# Multi-Scenario Unified Resource Scheduling in Edge Cloud-Native Environment

*Wei Xiong[1], Xinying Wang[1*], Zhao Wu[1], Qiaozhi Hua[1], Franz Wotawa[2]*

*[1] School of Computer Engineering, HuBei University of Arts and Science, China*
*[2] Institute for Software Technology, Graz University of Technology, Austria*
*xwei9093@126.com, wxy200888@126.com, wuzhao73@163.com, alex2441@163.com, wotawa@ist.tugraz.at*

## Abstract

The development of edge computing technologies has brought about challenges in resource management. Traditional resource scheduling policies often prove insufficient due to the dynamic nature of cloud-edge collaboration. Therefore, adopting an edge cloud-native approach becomes necessary. This paper proposed a unified resource scheduling approach for joint optimization across multiple scenarios in the edge cloud-native environment. Our approach can schedule dynamically mixed-service groups across multiple scenarios by utilizing the adversarial learning of the environment and agents. Our approach can address the latency issues arising from imbalances among multiple scenarios. We conduct experiments by considering some factors such as device-number, communication-distance, CPU-cycle, and task-generation-speed. The results show that our approach can achieve a higher offloading rate and better average performance.

**Keywords:** Edge cloud-native, Resource scheduling, Imitation learning, Reinforcement learning, Tiered traffic control

## 1 Introduction

As cloud-native technology matures, academia and business communities are exploring its practical implementation [1-2]. The integration of AI, IoT, and edge computing enhances the variety, scale, and complexity of business in edge computing scenarios [3]. As a result, there is a focus on new edge cloud-native platforms [4-5].

Edge cloud-native platform is utilized in various domains such as live video [6], cloud gaming [7], logistics and transportation [8], intelligent manufacturing [9], and urban brain [10]. These domains can be classified as mobile broadband services [11], large-scale IoT services connected to fixed sensors [12], and mission-critical IoT services [13-15] based on mobility, billing, security, policy control, delay and reliability. The ultra-large-scale edge cloud-native business is complex, and it faces technical challenges such as decentralized computing power, heterogeneous resources, and weak network connectivity. The principle of edge-cloud native is to combine discrete computing power into a larger resource pool. This optimizes resource scheduling and provides extreme energy efficiency by balancing peak and valley loads. When deploying mixed services and requesting resources from the management node, the scheduler is responsible for selecting appropriate physical machines to deploy these containers. Since the specifications of physical machines are not uniform and resource levels differ, different allocation methods yield different allocation rates. Therefore, a key task of the scheduler is to choose the most suitable physical machine from a pool of candidates based on a specific policy [16-17].

Scheduling computing resources is often seen as a vector-packing problem. If the number of containers for each application is predetermined, the scheduler can create an optimal deployment policy for all mixed services simultaneously. This situation can be framed as integer programming, solvable via general-purpose solvers or specifically designed algorithms. If requests from various applications reach the management node sequentially, the scheduler must generate deployment decisions immediately (online) with each request. The problem can be structured as a Markov Decision Process (MDP) [18], where the optimal policy can be determined through value iteration or policy iteration.

Scheduling policies include priority [19], DRF (dominant resource fairness) [20], binpack [21], Gang Scheduling [22], and other policies based on preset rules. Generally, these policies can achieve a good allocation rate. However, their effectiveness greatly diminishes when the bottlenecks are not concentrated in the same dimension [23]. In practical scheduling, considerations go beyond the resource dimension. They include factors such as disaster tolerance and interference isolation as well. For instance, services of the same application should not all be deployed on the same physical machine. Many applications even permit only one instance per machine. Some applications have a mutual exclusion relationship due to resource contention, which can significantly impact performance. For the same cluster resources, centralization is required for the scheduler. However, when multiple cluster schedulers exist simultaneously, decision conflicts may arise. The only solution is to forcibly divide the nodes using labels or deploy multiple clusters. Multiple sets of schedulers can also complicate maintenance and

*Corresponding Author: Xinying Wang; Email: wxy200888@126.com
DOI: https://doi.org/10.70003/160792642025112606002

create uncertainty in compatibility with the upstream Kube scheduler [24]. A pressing issue to address is how to build a unified scheduler based on the Kube-scheduler scheduling interface. Through a single scheduling protocol and system architecture, it should intelligently schedule underlying computing resources, support the deployment of mixed services, and improve resource utilization while guaranteeing application SLA. This is crucial to meet the high throughput, low latency service scheduling, and orchestration requirements for big data and AI.

Resource scheduling across the cloud and the edge utilizes prior knowledge and perceived context information to make decisions. These decisions are dynamically adjusted and executed, with their effectiveness determined by an evaluation system to ensure quality. Due to the large volume of micro-services, their wide distribution, and the need for low latency, there is a need for an approach that can combine perception with the lifecycle management of deployed services. This approach will provide support for system self-adaptation. Intelligent scheduling can be implemented according to runtime resources, which can reduce the complexity of the unified resource scheduler, improve runtime stability, and decrease resource costs.

This paper's contributions are as follows:

(1) We formulated the multi-scenario joint optimization problem as a cooperative, partially observable multi-agent system.

(2) We proposed a new multi-agent reinforcement learning called multi-scenario unified resource scheduling. This approach allows multiple agents to work together to achieve the best performance.

(3) We developed a hybrid architecture using Karmada, RunD, and Koordinator. We train the policy model through the duality and adversarial learning of the environment and agent and complete the online verification. Experiments revealed that our model significantly enhances the performance of resource scheduling.

The rest of this paper is organized as follows: Section 2 describes the motivation scenario. Section 3 presents our approach. Section 4 details our experiments. Section 5 discusses related work, and Section 6 concludes the paper.

## 2 Motivating Scenario

In the native edge cloud, there are numerous sub-scenarios. Each sub-scenario is optimized independently, creating a competitive relationship between them. Therefore, the performance improvement of each sub-scenario does not necessarily result in an overall one. The optimization policy addresses the sorting problem of multiple sub-scenarios by treating it as multiple fully cooperative and partially observable multi-agent sequential decision problems. Hence, the deployment policy for each scenario changes from independent to cooperative and mutually beneficial, as illustrated in Figure 1. Consequently, this paper studies the following topics:

(1) The dynamic equation of a time-varying system is fitted to represent the emergence of collective intelligence. A unified scheduler assesses overall reward through a comprehensive and global "referee". A communication module generates messages shared by multi-scenario mixed service groups. Each message encodes the historical observation and behavior of mixed service groups, approximating the global environment state. Every mixed service group gets its observations of a part of the environment and can receive messages sent by other groups. This model allows mixed service groups in different scenarios to work together to achieve the best global outcome.

(2) Flow control of mixed service groups in multiple scenarios is considered. The unified scheduler needs to address two key issues: efficiency and fairness. Under certain conditions, it's possible to achieve Pareto efficient allocation by altering the initial allocation state of endowments among individuals, thus considering fairness. Moreover, due to the unpredictability of mixed service cluster behavior, the instant reward can have a large variance, significantly impacting the learning. If the search is conducted in the entire real number space, it may not converge. To minimize instantaneous loss, we have designed upper and lower limits that enable only local search.

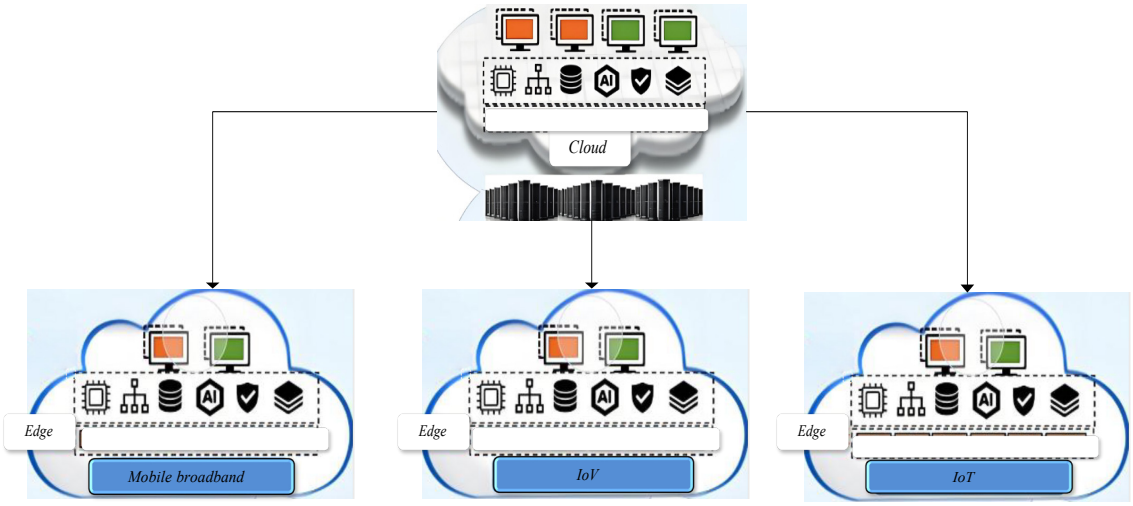



**Figure 1.** Multi-scenario under edge cloud-native environment

# 3 Our Approach

In this section, we give a formal description of the problem in Section 3.1, propose the framework in Section 3.2, build a multi-scenario virtual environment in Section 3.3, and train the environment and policy models simultaneously through the dual and adversarial learning in Section 3.4.

### 3.1 Problem Formulation

This paper proposes a unified resource scheduling approach to address the issues mentioned above. The goal is to improve overall performance metrics. The resource scheduling problem of multi-scenario joint optimization is considered a fully cooperative and partially observable multi-agent sequential decision problem. To model this problem, we use multi-agent reinforcement learning. Each scenario is viewed as an agent, and different resource scheduling policies in each scenario share the same objective. The scheduling results in one scenario consider the user's behavior and feedback in other scenarios. This approach transits each scenario's scheduling policy from independent to cooperative and mutually beneficial. Since we aim to utilize the user's behavior across all scenarios, the BiLSTM network is employed to remember historical information and explore continuous state and action spaces using the actor-critic method. Hence, we have named our algorithm MS-URS (Multi-Scenario Unified Resource Scheduling).

For the multi-agent model $\{A^1, A^1, ..., A^N\}$, each agent $A^i$ corresponds to a resource scheduling scenario. It is assumed that $o_t^{it}$ represents the state characteristics of all mixed service groups observed by the agent $A^i$ in time $t$. Because the action $a_t^i$ space is very large, the decision action space is formalized as <*scene*, *service*, *op*, *num*, *src*, *dst*>, where service represents mixed service groups, *op*∈{*Push*, *install*, *offload*, *upload*, *migrate*, *scale*}, *num* represents the number of service instances, *src* and *dst* represents the source and destination server nodes of service migration. $a^i = (\omega_1^i, ..., \omega_{N^i}^i)$, $a^i \in R^{N^i}$, where $\omega_{N^i}^i = \{e_{service}, e_{op}, e_{num}, e_{src}, e_{dst}\}$, $e_{service}$ represents the mixed service, $e_{op}$ represents action, $e_{num}$ represents the number of service instances, $e_{src}$ and $e_{dst}$ represents the source and target server nodes of service migration. The reward is

$$r(s_t, a_t^i) = \begin{cases} 1 & if\ QoS_{ij}\ \text{op}\ C_{ij} \\ -1 & otherwise \end{cases}, \quad \forall t \geq 0\ , \qquad (1)$$

where $QoS_{ij}$ op$C_{ij}$ represents that a j-th dimension of the service meets the agreed threshold $C_{ij}$. Multiple agents will cooperate to achieve the maximum reward.

### 3.2 Outline

As shown in Figure 2, the model includes a global "referee" to assess the overall reward. A communication module generates messages for multiple agents to share, encoding each agent's history of observations and actions, which serve to estimate the global state of the environment. Every agent processes its observations and receives information, subsequently producing an action. Specifically, this includes:

(1) Multi-agent: Each sub-scenario within the system has a unique policy. Every agent formulates a policy and simultaneously learns its policy function, mapping its state to an action.

(2) Sequential decision making: The agent sequentially interacts with the system, resulting in sequential actions. At any given time, the agent interacts with the actual scenario by executing a decision action. This current decision influences future decisions.

(3) Full cooperation: All agents collaborate to optimize the same target. Moreover, each agent communicates by sending messages to others, with the integrated referee assessing the overall reward.

(4) Partial observation: Every agent observes a segment of the environment and can receive messages from other agents.

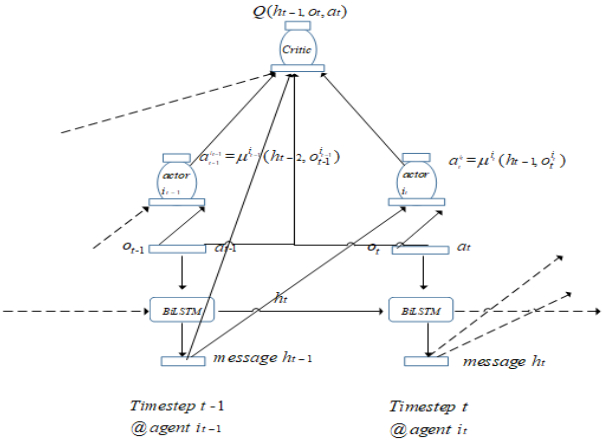



**Figure 2.** Framework of our approach

### 3.3 A Multi-Agent Reinforcement Learning Framework

The Critic in Figure 2 models the "action-value" function $Q(\delta_{t-1}, ot, a_t)$, which represents the overall reward you would get from taking an action $a_t$ when you receive information $\delta_{t-1}$ and observations $o_t$. Each agent produces a deterministic action, based on the function $a_t = \mu^i(\delta_{t-1}, o_t^i)$. The information is updated by the communication module, based on observations $o_t$ and actions $a_t$.

For simplicity, we'll focus on a case with 2 agents, each representing a self-optimizing policy and scenario. Our model, inspired by the Deep Policy Gradient method (DDPG [25-26]), is also based on the Actor-Critic method [27]. We've designed three crucial modules to foster coordination and cooperation among multiple agents: a global "Critic", individual agents, and a communication mechanism.

In a classical reinforcement learning problem, there will be an optimization problem of the form ($o_1$, $r_1$, $a_1$, …, $a_t$, $o_t$, $r_t$). The state represents experience, i.e. $s_t = f(o_1, r_1, a_1, …, a_{t-1}, o_t, r_t)$. We are addressing a challenge involving $N$ agents $\{A^1, A^1, ..., A^N\}$, where each agent is associated with a distinct scenario (e.g. mobile broadband, Internet

of things, Internet of vehicles, etc.). The observations ($o_t$ = ($o^1_t$, $o^2_t$, ..., $o^N_t$)), actions ($a_t$ = ($a^1_t$, $a^2_t$, ..., $a^N_t$)), and memories of short-term benefits ($r_t$ = $r(s_t, a^1_t)$, $r(s_t, a^2_t)$, ..., $r(s_t, a^N_t)$) are all owned by the individual agents.

The agent $A^i$ will take each decision action $a^i_t$ based on its policy $\pi^i(s_t)$ and state $s_t$, and then it will receive a temporary reward $r^i_t = r(s_t, a^i_t)$ from the environment while the state is updated from $s_t$ to $s_{t+1}$. The multiple agents will cooperate to achieve the maximum reward $Q(s_t, a^1_t, a^2_t, ..., a^N_t)$, which denotes the global reward due to action ($a^1_t$, $a^2_t$, ..., $a^N_t$).

The agent $A^{it}$ receives the current observation $o^{it}_t$ from the environment at time point $t$. The state is accessible to all agents and depends not solely by the prior state, actions and observation $o_t$. We have developed a communication module that employs a BiLSTM [28] network to encode prior observations and actions into a vector. Through the communication between the agents, the overall state can be approximated by $s_t \approx \{\delta_t, o_t\}$. Agent chooses an action $a^{it}_t = \mu^{it}(s_t) \approx \mu^{it}(\delta_{t-1}, o^{it}_t)$ to maximize the future reward, which is evaluated by a central critic $Q(s_t, a^1_t, a^2_t, ..., a^N_t)$. It is important to note that all combinations of observations $o_t$ = ($o^1_t$, $o^2_t$, ..., $o^N_t$) are obtained at time point $t$.

### 3.4 BiLSTM Message Mechanism

As shown in Figure 3, we have developed a messaging mechanism based on BiLSTM to allow agents to better cooperate by passing information to each other. More formally, the communication module works as follows:

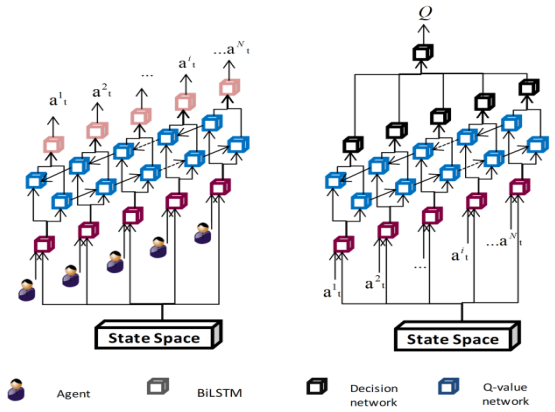



**Figure 3.** Message mechanism based on BiLSTM model

$$\delta_{t-1}=\text{BiLSTM}(\delta_{t-2},[o_{t-1};a_{t-1}];\varphi) \quad . \tag{2}$$

Benefiting from this message $\delta_{t-1}$, each agent can approximate the global state of the environment $s_t \approx \{\delta_{t-1}, o_t\}$. This solves the problem of each agent receiving local observations $o^i_t$ but not getting a global state $s_t$.

Loss is

$$L(\alpha,\omega,\phi;\tau_i)= \\ -\mathop{\mathrm{E}}_{q_\phi(z|x^i_{1:T_i})}\sum_{t=1}^{T_i}\left[\log\pi_\alpha(a^i_t \mid o^i_t,z)+\log p_\omega(o^i_{t+1} \mid o^i_t,z)\right] , \tag{3} \\ +D_{KL}(q_\phi(z \mid o^i_{1:T_i}) \,\|\, p(z))$$

where BiLSTM is used to encode $q_\phi(z|o^i_1{:}T_i)$, $z$ is the embedding obtained after encoder transformation, $p_\omega(o^i_{t+1}| o^i_t, z)$ is the state transition decoder, $\pi_\alpha(a^i_t| o^i_t, z)$ is the decoder of the policy, $D_{KL}(.||.)$ is the KL distance, and $L(\alpha, \omega, \phi; \tau_i)$ is the loss function. Its overall architecture is shown in Figure 3.

### 3.5 Private Actor

Each agent is an independent "actor" who takes in local observations and shared information and initially makes a decision. Since we are dealing with reinforcement learning with continuous actions, we define an action as a substantial vector $a^i = (\omega^i_1, ..., \omega^i_{N^i})$, $a^i \in R^{N^i}$. Thus, each action is a vector of $N^i$ dimensions, and each dimension is a real value. This vector will be used as a parameter to control the multi-scenario unified resource scheduling.

Since this is a continuous behavior type, similar work is common in control problems [12, 23, 34]. Inspired by related work, we utilize a deterministic policy approach rather than a random policy approach. Each agent has an actor corresponding function $\mu^i(s_i; \theta^i)$, where the parameters are $\theta^i$, and the function unambiguously maps a state to an action. At time $t$, the agent $A^{it}$ decides its action based on the actor-network:

$$a^{it}_t = \mu^{it}(s_t;\theta^{it}) \approx \mu^{it}(\delta_{t-1}, o^{it}_t;\theta^{it}) \quad , \tag{4}$$

where $s_t \approx \{\delta_{t-1}, o_t\}$ denotes the communication module. Thus, the actor's behavior depends on both the information $\delta_{t-1}$ and current observation $o^{it}_t$.

### 3.6 Centralized Critic

We have meticulously crafted a critical network architecture that is designed to precisely estimate the action-value function, adopting a sophisticated approach akin to the DDPG (Deep Deterministic Policy Gradient) algorithm, which is renowned for its ability to learn optimal policies in continuous action spaces. Because all agents can share a goal, we use a global critic function $Q(s_t, a^1_t, a^2_t, ..., a^N_t; \phi)$ to fit the future rewards.

In our approach, only one agent $A^{it}$ will be active at time $t$, $o_t = \{o^i_t\}$ $a_t = \{a^i_t\}$. Consequently, it is imperative that we streamline the action-value function to $Q(\delta_{t-1}, o_t, a_t; \phi)$, and similarly, the action selection function must be refined to $\mu^{it}(\delta_{t-1}, o_t; \theta^{it})$.

### 3.7 Model Training

The Critic Network $Q(\delta_{t-1}, o_t, a_t; \phi)$ is meticulously trained employing the Bellman equation. We minimize the following loss function:

$$L(\phi)= \mathop{\mathrm{E}}_{\delta_{t-1},o_t}[(Q(\delta_{t-1},o_t,a_t;\phi)-y_t)^2] \quad , \tag{5}$$

$$\text{where } y_t=r_t+\gamma Q(\delta_t,o_{t+1},a_{t+1},\ \mu^{it+1}(\delta_t,o_{t+1});\phi) \quad . \tag{6}$$

The update of the network of private actors is based on

maximizing the whole expectation. Suppose at time $t$, $A^{it}$ is active, then the objective function is:

$$J(\theta^{it}) = \underset{\delta_{t-1},o_t}{\mathrm{E}}[(Q(\delta_{t-1},o_t,a;\phi)\,|\,a=\mu^{it}(\delta_{t-1},o_t;\theta^{it})]. \quad (7)$$

According to the chain rule, the parameter gradient of each actor can be expressed as follows:

$$\begin{aligned}&\nabla_{\theta^{it}}J(\theta^{it})\\ &\approx \underset{\delta_{t-1},o_t}{\mathrm{E}}[\nabla_{\theta^{it}}Q(\delta_{t-1},o_t,a;\phi)\,|\,a=\mu^{it}(\delta_{t-1},o_t;\theta^{it})]\\ &= \underset{\delta_{t-1},o_t}{\mathrm{E}}[\nabla_a Q(\delta_{t-1},o_t,a;\phi)\,|\,a=\mu^{it}(\delta_{t-1},o_t;\theta^{it})\nabla_{\theta^{it}}\mu^{it}(\delta_{t-1},o_t;\theta^{it})]\end{aligned}. \quad (8)$$

The goal of communication module training is to minimize the following functions:

$$\begin{aligned}&L(\psi)\\ &= \underset{\delta_{t-1},o_t}{\mathrm{E}}[(Q(\delta_{t-1},o_t,a;\phi)-y_t)^2\,|\,\delta_{t-1}=BiLSTM(\delta_{t-2},[o_{t-1};a_{t-1};\psi)]\\ &- \underset{\delta_{t-1},o_t}{\mathrm{E}}[Q(\delta_{t-1},o_t,a;\phi)\,|\,\delta_{t-1}=BiLSTM(\delta_{t-2},[o_{t-1};a_{t-1};\psi)]\end{aligned}. \quad (9)$$

We make a replay buffer [23] to store each agent's interactions with the environment and update them in a minibatch. In each training, we select a set of mini-batches and train them in parallel, updating both the actor-network and the critic-network.

### 3.8 Multi-layer Flow Control Based on Cross-entropy

The optimization of competition among multiple scenarios occurs gradually. Can the scheduling process be personalized? We propose an intervention of various layers, referring to the division of sub-scenario types.

Earlier algorithms may not have adequately considered these factors. A typical method involves simple weighting, which often results in efficiency loss, leading to mostly meaningless outcomes. However, upon closer examination of this issue, it's clear that some loss is inevitable. In a purely competitive environment, optimization will gradually occur under the current supply and demand dynamics, reaching a local optimum. A significant disruption could break this local optimum, causing an immediate loss in efficiency. However, this disruption could potentially lead to a better position than the previous stable point.

This prompts us to consider two questions:

(1) How can we minimize the immediate loss as much as possible?

(2) How can we reach the new local optimum as quickly as possible?

The corresponding solution is straightforward:

(1) We will implement personalized interventions to minimize unnecessary losses. For instance, data flow timeliness can be stratified for intervention, exempting scenarios that are less sensitive to data flow.

(2) Upon further abstraction, this solution naturally defines a reinforcement learning problem: personalized intervention equates to taking different actions for different states, while broader and intelligent exploration corresponds to the learning process of reinforcement learning.

We interpret the resource scheduling behavior of multi-scenario joint optimization as a Markov Decision Process (MDP) where the scenario agent interacts with the management center. The information the management center observes is considered the state; the traffic control policy is regarded as the action, and the environmental feedback is seen as the reward. The optimization problem of traffic control policy can also be solved by Reinforcement Learning (RL). To consider the influence of traffic structure change, we combine the change in hierarchical traffic proportion and environmental feedback as the reward.

The scenario's context typically includes the long-term behavior preference of the agent, the real-time state, and the behavior sequence representation, which we denote as $s \in R^d$.

Suppose there are m signals that require intervention, and each layer is abstracted into a feature. If the agent belongs to the layer, the score is 1; otherwise, it's 0. The weight corresponding to each layer forms an action. Instead of directly outputting the absolute value of the action, a common technique involves using sigmoid (or tanh, which are interchangeable) in the final output layer of the neural network. This bounds the output of the actor-network to [0, 1] in each dimension, i.e. $o \in [0,1]^m$, That is, a transformation is then applied as follows:

$$a^k = L^k + (U^k - L^k)o^k, \forall\, k \in \{1,2,...,m\}, \quad (10)$$

Here $U^k$ and $L^k$ are the upper bound and low bound of the weight of the $k$-th dimensional hierarchical feature, which generally comes from domain knowledge, but is usually limited by experience.

The first element in reward design is the stratified proportion $p_i(\pi)$, which represents the ratio of stratified traffic to total traffic. However, efficiency must be considered during traffic regulation, so the behavior feedback of multi-scenario mixed services is a crucial reward factor. Different actions such as push, install, uninstall, update, migration and scale are factored into the feedback. Each action holds varying influence factors. All these actions in each scenario are collectively referred to as rewards. The stratified ratio also requires a control group. For example, since the performance difference is inherent to the scenario and unrelated to traffic control actions, its original value is subtracted from the stratified ratio in the baseline bucket when calculating the actual stratified ratio $p_i(\pi_{basic})$, i.e

$$r(s,a) = \mathrm{Re}\,ward(n_{actions}) + \sum_i^m \lambda_i(p_i(\pi) - p_i(\pi_{basic}), \quad (11)$$

The above model is based on the reward of a particular scenario. However, due to the unpredictability of scenario

behavior, instant rewards can exhibit high variance, which significantly impacts the learning process. Thus, searching for the entire real number space may lead to non-convergence. To address this, we establish upper and lower bounds, allowing the Reinforcement Learning (RL) algorithm to search within a local space. This approach simplifies the learning process but introduces two new challenges:

1. How can we ensure the validity of the upper and lower bounds?

2. How can we avoid settling on a locally optimal interval?

To solve these problems, we employ the Cross-Entropy Method (CEM) to dynamically update the action's upper and lower bounds in real-time. Specifically, we disregard the state and focus on a globally optimal action, which we assume follows a Gaussian distribution $a_k$, such as

$$a_{k^*} \in N(\mu_k, \sigma_k^2) \ , \tag{12}$$

At the beginning of each iteration, we sample s samples from this distribution, i.e. $\Omega_1$, $\Omega_2$, ..., $\Omega_s$, and then deliver sufficiently on these actions to obtain a sufficiently confident reward value for the corresponding action.

$$\begin{aligned} R(\Omega_1) &= \frac{1}{N_1}\sum_{i}^{N_1} r_i(\Omega_1) \\ R(\Omega_2) &= \frac{1}{N_1}\sum_{i}^{N_2} r_i(\Omega_2) \\ &\cdots \\ R(\Omega_s) &= \frac{1}{N_1}\sum_{i}^{Ns} r_i(\Omega_s) \end{aligned} \tag{13}$$

We then sort the pairs $R(\Omega_1)$, $R(\Omega_2)$, ..., $R(\Omega_s)$ and pick out the subset D of top p that maximizes the probability that the Gaussian distribution produces these samples, i.e

$$\max_{\mu_k^*, \sigma_k^{2^*}} f(\mu_k^*, \sigma_k^{2^*}) = \sum_{i \in D} \log N(\Omega_i \mid \mu_k^*, \sigma_k^{2^*}) \tag{14}$$

The above equation has an optimal solution, that is, $\mu_k^*$ is the mean of all samples in D, $\sigma_k^{2^*}$ is the variance of all samples, but if we solve it directly, the model will iterate too fast. On the one hand, it will completely forget the information of previous iterations. On the other hand, this will be directly output to the RL learning actor above. Therefore, the bound cannot be changed by more than one block, otherwise, RL may not be able to keep up with the change in time. Therefore, we adopt a slow update method, i.e

$$\mu_k \leftarrow \mu_k + \alpha \frac{\partial f}{\partial \mu_k} \tag{15}$$

$$\sigma_k \leftarrow \sigma_k + \alpha \frac{\partial f}{\partial \sigma_k} \tag{16}$$

After the update, we use a specific method to define the upper and lower bounds of the k-th dimensional action. This ensures the RL-regulated action is within a globally optimal space

$$L^k \leftarrow \mu_k - 2\sigma_k \tag{17}$$

$$U^k \leftarrow \mu_k + 2\sigma_k \tag{18}$$

The overall flow of our implementation is as follows:

The initial upper bound and low bound are selected to start RL learning. At the same time, the upper bound and low bound are dynamically adjusted.

**Algorithm** 1. Unified resource scheduling under multiple scenarios

**Input: The environment**
**Body**
Initialize the actor networks $\theta = \{\theta^1, ..., \theta^N\}$ and critic network $\phi$
Initialize the replay buffer $R$
  **for** $k$ steps **do**
    **for** i = 0,1,2,... ,M **do**
      **while** t<T and $o_t \neq$ terminal **do**
        Generate Action $\mu^{i_t}(h_{t-1}, o_t; \theta^{i_t})$ according to agent $i_t$
        Receive new observation $o_{t+1}$ and get reward $r_t$ with Equation(11)
        get $h_t$ with Equation (2)
        t=t+1
        **end**
        Store episode $\{h_0, o_1, a_1, r_1, h_1, o_2, a_2, r_2, ...\}$ in $R$
      **end for**
  **end for**
  sample minibatch $B$ from $R$
  **foreach** episode in $B$ **do**
    **for** i = T downto 1 **do**
      update the critic by Equation (5)
      update the $i_t$-th actor by Equation(15)(16)
      update the upper bound and low bound of action by Equation(17)(18)
      update the communication component by Equation(3)
    **end**
  **end for**
**Output: Private Actor** $\theta = \{\theta^1, ..., \theta^N\}$

# 4 Experiments

In this section, we conduct experiments to compare our approach with other state-of-the-art approaches. We aim to answer the following questions:

(1) How does our approach compare with published state-of-the-art approaches?

(2) How does the number of devices affect performance?

(3) What is the impact of communication distance on performance?

(4) How does task size affect performance?

(5) What is the effect of task generation speed on performance?

(6) How do CPU cycles impact performance?

(7) How does our approach which includes hierarchical flow control compare with the one excluding hierarchical flow control in terms of performance?

## 4.1 Experimental Design

We use an edge-cloud network composed of cloud servers, edge nodes, and terminal devices. The cloud server is a physical machine equipped with a 16-core Intel Xeon E5-2620 v4 CPU, 64GB memory, 1TB hard disk, and Ubuntu 18.04 operating system. It serves as a cloud computing center, offering powerful computing and storage capabilities. Edge nodes are laptops equipped with a 4-core Intel Core i5-8250U CPU, 8GB memory, 256GB hard disk, and Ubuntu 18.04 operating system. They form the edge computing layer, providing close computing and storage services. The connection between the cloud server, edge nodes, and terminal devices is established through wired or wireless networks, with network bandwidth and latency dynamically changing according to the actual situation.

We use the mixed architecture built by Karmada, RunD, and Koordinator as the native platform of edge cloud. Various resource scheduling policies are implemented using its APIs and tools. The performance of these policies is evaluated by monitoring and recording various performance indicators.

The training process involves obtaining user behavior logs in real time to provide training samples for our approach. These samples are stored in a replay buffer. Then, the model is updated and applied online. This process is repeated, resulting in a dynamically updated model that captures changes in user behavior.

Each agent's localized observations are encapsulated within a 52-dimensional vectorial space, while their respective behaviors are characterized by vectors of 7 and 3 dimen-sions. For simplicity, a 10-dimensional vector (with zeroes filling in the gaps) is used as the output of BiLSTM and the evaluation network.

The communication module takes a 62-dimensional vector (52 + 7 + 3 = 62) as input and outputs a 10-dimensional vector. The actor network also uses a 62-dimensional input, with a hidden layer size of 32/32/7. The initial two layers of the network architecture are equipped with Rectified Linear Unit (ReLU) activation functions to introduce non-linearity, facilitating the network's ability to learn complex patterns. In contrast, the final layer is endowed with a Softmax activation function, which is instrumental in transforming the output into a probability distribution, essential for making informed decisions in the context of multi-agent systems.

The Critic network is meticulously engineered with a dual-layered hidden architecture, each layer comprising 32 neurons, all of which are activated through the Rectified Linear Unit (ReLU) function, enabling the network to effectively capture and process the intricate dynamics of the multi-agent environment. The gain attenuation coefficient in the Bellman formula is set to $\gamma = 0.95$. We use RMSProp to learn the network parameters, with learning rates of 10-2-3 and 10, and a hidden layer of 128, corresponding to the ac-tor-network and the critic-network, respectively.

## 4.2 Comparison

To comprehensively assess the efficacy and performance of our algorithm, we conducted a rigorous comparison against a spectrum of benchmarks, including Ex-pert policies, SM-NE, DQN, and Random scheduling. We consider factors such as cost and time.

We compare our approach with the following approaches:

1. Expert policies: Experts use the ACKTR algorithm [29], which is based on centralized management, to address the problem. Each expert can observe the entire instantaneous system state.

2. SM-NE [30]: This decentralized task scheduling algorithm requires a centralized server for some information. SM-NE has been applied to pervasive edge computing networks through game theory and Nash equilibrium.

3. DQN-based solution [31]: This approach considers the nuanced dynamics of peer-to-peer offset loading. Furthermore, it is underpinned by a centralized scheduling paradigm, which harnesses the predictive capabilities of Deep Q-Networks (DQN) to optimize resource allocation and task distribution across the network.

4. Random scheduling: Devices are randomly selected for offloading, regardless of their performance. Once a suitable device is found, it can be used without further search.

In this section, we undertake an exhaustive assessment of the holistic efficacy across five distinct policy frameworks by Expert policies, SM-NE approach, DQN-based solutions, and Random scheduling. In Figure 4, we present a graphical representation of the mean total cost metrics over the final 500 epochs, utilizing this mean to mitigate the impact of algorithmic-induced variability. At task densities of 10, 12, and 14, the aggregate costs for both Expert policies and our proposed methodology are strikingly comparable and significantly lower than those associated with DQN-based solutions. A discernible divergence emerges at a density of 16, which intensifies at densities 18, 20, and 22. Nonetheless, our approach exhibits a commendable level of stability, with only a marginal escalation in costs attributable to the escalation

in the total volume of tasks. Our research findings indicate that an edge offloading system reliant on a solitary high-capacity edge server is capable of delivering consistent service within a limited range of task densities. As the task load increases, the efficiency of such a singular offloading framework diminishes markedly. Although our approach performs slightly worse than Expert policies at low densities, this difference is negligible due to task randomness. Our approach maintains stability as task density increases. Therefore, our approach can effectively perform offloading at a low cost in various scenarios.

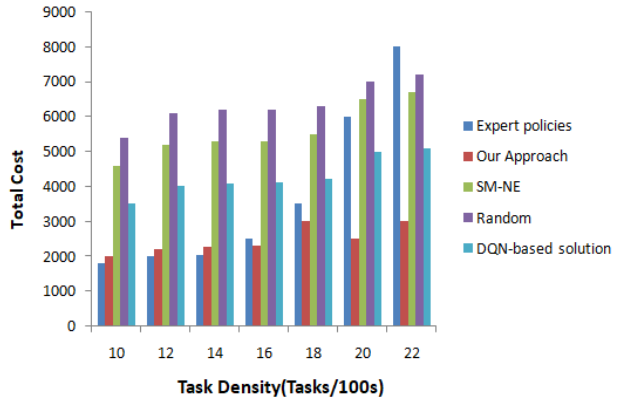



**Figure 4.** Total cost comparison on different task densities

### 4.3 Performance Evaluation

#### 4.3.1 Impacts of Total Device Number

Figure 5 provides a comprehensive visualization of the performance metrics across a spectrum of device counts. Our analysis reveals that the mean task completion times achieved by our approach are notably superior to those of the SM-NE, DQN-based solutions, and random scheduling strategies, yet marginally exceed the times recorded by the expert policy. To illustrate, in an environment with 30 devices, the average task completion times are as follows: the expert policy is 0.19 seconds, MILP is 0.18 seconds, SM-NE is 0.15 seconds, our DQN is 0.11 seconds, and the random scheduling is 0.21 seconds, respectively. This comparative analysis underscores the efficiency and effectiveness of our approach in optimizing task completion times across varying device configurations.

Our approach's effectiveness stems from its use of GAIL [32] to train its policy, which allows it to approximate the performance of an expert with an acceptable gap. The expert policy, grounded in the overall system state, solves the optimization problem by attaining a Nash equilibrium among different devices, much like a centralized control approach. In comparison, SM-NE calculates the Nash equilibrium based on factors like average task arrival intensity and transmission rate. However, in edge computing networks, infor-mation updates may lag, limiting SM-NE's performance relative to our approach. The random scheduling algorithm, which randomly selects devices for offloading, doesn't have predictable performance. Although centralized, the DQN-based solution primarily seeks to minimize system cost, while ensuring tasks can be executed before their deadlines. Thus, task completion delay may not be minimized significantly.

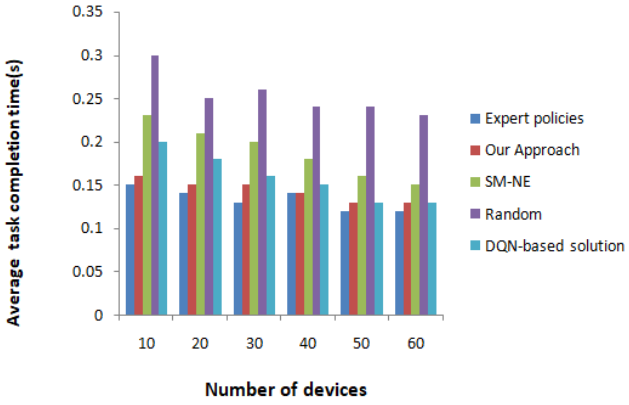



**Figure 5.** Average task completion time with different number of devices

Figure 6 delineates the offloading ratio across varying device numbers, which represents the proportion of tasks offloaded relative to the total tasks generated within the network. It is observed that the DQN-based solution exhibits the highest offloading rate, whereas the expert policy and our proposed methodology surpass the offloading rates of SM-NE and random scheduling algorithms. For instance, in a scenario with 20 devices, the offloading ratios are as follows: the expert policy at 0.15, MILP at 0.14, SM-NE at 0.1, the DQN scheme at 0.08, and the random scheduling scheme at 0.16. This data underscores the efficacy of our approach in effectively distributing tasks across devices, ensuring optimal resource utilization and performance.

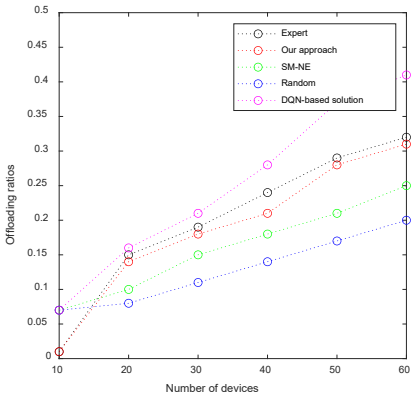



**Figure 6**. Offloading ratio with different numbers of devices

In scenarios where the device number is less than ten, the offloading rates achieved by the expert policy, MILP, and SM-NE are surpassed by those of the DQN-based solution and the random scheduling algorithm. This phenomenon arises from the fact that the Nash equilibrium conditions limit the potential for task offloading to other devices, thereby constraining the offloading rates. Conversely, the DQN-based solution is capable of ef-

fectively offloading tasks to edge servers or other devices, which leads to a substantial reduction in system costs, even when the number of devices is limited. As the device count escalates, the offloading rates for all five algorithms experience an uptick, attributable to the increased opportunities for task offloading that become available with a denser device network.

Figure 7 graphically represents the average performance enhancement across the five algorithms, measured by the reduction in task completion time achieved by offloading tasks between devices rather than executing them locally. Our approach exhibits the most significant alignment with the expert policy's performance and outperforms the remaining algorithms. This is attributed to our method's emulation of expert behavior, which is inherently optimized for task execution. The DQN-based solution, on the other hand, prioritizes meeting task deadlines over performance optimization.

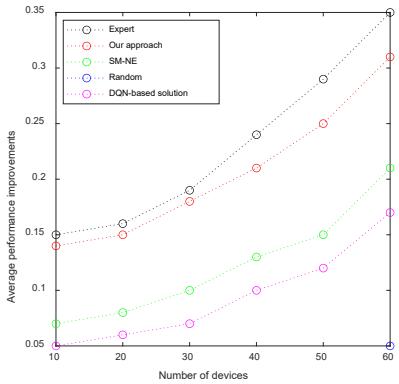



**Figure 7.** Average performance improvement with different number of devices

With an increasing number of devices, the average performance improvement for all algorithms is observed to increase, as the collective computational resources lead to a decrease in the average task completion time, thereby amplifying the overall efficiency of the system.

### 4.3.2 Generalization Ability of Our Approach

Figure 8 provides a detailed visualization of the average task completion time's performance trend concerning the communication distance. It is observed that as the communication distance extends, the performance metrics of all five algorithms tend to improve. For instance, at a communication distance of 60 meters, the average task completion times are recorded as 0.24 seconds, 0.21 seconds, 0.16 seconds, 0.12 seconds, and 0.26 seconds for the respective algorithms. Upon increasing the distance to 80 meters, the completion times are further optimized to 0.32 seconds, 0.29 seconds, 0.20 seconds, 0.14 seconds, and 0.35 seconds. This trend suggests that greater communication distances may paradoxically lead to enhanced task completion efficiency, potentially due to the system's ability to distribute tasks more effectively across a wider network of devices.

This enhancement is attributed to the fact that an extended communication range facilitates greater inter-device connectivity, thereby providing a broader array of options for task offloading, as more devices become accessible for this purpose. Furthermore, our policy's performance aligns most closely with that of the expert policy, reflecting a sophisticated understanding of task distribution and offloading strategies. As the communication distance expands, the performance disparities among the five algorithms diminish, owing to the augmented pool of devices that can participate in task offloading, which levels the playing field in terms of efficiency and effectiveness across the different algorithmic approaches.

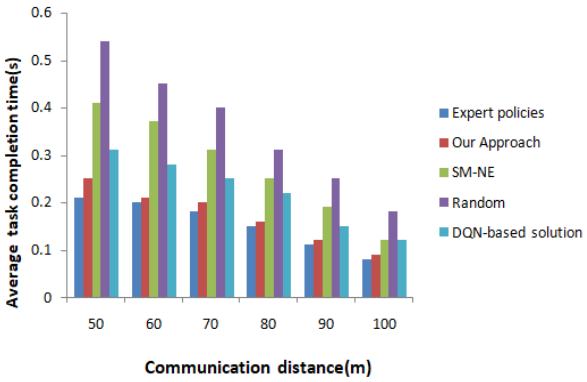



**Figure 8.** Average task completion time with different communication distance

Figure 9 delineates the offloading rate performance across varying communication distances. Our methodology achieves a higher offloading rate compared to the SM-NE and random scheduling algorithms, yet it falls short of the DQN-based solutions. This discrepancy arises from our approach's capability to effectively distribute tasks across a multitude of devices, thereby emulating the expert policy with high fidelity. The nuanced task scheduling mechanism inherent in our approach enables a more efficient offloading strategy, which, while not surpassing the DQN-based solutions, still outperforms the SM-NE and random scheduling algorithms in terms of offloading efficiency.

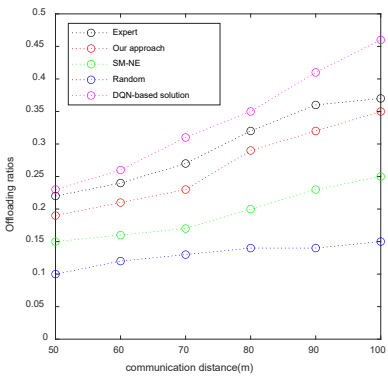



**Figure 9.** Offloading ratio with different communication distance

The offloading ratio exhibits a positive correlation with the expansion of communication distance. At a

communication distance of 50 meters, the offloading ratios for the expert policy, our approach, the SM-NE, the DQN-based solution, and the random scheduling algorithm are recorded as 0.22, 0.19, 0.15, 0.10, and 0.23 seconds, respectively. Upon increasing the distance to 70 meters, the offloading times for the aforementioned algorithms are adjusted to 0.27, 0.23, 0.17, 0.13, and 0.31 seconds. This trend indicates that with greater distances, the potential for task offloading is enhanced, allowing for more efficient resource utilization and improved system performance across the board.

This trend can be attributed to the limited inter-device communication opportunities at shorter communication distances, which consequently restricts the scheduling and processing of tasks across various devices.

### 4.3.3 Convergence Time

We evaluated the convergence time of schemes based on our approach, SM-NE, and DQN with varying numbers of devices, as depicted in Figure 10. When the network has a small number of devices, our approach and SM-NE show little difference in convergence time. Nonetheless, with an escalation in the number of devices, the merits of our approach become increasingly evident. It consistently demonstrates the shortest solution convergence time.

In our methodology, while individual devices may not have access to the comprehensive real-time network state, the learning agent is capable of emulating a centralized policy through training within a decentralized environment. The expert data thus gathered can be leveraged within a Generative Adversarial Network (GAN) to forecast rewards, effectively reducing the variability in the distribution of observation-action pairs.

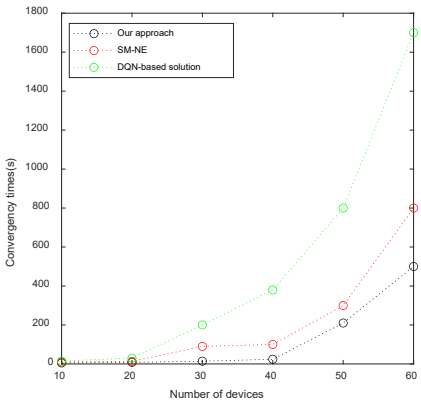



**Figure 10.** Convergence time with different number of devices

On the other hand, the SM-NE algorithm struggles to obtain immediate system state information in parallel and distributed systems. This includes device connections and updated average task arrival rates. Consequently, this could lead to decision-making based on outdated data. Furthermore, with an increasing number of devices, the computational time to determine the Nash Equilibrium (NE) also escalates, exacerbated by the latency in information synchronization across the network.

DQN-based solutions, which operate in a centralized manner, train a learning model that relies on the comprehensive state of the system to concurrently determine actions for all tasks at each time interval. However, the vastness of the state and action space necessitates extended training periods for the model, a duration that is further prolonged with an increase in the number of devices.

### 4.3.4 Impacts of Task Generation Speed

Figure 11 illustrates the average task completion time for the five algorithms in response to variations in task generation velocity. Task generation velocity is defined as the maximum quantity of tasks that a device can generate within a specific timeframe. As the task generation velocity intensifies, the average completion time for these algorithms correspondingly ascends. This escalation is attributed to the increased volume of tasks in each device's local queue at lower generation velocities. When the task generation rate surpasses 4 tasks per time slot, the completion time experiences a steep ascent due to the cumulative impact of computational and transmission delays.

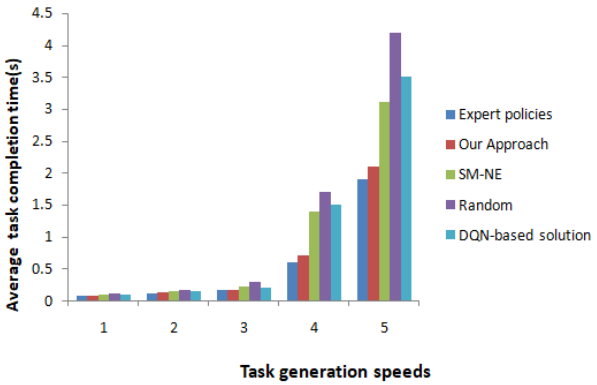



**Figure 11.** Average task completion time with different task generation speeds

### 4.3.5 Impact of CPU Cycles

This study delves into the performance trajectory of the average task completion time for five distinct algorithms, contingent upon varying CPU cycle counts. Here, CPU cycles denote the peak computational capacity that a device can deliver. With an augmentation in CPU cycles, the performance ratio diminishes.

As depicted in Figure 12, at a CPU cycle rate of 9 gigacycles, the average task completion times for the expert policy, our approach, the SM-NE algorithm, the DQN solution, and random scheduling are recorded at 0.14 seconds, 0.15 seconds, 0.20 seconds, 0.32 seconds, and 0.17 seconds, respectively. Upon elevating the CPU cycle to 12 gigacycles, these times are reduced to 0.09 seconds, 0.10 seconds, 0.12 seconds, 0.15 seconds, and 0.13 seconds, respectively.

This trend is attributable to the fact that an increase in CPU cycles equips each device with enhanced computational capabilities. As a result, the processing latency for individual tasks is mitigated, culminating in a reduction in the average task completion time.

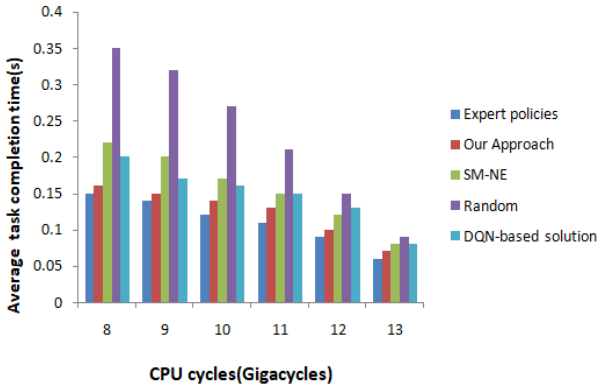



**Figure 12.** Average task completion time with different CPU cycles

However, there is a noticeable performance gap compared to the expert policy due to the imitation capability of our approach.

**4.3.6 Impact of Hierarchical Flow Control**

We examine the effects of managing the proportion of multi-layered traffic in a real environment. Figure 13 displays the increase in the offloading rate of our approach including layered traffic control compared to our approach without layered traffic control. Hierarchical traffic control policy consistently outperforms our standard policy.

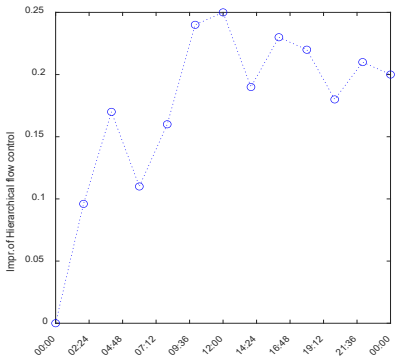



**Figure 13.** Average performance improvements with our approach including hierarchical flow control compared to our approach without hierarchical flow control

# 5 Related Work and Discussion

Resource management plays a pivotal role in both cloud and edge computing environments. The strategic allocation and deployment of storage, network bandwidth, and computing resources are instrumental in achieving substantial reductions in energy consumption. Offloading, as a technique, is designed to bolster the Quality of Service (QoS) by judiciously distributing the computational burden, thereby optimizing resource utilization and enhancing overall system efficiency.

Considering the inherent complexity and dynamism of real-world scenarios, the offloading algorithms and computing architectures must exhibit exceptional scalability to adapt effectively. Nonetheless, the majority of existing models grapple with the challenge of addressing issues across diverse scenarios due to the inherent limitations of both the models and the offloading strategies they employ.

In single-user edge computing systems, offloading is facilitated between an individual user and a single edge server or a cluster of edge servers. The distinction between deterministic and stochastic tasks necessitates the development of distinct task models and corresponding solutions. For example, Wang et al. [33] concentrate on reducing energy expenditure and execution latency by meticulously tuning the transmission power and rate. They transform the inherently non-convex optimization problem into a convex form, leveraging variable substitution methodologies to achieve a solution.

Mahmoodi et al. [34] address the challenges of energy and cost optimization in intricate scenarios characterized by arbitrary task dependencies, as opposed to linear sequences. They delineate task relation graphs that incorporate both parallel and sequential elements and devise a comprehensive offloading strategy aimed at reducing energy con-sumption and latency.

The stochastic task model presents a heightened level of complexity. Dong et al. [35] have crafted an adaptive offloading algorithm that adeptly balances power consumption with application execution duration, while maintaining a low computational complexity.

In reference [36], several dynamic issues are considered jointly, including resource allocation and control of the network interface. A DREAM algorithm is proposed to solve these problems and is verified in terms of energy saving and latency reduction on the Android platform.

In the realm of multi-user systems, cooperative resource management encompasses the strategic allocation and utilization of both radio and computing resources to optimize system performance. Different offloading schemes are required for centralized and distributed systems. In a centralized system, the MEC server processes computing requests and distributes resources among mobile devices.

The diversity of devices inherent in edge computing environments introduces complexities that challenge the efficiency of offloading processes. Conventional approaches frequently struggle to deliver optimal solutions when confronted with dynamic offloading challenges that entail multiple constraints. Nonetheless, the integration of reinforcement learning has marked a significant advancement in tackling these complexities.

Research such as [37] has constructed a simple Q-Learning-based offloading model, where an edge server receives several tasks from a single device. Despite the model's reliance on the Q-function to determine the most advantageous offloading decisions, contingent upon device capabilities and energy availability, it is not endowed with sufficient robustness. Consequently, the Deep Q-Network (DQN) has emerged as a favored instrument in the field of edge computing offloading research, offering a more resilient approach to decision-making.

Duc Van Le and Chen-Khong Tham have devised an algorithm that integrates Deep Q-Network (DQN) to concurrently account for task loss probability and user mobility when making optimal offloading decisions [38]. Parallel studies, such as [39] and [40], have also embraced DQN-based algorithms, with an emphasis on task allocation that takes into account radio resources, thereby crafting a more holistic model. Notably, [39] introduces an innovative action selection technique aimed at enhancing efficiency by partitioning the optimization problem into offloading and allocation components, effectively mitigating computational complexity.

While these studies demonstrate the potential of deep reinforcement learning for offloading optimization, their models do not reveal their adaptability to different scenarios, which is our main interest.

In response to this challenge, we have architected a two-tier framework that facilitates load offloading from the terminal devices to the edge servers and among the edge servers themselves. This system enables a rational distribution of regional resources at a more elevated level, thereby enhancing the efficiency of dynamic task offloading and averting the wastage of resources.

Subsequently, we implemented a deep reinforcement learning algorithm that is particularly tailored to the nuances of edge computing environments. In contrast to Q-learning, which is limited to addressing a finite number of states, our algorithm is adept at optimizing sequential decision-making processes within a significantly broader state space.

In contrast to policies derived from Deep Q-Networks (DQN), our approach demands a more modest computational footprint and achieves convergence at a more expedited pace. DQN-based policies rely on powerful GPUs, while our policy can be implemented on devices with only multi-core CPUs, making it more suitable for edge devices with limited computing resources.

In the realm of Multi-Agent Reinforcement Learning (MARL) [41], a collective of autonomous and interactive agents inhabit a shared environment. Each agent, leveraging its unique information, formulates actions through its policy function. The interactions among these agents span a spectrum: from complete cooperation, where all agents pursue a unified objective, to full competition, where goals are opposed. Hybrid models represent a nuanced blend that straddles these polarities.

# 6 Conclusion and Future Work

In conclusion, the approach we have developed presents a highly scalable and robust offloading solution that is well-suited for dynamic edge computing scenarios. This is accomplished by harnessing the transformative capabilities of deep reinforcement learning within the task-scheduling paradigm. Our two-tier architecture, combined with the use of Generative Adversarial Imitation Learning (GAIL), allows for efficient and strategic allocation of resources. This is achieved while maintaining a high level of adaptability across a variety of different scenarios, making our policy not only effective but also more feasible for implementation in edge devices that have limited resources. This is particularly important when compared to policies based on Deep Q-Network (DQN), which can be more demanding in terms of resources. Looking forward, we aim to enhance our model's gen-eralization capability as part of our future work. This is an important aspect that we believe will further improve the effectiveness of our approach. Moreover, we also plan to investigate the potential benefits of integrating federation learning into our framework. We hypothesize that this could further improve resource utilization and overall system performance, enhancing the robustness and scalability of our solution.

# Acknowledgment

This work was supported by the Natural Science Foundation Innovation and Development Joint Fund of Hubei Province in China (2025AFD031), Hubei Province Science and Technology Plan Project (2025DJC021), and Hubei Superior and Distinctive Discipline Group of "New Energy Vehicle and Smart Transportation".

# References

[1] A. Damiani, G. Fiscaletti, M. Bacis, R. Brondolin, M. D. Santambrogio, BlastFunction: A Full-stack Framework Bringing FPGA Hardware Acceleration to Cloud-native Applications, *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, Vol. 15, No. 2, pp. 1-27, June, 2022.
https://doi.org/10.1145/3472958

[2] S. Pardeshi, C. Khairnar, K. Alfatmi, Analysis of data handling challenges in edge computing, *International Journal of Performability Engineering*, Vol. 18, No. 3, pp. 176-187, March, 2022.
https://doi.org/10.23940/ijpe.22.03.p4.176187

[3] A. Boudi, M. Bagaa, P. Poyhonen, T. Taleb, H. Flinck, AI-Based Resource Management in Beyond 5G Cloud Native Environment, *IEEE Network*, Vol. 35, No. 2, pp. 128-135, March/April, 2021.
https://doi.org/10.1109/MNET.011.2000392

[4] G. A. Jimenez-Maggiora, S. Bruschi, H. Qiu, J. So, P. S. Aisen, Corrigendum to: ATRI EDC: a novel cloud-native remote data capture system for large multicenter Alzheimer's disease and Alzheimer's disease-related dementias clinical trials, *JAMIA Open*, Vol. 5, No. 1, pp. 1-14, April, 2022.
https://doi.org/10.1093/jamiaopen/ooac008

[5] M. Al-Quraan, L. Mohjazi, L. Bariah, A. Centeno, A. Zoha, K. Arshad, K. Assaleh, S. Muhaidat, M. Debbah, M. Imran, Edge-Native Intelligence for 6G Communications Driven by Federated Learning: A Survey of Trends and Challenges, *IEEE Transactions on Emerging Topics in Computational Intelligence*, Vol. 7, No. 3, pp. 957-979, June, 2023.
https://doi.org/10.1109/TETCI.2023.3251404

[6] H. Wang, G. Tang, K. Wu, J. Wang, PLVER: Joint Stable Allocation and Content Replication for Edge-assisted Live Video Delivery, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 33, No. 1, pp. 218-230, January, 2022.
https://doi.org/10.1109/TPDS.2021.3090784

[7] S. Kassir, G. Veciana, N. Wang, X. Wang, P. Palacharla, Joint Update Rate Adaptation in Multiplayer Cloud-Edge Gaming Services: Spatial Geometry and Performance Tradeoffs, *Twenty-second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, Shanghai, China, 2021, pp. 191-200.
https://doi.org/10.1145/3466772.3467048

[8] N. Sreekumar, A. Chandra, J. Weissman, Position Paper: Towards a Robust Edge-Native Storage System, *IEEE/ACM Symposium on Edge Computing (SEC)*, San Jose, CA, USA, 2020, pp. 285-292.
https://doi.org/10.1109/SEC50012.2020.00040

[9] G. Nain, K. K. Pattanaik, G. K. Sharma, Towards edge computing in intelligent manufacturing: Past, present and future, *Journal of Manufacturing Systems*, Vol. 62, pp. 588-611, January, 2022.
https://doi.org/10.1016/j.jmsy.2022.01.010

[10] Q. Huang, Y. Huang, The Significance of Urban Cockpit for Urban Brain Construction, *11th International Conference on E-business, Management and Economics*, Beijing, China, 2020, pp. 70-73.
https://doi.org/10.1145/3414752.3414800

[11] L. Loven, T. Leppnen, E. Peltonen, J. Partala, E. Harjula, P. Porambage, M. Ylianttila, J. Riekki, EdgeAI: A Vision for Distributed, Edge-native Artificial Intelligence in Future 6G Networks, 6*G Wireless Summit*, Levi, Finland, 2019, pp. 1-2.

[12] J. Okwuibe, J. Haavisto, E. Harjula, I. Ahmad, M. Ylianttila, SDN Enhanced Resource Orchestration of Containerized Edge Applications for Industrial IoT, *IEEE Access*, Vol. 8, pp. 229117-229131, December, 2020.
https://doi.org/10.1109/ACCESS.2020.3045563

[13] X. He, H. Lu, Y. Mao, K. Wang, QoE-driven Task Offloading with Deep Reinforcement Learning in Edge intelligent IoV, *IEEE Global Communications Conference*, Taipei, Taiwan, 2020, pp. 1-6.
https://doi.org/10.1109/GLOBECOM42002.2020.9348050

[14] Y. Zhai, W. Sun, J. Wu, L. Zhu, J. Shen, X. Du, M. Guizani, An Energy Aware Offloading Scheme for Interdependent Applications in Software-Defined IoV With Fog Computing Architecture, *IEEE Transactions on Intelligent Transportation Systems*, Vol. 22, No. 6, pp. 3813-3823, June, 2021.
https://doi.org/10.1109/TITS.2020.3044177

[15] M. Liwang, R. Chen, X. Wang, Resource Trading in Edge Computing-enabled IoV: An Efficient Futures-based Approach, *IEEE Transactions on Services Computing*, Vol. 15, No. 5, pp. 2994-3007, September-October, 2022.
https://doi.org/10.1109/TSC.2021.3070746

[16] M. C. Ogbuachi, A. Reale, P. Suskovics, B. Kovacs, Context-Aware Kubernetes Scheduler for Edge-native Applications on 5G, *Journal of Communications Software and Systems*, Vol. 16, No. 1, pp. 85-94, March, 2020.
https://doi.org/10.24138/jcomss.v16i1.1027

[17] S. H. VanderLeest, ARINC 653 hypervisor, *29th Digital Avionics Systems Conference*, Salt Lake City, UT, USA, 2010, pp. 5.E.2-1-5.E.2-20.
https://doi.org/10.1109/DASC.2010.5655298

[18] Y. Zhou, B. Li, T. R. Lin, Maintenance optimisation of multicomponent systems using hierarchical coordinated reinforce-ment learning, *Reliability Engineering and System Safety*, Vol. 217, Article No. 108078, January, 2022.
https://doi.org/10.1016/j.ress.2021.108078

[19] C. Li, J. Liu, W. Li, Y. Luo, Adaptive priority-based data placement and multi-task scheduling in geo-distributed cloud systems, *Knowledge-Based Systems*, Vol. 224, Article No. 107050, July, 2021.
https://doi.org/10.1016/j.knosys.2021.107050

[20] C. You, C. Ren, L. Li, Online Multi-resource Social Welfare Maximization for Non-Preemptive Jobs, *IEEE Access*, Vol. 8, pp. 97920-97934, May, 2020.
https://doi.org/10.1109/ACCESS.2020.2996630

[21] Y. Jeon, H. Baek, S. Pack, Mobility-Aware Optimal Task Offloading in Distributed Edge Computing, *International Conference on Information Networking (ICOIN)*, Jeju Island, Korea, 2021, pp. 65-68.
https://doi.org/10.1109/ICOIN50884.2021.9334008

[22] M. S. Mastoori, G. Rahmanian, The improved greedy gang scheduling by minimizing context switch condition, *26th International Computer Conference, Computer Society of Iran (CSICC)*, Tehran, Iran, 2021, pp. 1-5.
https://doi.org/10.1109/CSICC52343.2021.9420557

[23] Y. Mao, Y. Fu, W. Zheng, L. Cheng, Q. Liu, D. Tao, Speculative Container Scheduling for Deep Learning Applications in a Kuber-netes Cluster, *IEEE Systems Journal*, Vol. 16, No. 3, pp. 3770-3781, September, 2022.
https://doi.org/10.1109/JSYST.2021.3129974

[24] L. Xu, K. Wang, Z. Ouyang, X. Qi, An improved binary PSO-based task scheduling algorithm in green cloud computing, *9th International Conference on Communications and Networking*, Maoming, China, 2014, pp. 126-131.
https://doi.org/10.1109/CHINACOM.2014.7054272

[25] Y. Liu, H. Liang, Y. Xiao, H. Zhang, J. Zhang, L. Zhang, L. Wang, Logistics-involved service composition in a dynamic cloud manufacturing environment: A DDPG-based approach, *Robotics and Computer-Integrated Manufacturing*, Vol. 76, Article No. 102323, August, 2022.
https://doi.org/10.1016/j.rcim.2022.102323

[26] H. S. Chauhan, H. Babbar, S. Rani, D2PG: Deep Deterministic Policy Gradient-Based Vehicular Edge Caching Scheme for Digital Twin-Based Vehicular Networks, *International Journal of Performability Engineering*, Vol. 19, No. 5, pp. 350-358, May, 2023.
https://doi.org/10.23940/ijpe.23.05.p7.350358

[27] Q. Yang, R. Parasuraman, A Strategy-Oriented Bayesian Soft Actor-Critic Model, *arXiv*, arXiv:2303.04193, March, 2023.
https://arxiv.org/abs/2303.04193

[28] T. Chen, R. Xu, Y. He, X. Wang, Improving sentiment analysis via sentence type classification using BiLSTM-CRF and CNN, *Expert Systems with Applications*, Vol. 72, pp. 221-230, April, 2017.
https://doi.org/10.1016/j.eswa.2016.10.065

[29] Y. Wu, E. Mansimov, S. Liao, R. Grosse, J. Ba, Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation, *31st International Conference on Neural Information Processing Systems*, Long Beach, California, USA, 2017, pp. 5285-5294.

[30] S. Josilo, G. Dan, Decentralized algorithm for randomized task allocation in fog computing systems, *IEEE/ACM Transactions on Networking*, Vol. 27, No. 1, pp. 85-97, February, 2019.
https://doi.org/10.1109/TNET.2018.2880874

[31] Q. Luo, C. Li, T. H. Luan, W. Shi, Collaborative data scheduling for vehicular edge computing via deep reinforcement learning, *IEEE Internet of Things Journal*, Vol. 7, No. 10, pp. 9637-9650, October, 2020.
https://doi.org/10.1109/JIOT.2020.2983660

[32] X. Wang, J. Zhou, T. Song, D. Liu, Q. Wang, FlotGAIL: An operational adjustment framework for flotation circuits using generative adversarial imitation learning, *Minerals Engineering*, Vol. 183, Article No. 107598, June, 2022. https://doi.org/10.1016/j.mineng.2022.107598

[33] Y. Wang, M. Sheng, X. Wang, L. Wang, J. Li, Mobile-edge computing: Partial computation offloading using dynamic voltage scaling, *IEEE Transactions on Communications*, Vol. 64, No. 10, pp. 4268-4282, October, 2016. https://doi.org/10.1109/TCOMM.2016.2599530

[34] S. E. Mahmoodi, R. N. Uma, K. P. Subbalakshmi, Optimal joint scheduling and cloud offloading for mobile applications, *IEEE Transactions on Cloud Computing*, Vol. 7, No. 2, pp. 301-313, April-June, 2019. https://doi.org/10.1109/TCC.2016.2560808

[35] D. Huang, P. Wang, D. Niyato, A dynamic offloading algorithm for mobile computing, *IEEE Transactions on Wireless Communications*, Vol. 11, No. 6, pp. 1991-1995, June, 2012. https://doi.org/10.1109/TWC.2012.041912.110912

[36] J. Kwak, Y. Kim, J. Lee, S. Chong, DREAM: Dynamic resource and task allocation for energy minimization in mobile cloud systems, *IEEE Journal on Selected Areas in Communications*, Vol. 33, No. 12, pp. 2510-2523, December, 2015. https://doi.org/10.1109/JSAC.2015.2478718

[37] S. Nath, J. Wu, Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems, *Intelligent and Converged Networks*, Vol. 1, No. 2, pp. 181-198, September, 2020. https://doi.org/10.23919/ICN.2020.0014

[38] D. V. Le, C. K. Tham, A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds, *IEEE Conference on Computer Communications Workshops*, Honolulu, HI, USA, 2018, pp. 760-765. https://doi.org/10.1109/INFCOMW.2018.8406881

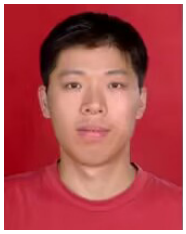
[39] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, M. Bennis, Performance optimization in mobile-edge computing via deep reinforcement learning, *arXiv*, arXiv:1804.00514, March, 2018. https://arxiv.org/abs/1804.00514

[40] L. Huang, S. Bi, Y. J. A. Zhang, Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks, *IEEE Transactions on Mobile Computing*, Vol. 19, No. 11, pp. 2581-2593, November, 2020. https://doi.org/10.1109/TMC.2019.2928811

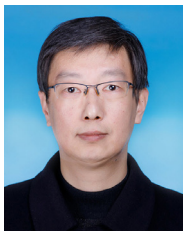
[41] L. Schfer, O. Slumbers, S. Mcaleer, Y. Du, S. V. Albrecht, D. Mguni, Ensemble Value Functions for Efficient Exploration in Multi-Agent Reinforcement Learning, *arXiv*, arXiv:2302.03439, February, 2023. https://arxiv.org/abs/2302.03439

## Biographies

**Wei Xiong** is an associate professor at Hubei University of Arts and Science, China. He received his Ph.D. degree in computer science from Wuhan University, China in 2015. His research interest includes edge computing and AI, as well as autonomous driving vehicles.



**Xinying Wang** is an associate professor at Hubei University of Arts and Science, China. He received his B.S. in education technology from Central China Normal University in 2000, M.S. degree in automatic control from Wuhan University of Technology in 2008. His research interest includes service computing and software engineering.



**Zhao Wu** is the professor at Hubei University of Arts and Science, China. He received his B.S. in computer application from China University of Geoscience in 1999, M.S. degrees in computer application from Wuhan University of Technology in 2003 and his Ph.D. degree in computer science from Wuhan University in 2007. He is a member of the China Computer Federation. His research interests include cloud computing, service computing and internet of things.

**Qiaozhi Hua** is an associate professor at Hubei University of Arts and Science, China. He received the PhD from Waseda University, Tokyo, Japan in 2019. His main fields of research interests include mobile communications, wireless sensor networks, intelligent transportation systems and optical communications.

**Franz Wotawa** is a PhD graduate from the Vienna University of Technology in Austria. He currently holds a professorship at Graz University of Technology. Professor Wotawa's research focuses on intelligent systems, software verification, system testing, and autonomous driving vehicles.