Journal of Computer Science and Artificial Intelligence
ISSN: 3078-8242 (Print), ISSN: 3078-2309 (Online) | Vol. 2, No. 1, 2025

Optimization of Multi-Task Efficient Offloading Strategy for Heterogeneous Edge Computing Based on Deep Reinforcement Learning

Sen Zhu *

School of Software, Henan Polytechnic University, Jiaozuo 454003, China
* **Corresponding author**: Sen Zhu

Abstract: Edge computing is a computing paradigm that provides fast and efficient computing services to mobile devices by bringing resources closer to the edge of the network. However, current edge servers lack the computational power to handle the large volume of tasks generated by connected devices. Moreover, some mobile devices may not fully utilize their computational resources. In order to maximize the use of resources, a more sustainable and energy efficient computing paradigm was developed. Minimize the use of Mobile Edge Computing (MEC) servers for applications that run in random environments with unpredictable workloads. And utilizing available resource-constrained edge devices to keep resource-rich servers idle to handle any unpredictable larger workloads. Our focus is on optimizing device computation offloading to minimize the maximum task processing delay in the system. This problem has been proven to be NP-hard. To address this problem, we propose an efficient offloading algorithm based on deep Q-network (DQNEO). The algorithm effectively utilizes computational resources in the system and makes intelligent scheduling decisions based on system state information using deep reinforcement learning. Experimental results show that the DQNEO algorithm can reduce the system processing delay by converging to the optimal solution.

Keywords: Edge Computing, Deep Reinforcement Learning, Resource Adaptation, Task Scheduling.

1. Introduction

With the development of digital cities and network technologies, central traffic congestion and explosive growth in end-user devices have led to issues such as computational resource scarcity and uneven distribution. In MEC networks, large amounts of computational resources and storage are distributed at the network edge. The MEC architecture utilizes available edge nodes and wireless network transmission to provide fast data processing services to surrounding end-users, especially for latency-sensitive and computation-intensive tasks. End-users can offload tasks to MEC servers within the region to reduce their own overhead [1]. In the edge node network, edge server nodes can provide virtualized computing, storage, and network resources to users. Once users request task offloading for computation, the edge network nodes allocate computing resources on-demand. Compared to IoT edge computing, MEC servers in edge networks are typically equipped with limited computational and storage resources. When the resource requests of terminal tasks are unevenly distributed, uneven utilization of server resources within the network can occur. A significant challenge is how to coordinate the limited resources of different edge nodes in the network to achieve high resource utilization. Some work on offloading computational tasks to edge servers has already been done.

To address the issue of limited computational resources in edge server nodes, although some researchers have proposed different solutions to the task scheduling problem, most of these optimizations still focus on a single objective. These studies rely on simple trade-off functions between objectives, which do not ensure effective trade-off solutions that meet all user goals. Therefore, a multi-objective joint optimization problem has been proposed. Considering the complex and dynamic environment of edge computing networks, most researchers' work primarily focuses on two methods: traditional methods and intelligent methods. Traditional methods emphasize adjusting and extending task scheduling strategies in the network, such as First-Come-First-Served, Earliest Job First, Shortest Job First, Round Robin, and Greedy algorithms. The main limitation of traditional methods is that they can only support the optimization of a limited number of parameters. This makes them unsuitable for dynamically changing scenarios, such as edge computing environments that require simultaneous optimization of multiple parameters, such as task transmission delay and transmission cost.

On the other hand, traditional intelligent methods leverage artificial intelligence technologies, such as fuzzy logic, Particle Swarm Optimization, and Genetic Algorithm, to design more reliable scheduling methods while optimizing multiple parameters. However, similar to traditional methods, intelligent scheduling methods operate offline by attempting to optimize a series of parameters upon receiving a specific task. This leads to long execution times, making them inefficient for delay-sensitive tasks, such as IoT and big data analysis tasks. As a result, methods based on Deep Reinforcement Learninghave been proposed to address the task scheduling problem in edge node networks. These algorithms do not require any prior knowledge and can provide optimal scheduling solutions by learning from historical experiences and analyzing the current state information of each network node, thus improving the shortcomings of traditional algorithms that are not suitable for rapidly changing scenarios. To achieve this goal, we have developed an optimized algorithm based on Deep Q-Network(DQN), which takes into account various factors such as task characteristics, device capabilities, and network

conditions. It also considers the current workload and resource constraints of edge servers and devices to determine whether to offload tasks or execute them locally.

The main contributions of this paper are summarized as follows:

The computational offload optimization problem is modeled by considering the computational power of the mobile device, the available computational resources on the edge server, and the network conditions. We then prove that the problem is NP-hard.

Efficient DQN-based offloading algorithm (DQNEO) is proposed to solve the computational offloading optimization problem. DQNEO first modifies the operations based on the available computational resources on the edge servers. Subsequently, it assigns appropriate values to the scheduling solution considering the computational resources of the mobile device.

2. Related Work

Computational offloading is a significant research problem in edge computing, and many researchers focus on developing and implementing efficient task scheduling algorithms for resource allocation, each designed to meet the service quality requirements of tasks in the system. Additionally, these algorithms aim to optimize other aspects of the system, such as response time, energy consumption, utilization, and rental costs. For example, in Infrastructure-as-a-Service [2], users obtain services by renting computational resources such as servers, storage, and cloud networks, which provides strong advantages in flexibility, scalability, and cost. Based on this, some scholars have proposed a series of scheduling algorithms, such as workflow-based dynamic scheduling algorithms [3-6], adaptive particle swarm optimization algorithms [7], and the Kuhn-Munkres algorithm [8]. Yuan et al. proposed an improved differential evolution algorithm that optimizes the distribution of tasks across different virtual machine instances by encoding, mutation, crossover, and selection operations [9]. Ben et al. proposed a spatial task scheduling and resource optimization method that is solved using the simulated annealing-based bat algorithm, providing near-optimal real-time solutions [10]. Another widely adopted metaheuristic approach is the particle swarm optimization algorithm [11], with literature proposing a series of improvements and optimizations to PSO to enhance its convergence speed [12, 13]. However, these algorithms still have some limitations. On the one hand, the scheduling efficiency under resource constraints is not guaranteed in the aforementioned scheduling algorithms. On the other hand, heuristic-based methods do not account for the dynamicity and diversity of task transmission, and they also face difficulties in accurately modeling complex, time-varying edge node networks.

Inspired by the advantages of machine learning in handling large state spaces [14] and action spaces [15], deep reinforcement learning-based methods have gradually been applied to the task scheduling problem in MEC systems. Without any prior knowledge, deep reinforcement learning algorithms can provide optimal scheduling solutions by learning from historical experiences and analyzing the current state information of each network node, thus addressing the shortcomings of traditional algorithms that are unsuitable for rapidly changing scenarios. Reinforcement learning (RL) is a data-driven, model-free algorithm that has been widely applied in task scheduling systems [16]. Its fundamental idea is to adjust the policy by maximizing the reward received from the changes in the environment state to achieve the desired goal. Due to its own limitations, it does not handle high-dimensional and continuous problems well. Deep learning (DL) methods, on the other hand, excel at discovering features in input data and are adept at summarizing perceptual and feature expressions of inputs. Therefore, combining deep reinforcement learning with deep learning can complement each other. Currently, deep reinforcement learning has been successfully applied to task scheduling systems in edge scenarios to solve computational resource allocation problems. For example, Sinde et al., based on using ant colony optimization to search resources, introduced deep reinforcement learning to divide resources into state space and action space, reducing spatial complexity, shortening waiting time, and improving idle resource utilization [17]. Literature considers the complexity of parallel computing environments and the curse of dimensionality and designs a multi-task deep reinforcement learning method for scalable parallel task scheduling, which extends the action selection to multi-task decisions, with resource competition between parallel tasks shared through neural network layers [18]. Experimental results show that MDTS significantly improves reward values compared to earliest connection methods and particle swarm optimization algorithms.

Zeng et al. used a scheduling strategy based on the DQN algorithm to improve computational performance and resource allocation at the network edge [19]. DQN is a typical value-function-based deep reinforcement learning method that can extract patterns from model changes and make correct decisions to balance VM migration costs and data transmission costs, thereby reducing the overall cost in the long run. In a similar network environment, Sheng et al. adopted the Double Deep Q-Learning Network (DDQN) algorithm, an improvement of DQN, in their model to solve the optimal scheduling strategy and minimize the total system consumption cost [20]. Simulation results show that the proposed algorithm achieves the highest task completion rate. However, a series of DQN-based algorithms perform poorly in high-dimensional action spaces. In 2016, the DeepMind team proposed the Deep Deterministic Policy Gradient (DDPG) algorithm, which addresses this issue effectively. Therefore, the algorithm in this paper is based on the DDPG method, providing stable performance and fast convergence speed.

3. System Model and Problem Formulation

This section provides an in-depth discussion of the interconnection mechanisms between edge devices and explores the conceptual modeling of edge device resources in the MEC environment. The main symbols used in this paper are summarized in Table 1.

Table 1. Key Symbols Used in This Paper

Attributes	Description
a_i^{id}	ID of the task
a_i^{siz}	Size of the task
a_i^{sd}	Size of the offloading request data
a_i^{rd}	Size of the result data
a_i^{lat}	Maximum acceptable latency for the task
a_i^{mo}	Mobility state of the device that requested the task
f_j^{id}	Device ID
f_j^{co}	Total MIPS (Million Instructions Per Second) available per core in the device
f_j^{nc}	Number of available CPU cores in the device
f_j^{cu}	Current CPU utilization of the device
f_j^{bc}	Battery capacity (for battery-powered devices)
f_j^{bl}	Remaining battery level (for battery-powered devices)
f_j^{d}	Dynamic energy consumed by the device per second for computing (W)
f_j^{s}	Static energy consumed by the device to stay connected to the network and remain powered on (W)
f_j^{mo}	Mobility state of the device
f_j^{qu}	Number of tasks waiting for execution on the device
f_j^{off}	A binary value indicating whether the device is within the offloading coverage of the task generator device
$t_{i,j}^{se}$	Orchestrator service time
$t_{i,j}^{nt}$	Network time
$t_{i,j}^{wt}$	Waiting time
$t_{i,j}^{e}$	Execution time
$t_{i,j}^{p}$	Total task processing time
$n_{i,j}^{trn}$	Total energy consumed for data transmission over the network (W)
$n_{i,j}^{com}$	Total energy consumed by the device for computing a task (W)
n^{tb}	Energy consumed for each bit of data transmission over the network (W)
$y_{i,j}$	Task success indicator
$w_{i,j}^{rsc}$	Number of wasted resources

3.1. Heterogeneous Edge Computing Environment

Heterogeneous Edge Computing (HEC) refers to a computing environment composed of different types of edge devices, which vary in hardware, software configuration, and computational capacity. HEC aims to handle diverse computing demands by assigning tasks to appropriate devices, thereby improving resource utilization, reducing latency, and optimizing system performance. As shown in Figure.1.

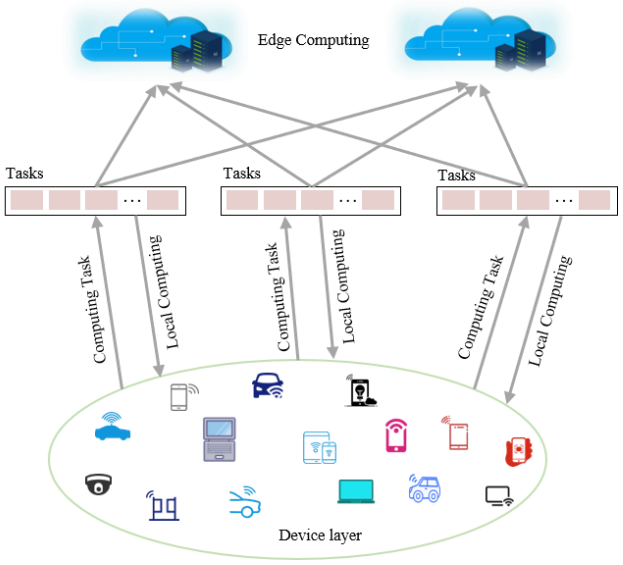

Figure 1. Heterogeneous Edge Computing Environment Model

3.2. Delay and Energy Consumption Model

The model consists of a system of heterogeneous edge devices, with a total of F devices equipped with computational resources, represented as a set $F_c = \{f_1, f_2, f_3, \ldots f_F \mid j \in [1, F]\}$. These devices are interconnected through a high-performance coordinator H_c. Each computationally capable device is represented as $f_j = [f_j^{id}, f_j^{comp}, f_j^{nc}, f_j^{bc}, f_j^{bl}, f_j^{d}, f_j^{s}, f_j^{mo}, f_j^{qu}]$, where the meaning of $f_j^{id}, f_j^{comp}, f_j^{nc}, f_j^{bc}, f_j^{bl}, f_j^{d}, f_j^{s}, f_j^{mo}, f_j^{qu}$ is shown in Table 1. Due to the differences in device types, the parameter values of each device vary. It is assumed that all devices f_j share the network bandwidth B (Hz) equally, enabling communication with the coordinator H_c to ensure that device requests are sent and responses are received with equal priority.

Edge devices generate tasks of varying sizes and service level agreements (SLA). Assume that at each specific time step, A tasks are generated, represented as $A = \{a_1, a_2, a_3, \ldots a_A \mid i \in [1, A]\}$. The number of tasks at each time step dynamically varies and is not fixed. Each task is independently generated by an edge device, represented as $a_i = [a_i^{id}, a_i^{siz}, a_i^{sd}, a_i^{rd}, a_i^{lat}, a_i^{mo}]$, where the meaning of $a_i^{id}, a_i^{siz}, a_i^{sd}, a_i^{rd}, a_i^{lat}, a_i^{mo}$ is detailed in Table 1. When the coordinator H_c receives task a_i at timestamp t, it selects device f_j as the offloading target for task a_i. The computation time for task a_i on device f_j can be expressed as follows:

$$t_{i,j}^{e} = \frac{a_i^{siz}}{f_j^{comp}} \tag{1}$$

The total processing time $t_{i,j}^{p}$ is the time from when task a_i is submitted to when the result is returned, calculated can be defined as:

$$t_{i,j}^{p} = \left(t_{i,j}^{nt} + t_{i,j}^{wt} + t_{i,j}^{se} + t_{i,j}^{e}\right) \quad (2)$$

Task a_i is considered successful if the total processing time $t_{i,j}^{p}$ does not exceed its maximum delay a_i^{lat}. A binary variable $y_{i,j}$ is defined to indicate whether task a_i is successfully executed, as shown in the following equation:

$$y_{i,j} = \begin{cases} 1, & if\ t_{i,j}^{p} \le a_i^{lat} \\ 0, & otherwise \end{cases} \quad (3)$$

This study aims to minimize resource wastage, which refers to resources wasted due to task failures or when the computational capacity of devices exceeds the demand. The resource wastage $W_{i,j}^{rsc}$ is calculated as follows:

$$W_{i,j}^{rsc} = \begin{cases} f_j^{comp}, & if\ y_{i,j} = 0 \\ f_j^{comp} - a_i^{siz}, & if\ y_{i,j} = 1 \\ 0, & if\ y_{i,j} = 1\ and\ (f_j^{comp} - a_i^{siz}) \le 0 \end{cases} \quad (4)$$

The energy model in this paper is based on the design from related literature [21]. The energy consumption for device f_j to transmit the offloading request of task a_i and receive a response over the network, denoted as $n_{i,j}^{trn}$, can be estimated as follows:

$$n_{i,j}^{trn} = n^{tb} t_{i,j}^{nt} \left(a_i^{sd} + a_i^{rd}\right) \quad (5)$$

Where $\left(a_i^{sd} + a_i^{rd}\right)$ represents the total size of the offloading request and the result data transmitted over the network. For the selected edge device f_j during the execution of task a_i, the computational energy consumption $n_{i,j}^{com}$ is calculated as follows:

$$n_{i,j}^{com} = \left(f_j^{d} + f_j^{s}\right) t_{i,j}^{e} \quad (6)$$

The energy required by the coordinator H_c to select the optimal device f_j is related to algorithmic efficiency and other factors, some of which are beyond the scope of this paper. For simplicity, the energy consumption of the coordinator is $n_{i,j}^{H} = 0$. Therefore, the total energy consumption for completing task offloading from edge devices to the target device is as follows:

$$N_{i,j}^{tot} = n_{i,j}^{trn} + n_{i,j}^{com} + n_{i,j}^{H} \quad (7)$$

For battery-powered devices, the remaining battery capacity can be calculated based on the battery capacity f_j^{bc} and the energy consumed for computation and network transmission, with the calculation formula as follows:

$$f_j^{bl} = \frac{f_j^{bc} - \sum_{i=1}^{A}\left(n_{i,j}^{trn} + n_{i,j}^{com}\right)}{f_j^{bc}} \times 100 \quad (8)$$

3.3. Problem Formulation

This paper proposes a solution to achieve efficient workload orchestration in the EEC environment. The aim is to reduce resource wastage during task allocation, ensure that small tasks do not occupy excessive computational resources, maximize resource utilization, and thereby reduce energy consumption and costs. Another goal is to maximize the task success rate by reducing the number of failed tasks, thereby improving task completion within a specific time frame, as follows:

$$\begin{cases} min \sum_{i=1,j\in[1,F]}^{A} W_{i,j}^{rsc} \\ min \sum_{i=1,j\in[1,F]}^{A} \left(1 - y_{i,j}\right) \end{cases} \quad (9)$$

The above two objectives can be combined into a multi-objective optimization problem, expressed as:

$$\begin{aligned} P: min & \sum_{i=1,j\in[1,F]}^{A} \left(W_{i,j}^{rsc} + \left(1 - y_{i,j}\right)\right) \\ s.t.: & C_1 \left(f_j^{comp} - a_i^{siz}\right) = \left(f_j^{comp} - a_i^{siz}\right)_{[min]}, \\ & \forall i \in [1, A], \forall j \in [1, F] \\ & C_2: t_{i,j}^{p} \le a_i^{lat}, \\ & \forall i \in [1, A], \forall j \in [1, F] \end{aligned} \quad (10)$$

This problem has two constraints. The first constraint ensures that resource wastage is minimized. The second constraint ensures that the task processing time does not exceed the maximum acceptable delay a_i^{lat}. Only tasks that satisfy this constraint are considered successful.

It is important to note that these two constraints conflict, as selecting a high-computing-capacity device f_j^{comp} can reduce task processing time $t_{i,j}^{p}$ but increases resource wastage. Since total energy consumption $N_{i,j}^{tot}$ depends on task size a_i^{siz} and device computational capacity f_j^{comp}, reasonably reducing computational resources helps optimize energy consumption. This problem is modeled as a nonlinear mixed-integer problem, and a deep reinforcement learning approach is employed to find the optimal solution, enabling efficient workload coordination and objective optimization.

4. Design and Implementation

4.1. Efficient Offloading Based on DQNEO

To effectively solve problem P, we designed an efficient offloading algorithm based on deep reinforcement learning, named DQNEO. In DQNEO, we define states, actions, and rewards based on problem P as follows:

4.1.1. State Representation

In our architecture, the state consists of two fundamental feature sets: a subset of task attributes and a subset of resource device attributes. Task attributes describe the requirements necessary to ensure the application's QoS, such as task size, task delay, and the mobility status of the task, which can be represented as: $a_i^{'} = \left[a_i^{id}, a_i^{siz}, a_i^{lat}, a_i^{mo}\right]$. The subset of device attributes describes the current state of each available resource at timestamp t, such as device computational capacity, current CPU utilization, remaining battery power, device mobility status, and offloading capability, which can

be represented as: $f_j^{'} = \left[f_j^{id}, f_j^{comp}, f_j^{cu}, f_j^{bl}, f_j^{off}, f_j^{mo}, f_j^{qu} \right]$. In other words, each state captures the attributes of incoming tasks and the attribute states of all available devices at timestamp t. To ensure each state is unique and address the challenges of satisfying the Markov Decision Process (MDP), we include the ID of tasks that have arrived at the scheduler in the state representation. This ensures that each state is distinct from the subsequent state.

4.1.2. Resource Selection

The coordinator's action scope is limited to the number of edge devices with computational resources in the environment. Thus, if there are N computationally capable edge devices, the coordinator selects one of these devices for task offloading. At timestamp t, the action space is defined as $m_i \in \{m_1, m_2, ...m_n\}$. The neural network embedded in the coordinator's DQNEO algorithm is responsible for executing these actions. The coordinator agent needs to select a target device for task offloading. However, not all devices in the action space are available or valid at every time point. To address this issue, the algorithm introduces an action masking mechanism, which excludes unavailable or invalid devices, retaining only valid actions to improve the exploration efficiency of the DQNEO algorithm. In this paper, the primary responsibility of the resource selection module is to detect and mask invalid or infeasible actions to prevent their inclusion in the final selection. Through this process, the algorithm identifies and masks two types of devices: non-offloadable devices and incapable devices.

4.1.3. Reward Function Design

The reward function is key to the DQNEO algorithm, guiding the agent to select optimal actions to achieve multi-objective optimization. We define the reward function as:

$$r = \begin{cases} 1, & 0 \le \dfrac{T_{max} - max\{T_i^t\}}{T_{max}} < 1, \\ 0, & -0.5 \le \dfrac{T_{max} - max\{T_i^t\}}{T_{max}} < 0, \\ -1, & otherwise \end{cases} \tag{11}$$

Where T_{max} represents the system's maximum tolerable delay, and $\max\{T_i^t\}$ represents the maximum total processing time of all tasks in the system.

4.2. Overview of the DQNEO Algorithm

The DQNEO algorithm is a task scheduling method based on deep reinforcement learning, designed to address the complexity of task allocation in edge computing environments. First, the system initialization includes the following components: setting the system state S, learning rate α, experience pool size R, mini-batch size K, update interval I, and maximum iterations T. The system state S describes the features of the task queue and the current resource distribution of edge devices, forming the fundamental input for task scheduling. Two deep neural networks are defined in the algorithm: an evaluation network F_{eval} and a target network F_{target}, both with identical structures but independent parameters. During initialization, the parameters F_{eval} of the evaluation network W_{eval} are randomly initialized and then copied to the parameters F_{target} of the target network W_{target} to ensure stability during the initial training phase.

During the algorithm's iterative process, the current system state S_t is fed into the evaluation network F_{eval}, which outputs the value $Q(S_t, B)$ of all possible task scheduling strategies, where B represents the set of all possible actions. To balance exploration and exploitation, the algorithm employs an $\in$-greedy strategy to select an action B_t. Specifically, an action B_t is randomly chosen with a probability of $\in$ to increase exploration, while the action with the highest current value $Q(S_t, B)$ is selected with a probability of $1-\in$, thereby improving the efficiency of task scheduling.

Once the action B_t is selected, the algorithm calculates the corresponding task scheduling plan P and optimizes P based on system resource constraints to ensure the following conditions are met: (1) the total delay for task completion does not exceed the maximum allowable delay L_{max}, and (2) the total resource usage does not exceed the available resources R_{max} of the edge devices. Subsequently, the task offloading process is performed according to the optimized scheduling plan P, and the scheduling reward R_t is computed. The reward R_t is designed to comprehensively evaluate the quality of the current action B_t by considering task completion rate, resource utilization, and delay.

Next, the algorithm stores the current state S_t, the selected action B_t, the computed reward R_t, and the new state S_{t+1} after task offloading into the experience pool R, forming a quadruple (S_t, B_t, R_t, S_{t+1}). To optimize the evaluation network F_{eval}, a mini-batch $\{(S, B, R, S')\}$ is randomly sampled from the experience pool. The target value is computed based on the reward R and the output of the target network F_{target}, using the following formula: $Q_{target} = R + \gamma \cdot \max_{B} Q_{target}(S', B'; W_{target})$, where γ is the discount factor that balances the weight of immediate rewards and future returns.

The parameters W_{eval} of the evaluation network are updated by minimizing the mean squared error $\zeta = \frac{1}{K}\sum_{i=1}^{K}\left(Q_{target}(S_i, B_i; W_{eval}) - Q_{target}\right)^2$. Gradient descent is employed to optimize this loss function, and at fixed update intervals I, the parameters W_{eval} of the evaluation network are synchronized with the parameters W_{target} of the target network to enhance the stability of the training process.

Finally, the system state is updated to S_{t+1}, and the next iteration begins. Through multiple iterations, the DQNEO algorithm continuously refines the task scheduling strategy,

ultimately achieving efficient task allocation in the edge computing environment. This method dynamically learns to effectively balance resource utilization and task completion rates, reducing task execution delays while minimizing computational resource wastage.

5. Performance Evaluation

This section evaluates the performance of the DQNEO algorithm through computer simulations, focusing on resource utilization, task acceptance rate, task rejection rate, and cost ratio. The evaluation of the DQNEO algorithm is compared with three heuristic algorithms (h1, h2, h3) and DQN. The heuristic method h1 schedules tasks using FIFO (First In, First Out) and prioritizes edge resources if available. h2 prioritizes tasks with high resource demands. h3 employs the 0/1 knapsack algorithm to maximize utilization and benefits. Simulation parameters are shown in Table 2, and the experimental environment configuration is listed in Table 3. The proposed scheme uses the DQNEO algorithm to decide whether to offload or reject tasks based on resource availability and pending tasks. It also considers network information to optimize scheduling performance.

Table 2. Simulation environment configuration

Definitions and description	Values
Edge Link Bandwidth (Mbps)	100
Cloud Link Bandwidth (Mbps)	200
Number of Edge Servers	30
Edge Server CPU Capacity	40-60(evenly distributed)
Task CPU Requests	10-20
Task Data Size (MB)	10-20
Task Tolerable Latency (ms)	10-15
Number of Offloaded Tasks	50,100,150,200,250

Table 3. Learning parameters of the DQNEO algorithm

Definitions and description	Values
Learning rate	0.001
Training batch size	32
Discount factor	0.99
Dense-layer setup (Hidden)	256
The target network smoothly copies the parameter	0.005
N-step for Q-learning	1
Initial epsilon (Exploration)	1.0
Final epsilon (Exploration)	0.1
Target synchronization interval training steps	1000
Replay Buffer Capacity (Size of the replay buffer)	10,000

Figure 2 presents the comparison of average resource utilization between DQNEO and other schemes. As the number of tasks increases, the average resource utilization of all five schemes also rises. However, it can be observed that DQNEO consistently achieves higher utilization compared to the other algorithms, as shown in Figure 4. Task rejection rate is a critical metric that influences resource utilization. A lower task rejection rate corresponds to higher resource utilization. DQNEO adopts a robust mechanism to select the optimal server based on task requirements, thereby improving the efficiency of the edge system. Moreover, DQNEO utilizes intelligent resource allocation strategies to enhance task acceptance rates while maintaining resource utilization. High task acceptance rates generally lead to higher average utilization and lower costs.

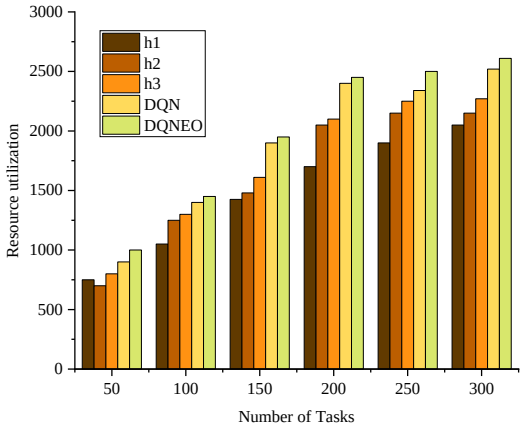

Figure 2. Resource utilization comparison

Figure 3 provides a comprehensive comparison of the task rejection rates among the five different schemes. As the number of tasks increases, the task rejection rate rises for all five schemes. However, DQNEO exhibits a significantly lower growth compared to the other four schemes. This demonstrates the superior performance of DQNEO in terms of task acceptance rate and lower rejection rate. The ability of DQNEO to accept more tasks is attributed to its intelligent task allocation mechanism, which assigns tasks to servers that best match their resource requirements and availability. This mechanism not only conserves and reserves resources for future use but also allows more tasks to be accepted. The advantage of a high task acceptance rate lies in its ability to improve resource utilization and reduce system idle time. Therefore, DQNEO outperforms the other methods in resource utilization, demonstrating its effectiveness in enhancing edge system performance.

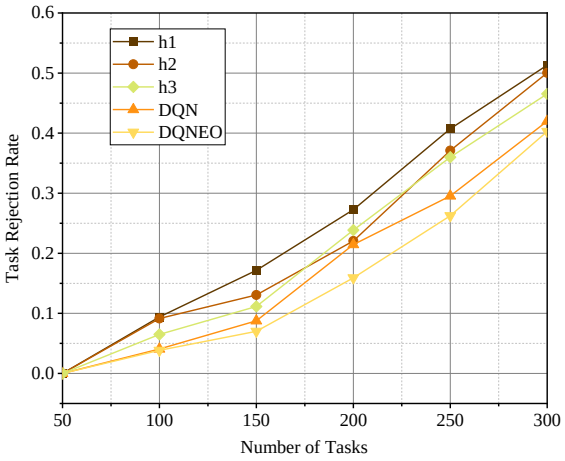

Figure 3. Task rejection comparison

Figure 4 illustrates the cost ratio comparison among the five schemes. As the number of tasks increases, the cost ratio grows for all five schemes. However, DQNEO exhibits significantly lower growth in cost ratio compared to the other four schemes, demonstrating its superior performance in cost ratio reduction. Additionally, compared to heuristic methods and DQN, DQNEO achieves a significantly lower task rejection rate, indicating that it accepts more offloaded tasks and thereby increases the utilization of the edge system. The key to DQNEO's performance lies in its intelligent task allocation capability. It efficiently assigns tasks to servers that best match resource requirements and availability, thereby minimizing overall costs and optimizing the utilization of

edge systems.

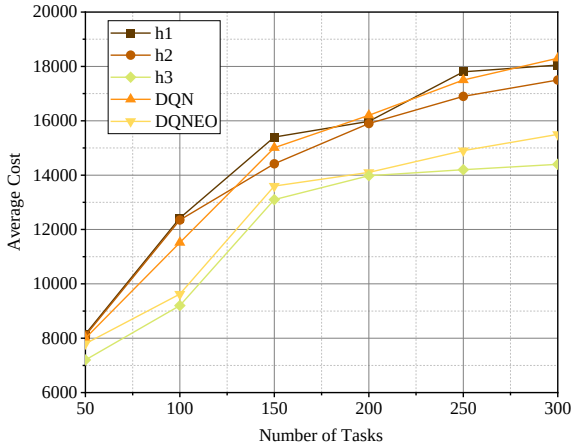

Figure 4. Average cost comparison

6. Conclusion

This paper defines a multi-objective problem for workload coordination in resource-constrained environments, aiming to optimize task success rates while minimizing resource wastage. To address this problem, we designed an efficient offloading algorithm based on deep reinforcement learning. Experimental results demonstrate that the DQNEO algorithm effectively reduces system processing delays while optimizing the utilization of computational resources. By integrating system states, the DQNEO algorithm can make intelligent task offloading decisions, achieving efficient and optimal resource utilization. Additionally, DQNEO considers the realistic characteristics of extreme edge environments, such as device heterogeneity, mobility, and QoS. However, our assumed MEC environment does not involve high-capacity servers. In real-world scenarios, balancing workloads between high-capacity edge servers and edge devices, as well as operating based on predefined objectives, remains a challenge that requires further improvement in future work.

References

[1] Wu Y, Ni K, Zhang C, Qian LP, Tsang DH. NOMA-assisted multi-access mobile edge computing: A joint optimization of computation offloading and time allocation. IEEE Transactions on Vehicular Technology. 2018; 67(12):12244-58.

[2] Li J, Qiu M, Ming Z, Quan G, Qin X, Gu Z. Online optimization for scheduling preemptable tasks on IaaS cloud systems. Journal of Parallel and Distributed Computing. 2012; 72(5):666-77.

[3] Calheiros RN, Buyya R. Meeting deadlines of scientific workflows in public clouds with tasks replication. IEEE Transactions on Parallel & Distributed Systems. 2013; 25(7):1787-96.

[4] Abrishami S, Naghibzadeh M, Epema DH. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Generation Computer Systems. 2013; 29(1):158-69.

[5] Cai Z, Li X, Ruiz R, Li Q. A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds. Future Generation Computer Systems. 2017; 71:57-72.

[6] Jiang H, Song M. Dynamic scheduling of workflow for makespan and robustness improvement in the IaaS cloud. IEEE Transactions on Information and Systems. 2017; 100(4):813-21.

[7] Zuo X, Zhang G, Tan W. Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud. IEEE Transactions on Automation Science and Engineering. 2013; 11(2):564-73.

[8] Fu Z, Tang Z, Yang L, Liu C. An optimal locality-aware task scheduling algorithm based on bipartite graph modelling for spark applications. IEEE Transactions on Parallel and Distributed Systems. 2020; 31(10):2406-20.

[9] Yuan H, Bi J, Zhou M. Energy-efficient and QoS-optimized adaptive task scheduling and management in clouds. IEEE Transactions on Automation Science and Engineering. 2020; 19(2):1233-44.

[10] Bezdan T, Zivkovic M, Bacanin N, Strumberger I, Tuba E, Tuba M. Multi-objective task scheduling in cloud computing environment by hybridized bat algorithm. Journal of intelligent & fuzzy systems. 2022; 42(1):411-23.

[11] Attiya I, Abd Elaziz M, Abualigah L, Nguyen TN, Abd El-Latif AA. An improved hybrid swarm intelligence for scheduling iot application tasks in the cloud. IEEE Transactions on Industrial Informatics. 2022; 18(9):6264-72.

[12] Yuan H, Bi J, Zhou M. Spatial task scheduling for cost minimization in distributed green cloud data centers. IEEE Transactions on Automation Science and Engineering. 2018; 16(2):729-40.

[13] Verma A, Kaushal S. A hybrid multi-objective particle swarm optimization for scientific workflow scheduling. Parallel Computing. 2017; 62:1-19.

[14] Wu Z, Xiong J. A novel task-scheduling algorithm of cloud computing based on particle swarm optimization. International Journal of Gaming and Computer-Mediated Simulations. 2021; 13(2):1-15.

[15] Domanal SG, Guddeti RMR, Buyya R. A hybrid bio-inspired algorithm for scheduling and resource management in cloud environment. IEEE Transactions on Services Computing. 2017; 13(1):3-15.

[16] Shiue Y-R, Lee K-C, Su C-T. Real-time scheduling for a smart factory using a reinforcement learning approach. Computers & Industrial Engineering. 2018; 125:604-14.

[17] Sinde R, Begum F, Njau K, Kaijage S. Refining network lifetime of wireless sensor network using energy-efficient clustering and DRL-based sleep scheduling. Sensors. 2020; 20(5):1540.

[18] Qi Q, Zhang L, Wang J, Sun H, Zhuang Z, Liao J, et al. Scalable parallel task scheduling for autonomous driving using multi-task deep reinforcement learning. IEEE Transactions on Vehicular Technology. 2020; 69(11):13861-74.

[19] Zeng D, Gu L, Pan S, Cai J, Guo S. Resource management at the network edge: A deep reinforcement learning approach. IEEE Network. 2019; 33(3):26-33.

[20] Sheng S, editor A resource collaborative scheduling strategy based on cloud edge framework. 2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC); 2021: IEEE.

[21] Jalali F, Hinton K, Ayre R, Alpcan T, Tucker RS. Fog computing may help to save energy in cloud computing. IEEE Journal on Selected Areas in Communications. 2016; 34(5):1728-39.