization neural network; Finally, the stability of the algorithm is improved using soft update methods, and the dynamic greedy exponential descent method is used for optimization. The Pareto optimal solution is obtained through different linear weighted combinations to minimize response time and energy consumption. Although the improved competitive deep double-Q network can reduce fluctuations and uncertainties in the decision-making process and enhance system stability. However, due to the complexity of the algorithm and limitations of the training data, the improved competitive deep Q-network may have limited generalization ability in new environments or unseen situations, resulting in performance degradation, reduced resource utilization, and increased task latency and energy consumption.

Hu et al. [7] proposed a particle swarm optimization algorithm based on hierarchical learning for allocating computing resources. Firstly, the particle swarm algorithm is used to optimize the computing speed, download power consumption, data offloading percentage, and network bandwidth of mobile devices to minimize total energy consumption. Then, in response to the problem of particle swarm optimization being prone to local optima and low diversity, a hierarchical learning strategy is introduced to improve the algorithm and enhance its global search capability. Finally, improving the algorithm effectively achieves rational allocation of computing resources and reduces total energy consumption. Although the algorithm can search at different levels through a hierarchical learning strategy, it can explore the solution space more comprehensively and avoid getting stuck in local optima. This helps to find better scheduling solutions in situations where computing power resources are limited. However, due to the introduction of hierarchical learning strategies, the computational complexity of the algorithm increases, especially when dealing with large-scale problems or complex environments, requiring more computing resources to perform hierarchical learning and particle update operations. Under the constraint of low computational resource consumption, the execution efficiency of the algorithm decreases.

Li et al. [8] proposed a resource scheduling method based on Lagrange multiplier method and joint optimization strategy for communication resources. Firstly, based on signal strength and traffic load, the access point (AP) switching strategy utilizes the global view and centralized control capabilities of the SDN controller to obtain, manage, and analyze information, calculate weights, and compare them to formulate switching strategies; Then, in order to improve system resource utilization, meet different application performance requirements, and maintain user QoS experience, a joint optimization strategy for computing and communication resources based on Lagrange multiplier method is proposed. By calculating and analyzing the task execution delay and energy consumption of edge servers and local terminals, optimization schemes for sub channel allocation and resource allocation are formulated. Although the Lagrange multiplier method transforms the allocation of computing power resources and communication constraints into Lagrange functions, it achieves unconstrained solving of constrained optimization problems. However, this method is usually optimized based on static or quasi-static assumptions, but in dynamic scenarios such as wireless communication, network topology, user demands, and channel states may change rapidly. If the scheduling strategy fails to respond to these changes in a timely manner, it may lead to delayed resource allocation and inability to meet real-time QoS requirements.

The Markov decision process models the dynamic environment as a framework of states, actions, transition probabilities, and reward functions based on Markov properties, accurately describing the relationship between edge node computing power state changes and scheduling decisions. By solving the optimal strategy, it maximizes long-term benefits and effectively addresses the uncertainty and randomness of network states; The hybrid actor critic algorithm integrates strategy gradient and value function estimation, with the actor network directly learning the optimal scheduling strategy and the critic network evaluating the strategy value. The two guide each other to improve performance, especially in continuous action space tasks. This algorithm can efficiently handle complex computing power allocation problems and reduce computational complexity [9]; After combining the two, the Markov decision process provides a theoretical framework and optimization objectives, while the hybrid actor critic algorithm provides an efficient solution method. While reducing the cost of computing resources, the scheduling strategy can be dynamically adjusted based on real-time network status and task requirements, achieving a dual improvement in computing resource utilization and task execution efficiency, and enhancing the system's adaptability to dynamic network environments [10]. Therefore, based on the advantages of Markov decision process and hybrid actor critic algorithm, this paper studies and analyzes the low-cost scheduling of computing resources. The core route of this paper is as follows:

(1) Construction of large-scale edge computing network structure. Build a three-tier "end edge cloud" architecture for large-scale edge computing. Terminal collection, edge processing upload, cloud storage analysis, reducing transmission bandwidth and cloud load, and improving system performance.

(2) Establish goals for computing resource scheduling. Clearly define transmission efficiency, computation delay, transmission delay, and energy consumption as scheduling objectives, measured by cumulative weighted overhead. Set weights, balance objectives, and achieve efficient resource utilization with low costs.

(3) Low-cost scheduling strategy for computing power resources. The dynamic characteristics of edge computing are modeled based on MDP. For mixed action spaces, the HA2C algorithm is used to split actions and introduce experience replay to achieve low-cost scheduling of computing resources. Low-cost scheduling of computing power resources

1.1 Large scale edge computing network architecture

Large scale edge computing refers to the deployment of a large number of computing, storage and network resources at the network edge to support the real-time data processing requirements of large-scale IoT devices and applications. With the rapid development of IoT technology, more and more devices are connected to the network, generating massive amounts of data [11]. If all of this data is transmitted to the cloud for processing, it will not only occupy a large amount of network bandwidth, but may also lead to network congestion and latency. Large scale edge computing transfers data processing and analysis tasks from the cloud to the network edge to improve the real-time, reliability and flexibility of the system [12].

Edge computing is a distributed computing model, which pushes data processing and storage capacity from the centralized cloud data center to edge devices closer to the source or end user of data generation. The core idea of edge computing is to perform computing tasks as close to the data source as possible to reduce network transmission time and cost [13]. By deploying intelligent hardware at the network "edge", such as smart phones, tablets, sensors, routers, etc., edge computing can complete the collection, processing and decision-making process of some data locally, and only upload necessary information to the cloud.

According to different composition and functions, edge computing can be divided into a three-tier architecture of "end edge cloud". The edge layer is a computing layer inserted between the traditional terminal layer and the cloud layer. The specific composition of each layer is as follows:

(1) Terminal layer: The terminal layer is located at the bottom of the architecture and consists of sensing devices that are close to the user and have data collection and information perception functions, such as smartphones, sensor modules, etc. The terminal layer is mainly responsible for perceiving and collecting user information and event data, and uploading these data to the edge layer for processing [14].

(2) Edge layer: The edge layer is located between the terminal layer and the cloud layer, mainly composed of devices such as gateways and set-top boxes with certain storage and computing capabilities, also known as "edge nodes". This layer can localize the data collected by the terminal layer, provide localization services, and upload the data to the cloud layer according to needs; It can also receive instruction information sent by the upper layer and execute corresponding operations.

(3) Cloud layer: The top layer of the architecture is the cloud computing center layer, which is composed of servers with high computing and storage capabilities. The cloud center adopts a centralized computing model, which can store and analyze massive amounts of data transmitted from the bottom layer for a long time, providing comprehensive information decision-making services [15].

Edge computing is responsible for local analysis and processing of nearby data and provides real-time services by introducing an edge layer with certain storage and computing capabilities, which greatly reduces the network bandwidth pressure of data transmission and the load pressure of cloud computing center layer [16]. Build the overall architecture of edge computing.

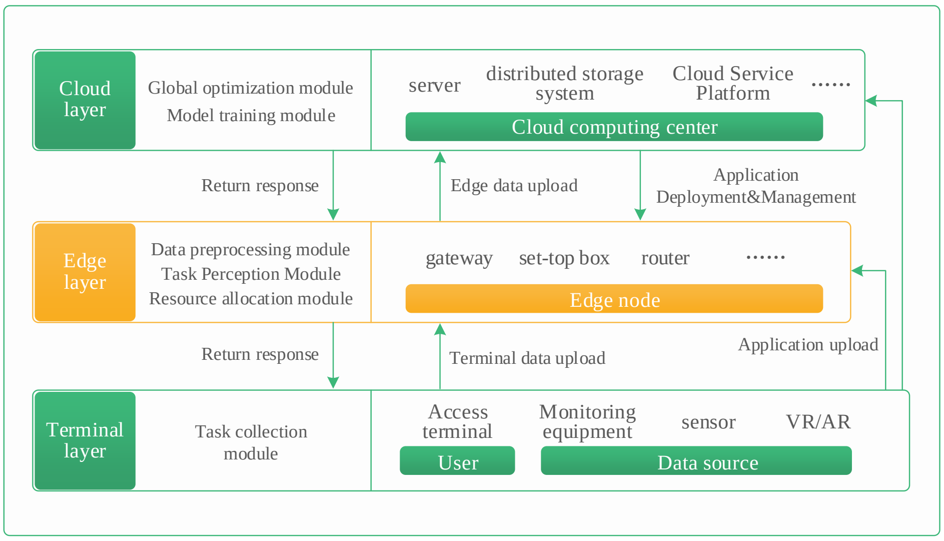


Figure 1: Overall architecture of edge computing

1.2 Scheduling objective setting

In the low overhead scheduling of computing resources in large-scale edge computing networks, there are multiple conflicting goals. By setting scheduling objectives, the importance and priority of each objective can be clarified, providing guidance for the formulation of scheduling strategies and finding a balance point between multiple conflicting objectives to ensure that scheduling methods achieve efficient resource utilization and low overhead while meeting system real-time, reliability, and flexibility requirements [17-18].

1.2.1 Transmission efficiency

Transmission efficiency is committed to improving the transmission efficiency of data in the network, ensuring that data can quickly and accurately reach the destination from the source, while reducing packet loss and error rates during transmission, thereby enhancing the overall performance of the system. When the computing resource scheduling task is unloaded to the edge computing node for processing, the task related data will be uploaded to the computing node through the wireless network [19-20]. The transmission delay is affected by the amount of data uploaded by the task and the channel state. In the actual transmission process, the amount of uploaded data is determined by the actual task, while the channel state follows a fading model, that is, the channel state changes at any time. According to Shannon's law, the channel data transmission rate can be expressed as:

$$Tr = \delta \cdot \log_2 \left(1 + \frac{SNR}{noise} \right) \qquad (1)$$

In the formula, δ represents channel bandwidth; $noise$ represents Gaussian white noise; SNR follows the Nakagami-m distribution, representing the channel fading model, and its distribution function calculation formula is as follows:

$$Q(z) = \frac{2m^m z^{2m-1}}{\Gamma(m) Qr^m} \exp\left(-\frac{-mz^2}{Qr^m} \right) \qquad (2)$$

In the formula, Qr represents the average power; $\Gamma(m)$ represents gamma function; m represents the fading coefficient. When $m = 1$, formula (2) degenerates into Rayleigh fading; When $m = \infty$, it represents no decay. By changing the value of m, SNR can be transformed into various fading models.

1.2.2 Calculate delay

After the task offloading decision is given by the agent, the task is scheduled to the corresponding computing node for processing, and the computation delay of the task is determined by the computation amount CPU_{task} of the task and the computation capacity CPU_{node} of the corresponding computing node. The calculation formula is as follows:

$$L_c = {CPU_{task}}/{CPU_{node}} \qquad (3)$$

1.2.3 Transmission delay

When a pending task is scheduled to be processed at the edge, the processing delay of the task is composed of computation delay and transmission delay. The formula for calculating transmission delay is as follows:

$$L_\tau = {Data_{task}}/{Tr} \qquad (4)$$

In the formula, $Data_{task}$ represents the total amount of data that needs to be uploaded when the task is offloaded to the edge end; Tr represents the current channel data transmission rate.

1.2.4 Energy consumption

The total energy consumption of a task from start to finish is composed of transmission energy consumption and computation energy consumption. The determining factors of transmission energy consumption include the unit time transmission energy consumption NC_u and the task transmission delay, and their calculation formulas are as follows:

$$NC_\tau = NC_u \cdot L_\tau \qquad (5)$$

The determining factor for calculating energy consumption is the calculation delay of the energy consumption task per unit time, and its calculation formula is as follows:

$$NC_c = NC_u \cdot L_c \qquad (6)$$

The total energy consumption for processing a task can be expressed as:

$$NC = NC_\tau + NC_c \qquad (7)$$

1.2.5 Accumulated weighted expenses

The comprehensive evaluation index for the performance of scheduling strategies is the long-term weighted overhead $Cost$, which is the weighted sum of the calculation delay, transmission delay, and energy consumption of the computing system. The overall weighted cost is expressed as:

$$Cost_i = \sum_{b=1}^{B}\sum_{d=1}^{D}\alpha L_c + \sum_{b=1}^{B}\sum_{d=1}^{D}\beta L_\tau + \sum_{b=1}^{B}\sum_{d=1}^{D}\gamma NC \qquad (8)$$

$$Cost = \sum_{i}^{itr} Cost_i \qquad (9)$$

In the formula, α 、 β 、 γ represents the weight of computation delay, transmission delay, and energy consumption, with a higher value indicating greater emphasis on this part of the overhead and $\alpha+\beta+\gamma=1$; $Cost_i$ represents the cost of the i-th iteration; B represents the number of tasks per iteration; D represents the number of subtasks included in each task. $Cost$ represents the overall weighted cost, where itr represents the total number of iterations. Thus, an effective balance between the objectives of computing resource scheduling has been achieved.

1.3 Low cost scheduling strategy for computing power resources

Large scale edge computing networks are highly dynamic, including the joining and leaving of devices, changes in network topology, randomness of task arrival, etc. [21]. Setting only static goals cannot adapt to this dynamically changing environment. In actual operation, it is necessary to adjust the scheduling strategy in a timely manner according to the dynamic changes of the system to ensure comprehensive optimization of multiple objectives in different situations. Markov decision process (MDP) can model the offloading decision and resource allocation based on the probability of system state transition. MDP models the dynamic behavior of the system through state space, action space, and reward function, and then selects the optimal scheduling action based on the current system state, and achieves the global optimization goal with long-term cumulative rewards [22]. MDP has no aftereffect and is highly adaptable to the dynamic characteristics of edge computing, thus providing a solid theoretical support and an effective solution framework for low overhead scheduling problems. The following is the Markov decision process transformation process [23].

The MDP model in the edge computing scenario can be expressed as a five tuple: $\chi = (S, A, P, R, \Upsilon)$, where S is the state space; A is the action space; P represents the state transition rate; R is the reward function, $\Upsilon \in [0,1]$ is the discount factor, balancing the importance of current and future expenses (usually taking $\Upsilon \geq 0.9$).

(1) State space: describes the current state of the system, including: the current utilization rate of computing power resources of edge nodes; Task queue status (calculation amount CPU_{task}, data amount $Data_{task}$ of pending tasks); Channel state (such as instantaneous signal-to-noise ratio SNR of Nakagami-m fading model). The state vector is represented as:

$$s(t) = [CPU_{node}, CPU_{task}, Data_{task}, SNR] \in S \qquad (10)$$

(2) Action space: After collecting status information, the scheduler will make decisions and send control action signals to the MEC server. The decisions of the scheduler include: task offloading decisions (local processing, edge node processing, or cloud offloading); Resource allocation ratio (such as the number of CPU cores allocated to tasks and bandwidth allocation). The action vector is represented as:

$$a_t \in A = \{Local, Edge_1, \cdots, Edge_N, Cloud\} \qquad (11)$$

In the formula, a_t represents the decision action made by the scheduler at time t; A represents the action space, which is the set of all possible actions that the scheduler can take; $Local$ represents local processing action; $Edge_N$ represents the N-th edge node processing action; $Cloud$ represents the cloud uninstallation action.

(3) State transition probability: Given state s_t and action a_t, the probability of transitioning to the next state s_{t+1} is $P(s_{t+1} | s_t, a_t)$. In edge computing, state transition is affected by the randomness of task arrival, channel fading formula (2) and computing resource dynamics.

(4) Reward function: The reward function $R(t)$ represents the instantaneous reward obtained when selecting action a_t in state s_t. Measuring the immediate cost of an action, designed as the negative value of weighted overhead (minimization objective):

$$R(s_t, a_t) = -(\alpha L_c + \beta L_\tau + \gamma NC) \quad (12)$$

In the formula, the weight must satisfy $\alpha + \beta + \gamma = 1$. By minimizing the negative value of weighted overhead, it is actually maximizing the long-term performance of the system.

By solving MDP, the optimal task scheduling strategy is found to maximize the long-term expected reward (i.e. minimize weighted overhead). The calculation formula is as follows:

$$\pi^* = \arg\max_\pi E\left[\sum_{t=0}^{\infty} \Upsilon^t R(s_t, a_t)\right] \quad (13)$$

In the formula, π represents the scheduling strategy; π^* represents the optimal scheduling strategy; E represents expectation.

In conclusion, in the edge computing network computing resource scheduling strategy, the action space is defined when constructing the Markov decision process model. This action space covers different types of actions such as task offloading decisions and resource allocation ratios, exhibiting the characteristics of a mixed action space that simultaneously encompasses both discrete and continuous actions [24].

SAC typically requires the construction of a joint strategy network and value function for mixed action spaces, and the introduction of additional temperature parameter automatic adjustment mechanisms. This leads to a complex model structure, difficulty in hyperparameter tuning, and the need for joint optimization of discrete and continuous actions for each policy update. In large-scale edge computing networks, the state and action space is already huge. This complexity of SAC will significantly increase the computational overhead and convergence time. PPO is essentially an online strategy algorithm, and its exploration efficiency depends on the current strategy. In complex mixed action spaces, PPO may find it difficult to efficiently explore all possibilities of "discrete continuous" combinations and may easily fall into local optima. The HA2C algorithm is specially designed for the mixed action space, which can implement targeted processing for different types of actions. It effectively overcomes the shortcomings of traditional methods in the mixed action space that it is difficult to effectively explore the optimal strategy and balance the impact of different types of actions on system performance, so as to more accurately explore the optimal task scheduling strategy, minimize the weighted overhead, and meet the demand for low-cost scheduling of large-scale edge computing network computing resources [25]. Therefore, the HA2C algorithm is adopted to accurately explore the optimal scheduling strategy.

The workflow of the HA2C algorithm consists of two parts, namely the environment interaction part and the network update part. In the environment interaction section, the scheduler takes the environment state as input to the actor network, outputs a mixed action a_t, and obtains the next state S_{t+1} and current reward $R(t)$ after interacting with the environment. The network update section updates the actor network and critic network parameters based on the interaction parameters. Constructing the Structure Diagram of HA2C Reinforcement Learning Algorithm [26].

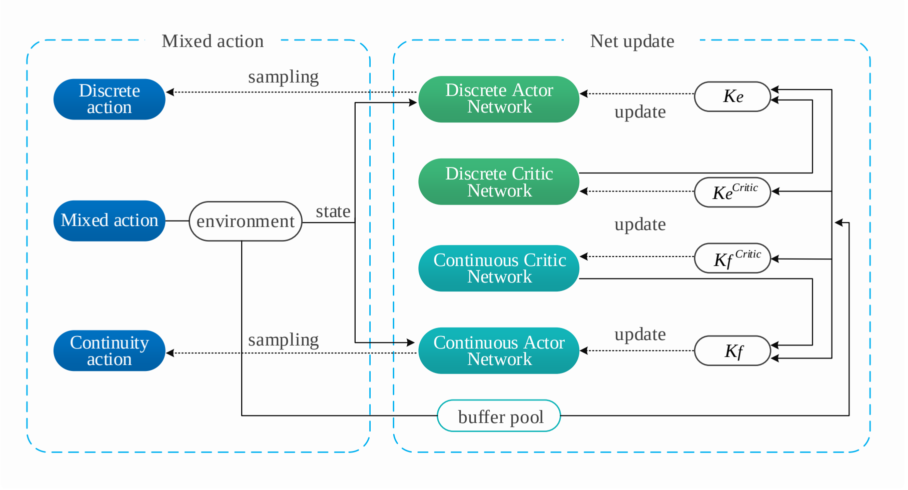


Figure 2: HA2C reinforcement learning algorithm

It is difficult to directly output continuous discrete mixed actions on the neural network in the environmental interaction part, so the mixed action a_t is divided into discrete action a_t^e and continuous action a_t^f, and discrete actor network, continuous actor network, and corresponding critic network are constructed. Discrete strategies, continuous strategies, and their corresponding value functions are parameterized as:

$\mu_e\left(a_t^e \middle| s_t,\theta\right)$, $\mu_f\left(a_t^f \middle| s_t,\omega\right)$, and $V_e\left(s_t;\varphi_e\right)$, $V_e\left(s_t;\varphi_e\right)$.

In practical scenarios with low computational resource overhead scheduling, difficulty in obtaining real-time rewards in asynchronous environments is a common problem. To address this issue, an experience replay mechanism has been introduced. This mechanism can store and transfer tuple $\rho_t = \left\{s_t, a_t, R(t), s_{t+1}\right\}$ to a buffer pool for subsequent network updates.

In the network update section, the HA2C algorithm is an improvement of the A2C algorithm, and the two neural network update methods are similar. The advantage actor critic (A2C) measures the selected actions compared to the average actions through the advantage function. The advantage function of the discrete network in the HA2C algorithm is defined as:

$$A_e\left(s_t,a_t\right) \approx R(t) + \Upsilon V\left(s_{t+1} \middle| s_t,a_t\right) - V\left(s_t\right) \quad (14)$$

In the formula, Υ represents the discount factor.

In the HA2C algorithm, actions generated by discrete and continuous networks are combined into mixed actions to interact with the environment and receive environmental rewards [27]. For discrete actor networks, the scheduler randomly samples task scheduling actions from the Categorical distribution, and the loss function of the discrete actor network is:

$$K_e = -\left\{\ln \mu_e\left(a_t^e \middle| s_t;\theta\right) A_e\left(s_t,a_t;\varphi\right) + \eta_e H_e\left(\mu_e\left(s_t;\theta\right)\right)\right\} \quad (15)$$

In the formula, η_e represents the entropy coefficient, which is used to control the regularization strength of entropy; $H_e\left(\mu_e\left(s_t;\theta\right)\right)$ encourages the exploration of the scheduler to increase the entropy of the discrete strategy $\mu_e\left(s_t;\theta\right)$, preventing premature convergence to suboptimal strategies during training. The detailed representation of parameter updates for discrete actor networks is:

$$\theta \leftarrow \theta + \frac{\partial \ln \mu_e\left(a_t^e \middle| s_t;\theta\right)}{\partial\theta} A_e\left(s_t,a_t;\varphi\right) + \eta_e \frac{\partial H_e\left(\mu_e\left(s_t;\theta\right)\right)}{\partial\theta} \quad (16)$$

Due to the asynchronous nature of task scheduling and resource allocation, even if all tasks have been scheduled, MEC servers still cache some tasks that require custom adjustment of computing resource allocation based on their status. In the context of low-cost scheduling of computing resources, this custom adjustment is particularly important to ensure that resources are utilized more reasonably and efficiently. Therefore, in order to assist in the subsequent allocation of computing resources, rewards are designed for both discrete and continuous actions, that is, there is a critic network for each. The loss functions of discrete critic networks are:

$$K_e^{Critic} = \left(R(t) + \Upsilon V_e\left(s_{t+1};\varphi\right) - V_e\left(s_t\right)\right)^2 \quad (17)$$

The parameter update formula for discrete Critic networks is formulated as:

$$\varphi \leftarrow \varphi + \left(R(t) + \Upsilon V_e\left(s_{t+1};\varphi\right) - V_e\left(s_t\right)\right) \frac{\partial V_e\left(s_t;\varphi\right)}{\partial\varphi} \quad (18)$$

The scheduler randomly samples the MEC server frequency allocation action for the number of sensors from the Gaussian distribution. Combined with servers and sensors in edge computing, the system can obtain new parameters in time to achieve scheduling optimization. By updating the action frequency of sensors, the process of real-time updating parameters is optimized to reduce the overhead, and finally the low-cost scheduling of large-scale edge computing network computing resources is realized. Build a low-cost scheduling flowchart for computing resources.

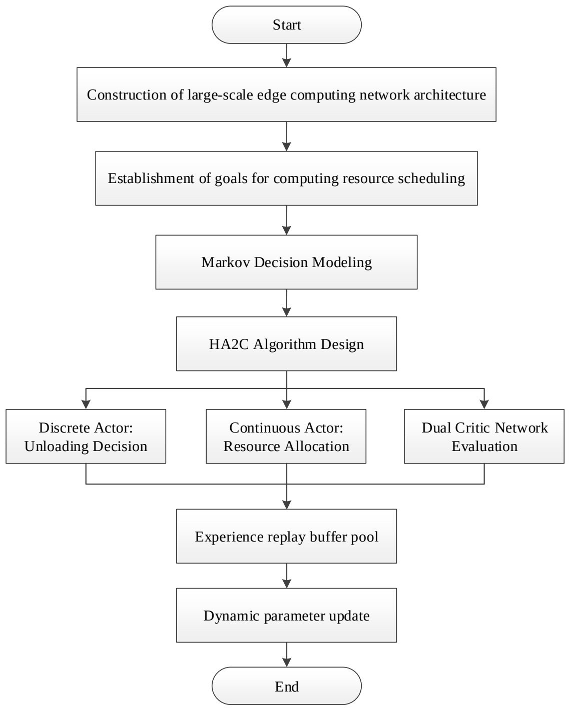

Start
Construction of large-scale edge computing network architecture
Establishment of goals for computing resource scheduling
Markov Decision Modeling
HA2C Algorithm Design
Discrete Actor: Unloading Decision
Continuous Actor: Resource Allocation
Dual Critic Network Evaluation
Experience replay buffer pool
Dynamic parameter update
End

Figure 3: Flow chart of low-cost scheduling of computing resources

In summary, the pseudocode for implementing the HA2C algorithm is as follows:

```
import torch
from torch.optim import Adam

class HA2C:
    def __init__(self, state_dim, discrete_action_dim, continuous_action_dim):
        # Core network architecture
        self.discrete_policy = DiscretePolicyNetwork(state_dim, discrete_action_dim)
        self.continuous_policy = ContinuousPolicyNetwork(state_dim, continuous_action_dim)
        self.critic = CriticNetwork(state_dim)  # Simplified to single critic

        # Combined optimizer
        self.optimizer = Adam([
            {'params': self.discrete_policy.parameters()},
            {'params': self.continuous_policy.parameters()},
            {'params': self.critic.parameters()}
        ], lr=0.0005)

        # Key hyperparameters
        self.gamma = 0.99
        self.batch_size = 256
    def collect_trajectories(self, env, n_episodes):
        for _ in range(n_episodes):
            state = env.reset()
            while True:
                # Hybrid action generation
                d_action, _ = self.discrete_policy.sample(state)
                c_action, _ = self.continuous_policy.sample(state)
                next_state, reward, done = env.step((d_action, c_action))

                # Store transition
                self.replay_buffer.push(state, (d_action, c_action), reward, next_state, done)
                if done: break
                state = next_state
    def update(self):
        if len(self.replay_buffer) < self.batch_size: return

        states, actions, rewards, next_states, dones = self.replay_buffer.sample(self.batch_size)

        # Value function update
        with torch.no_grad():
            target_v = rewards + self.gamma * (1 - dones) * self.critic(next_states)
        critic_loss = torch.nn.functional.mse_loss(self.critic(states), target_v)

        # Policy update
        _, d_logprob = self.discrete_policy.evaluate(states, actions[0])
        _, c_logprob = self.continuous_policy.evaluate(states, actions[1])
        policy_loss = -(d_logprob + c_logprob) * (target_v - self.critic(states)).detach()

        # Combined optimization
        self.optimizer.zero_grad()
        (critic_loss + policy_loss.mean()).backward()
        self.optimizer.step()
```

2 Experimental analysis

2.1 Experimental setup

To verify the effectiveness of the proposed method, set the number, type, resource requirements, deadline and other parameters of tasks to simulate the actual task scenarios in large-scale edge computing networks. Generate 10000 scheduling tasks, all divided into ten groups. Each task has random resource requirements and deadlines, and task types include compute intensive, data intensive, and hybrid tasks to simulate the diversity of tasks in practical application scenarios. This study conducted experiments using a custom discrete event simulator built on Python and SimPy libraries. Build a large-scale edge computing network structure diagram.

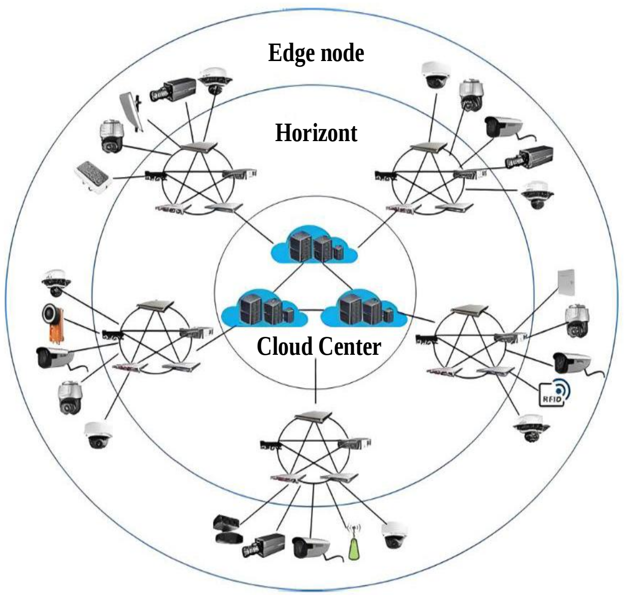


Figure 4: Large scale edge computing network structure

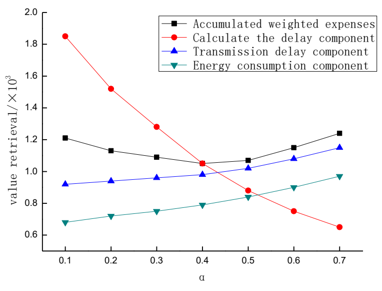
This network employs a three-layer "end-edge-cloud" architecture. The terminal layer, at the bottom, comprises devices like smartphones and sensor modules that collect and upload user and event data to the edge layer. The edge layer, situated between the terminal and cloud layers, consists of edge nodes such as gateways and set-top boxes with storage and computing capabilities; it localizes and processes terminal data, uploads it to the cloud as needed, reducing network bandwidth pressure and cloud computing center load. The cloud center, at the top, utilizes servers with high computing and storage capabilities in a centralized model to store and analyze massive data from the lower layers, providing comprehensive decision-making services for the system.

For the convenience of data calculation, construct an experimental environment table.

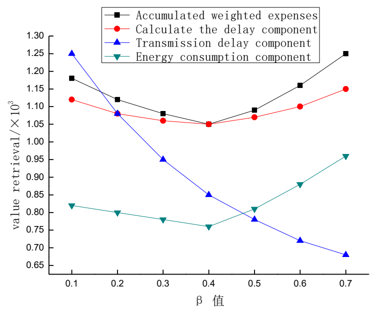
Table 1: Experimental environment

Project	Configuration
System	WIN 10
CPU	Intel Pentium Dual-Core 2.4GHZ
Programming Language	Python 3.8
Graphics card	RTX 3090
Monitor	Asus VG27AQL3A

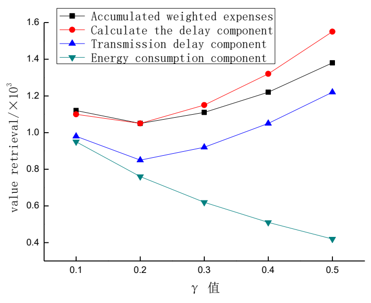
Among the parameters of the proposed method, the reward function weights α, β, and γ directly determine the optimization direction of the scheduling strategy. To evaluate the sensitivity of the proposed scheduling strategy to these hyperparameters and gain a deeper understanding of the trade-off between computational delay, transmission delay, and energy consumption, a systematic sensitivity analysis experiment was conducted, and a reward function weight sensitivity result graph was constructed.



（a）Sensitivity analysis data of delay weight α



（b）Sensitivity analysis data of transmission delay weight (β)



（c）Sensitivity analysis data of energy consumption weight (γ)

Figure 5: Sensitivity results of reward function weight

As α increases (with greater emphasis on reducing computational latency), the average computational latency of the system significantly decreases. At this point, the strategy tends to offload tasks to high-performance (but more energy consuming) edge nodes or the cloud, which may overlook energy efficiency optimization and cause fluctuations in transmission latency. The cumulative weighted cost reaches its lowest point near α=0.4, indicating that the benchmark weight is close to the local optimal solution; Both too high and too low alpha will increase the total cost, and the strategy is sensitive to extreme weights but performs stably within a reasonable range. Increasing β can effectively reduce the average transmission delay of the system. The strategy prioritizes selecting links with good channel conditions and high upload rates, but may sacrifice computational delay and increase energy consumption. The cumulative weighted overhead also reaches its minimum value around β=0.4. Increasing the gamma energy suppresses the total energy consumption of the system, and the strategy tends to use nodes with higher energy efficiency, even leaving some tasks for local processing. But this' energy-saving mode 'will lead to a significant increase in computation and transmission latency. When the gamma is too high (such as 0.5), the energy consumption is the lowest but the latency skyrockets, and the cumulative weighted overhead increases significantly.

During the training phase, the model only deals with two types of tasks: computationally intensive and data intensive. In the testing phase, introduce a new type of task that the model has never seen before: delay sensitive tasks. This type of task has extremely strict requirements for latency, but its computational and data volumes are moderate. The test set consists of a mixture of 80% known tasks (40% compute intensive and 40% data intensive) and 20% delay sensitive tasks. Test the generalization ability of the proposed method in a mixed task flow containing this new task type, and construct a performance result table for unknown task types based on the test results.

Table 2: Performance comparison of unknown task types before and after the application of the method in this article.

	Known task type overhead	Includes unknown task type overhead	Degree of performance degradation
After application	1.05×10^3	1.22×10^3	+16.2%
Before application	1.87×10^3	2.45×10^3	+31.0%

When faced with challenges of unknown task types, the performance of all methods deteriorates, but the degradation degree of our method is the smallest. This indicates that HA2C learns general scheduling principles through the MDP framework, rather than memorizing patterns for specific task types. Therefore, when encountering delay sensitive tasks with novel deadline constraints, it can quickly adjust strategies based on its critic network's accurate evaluation of the system state, prioritize resource allocation to these more urgent tasks, and thus obtain stronger generalization ability.

2.2 Experimental results and analysis

2.2.1 Resource utilization rate

The method in this paper, the method in literature [5] and the method in literature [6] are used to study the low overhead scheduling of large-scale edge computing network computing resources, and the utilization efficiency of edge device resources is compared and analyzed. Construct a comparison chart of low-cost scheduling resource utilization rates for computing power resources.

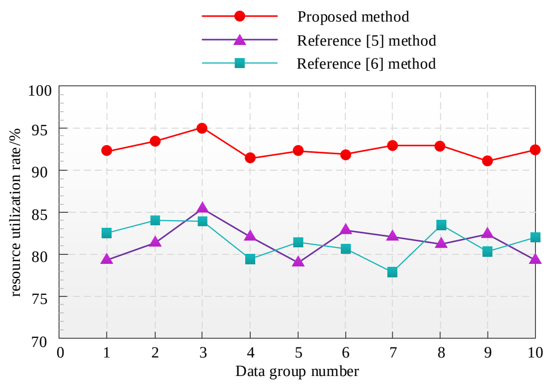


Figure 6: Comparison of low-cost scheduling resource utilization for computing power resources

It can be seen from Figure 6 that the resource utilization rate of low-cost scheduling of large-scale edge computing network computing resources in the literature [5] method is between 79% and 86%; The resource utilization ratio of low overhead scheduling of large-scale edge computing network computing resources in literature [6] is between 78% and 84%; However, the resource utilization ratio of the low overhead scheduling of large-scale edge computing network computing resources in this method is between 91% and 95%, and the resource utilization ratio is always above 90%. This is because the method proposed in this paper sets multiple objectives and evaluates the scheduling performance comprehensively through cumulative weighted overhead, so that the scheduling strategy needs to consider the impact of each objective from a global perspective when formulating. Avoiding the problem of performance degradation of other goals due to excessive bias towards one goal, it can find a balance point between multiple goals, achieve global optimal or near global optimal scheduling effects, and fully utilize the computing potential of edge devices.

To verify the statistical significance of the results, each experiment was independently repeated 30 times, and the mean, 95% confidence interval, and standard deviation were calculated. Use paired t-test (significance level α =0.05) to compare the performance differences between our method and the comparison method. Construct resource utilization statistics for three methods.

Table 3: Statistical results of resource utilization rates for three methods

Method	Mean	Standard deviation	Confidence interval
Proposed Method	93.2	1.05	[92.8, 93.6]
Reference [5] method	82.4	2.31	[81.5, 83.3]
Reference [6] method	80.9	2.67	[79.9, 81.9]

From Table 3, it can be seen that our method achieved an average resource utilization rate of 93.2%, significantly higher than the 82.4% of the method in reference [5] and the 80.9% of the method in reference [6], with an absolute improvement of over 10 percentage points. The standard deviation of this method (1.05) is much smaller than the two comparison methods. This indicates that the method proposed in this paper exhibits smaller fluctuations in different experimental rounds, and has stronger robustness and stability. The 95% confidence intervals for the method presented in this article are [92.8, 93.6], while the confidence intervals for the two baseline methods are [81.5, 83.3] and [79.9, 81.9], respectively. The three confidence intervals do not overlap at all, which proves the significance of the differences from a descriptive statistical perspective.

To conduct rigorous statistical inference, construct a paired t-test result table.

Table 4: Paired t-test results

Control group	t	p
This article's method vs the method in reference [5]	29.34	3.2×10^{-26}
This article's method vs the method in reference [6]	31.67	8.7×10^{-28}

Note: The significance level α is set to 0.05.

From Table 4, it can be seen that the obtained p-value is much smaller than the significance level α =0.05, which provides decisive evidence to reject the null hypothesis of "no difference in performance between the two methods". Therefore, it can be confirmed that the improvement in resource utilization efficiency of the method proposed in this article is not caused by random fluctuations, but rather an essential improvement with high statistical significance.

2.2.2 Response time

The method in this paper, the method in literature [5] and the method in literature [6] are used to study the low-cost scheduling of computing resources in large-scale edge computing networks, and the response time of resource scheduling is compared and analyzed. Build a response time comparison table for low-cost scheduling of computing resources.

Table 5: Comparison of response time for low-cost scheduling of computing resources

Data group number	Resource scheduling response time / ms		
	Proposed method/	Reference [5] method	Reference [6] method
01	3.22	11.53	12.47
02	2.56	12.26	16.87
03	2.57	12.58	18.38
04	2.87	11.82	16.17
05	3.58	11.42	15.85
06	2.92	11.87	18.31
07	2.73	10.24	12.73
08	2.52	12.33	13.52
09	3.61	13.23	17.17
10	2.85	11.86	18.82

Table 5 shows: literature [5] method has response times of 10.24ms - 13.23ms for low-cost scheduling in large-scale edge computing; [6] method ranges from 12.47ms - 18.82ms. Our method, however, achieves 2.52ms - 3.61ms, significantly lower and always under 4ms, indicating high scheduling efficiency. This is because our MDP modeling accurately captures dynamic edge environment characteristics via state, action spaces, and reward functions, enabling real-time status reflection, comprehensive operation coverage, and timely strategy adjustment for faster response.

2.2.3 Energy efficiency decline rate

To objectively evaluate the performance of the methods proposed in this article, literature [5], and literature [6] in terms of energy efficiency, the average task energy consumption, which is the average energy consumed by the system to process a single task, is used as the evaluation index to construct an energy efficiency comparison analysis chart, which intuitively reflects the energy utilization efficiency of different scheduling strategies when completing the same workload.

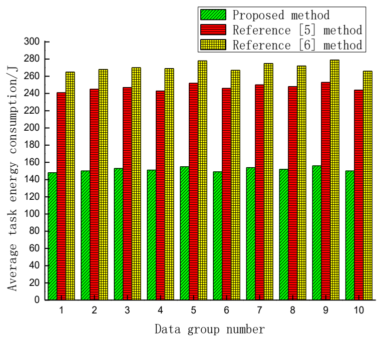


Figure 7: Comparative analysis results of energy efficiency

From Figure 7, it can be seen that in all 10 independent experiments, the HA2C scheduling strategy proposed in this paper achieved the lowest average task energy consumption, fluctuating within the range of 148-156 J with an average of 152 J. This is mainly attributed to the fine optimization ability of the HA2C algorithm in the mixed action space. Through a continuous Actor network, the algorithm can dynamically allocate "just right" computing resources based on the real-time computing requirements of the task and the current system state, avoiding the common over configuration or underconfiguration of resources in traditional methods. It shows that the scheduling strategy based on HA2C proposed in this paper can significantly and stably reduce the average energy consumption of system processing tasks through accurate, dynamic, and globally optimized resource allocation and task unloading decisions, so as to achieve a truly high-efficient and low-cost scheduling in large-scale edge computing networks.

3.2.4 Ablation experiment

To further explore the contribution of each core component in the HA2C algorithm to overall performance, three ablation variants were set up: no empirical replay mechanism, no dual critic network, and no mixed action processing. In the scenario of 10000 tasks, the performance of the complete HA2C algorithm was compared with the three ablation variants mentioned above, and a result table was constructed based on the cumulative weighted cost as the measurement standard.

Table 6: Comparison of ablation experimental performance

	Accumulated weighted expenses
The proposed method	1.05×10^3
Inexperienced replay	1.38×10^3
Unparalleled critic	1.29×10^3
No mixed action processing	1.67×10^3

The ablation experiment in Table 6 shows that mixed action processing is the fundamental architecture for achieving low-cost scheduling and solving the core problem; The experience replay mechanism is a key component to ensure efficient and stable learning of algorithms; The dual critic network provides precise guidance for collaborative optimization of mixed actions. These three components work together and are indispensable, which constitutes the fundamental reason why HA2C algorithm achieves significant performance gains in large-scale edge computing networks.

3.2.5 Scalability assessment

To verify the effectiveness and scalability of the proposed method, 10000 scheduling tasks were generated from the basic scenario and divided into ten groups. To conduct comprehensive load stress testing on the proposed method, two additional scenarios were set up: 50000 tasks (large-scale) and 100000 tasks (ultra large scale), all of which randomly generated corresponding resource requirements and deadlines. Task types include computation intensive, data intensive, and hybrid tasks to simulate the diversity of tasks in practical application scenarios. Thus, construct a weighted cost result table for the system under different task scales.

Table 7: Weighted cost results

Method	Task Scale		
	10,000	50,000	100,000
Proposed Method	1.05×10^3	5.32×10^3	10.81×10^3
Reference [5] method	1.87×10^3	9.91×10^3	20.76×10^3
Reference [6] method	2.14×10^3	11.52×10^3	24.13×10^3

From Table 7, it can be seen that in the large-scale scenarios of 50000 and 100000 tasks, the cumulative weighted overhead increases with the increase of task volume, which is an inevitable trend for all methods. However, the growth rate of the overhead of our method is much lower than that of the comparative method. When the task size increased from 10000 to 100000 (a tenfold increase), the cumulative weighted overhead of our method increased by about 10.3 times, showing an almost ideal linear growth trend. The cost of the methods in references [5] and [6] increased by approximately 11.1 times and 11.3 times, respectively, indicating a faster rate of cost growth. This almost linear low growth rate proves that the method proposed in this paper has excellent scalability and can efficiently cope with sharp increases in load. This is because the proposed method accurately collaborates to handle "task offloading" and "resource allocation" through separate discrete and continuous Actor networks. This decoupling architecture can more effectively find scheduling solutions that are close to the global optimum in a large mixed action space when facing large-scale tasks, avoiding the decline in decision quality caused by action space explosion. This is the main reason why baseline methods have faster cost growth when the load increases.

4 Conclusion

In order to solve the problems of low resource utilization rate of edge nodes and high overhead of dynamic task scheduling, this paper constructs an algorithm based on MDP and HA2C to study the low overhead scheduling method of large-scale edge computing network computing resources.

(1) By constructing a three-layer architecture of "end edge cloud" to optimize data flow and load distribution, combined with a scheduling target balancing strategy of cumulative weighted overhead, and utilizing MDP modeling dynamic characteristics and HA2C algorithm to handle mixed action space, efficient and low-cost scheduling of computing resources has been successfully achieved.

(2) The experimental results show that this method significantly shortens the response time of resource scheduling, improves the efficiency of resource

scheduling, maintains the resource utilization rate at more than 90%, effectively reduces energy consumption, and maintains a high scheduling accuracy, which provides an effective solution for resource optimization management of large-scale edge computing networks.

Funding

Key Scientific Research Project of Henan Province Universities in 2025; Project Name: Research on the Transformation and Upgrading Path of the New Quality Education Ecosystem in Universities in the Artificial Intelligence Era" by the Henan Provincial Department of Education; Project Number: 25B520047.

References

[1] Zhang F.F., Ge J.D., Li Z.J., et al. (2023). Cooperative Computation Offloading and Dynamic Task Scheduling in Edge Computing. *Journal of Software*, 34 (12): 5737-5756. http://dx.doi.org/10.13328/j.cnki.jos.006797

[2] Cang Y., Chen M., Pan Y., et al. (2024). Joint User Scheduling and Computing Resource Allocation Optimization in Asynchronous Mobile Edge Computing Networks. *IEEE Transactions on Communications*, 72 (6): 3378-3392. https://doi.org/10.1109/TCOMM.2024.3358237

[3] Wang Y., Lin X., Lou Z.L., et al. (2024). Uplink resource coordinated scheduling algorithm for edge-oriented optical computing power networks. *Optical Communication Technology*, 48(3), 45-51. https://doi.org/10.13921/j.cnki.issn1002-5561.2024.03.009.

[4] Xie Y.Y. and Chen L.S. (2025). Multi-objective Task Scheduling Method Based on Edge Computing. *Journal of University of Jinan (Science and Technology)*, 39 (01): 117-122. https://doi.org/10.13349/j.cnki.jdxbn.20241118.001

[5] Liu Y.P., Zhu Y.J., Bin Y.R., et al. (2022). Review of Research on Computing-Intensive Task Scheduling in Edge Environments. *Computer Engineering and Applications*, 58 (20): 28-42. https://doi.org/10.3778/j.issn.1002-8331.2202-0243

[6] Sheng Y., Zhu Z.W., Zhu C.Y., et al. (2023). Research on multi-objective edge task scheduling based on deep reinforcement learning. *Electronic Measurement Technology*, 46 (08): 74-81. https://doi.org/10.19651/j.cnki.emt.2210891.

[7] Hu X.M., Chen Z.F. and Li M. (2022). Level-Based Learning Swarm Optimization Algorithm for Resource Scheduling in Edge Computing. *Computer Engineering and Applications*, 58 (24): 107-115. https://doi.org/10.3778/j.issn.1002-8331.2110-0060

[8] Li C, Yu Z, Li X, et al. (2023). Low-latency AP handover protocol and heterogeneous resource scheduling in SDN-enabled edge computing. *Wireless networks*, 29(5): 2171-2187. https://doi.org/10.1007/s11276-023-03302-y

[9] Li J., Fan T.F., Gao H.L., et al. (2025). Research on Multi-core Task Offload Scheduling and Resource Adaptation in Edge Computing Networks. *Ordnance Industry Automation*, 44 (03): 29-34.

[10] Han D., Chen W. and Fang Y. (2020). Joint Channel and Queue Aware Scheduling for Latency Sensitive Mobile Edge Computing With Power Constraints. *IEEE Transactions on Wireless Communications*, 19(6): 3938-3951. https://doi.org/10.1109/TWC.2020.2979136

[11] Yamanaka H., Teranishi Y. and Kawai E. (2021). Service Migration Scheduling with Bandwidth Limitation against Crowd Mobility in Edge Computing Environments. *IEICE Transactions on Communications*, E104.B(3): 240-250. https://doi.org/10.1587/transcom.2020NVP0003

[12] Liu M. and Gong W. (2021). Resource Allocation of IoT Edge Computing Based on Joint Decision Model. *Computer Simulation*, 38 (12): 299-303.

[13] Cheng Y., Cao Z., Zhang X., et al. (2024). Multi objective dynamic task scheduling optimization algorithm based on deep reinforcement learning. *Journal of Supercomputing*, 80(5): 6917-6945. https://doi.org/10.1007/s11227-023-05714-1

[14] Wang Z, Wang G, Jin X, et al. (2022). Caching-based task scheduling for edge computing in intelligent manufacturing. *Journal of supercomputing,* 78 (4):5095-5117. https://doi.org/10.1007/s11227-021-04071-1

[15] Cha M.X., Zhu Y.J., Zhu L., et al. (2021). Resource scheduling algorithm for real-time stream data processing in edge computing. *Journal of Computer Applications*, 41 (S1): 142-148.

[16] Du R., Wang J. and Gao Y. (2024). Computing offloading and resource scheduling based on DDPG in ultra-dense edge computing networks. *Journal of supercomputing*, 80 (8):10275-10300. https://doi.org/10.1007/s11227-023-05816-w

[17] Qin B., Lei Q.Y. and Wang X. (2024). DGCQN: a RL and GCN combined method for DAG scheduling in edge computing. *The Journal of Supercomputing,* 80(13): 18464-18491. https://doi.org/10.1007/s11227-024-06140-7

[18] Zhang M., Liang S., He X.L., et al. (2020). Research on Engineering Project Resource Scheduling System Based on BIM and edge computing. *Construction Economics*, 41 (S1): 171-174.

https://doi.org/10.14181/j.cnki.1002-851x.2020S1171.

[19] Song S, Ma S, Zhao J, et al. (2022). Cost-efficient multi-service task offloading scheduling for mobile edge computing. *Applied Intelligence*, 52(4): 4028-4040. https://doi.org/10.1007/s10489-021-02549-2

[20] He H, Shan H, Huang A, et al. (2020). Edge-Aided Computing and Transmission Scheduling for LTE-U-Enabled IoT. *IEEE transactions on wireless communications*, 19(12):7881-7896. https://doi.org/10.1109/TWC.2020.3017207

[21] Ren J, Hou T, Wang H, et al. (2024). Collaborative task offloading and resource scheduling framework for heterogeneous edge computing. *Wireless Networks*, 30(9): 3897-3909. https://doi.org/10.1007/s11276-021-02768-y

[22] Wang L., Wu C.G. and Fan W.H. (2021). A Survey of Edge Computing Resource Allocation and Task Scheduling Optimization. *Journal of System Simulation*, 33 (03): 509-520. https://doi.org/10.16182/j.issn1004731x.joss.20-0584

[23] Diao X., Yang W., Yang L., et al. (2021). UAV-Relaying-Assisted Multi-Access Edge Computing with Multi-Antenna Base Station: Offloading and Scheduling Optimization. *IEEE Transactions on Vehicular Technology*, 70 (9): 9495-9509. https://doi.org/10.1109/TVT.2021.3101298

[24] Huang Y.F., Zeng W., Chen Z.Y., et al. (2024). Task scheduling method with Actor-Critic deep reinforcement learning in mobile edge computing. *Journal of Computer Applications*, 44 (S1): 150-155.

[25] Huang R. and Song G.L. (2024). Task Scheduling Optimization of Internet of Things Edge Computing Based on Offloading Strategy. *Journal of East China University of Science and Technology (Natural Science Edition)*, 50 (02): 264-273. https://doi.org/10.14135/j.cnki.1006-3080.20230320002

[26] Zhai Y.L., Bao T.H., Zhu L.H., et al. (2020). Toward Reinforcement-Learning-Based Service Deployment of 5G Mobile Edge Computing with Request-Aware Scheduling. I*EEE Wireless Communications*, 27(1): 84-91. https://doi.org/10.1109/MWC.001.1900298

[27] Liu F., Chen Z.Y., Ma K., et al. (2025). Real-time Low-power Scheduling Algorithm Based on Correlation for Task Partitioning in Edge Computing Environments. J*ournal of Chinese Computer Systems*, 46 (02): 289-296. https://doi.org/10.2009/j.cnki.21-1106/TP.2023-0459