Computation Cost-Driven Offloading Strategy Based on Reinforcement Learning for Consumer Devices

Rui Zhang, *Student Member, IEEE*, Hui Xia, *Member, IEEE*, Zijun Chen, *Student Member, IEEE*, Zi Kang, *Student Member, IEEE*, Kai Wang, *Member, IEEE*, and Wei Gao, *Member, IEEE*

***Abstract*—Edge computing extends computational capabilities to the physical edge of the world, providing faster and more efficient responses to consumer devices such as smart home appliances, smartphones, and wearable devices. However, it raises a critical question: How can we efficiently manage time-sensitive and computationally intensive tasks to meet device performance needs? To address this, we propose a computational cost-driven strategy for computation offloading based on reinforcement learning for consumer devices. Firstly, we convert the problem of minimizing computational costs, including latency and energy consumption, into maximizing the cumulative rewards problem of consumer devices. Second, we design a task completion time estimation method to ensure a favorable user experience. Third, by comprehensively considering the status of the network, computational demands, and task completion times, we employ reinforcement learning techniques to determine the optimal task offload strategy for consumer devices by maximizing cumulative rewards. This ensures efficient task completion with lower computational costs. The simulation results demonstrate that our method achieves the highest rewards in different scenarios. Our method reduces the task discard ratio by 25% compared to the PPO_KLP-based offload strategy.**

***Index Terms*—Mobile edge computing, computation offloading, task characteristics, completion time estimation.**

I. Introduction

SMART home devices, smartphones, wearable devices, and other intelligent consumer devices are proliferating at an unprecedented rate in today's digital world, playing an increasingly crucial role in people's lives [1]. The emergence of mobile edge computing (MEC) [2] technology has enabled these applications to achieve faster and more efficient responses to user demands, resulting in low latency and personalized user experiences [3]. Smart surveillance cameras [4] can perform image analysis locally, detecting intrusions or fires in real time without transmitting massive video data to the cloud. Health and fitness trackers [5] can locally monitor and analyze physiological data from users, such as heart rate and step count, in real time, eliminating the need to wait for the data to be uploaded to the cloud for processing.

However, with the rapid development of smart consumer devices, the demand for time-sensitive and computationally intensive tasks continues to grow. For example, smart home devices [4] need to collect, process, and analyze a large amount of data to meet the manufacturers' requirements to improve product performance and predictive maintenance. Despite collecting data locally, these devices may be unable to complete complex data analysis and inference tasks within a reasonable timeframe due to limited computational resources, thus limiting their functionality. Smart speakers and voice assistants [6] need to process voice input, perform speech recognition, and interpret commands in real time to respond quickly to user instructions. However, due to the complexity of computational tasks, these devices may require more time to complete processing, significantly affecting user experience and interaction smoothness. In this context, research on task-offloading strategies for these tasks has become particularly urgent.

Research on computation offloading strategies [7], [8] mainly focuses on latency optimization, energy consumption optimization, and joint optimization of latency and energy consumption. Latency optimization aims to efficiently migrate computation tasks to faster computing resources to meet the requirements of time-sensitive tasks [9], [10], [11], [12], [13], [14], [15], [16], [17]. However, these studies often overlook the energy consumption issues of terminal devices. Given the limited battery capacity of the terminal devices, rapid battery drain can cause interruptions of tasks during computation. Therefore, some researchers have proposed computation offloading strategies based on energy consumption optimization [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28]. However, such studies address concerns about energy consumption, they can lead to relatively higher latency, which is not very user-friendly in terms of the overall experience. To overcome these challenges, joint optimization of latency and energy consumption [23], [29], [30], [31], [32], [33] has become a focus of research. This method balances task execution latency and device energy consumption to achieve optimal system performance. Consequently, we focus on the research of computation offloading strategies that jointly

Manuscript received 11 September 2023; revised 24 November 2023; accepted 18 January 2024. Date of publication 23 January 2024; date of current version 26 April 2024. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62172377 and Grant 62272129, and in part by the Startup Research Foundation for Distinguished Scholars under Grant 202112016. *(Corresponding authors: Hui Xia; Wei Gao; Kai Wang.)*

Rui Zhang, Hui Xia, Zijun Chen, and Zi Kang are with the School of Computer Science and Technology, Ocean University of China, Qingdao 266100, China (e-mail: zhangrui0504@stu.ouc.edu.cn; xiahui@ouc.edu.cn; 18737684301@163.com; kangzi@stu.ouc.edu.cn).

Kai Wang is with the School of Computer Science and Technology, Harbin Institute of Technology, Weihai 264200, China (e-mail: dr.wangkai@hit.edu.cn).

Wei Gao is with the College of Ocean Technology, Ocean University of China, Qingdao 266100, China (e-mail: gaowei@ouc.edu.cn).

Digital Object Identifier 10.1109/TCE.2024.3357459

optimize latency and energy consumption. However, joint optimization of latency and energy consumption increases the complexity of the problem, making it a challenge to find the optimal or near-optimal solutions to obtain the model.

Reinforcement learning is a practical method for addressing the aforementioned problems [35], [36], [37], with its core concept of treating the terminal device as an intelligent agent. The primary task of this agent is to select the optimal computational offloading strategy while considering both latency and energy consumption to maximize user experience and device performance. Researchers have explored various reinforcement learning methods in the field of computational offloading decisions, including Deep Q-Networks (DQN) and their enhancements, policy gradient methods, and Deep Deterministic Policy Gradients (DDPG) [38]. Each of these methods has its strengths, and the choice depends on the specific requirements of the problem. DQN is suitable for discrete action and state spaces, offering high stability and effectiveness, especially in cases where reward signals are well-defined. Policy gradient methods apply to continuous action and high-dimensional state spaces, making them capable of handling complex parameterized policies. DDPG combines the advantages of value iteration and policy gradient methods, excelling in continuous action and state spaces with faster convergence rates and stability. Consumer devices often require choosing the best offloading decisions within a continuous action space while making decisions in a high-dimensional state space. Such offloading decisions often involve complex objective functions and reward signals, requiring a balance between latency and energy consumption. Double Deep Q-Network (DDQN) performs exceptionally well in this context because it can effectively learn and optimize policies to maximize cumulative rewards while adapting to different tasks and changing environments. Therefore, we choose to employ DDQN to derive offloading strategies.

Considering network conditions, computational demands, and task completion times, we use reinforcement learning techniques to assist consumer devices in flexibly adjusting their offloading strategies in various situations to achieve optimal performance and efficiency. To this end, we propose a method called 'Computation Cost-Driven Offloading Strategy based on Reinforcement Learning for Consumer Devices' and evaluate its effectiveness in three scenarios. The main contributions are as follows:

- We present a Computation Cost-Driven Offloading Strategy based on Reinforcement Learning for time-sensitive and energy-intensive tasks. This method enables consumer devices to dynamically adapt their offloading strategies in response to environmental fluctuations and real-time task demands, aiming to optimize performance and efficiency.
- We design a task completion time estimation method that breaks tasks down into multiple modules and estimates the time required for each module under different offloading strategies. This method efficiently estimates task completion time, ensuring a good user experience.

II. Related Work

We will discuss the research status of computation offloading strategies from three perspectives: latency optimization [9], [10], [11], [12], [13], [14], [15], [16], [17], energy consumption optimization [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], and joint optimization of latency and energy consumption [23], [29], [30], [31], [32], [33].

A. Latency Optimization

Latency optimization refers to the efficient migration of computational tasks to faster computing resources to meet the demands of time-sensitive tasks while ensuring that tasks are completed within specified timeframes, thereby enhancing system performance and responsiveness [9], [10], [11]. Feng et al. [12] transformed the problem of minimizing system latency into a mixed-integer nonlinear programming issue by optimizing reverse offloading decisions and resource allocation and used a distributed dual-decomposition approach to achieve optimal resource allocation. Zhou et al. [13] comprehensively considered the high delay risk caused by network uncertainty and the average delay of the system, formulated the distributed robust offloading optimization problem, and proposed a risk-sensitive task offloading scheme. Liu and Liu [14] developed the partitioning and offloading problem as a two-objective optimization problem and designed a delay-priority oriented offloading strategy to help edge nodes make offloading decisions. Saleem et al. [15] explored the reduction of task execution delay in the shared spectrum by employing a low-complexity heuristic algorithm that combines MEC and Device-to-Device partial computation offloading. Tang [16] used DDQN, Dueling DQN algorithms, and the Long Short-Term Memory network to estimate costs and introduced a deep reinforcement learning-based offloading algorithm to minimize latency costs and task loss. Zhao et al. [17] introduced a caching-assisted framework and proposed a task-offloading solution based on the DDPG task preprocessing algorithm. However, these studies often overlook the energy consumption of mobile devices, making them less applicable to devices with limited battery capacity.

B. Energy Consumption Optimization

Energy consumption optimization effectively offloads computing tasks from mobile devices to other computing resources, such as cloud servers or edge servers, to reduce device energy consumption during task execution [18], [19]. Wang and Zhu [20] formulated an optimization problem aimed at minimizing energy consumption and offloading costs, proposing a dynamic offloading strategy. Khune and Pasricha [21] proposed a Q-learning-based offloading decision method to minimize energy consumption. Zhou et al. [22] designed a winning bid scheduling method based on a greedy stochastic adaptive search process to determine the offloading strategy. Chen et al. [23] introduced an advanced DQN that incorporates a priori buffer mechanism and an expert buffer mechanism. Fang et al. [24] considered energy consumption and energy harvesting and proposed an online energy-saving auction algorithm for MEC systems consisting of multiple

servers and mobile devices. Saranya and Sasikala [25] made computational offload decisions based on the tasks assigned to processing, designing types of offloads such as partial offloads, full offloads, and distributed offloads to save energy. Cao et al. [26] considered the MEC system with auxiliary nodes and jointly optimized the allocation of computational and communication resources to users and additional nodes to meet computational latency constraints while reducing overall energy consumption. Zhou et al. [27] proposed a joint optimization of resource allocation and task-offloading strategy to minimize device energy consumption, solved using a low-complexity algorithm. Huang et al. [28] combined offloading and scheduling strategies, using Double Q-learning with intervention to meet real-time task requirements and reduce the energy consumption of consumer devices. Although these studies address energy consumption issues, they may lead to higher latency, which is not user-friendly.

C. Joint Optimization of Latency and Energy Consumption

The core idea of joint optimization of latency and energy consumption is to balance task execution latency and device energy consumption to achieve optimal system performance and maximize efficiency. Kumar et al. [29] made real-time offload decisions based on the requirements of the end application, analyzed them by a fuzz-based offload controller, and proposed a computing offload framework based on artificial intelligence. Yuan et al. [30] proposed a DQN-based frame aggregation and task offloading method by considering delay, energy consumption, and throughput. Mahmoodi et al. [31] modeled task dependencies as general topological graphs and employed linear programming to solve the joint optimization problem of offloading decisions, latency, and energy consumption. Chen et al. [23] introduced an advanced DQN method using the experience replay mechanism to select samples for network training, minimizing total weighted latency and energy consumption. Truong et al. [33] employed a three-step Actor-critical DQN method, in which the agent interacts with the environment and accumulates experiences, subsequently learning the optimal task-offloading strategy. Cheng et al. [34] used an online policy gradient-based Actor-Critic algorithm to manage state and action spaces, reducing the need for training samples, speeding up the learning process, and reducing server latency, energy consumption, and costs. Wang et al. [32] proposed an intelligent task-offloading solution using non-policy reinforcement learning supported by sequence-to-sequence neural networks. However, the combination of optimization of latency and energy consumption increases the complexity of the problem and may not be suitable for applications with specific latency or energy consumption requirements.

Inspired by the above proposal, we first consider the characteristics of tasks, network conditions, and edge server resources. Then, we utilize the reinforcement learning technique to assist in offloading decisions of customer devices, aiming to maximize consumer rewards and minimize computational cost and energy consumption.

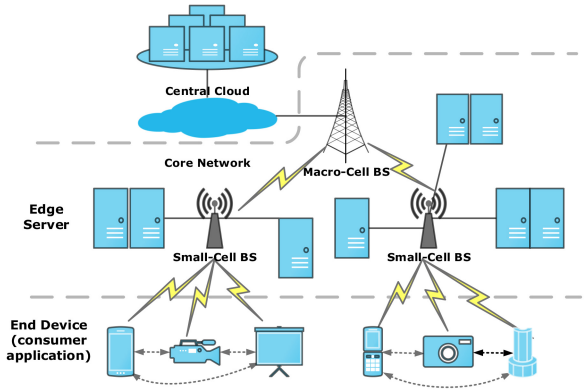


Fig. 1. Computation offloading model.

III. System Model

We consider a wireless cloud edge network that consists of three types of computing nodes: customer devices, edge servers, and central cloud, as shown in Fig. 1. There are $\mathcal{N}$ customer devices and $\mathcal{M}$ edge servers, denoted as set $\mathcal{N} = \{1, 2, \ldots, \mathcal{N}\}$ for customer devices and set $\mathcal{M} = \{1, 2, \ldots, \mathcal{M}\}$ for edge servers. We assume that each customer device has multiple computation-intensive tasks to be completed and that each customer device can offload tasks to edge servers or execute them locally. Edge servers may have limited computing resources and may not be able to fulfill the offloading requirements of consumer devices. In such cases, the edge server may offload some tasks to the central cloud or other edge servers.

A. Local Computing

The task latency computed locally $\mathcal{T}_{t_k}^{local}$ for a task t_k of size S_k depends on the computing power f^{local} and the resources required D_k, such as the frequency of CPU cycles. That is

$$\mathcal{T}_{t_k}^{local} = \frac{D_{t_k}}{f^{local}} \tag{1}$$

The energy consumed is

$$E_{t_k}^{local} = \mathcal{P}^c \frac{D_{t_k}}{f^{local}} \tag{2}$$

$\mathcal{P}^c$ is transmission energy consumption per unit time. Integrating Eq. (1) and (2), we get the local computing cost

$$C_{t_k}^{local} = \mathcal{I}_{t_k}^t \mathcal{T}_{t_k}^{local} + \mathcal{I}_{t_k}^e E_{t_k}^{local} \tag{3}$$

$\mathcal{I}_{t_k}^t$ and $\mathcal{I}_{t_k}^e$ are the time and energy weights of the task cost.

B. Offload Computing

The latency of offloading task t_k from customer devices to edge servers is related to the offloading ratio α, the upload rate of the wireless network channel R^u, the computational capacity f^{edge_m} of the edge server m, and the number of resources required to complete the task D_{t_k},

$$\mathcal{T}_{t_k}^{edge_m} = \alpha \left(\frac{S_{t_k}}{R^u} + \frac{D_{t_k}}{f^{edge_m}} \right) \tag{4}$$

The energy consumed is

$$E_{t_k}^{edge_m} = \alpha\left(\mathcal{P}^u \frac{S_{t_k}}{R^u} + \mathcal{P}^{idle}\frac{D_{t_k}}{f^{edge_m}}\right) \tag{5}$$

$\mathcal{P}^u$ and $\mathcal{P}^{idle}$ are transmission energy consumption and task processing energy consumption per unit time. Integrating Eq. (4) and (5), we get the computational cost of the edge server m,

$$C_{t_k}^{edge_m} = \mathcal{I}_{t_k}^t \mathcal{T}_{t_k}^{edge_m} + \mathcal{I}_{t_k}^e E_{t_k}^{egde_m} \tag{6}$$

if the edge server does not have enough resources to handle the task t_k, it is offloaded to the central cloud or another edge server. In this case, the delay of central cloud $\mathcal{T}_k^{cloud}$ and other edge server $\mathcal{T}_{t_k}^{edge_i}$ is

$$\mathcal{T}_{t_k}^{cloud} = \alpha\lambda\left(\frac{S_{t_k}}{R^{trans}} + \frac{D_{t_k}}{f^{cloud}}\right) \tag{7}$$

$$\mathcal{T}_{t_k}^{edge_i} = \alpha\lambda\left(\frac{S_{t_k}}{R^{trans}} + \frac{D_{t_k}}{f^{edge_{i\in\{1,\ldots,m-1,m+1,\ldots,M\}}}}\right) \tag{8}$$

R^{trans} is the channel upload rate to the central cloud or other edge servers. f^{cloud} is the computational capacity of the central cloud. $f^{edge_{i\in\{1,\ldots,m-1,m+1,\ldots,M\}}}$ is the computing capacity of any server another than edge server m. λ is the proportion of tasks offloaded from edge servers m to the central cloud or other edge server. The energy consumed is

$$E_{t_k}^{-} = \alpha\lambda\left(\mathcal{P}^u \frac{S_{t_k}}{R^{trans}} + \mathcal{P}^{idle}\frac{D_{t_k}}{f^{-}}\right) \tag{9}$$

f^- represents the computational capacity of the central cloud or edge server. Integrating Eq. (7) and (8), the computing cost is

$$C_{t_k}^{-} = \mathcal{I}_{t_k}^t \mathcal{T}_{t_k}^{-} + \mathcal{I}_{t_k}^e E_{t_k}^{-} \tag{10}$$

The total cost of task-offloading is

$$C_{call} = \sum_{k=1}^{\hat{\mathcal{N}}} (1-\alpha) C_{t_k}^{local} + \alpha(1-\lambda) C_{t_k}^{server} + \alpha\lambda C_{t_k}^{-} \tag{11}$$

We model task-offloading as an optimization problem with the objective of minimizing the cumulative cost, which is the sum of latency and energy consumption. Under constraints of maximum tolerable delay and computational capacity, the problem can be formulated as follows:

$$\begin{aligned}
&\min_{\mathcal{A},\mathcal{S}} \sum_{k=1}^{\hat{\mathcal{N}}} {B_{t_k}}^T \left[\, C_{t_k}^{local} \;\; C_{t_k}^{server} \;\; C_{t_k}^{-} \,\right]^T \\
&s.t., \\
&\mathcal{C}_1 : B_{t_k} = \left[\, b_1 \; b_2 \; b_3 \,\right], b_{-} \in \{0, 1\} \\
&\mathcal{C}_2 : {B_{t_k}}^T \cdot \left[\, \mathcal{T}_{t_k}^{local} \; \mathcal{T}_{t_k}^{server} \; \mathcal{T}_{t_k}^{-} \,\right]^T < \kappa_{t_k}, \forall k \in \hat{\mathcal{N}} \\
&\mathcal{C}_3 : 0 < f_{t_k} < F^*, \forall k \in \hat{\mathcal{N}}, * \in \{local, edge, cloud\} \\
&\mathcal{C}_4 : \sum_{k=1}^{\hat{\mathcal{N}}} f_{t_k} \le F^*, \forall k \in \hat{\mathcal{N}}, * \in \{local, edge, cloud\}
\end{aligned} \tag{12}$$

where $\hat{\mathcal{N}}$ is the number of tasks in the customer device. $\mathcal{A} = [B_{t_1}, B_{t_2}, \ldots, B_{t_{\hat{\mathcal{N}}}}]$ is the decision vector for the offload of the task, $\mathcal{S} = [f_{t_1}, f_{t_2}, \ldots, f_{t_{\hat{\mathcal{N}}}}]$ is the allocated computing resources, $\mathcal{C}_1$ indicates whether the customer device chooses local computation or offloading to complete the task, $\mathcal{C}_2$ represents the constraint that the task completion time cost should not exceed a time threshold κ_{t_k}, F^* represents the resource threshold, $\mathcal{C}_3$ denotes that the allocated computing resources should not exceed the computing resources of the offload target, and $\mathcal{C}_4$ represents the total allocated computing resources should not exceed the computing resources of the offloading target.

Eq. (12) can be solved by finding the optimal values of the vector of execution of the decision and the computation of the allocation of resources. However, the feasible set and objective function of Eq. (12) is not convex. Moreover, as the number of users increases, the scale of Eq. (12) overgrows, making it a hard NP problem. Instead of traditional optimization methods to solve the NP-hard problem (12), we propose a reinforcement learning technique to find the optimal values of $\mathcal{A}$ and $\mathcal{S}$.

IV. Problem Solving

A. Determining the Computation Offloading Strategy

Compared to the traditional DQN algorithm, DDQN demonstrates higher stability and convergence speed, allowing more efficient learning and policy optimization to maximize cumulative rewards. Meanwhile, DDQN can continuously learn from the environment, updating based on real-time observations and reward signals. Hence, it can adapt to the dynamic changes in the environment and task requirements, swiftly adjusting to new conditions and scenarios based on the latest information to make offloading decisions. Thus, we use the DDQN algorithm to make offload computation decisions. The basic elements are defined as follows:

State: The state space of the system consists of three parts, including the computing capabilities of customer devices, edge servers, and the central cloud.

Action: The system's action space consists of task-offloading strategies $\mathcal{A} = [B_{t_1}, B_{t_2}, \ldots, B_{t_{\hat{\mathcal{N}}}}]$ and allocation of computational resources $\mathcal{S} = [f_{t_1}, f_{t_2}, \ldots, f_{t_{\hat{\mathcal{N}}}}]$.

Reward: After taking each possible action a in state s, the agent receives a reward $R(s,$ a). Stable rewards can help the agent learn optimal strategies more rapidly and exhibit robustness across varying environmental conditions. Our optimization objective is to minimize the total cost, while the goal of reinforcement learning is to maximize rewards. Hence, rewards should be inversely correlated with the total cost. According to the above definition, costs encompass processing time and resource consumption. Our goal is to ensure timely task completion and maximise resource utilization to enhance the return on investment for client devices. However, to ensure that client devices can complete latency-sensitive tasks on time, an understanding of the computing capability of the offloading destination is required. Unfortunately, this is impractical. Therefore, we design a task completion time estimation method to assist consumer devices in making offloading decisions, ensuring a positive user experience. For specific calculations, please refer to Section IV-B.

Algorithm 1 Computation Cost-Driven Offloading Strategy

Input the size of the replay buffer, training batchsize, target network update frequency;
Initialize the main network parameters θ and the target network parameters θ_{target};
for $e \in \{1, 2, \cdots, epoch\}$ **do**
Select action a using Eq. (13) based on the current state s;
Perform action a and observe the new state s' after transfer and reward;
Calculate the target Q-value;
Calculate the Q-value function of the primary network;
Calculate the loss using Eq. (15);
Update the parameters of the primary network;
Copies the parameters of the primary network θ to those of the target network θ_{target} at a defined update frequency;
end

Q-Value Function: DDQN utilizes the Q-value function to evaluate the value of each state-action pair. The Q-value function is $Q(s, a; \theta)$. The Q-value function represents the long-term cumulative reward obtained by taking a particular action in the current state. To stabilize the learning process, DDQN introduces the target Q-value function $\mathrm{Q}(s, a; \theta_{target})$ to calculate the target Q-value.

$$Q = Q_{t\arg et}\left(s, a; \theta_{target}\right) + \left(Q(s, a; \theta) - \frac{1}{O}\sum_{o=1}^{O} Q(s, a; \theta)\right) \tag{13}$$

Our primary and target networks utilize the same neural network structure, including three hidden layers with 64 neurons each. Each hidden layer applies the ReLU activation function for non-linear transformations. We use the ε-greedy strategy to select the action, which means that we randomly select an action with a probability of ε and select the action with the maximum Q-value with a probability of (1-ε).

$$\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}(s)|} & \text{if} \quad \text{a=arg max}_a Q(s, a; \theta) \\ \frac{\varepsilon}{|\mathcal{A}(s)|} & \text{if} \quad \text{a} \neq \arg\max_a Q(s, a; \theta) \end{cases} \tag{14}$$

We use Mean Squared Error to measure the difference between the predicted values of the current and target Q-value functions, denoted $L(\theta)$.

$$L(\theta) = \frac{1}{O}\sum_{o=1}^{O}\left(Q^{o}_{target}\left(s, a; \theta_{target}\right) - Q^{o}(s, a; \theta)\right)^2 \tag{15}$$

O is the number of samples sampled from the experience replay collection. The pseudocode for the above procedure is shown in Algorithm 1.

B. Task Completion Time Estimate

To ensure a good user experience, we design a task completion time estimation method to ensure that tasks can be completed within a threshold. A task t_k can be characterized by H attributes, $t_k = \{a_1, a_2, \ldots, a_H\}$, a_h represents input data features, computational complexity, resource requirements, real-time constraints, and priority. To estimate the completion time of a task, we divide tasks into two categories: simple and complex. Simple tasks consist of a single module that cannot be further divided, whereas complex tasks are divided into multiple modules.

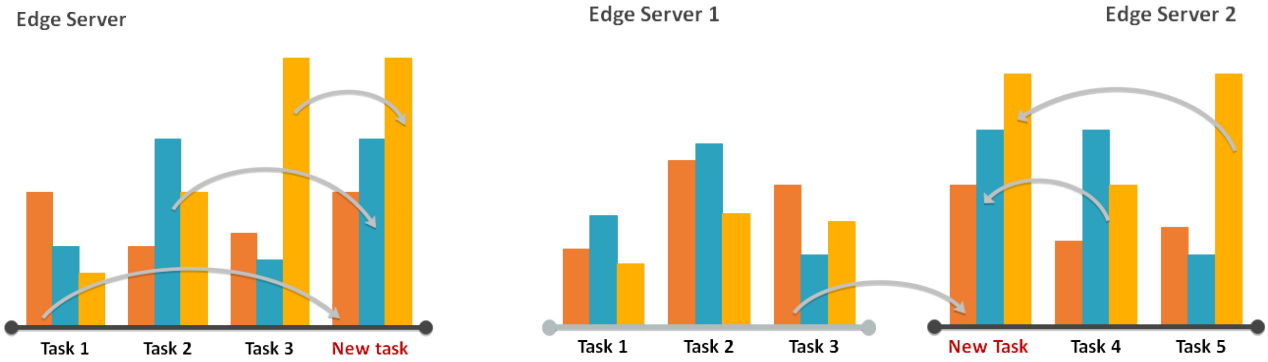
Simple tasks: By analyzing the historical data of the features of the tasks and the completion time, we can build models to predict the completion time of new tasks. Thus, we employ clustering methods to classify and analyze tasks. Specifically, we assign tasks to respective clusters based on their similarity and then calculate the average completion time of tasks in each cluster. This average completion time is considered the current cluster task completion time, from which we can obtain the completion time for new tasks t'_k. However, task features and requirements diversity can result in tasks exhibiting nonconvex distributions in MEC. The Density-Based Spatial Clustering of Applications with Noise algorithm (DBSCAN) can handle nonconvex clusters and is insensitive to data distribution. It is suitable for addressing this issue in clustering edge computing tasks. In particular, we use cosine similarity to measure task similarity in the DBSCAN algorithm.

$$cs = \frac{\sum_{h=1}^{H} a_h a'_h}{\sqrt{\sum_{h=1}^{H} a_h^2}\sqrt{\sum_{h=1}^{H}\left(a'_h\right)^2}} \tag{16}$$

The completion time of the task $\mathcal{T}_{t'_k}$ is the average completion time of all tasks $N_{cluster}$ within the same cluster $cluster_c$.

$$\mathcal{T}_{t'_k} = \frac{1}{N_{cluster}}\sum_{c=1}^{N_{cluster}} \mathcal{T}_{cluster_c} \tag{17}$$

Complex tasks: We estimate the completion time for complex tasks by dividing them into multiple modules. A complex task t_k can be divided into n modules, $t_k = \{\varphi_1(t_k), \ldots, \varphi_n(t_k)\}$, where $\varphi_{\hat{i}}(t_k)$ represents the $\hat{i}$-th module of task t_k. The historical task set submitted by the customer device to the edge server m is $\{t_1, t_2, \ldots, ., t_{k_0}, \ldots, t_{n_1}\}$, the historical tasks t_{k_0} represented by $t_{k_0} = \{\varphi_1(t_{k_0}), \ldots, \varphi_{\hat{i}}(t_{k_0}), \ldots, ., \varphi_n(t_{k_0})\}$. For a new task $t'_k = \{\varphi_1(t'_k), \ldots, \varphi_{\hat{i}}(t'_k), \ldots, ., \varphi_{n'}(t'_k)\}$, if $\varphi_{\hat{i}}(t'_k) = \varphi_{\hat{i}}(t_{k_0})$, then t'_k can be defined based on the modules of previous tasks. Therefore, based on whether the modules of the new task fully or partially exist in the modules of the historical tasks on the server or the central cloud, the definition of the new task can be divided into the following two cases: (*i*) the modules of the new task fully exist in the modules of the historical tasks on the edge server m, i.e., $t'_k = (\varphi_1(t_1), \varphi_2(t_2), \ldots, \varphi_n(t_{n_1}))$, where $\varphi_1(t_1) = \varphi_1(t'_k), \varphi_2(t_2) = \varphi_2(t'_k), \ldots, \varphi_n(t_{n_1}) = \varphi_{n'}(t'_k)$; (*ii*) some modules of the new task partially exist in the modules of the historical tasks on the current server, $t'_k = \{\varphi_1(t_1), \varphi_2(t_2), \ldots, \varphi_{\hat{i}}(t_k), \varphi_{\hat{i}+1}(t'_k), \ldots, \varphi_{n'}(t'_k)\}$, where $\varphi_{\hat{i}+1}(t'_k), \ldots, \varphi_{n'}(t'_k)$ represent the modules only present in the new task t'_k.



(a) The task modules come from the same edge server (b) The task modules come from different edge servers.

Fig. 2. Task completion time estimate (The orange, yellow, and blue bars represent the attributes of the task).

For (*i*), as shown in Fig. 2(a), we evaluate according to the task completion time inference function $\mathcal{J}$. For the completion time of t'_k,

$$\mathcal{T}_{t'_k} = \mathcal{J}\left(\varphi_1(t_1), \varphi_2(t_2), \ldots, \varphi_n(t_{n_1})\right) \tag{18}$$

If each module of the current task can be completed in parallel,

$$\mathcal{J} = \max\{\mathcal{T}_{\varphi_1(t_k)}, \ldots, \mathcal{T}_{\varphi_{n'}(t_k)}\} \tag{19}$$

otherwise,

$$\mathcal{J} = \sum_{\hat{i}=1}^{n'} w_{\hat{i}}(t_k)\mathcal{T}_{\varphi_{\hat{i}}}(t_k) \tag{20}$$

$w_{\hat{i}}(t'_k)$ represents the weight of the $\hat{i}$-th module in t'_k.

For (*ii*), as shown in Fig. 2(b), if the attribute $\varphi_{\hat{i}+1}(t'_k), \ldots, \varphi_{n'}(t'_k)$ is the same as the historical task attribute from other servers, $\varphi_{\hat{i}+1}(t'_k) = \varphi_{n_1+1}(t_k^2), \ldots, \varphi_{n'}(t'_k) = \varphi_n(t_k^i)$, $\varphi_n(t_k^m)$ is the module of t'_k from edge server *i*. Then, the customer device completion time inference function $\mathcal{J}$ can be defined as follows.

$$\mathcal{T}_{t'_k} = \mathcal{J}\left(\varphi_1(t_1), \varphi_2(t_2), \ldots, \varphi_{n_0}(t_{n_1}), \varphi_{n_0+1}\left(t_k^2\right), \ldots, \varphi_n\left(t_k^m\right)\right) \tag{21}$$

If each module of the current task can be completed in parallel

$$\mathcal{J} = \max\left\{\mathcal{T}_{\varphi_1(t_1)}, \ldots, \mathcal{T}_{\varphi_{n_0}(t_{n_1})}, \mathcal{T}_{\varphi_{n_0+1}(t_k^2)}, \ldots, \mathcal{T}_{\varphi_n(t_k^i)}\right\} \tag{22}$$

otherwise,

$$\mathcal{J} = (1-\beta)\sum_{\hat{i}=1}^{n_0} w_{\hat{i}}(t_k)\mathcal{T}_{\varphi_{\hat{i}}}(t_k) + \beta \sum_{\hat{i}=n_0+1}^{n} w_{\hat{i}}\left(t_k^i\right)\mathcal{T}_{\varphi_{\hat{i}}}\left(t_k^i\right) \tag{23}$$

β indicates the trust degree of the current server to other servers, n_0 indicates the number of t'_k's modules that the current edge server (such as Edge Server *m*) contains, $n_0 + 1 \sim n$ indicates the number of t'_k's modules that other edge servers (such as Edge Server *i*) contain.

Please note that if the DBSCAN algorithm classifies a simple task as noise, the calculation of the task completion time is

$$\mathcal{T}_{t'_k} = \sum_{i=1}^{\hat{I}} w_i \mathcal{T}^i_{t'_k} \tag{24}$$

$\mathcal{T}^i_{t'_k}$ is the task completion time on edge server *i*. w_i is the degree of trust of the current edge server with the edge server *i*. $\hat{I}$ is the number of other edge servers.

V. Experimental Result

A. Experimental Setup

To validate the effectiveness of the proposed method, we first analyze the rewards in the same environment settings obtained by five methods (Proximal Policy Optimization (PPO)-based offload strategy [32], Kullback-Leibler Proximity (PPO_KLP)-based offload strategy [35], DDPG-based offload strategy [38], Soft Actor-Critic (SAC)-based offload strategy [39], OUR) in three scenarios to verify that our method can achieve the lowest cost. Second, we analyze the rewards in different environments of five baseline methods (Random, Local, Particle [20], MEC [21], ADDQN [23], and OUR), the latency three benchmark methods (FUV [12], FUR [12], Greedy [22], and OUR), to verify that our method achieves the highest rewards and incurs the minimum delay. Finally, we assess the accuracy of seven completion time estimation methods (Kmeans estimation, Optics estimation, Mean Shift estimation, Analog estimation, Programmer Evaluation and Review Technique (PERT), OUR) to verify the validity of task completion time estimation method.

Baseline methods: The PPO-based offload strategy [32] uses PPO to make offloading decisions. PPO [35] is a policy optimization-based reinforcement learning algorithm designed to improve stability by limiting the magnitude of policy changes in each update. The PPO_KLP-based offload strategy uses PPO_KLP to make offloading decisions. PPO_KLP [35] is a variant of the PPO that introduces a penalty term based on the Kullback-Leibler divergence in the optimization objective. The DDPG-based offload strategy [38] uses DDPG to make offload decisions. DDPG [38] is a deep reinforcement learning algorithm based on policy gradients, particularly suitable for problems with continuous action spaces. The SAC-based offload strategy [39] uses SAC to make offload decisions. SAC [39] is a deep reinforcement learning algorithm based on the maximum entropy framework.

FUV [12] does not employ reverse offloading and instead offloads all tasks to edge servers for execution. FUR [12]

TABLE I
NETWORK RESOURCE

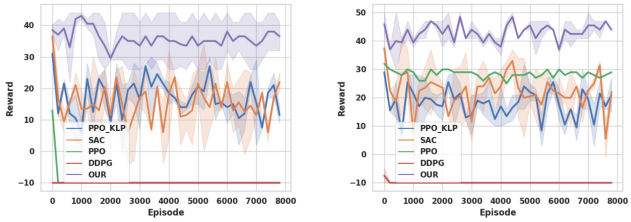
Configuration	Scenario 1	Scenario 2	Scenario 3
(1, 3, 1)	(30, (20, 50, 50), 50)	(0, (20, 50, 50), 50)	(30, (0, 0, 50), 50)
(1, 5, 1)	(30, (20, 50, 50, 20, 30), 50)	(0, (20, 50, 50, 20, 30), 50)	(30, (0, 0, 0, 20, 30), 50)

employs a reverse offloading framework to offload all tasks to consumer devices for execution. Particle [20] treats the state of the system as relevant variables, using the K-nearest neighbors algorithm to quantify the neural network. MEC [21] employs Q-learning for reward-based offloading decisions, allowing edge devices to consider the context for meaningful offloading and network selection. Greedy [22] uses a greedy randomized adaptive search procedure to determine the offload strategy. ADDQN [23] maximizes the benefits of offloading by combining priority buffer and expert buffer mechanisms, determining the reverse offloading strategy to the fullest extent possible.

Parameter settings: For reward analysis and task discard ratio analysis using three baseline methods (DDPG, PPO_KLP, and SAC), we use environment variables consistent with those in our study. The environment variables for the ADDQN, Particle, and MEC methods are referenced from their settings. In the case of delay analysis using the Greedy, FUV, and FUR benchmark methods, the environment variables are all based on those set in the FUR. We conduct experiments with assigned and random sequences of 10 and 20 tasks. The resource configurations shown in Table I. (1, 3, 1) represents a network with one customer device, three edge servers connected to the customer device, and one central cloud. Similarly, (30, (20, 50, 50), 50) indicates that the resource capacity of the customer device is 30, there are three edge servers with resource capacities of 20, 50, and 50, and the resource capacity of the central cloud is 50. Likewise, (30, (20, 50, 50, 20, 30), 50) represents a configuration where the computational resources of the customer device is 30. There are five edge servers with resource capacities of 20, 50, 50, 20, and 30, and one central cloud with a resource capacity of 50. The assigned task sequences are presented in Table II.

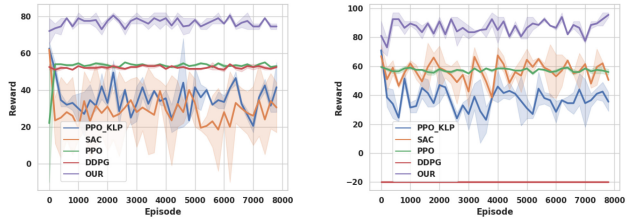
TABLE II
RESOURCE REQUIREMENTS AND TIME THRESHOLDS FOR GIVEN TASK SEQUENCE

Task number: 10			Task number: 20				
Name	Resource	Deadline	Name	Resource	Deadline	Name	Resource
1	8	10	1	9	6	11	3
2	1	7	2	7	6	12	3
3	7	4	3	5	7	13	8
4	8	1	4	7	3	14	1
5	3	9	5	2	8	15	2
6	6	3	6	7	5	16	1
7	9	7	7	10	2	17	8
8	8	5	8	2	2	18	5
9	10	1	9	6	2	19	3
10	7	9	10	9	1	20	8



(a) Assign (b) Random

Fig. 3. Reward analysis (Computing resources: customer device, edge server, and central cloud all have to compute resources, tasks: 10, customer devices: 1, central cloud: 1, edge servers: 3).



(a) Assign (b) Random

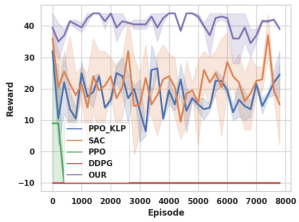
Fig. 4. Reward analysis (Computing resources: customer device, edge server, and central cloud all have to compute resources, tasks: 20, customer devices: 1, central cloud: 1, edge servers: 3).

B. Reward Analysis in the Same Environment Setting

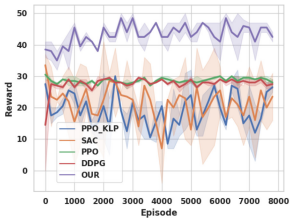
1) Reward Analysis of Different Offloading Strategies for Customer Devices in Scenario 1: Figs. 3 to 6 present an analysis of the rewards obtained by customer devices when employing five offloading strategies in Scenario 1.

Fig. 3 illustrates the rewards obtained by the consumer device using different strategies when ten tasks and three edge servers are connected to the consumer device. Fig. 3(a) shows the rewards obtained by customer devices with different task-offloading strategies when assigning ten tasks. SAC and PPO show similar reward performance with little difference, while DDPG and PPO exhibit the worst results. Fig. 3(b) presents the rewards obtained by customer devices with different task-offloading strategies when randomly generating the sequence of ten tasks. It can be seen from this figure that our method's derived task-offloading strategy achieves the highest reward, followed by PPO, SAC, and PPO_KLP, which perform similarly, and DDPG performs the worst. This is because DDPG is unsuitable for this scenario, where the performance of DDPG is highly affected by the initial parameters of the model. Our method is more stable from these two figures and achieves the highest rewards compared to the other four methods.

Fig. 4 demonstrates the rewards obtained by the consumer device using different strategies when 20 tasks and three edge servers are connected to the consumer device. Fig. 4(a) shows the rewards obtained by customer devices with different

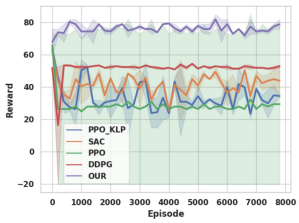


(a) Assign

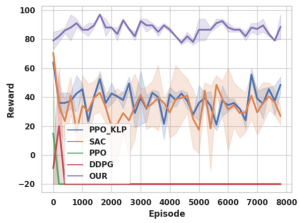


(b) Random

Fig. 5. Reward analysis (Computing resources: customer device, edge server, and central cloud all have to compute resources, tasks: 10, customer devices: 1, central cloud: 1, edge servers: 5).



(a) Assign



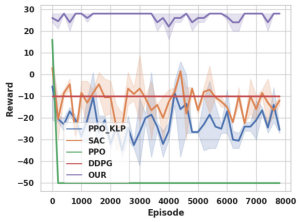
(b) Random

Fig. 6. Reward analysis (Computing resources: customer device, edge server, and central cloud all have to compute resources, tasks: 20, customer devices: 1, central cloud: 1, edge servers: 5).

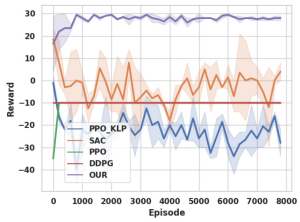
task-offloading strategies when assigning 20 tasks. From the figure, our method's derived task-offloading strategy achieves the highest and fastest convergence of rewards, followed by DDPG and PPO. Still, DDPG is more stable, and SAC performs the worst. Fig. 4(b) presents the rewards obtained by customer devices with different task-offloading strategies when randomly generating the sequence of 20 tasks. It can be observed from this figure that our method's derived task-offloading strategy achieves the highest reward, followed by PPO and SAC, with DDPG performing the worst. These two figures indicate that our method is more stable and achieves the highest rewards compared to the other four methods.

Fig. 5 presents the rewards obtained by the consumer device using different strategies when ten tasks and five edge servers are connected to the consumer device. Fig. 5(a) illustrates the rewards obtained by customer devices with different task-offloading strategies when assigning ten tasks. It is evident from this figure that our method's derived task-offloading strategy achieves the highest and most stable rewards, with DDPG and PPO yielding negative rewards and performing the worst. Fig. 5(b) shows the rewards obtained by customer devices with different task-offloading strategies when randomly generating the sequence of 10 tasks. It can be seen from this figure that our method's derived task-offloading strategy obtains the highest reward, followed by DDPG and PPO, with SAC and PPO_KLP performing similarly. Based on these two figures, our method is more stable and achieves the highest rewards compared to the other four methods.

Fig. 6 shows the rewards obtained by the consumer device using different strategies when 20 tasks and five edge servers are connected to the consumer device. Fig. 6(a) presents the rewards obtained by customer devices with different task-offloading strategies when assigning 20 tasks. It is evident



(a) Assign



(b) Random

Fig. 7. Reward analysis (Computing resources: no local computing resources, tasks: 10, customer devices: 1, central cloud: 1, edge servers: 3).

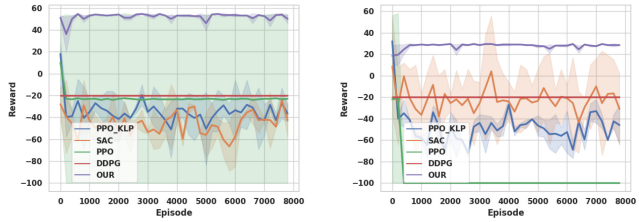
from this figure that our method's derived task-offloading strategy achieves the highest and most stable rewards, with DDPG following and PPO exhibiting the largest reward fluctuations, and thus performing the worst. Fig. 6(b) shows the rewards obtained by customer devices with different task-offloading strategies when randomly generating the sequence of 20 tasks. It can be observed from this figure that our method's derived task-offloading strategy obtains the highest reward, with SAC and PPO_KLP performing similarly and PPO and DDPG yielding negative rewards and performing the worst. These two figures indicate that our method is more stable and achieves the highest rewards compared to the other four methods.

In summary, when consumer devices, edge servers, and cloud servers have computing resources available, the computed offloading strategies derived from our method exhibit better stability and yield higher rewards than the four baseline methods.

2) Reward Analysis of Different Offloading Strategies for Customer Devices in Scenario 2: Figs. 7 to 9 depict the rewards obtained by the customer device using different algorithms to derive task-offloading strategies in Scenario 2.

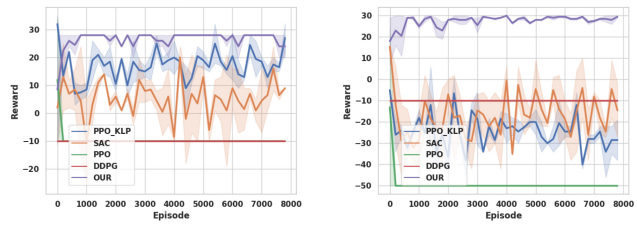
Fig. 7 illustrates the rewards obtained by the consumer device using different strategies when ten tasks and three edge servers are connected to the consumer device. Fig. 7(a) illustrates the rewards of different task-offloading strategies when ten tasks are assigned. Our method yields the highest reward, while DDPG and PPO achieve approximately -10 and -50 rewards, respectively. SAC and PPO_KLP exhibit varying rewards that fluctuate between -30 and 0, indicating that our method results in the highest task completion rate. Fig. 7(b) shows the rewards obtained by different task-offloading strategies when ten tasks are randomly generated. It can be seen that our method achieves the highest reward, followed by SAC, while PPO_KLP shows the least effective performance. This reflects that our method leads to the highest task completion quantity. These two figures reveal that our method is more stable and achieves the highest reward and task completion rate compared to the other four methods.

Fig. 8 presents the rewards obtained by the consumer device using different strategies when 20 tasks and three edge servers are connected to the consumer device. Fig. 8(a) shows the rewards of different task-offloading strategies when 20 tasks are assigned. Our method yields the highest, fastest converging, and most stable rewards, followed by DDPG and PPO,



(a) Assign (b) Random

Fig. 8. Reward analysis (Computing resources: no local computing resources, tasks: 20, customer devices: 1, central cloud: 1, edge servers: 3).



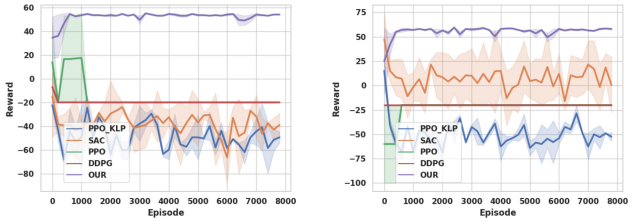
(a) Assign (b) Random

Fig. 9. Reward analysis (Computing resources: no local computing resources, tasks: 10, customer devices: 1, central cloud: 1, edge servers: 5).

with DDPG exhibiting more stability and SAC showing the least effective performance. Fig. 8(b) illustrates the rewards obtained by different task-offloading strategies when 20 tasks are randomly generated. Our method achieves the highest reward, followed by SAC, while PPO demonstrates the least effective performance. This is because SAC and PPO are not suitable for this scenario, and the performance is heavily affected by the initial parameters of the model. These two figures indicate that our method is more stable and achieves the highest reward and task completion rate compared to the other four methods.

Fig. 9 displays the rewards obtained by the consumer device using different strategies when ten tasks and five edge servers are connected to the consumer device. Fig. 9(a) shows the rewards of different task-offloading strategies when assigned ten tasks. Our method yields the highest and most stable rewards, followed by PPO_KLP, while PPO and DDPG achieve negative rewards with the least effective performance. Fig. 9(b) illustrates the rewards obtained by different task-offloading strategies when ten tasks are randomly generated. Our method achieves the highest reward, followed by DDPG, with SAC and PPO_KLP exhibiting similar performance and PPO demonstrating the least effective performance. These two figures reveal that our method is more stable and achieves the highest reward compared to the other four methods.

Fig. 10 presents the rewards obtained by the consumer device using different strategies when 20 tasks and five edge servers are connected to the consumer device. Fig. 10(a) shows the rewards of different task-offloading strategies when 20 tasks are assigned. Our method yields the highest and most stable rewards, followed by PPO and DDPG, while PPO_KLP demonstrates the least effective performance. Fig. 10(b) illustrates the rewards obtained by different task-offloading strategies when 20 tasks are randomly generated.



(a) Assign (b) Random

Fig. 10. Reward analysis (Computing resources: no local computing resources, tasks: 20, customer devices: 1, central cloud: 1, edge servers: 5).

Our method achieves the highest and most stable reward, followed by SAC, and PPO_KLP demonstrates the least effective performance. This is because PPO_KLP is not suitable for this scenario, causing performance to be severely affected by the initial parameters. These two figures indicate that our method is more stable and achieves the highest reward compared to the other four methods.

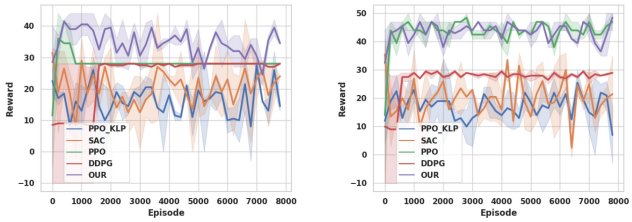
In summary, consumer devices lack computational resources, while edge servers and cloud servers possess computational resources. The computational offloading strategy derived by our method outperforms the four baseline methods by achieving higher and more stable rewards.

3) Analysis of Reward for Different Offloading Strategies in Scenario 3: Fig. 11 to 14 analyze the rewards obtained by customer devices using different methods for deriving task-offloading strategies in Scenario 3.

Fig. 11 illustrates the rewards obtained by the consumer device using different strategies when ten tasks and three edge servers are connected to the consumer device. Fig. 11(a) shows the rewards of different task-offloading strategies for the 10 assigned tasks. From this figure, it can be seen that our method achieves higher and more stable rewards, followed by PPO and DDPG, while PPO_KLP performs the worst. Fig. 11(b) presents the rewards of different task-offloading strategies for the randomly generated 10 tasks. It can be seen that our method and PPO produce the highest rewards, followed by DDPG, while PPO_KLP performs the worst. These two figures demonstrate that our method outperforms the other four stability and reward acquisition methods.

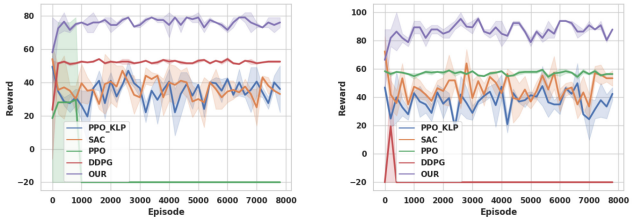
Fig. 12 shows the rewards obtained by the consumer device using different strategies when 20 tasks and three edge servers are connected to the consumer device. Fig. 12(a) illustrates the rewards of different task-offloading strategies for the 20 assigned tasks. Our method achieves the highest and fastest converging rewards, followed by DDPG, while PPO performs the worst. Fig. 12(b) demonstrates the rewards of different task-offloading strategies for the randomly generated 20 tasks. It can be seen that our method yields the highest rewards, followed by PPO, while DDPG performs the worst. These two figures indicate that our method is more stable and acquires higher rewards than the other four methods.

Fig. 13 displays the rewards obtained by the consumer device using different strategies when 10 tasks and five edge servers are connected to the consumer device. Fig. 13(a) shows the rewards of different task-offloading strategies for the 10



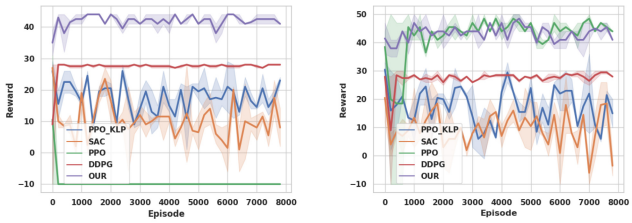
(a) Assign (b) Random

Fig. 11. Reward analysis (Computing resources: no computing resources on individual edge servers, tasks: 10, customer devices: 1, central cloud: 1, edge servers: 3).



(a) Assign (b) Random

Fig. 12. Reward analysis (Computing resources: no computing resources on individual edge servers, number of tasks: 20, customer devices: 1, central cloud: 1, edge servers: 3).

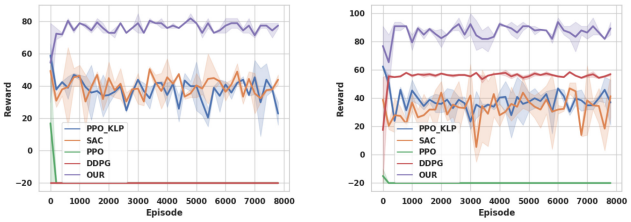


(a) Assign (b) Random

Fig. 13. Reward analysis (Computing resources: no computing resources on individual edge servers, number of tasks: 10, customer devices: 1, central cloud: 1, edge servers: 5).

assigned tasks. Our method achieves the highest and most stable rewards, followed by DDPG, while SAC and PPO_KLP perform similarly and PPO performs the worst. Fig. 13(b) presents the rewards of different task-offloading strategies for the 10 randomly generated tasks. Our method achieves the most stable rewards, followed by DDPG, while SAC and PPO_KLP perform similarly. These two figures demonstrate that our method is more stable and acquires higher rewards than the other four methods.

Fig. 14 displays the rewards obtained by the consumer device using different strategies when 20 tasks and five edge servers are connected to the consumer device. Fig. 14(a) illustrates the rewards of different task-offloading strategies for the 20 assigned tasks. Our method achieves the highest and most stable rewards, followed by SAC and PPO_KLP, with DDPG performing slightly worse and PPO and DDPG performing the worst. Fig. 14(b) shows the rewards of different task-offloading strategies for the randomly generated 20 tasks. It can be seen that our method yields the highest rewards, followed by DDPG, while PPO performs the worst. This is because PPO_KLP is not suitable for this scenario,



(a) Assign (b) Random

Fig. 14. Reward analysis (Computing resources: no computing resources on individual edge servers, number of tasks: 20, customer devices: 1, central cloud: 1, edge servers: 5).

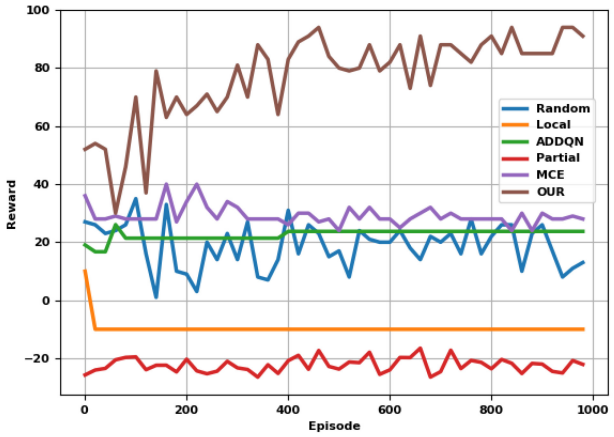


Fig. 15. Reward analysis in the same environment settings.

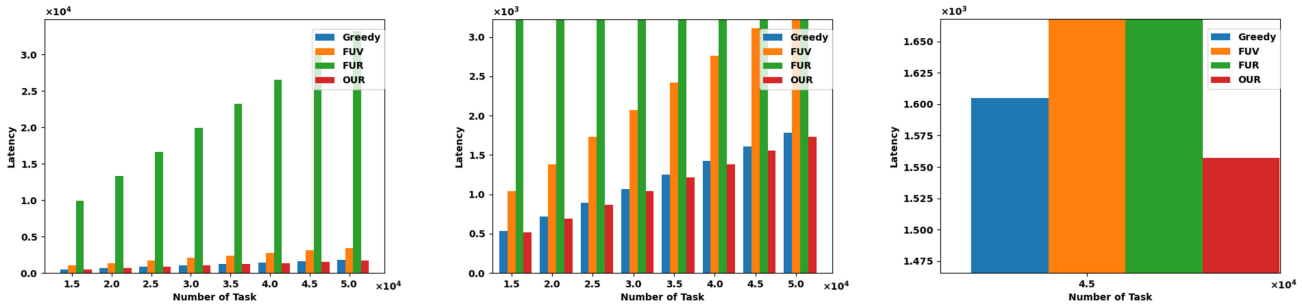
which causes performance to be severely affected by the initial parameters. These two figures indicate that our method is more stable and acquires higher rewards than the other four methods. From this, it can be seen that similar to the previous two scenarios, our method exhibits higher rewards in this scenario than the four baseline methods. Additionally, there is less fluctuation in rewards during the training process, indicating a more stable performance relative to the baselines.

In conclusion, analyzing the rewards obtained by the customer devices in the three extreme scenarios, it is evident that our method achieves the highest and most stable rewards.

C. Reward Analysis and Latency Analysis in the Different Environment Setting

Fig. 15 illustrates the evolution of rewards with increasing episodes for six methods (Random, Local, ADDQN, Partial, MCE, and OUR). 'Random' indicates that the offload strategy is selected at random. 'Local' means that all tasks are completed locally. This figure shows that the cumulative reward obtained by the six offloading strategies gradually converges with the increase in episodes. OUR achieves the highest cumulative reward, significantly outperforming the other five offloading strategies. This validates the effectiveness of our method.

Fig. 16 shows the variation of task-offloading latency for four strategies (Greedy, FUR, FUV, and OUR) as the task quantity increases. Due to significant differences in latency among the four offloading strategies, we present experimental results for the original, scaled, and locally scaled graphs.



(a) original graph (b) scaled graph (c) locally scaled graph

Fig. 16. Latency analysis.

TABLE III
TASK COMPLETION STATUS

Methods \ Type	Timeout	Timely	Discard	Reward
PPO_KLP [36]	10.0%	47.5%	42.5%	31.00
PPO [33]	10.0%	40.0%	50.0%	18.00
DDPG [39]	20.0%	30.0%	50.0%	16.00
SAC [11]	7.5%	40.0%	52.5%	25.50
OUR	5.0%	77.5%	17.5%	72.00

TABLE IV
COMPLETION TIME ESTIMATION

Tasks \ Methods	Time	Kmeans	Optics	Mean Shift	DBSCAN
Simple task 1	**5.70**	5.13	6.50	6.00	**5.57**
Tasks \ Methods	**Time**	**Analog**	**Mean**	**PERT**	**OUR**
Complex task 2	**15.00**	14.00	16.75	22.39	**14.14**
Complex task 3	**17.00**	14.00	16.75	25.20	**17.27**
Complex task 4	**5.00**	14.00	16.75	8.40	**4.06**

From Fig. 16(a), it can be observed that the latency for four offloading strategies increases with increased task quantity. Fig. 16(b) reveals that among the four offloading strategies, OUR performs the best, followed by Greedy, and FUR exhibits the maximum latency. Taking the number of tasks as 45,000, compared with FUR, the latency of OUR is reduced by 28,327.50, and compared with Greedy, the latency of OUR is reduced by 47.64. Thus, it can verify the validity of OUR.

D. Analysis of Task Completion Status and Completion Time Estimation

Table III displays the status of task completion and the average rewards for five computational offloading strategies in Scenario 1 after 100 training epochs. These strategies are evaluated based on 20 randomly generated tasks. 'Timely' indicates tasks are completed within the specified time threshold, ensuring a satisfactory user experience. 'Timeout' indicates that task completion times slightly exceed the time threshold, causing some impact on the user experience but not severely. 'Discard' indicates that consumer devices have been abandoned and do not process tasks. The table shows that our derived computation offloading strategy achieves the highest task completion rate and the lowest task discard ratio, a significant improvement compared to the four baseline methods. This performance can be attributed to our task completion time estimation method. The PPO_KLP strategy has the second highest task completion rate but a significantly higher task discard ratio of 42.5%. Compared to the PPO_KLP method, the task discard ratio of our method is reduced by 25%.

Table IV shows the estimated completion time for the new task. 'Time' indicates the official completion time of the new task. It can be seen from the table that compared with Kmeans, Optics, and Mean Shift, DBSCAN has the highest estimation accuracy for simple tasks, and the error is only 0.13 compared with the official completion time. For complex tasks, compared with the three benchmark methods (Analog estimation, Mean estimation, PERT estimation), the proposed method has the highest estimation accuracy, and the minimum mean error is only 0.69 ((15-14.14+17.27-17+5-4.06)/4=0.69). In conclusion, the validity of our proposed completion time estimation method can be verified.

VI. CONCLUSION

Researches on computation offloading strategies for time-sensitive and compute-intensive tasks have become crucial to improving the performance of consumer devices. By considering three factors (network conditions, computational demands, and task completion times) and using reinforcement learning algorithms, we propose a 'Computation Cost-driven Offloading Strategy based on Reinforcement Learning for Consumer Devices' to optimize resource utilization, reduce latency, and meet task requirements. Detailed experimental results validate the effectiveness of the proposed method. In future research, we will strive to refine further computation offloading strategies for customer devices to meet the evolving demands of edge computing and facilitate its widespread adoption across various industries.

REFERENCES

[1] F. Yu and H. Yan, "An efficient hot-cold data separation garbage collection algorithm based on logical interval in NAND flash-based consumer electronics," *IEEE Trans. Consum. Electron.*, vol. 69, no. 3, pp. 431–440, Aug. 2023.

[2] H. Qiu, K. Zhu, N. C. Luong, C. Yi, D. Niyato, and D. I. Kim, "Applications of auction and mechanism design in edge computing: A survey," *IEEE Trans. Cogn. Commun. Netw.*, vol. 8, no. 2, pp. 1034–1058, Jun. 2022.

[3] S. Saraswat, A. Gupta, H. P. Gupta, and T. Dutta, "An incremental learning based gesture recognition system for consumer devices using Edge-Fog computing," *IEEE Trans. Consum. Electron.*, vol. 66, no. 1, pp. 51–60, Feb. 2020.

[4] Q. Ma, H. Tan, and T. Zhou, "Mutual authentication scheme for smart devices in IoT-enabled smart home systems," *Comput. Stand. Interf.*, vol. 86, Aug. 2023, Art. no. 103743.

[5] M. Holko et al., "Wearable fitness tracker use in federally qualified health center patients: Strategies to improve the health of all of us using digital health devices," *NPJ Digit. Med.*, vol. 5, no. 1, pp. 1–7, 2022.

[6] S. R. Brause and G. Blank, "There are some things that I would never ask Alexa'–privacy work, contextual integrity, and smart speaker assistants," *Inf., Commun. Soc.*, 2023, submitted for publication.

[7] H. Jeong, T. Lee, and Y. I. Eom, "WebRTC-based resource offloading in smart home environments," in *Proc. IEEE Int. Conf. Consum. Electron.*, Las Vegas, NV, USA, 2022, pp. 1–6.

[8] F. Tian, Y. Yu, D. Li, J. Cui, and Y. Dong, "QoE optimization for traffic offloading from LTE to Wi-Fi," in *Proc. 8th Global Conf. Consum. Electron.*, Osaka, Japan, 2019, pp. 115–116.

[9] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.

[10] M. Hosseinzadeh, A. Wachal, H. Khamfroush, and D. E. Lucani, "QoS-aware priority-based task offloading for deep learning services at the edge," in *Proc. 19th Annu. Consum. Commun. Netw. Conf.*, Las Vegas, NV, USA, 2022, pp. 319–325.

[11] E. Mustafa et al., "Reinforcement learning for intelligent online computation offloading in wireless powered edge networks," *Clust. Comput.*, vol. 26, no. 2, pp. 1053–1062, 2023.

[12] W. Feng, S. Yang, Y. Gao, N. Zhang, R. Ning, and S. Lin, "Reverse offloading for latency minimization in vehicular edge computing," in *Proc. IEEE Int. Conf. Commun.*, Chengdu, China, 2021, pp. 1–6.

[13] S. Zhou, A. Ali, A. Al-Fuqaha, M. Omar, and L. Feng, "Robust risk-sensitive task offloading for edge-enabled industrial Internet of Things," *IEEE Trans. Consum. Electron.*, vol. 22, no. 1, pp. 500–514, Oct. 2023.

[14] C. Liu and K. Liu, "Toward reliable DNN-based task partitioning and offloading in vehicular edge computing," *IEEE Trans. Consum. Electron.*, early access, May 29, 2023, doi: 10.1109/TCE.2023.3280484.

[15] U. Saleem, Y. Liu, S. Jangsher, X. Tao, and Y. Li, "Latency minimization for D2D-enabled partial computation offloading in mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 4, pp. 4472–4486, Apr. 2020.

[16] M. Tang, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.

[17] J. Zhao, L. He, D. Zhang, and X. Gao, "A TP-DDPG algorithm based on cache assistance for task offloading in urban rail transit," *IEEE Trans. Veh. Technol.*, vol. 72, no. 8, pp. 10671–10681, Aug. 2023.

[18] C. Swain, M. N. Sahoo, A. Satpathy, K. Muhammad, S. Bakshi, and J. J. P. C. Rodrigues, "A-DAFTO: Artificial cap deferred acceptance based fair task offloading in complex IoT-Fog networks," *IEEE Trans. Consum. Electron.*, early access, Mar. 29, 2023, doi: 10.1109/TCE.2023.3262995.

[19] Z. Xu, Y. Zhang, X. Qiao, H. Cao, and L. Yang, "Energy-efficient offloading and resource allocation for multi-access edge computing," in *Proc. IEEE Int. Conf. Consum. Electron. Taiwan*, Taiwan, China, 2019, pp. 1–2.

[20] Z. Wang and Q. Zhu, "Partial task offloading strategy based on deep reinforcement learning," in *Proc. 6th Int. Conf. Comput. Commun.*, Chengdu, China, 2020, pp. 1516–1521.

[21] A. Khune and S. Pasricha, "Mobile network-aware middleware framework for cloud offloading: Using reinforcement learning to make reward-based decisions in smartphone applications," *IEEE Consum. Electron. Mag.*, vol. 8, no. 1, pp. 42–48, Jan. 2019.

[22] H. Zhou, T. Wu, X. Chen, S. He, D. Guo, and J. Wu, "Reverse auction-based computation offloading and resource allocation in mobile cloud-edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 10, pp. 6144–6159, Oct. 2023.

[23] N. Chen, S. Zhang, Z. Qian, J. Wu, and S. Lu, "When learning joins edge: Real-time proportional computation offloading via deep reinforcement learning," in *Proc. 25th Int. Conf. Parallel Distrib. Syst.*, Tianjin, China, 2019, pp. 414–421.

[24] G. Fang, R. Zhou, X. Li, and Z. Li, "A cross-edge offloading framework for green MEC systems," *IEEE Trans. Consum. Electron.*, early access, Sep. 27, 2023, doi: 10.1109/TCE.2023.3319816.

[25] G. Saranya and E. Sasikala, "Offloading methodologies for energy consumption in mobile edge computing," in *Proc. 2nd Int. Conf. Smart Electron. Commun.*, 2021, pp. 832–838.

[26] X. Cao, F. Wang, J. Xu, R. Zhang, and S. Cui, "Joint computation and communication cooperation for energy-efficient mobile edge computing," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4188–4200, Jun. 2019.

[27] J. Zhou, X. Zhang, W. Wang, and Y. Zhang, "Energy-efficient collaborative task offloading in D2D-assisted mobile edge computing networks," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Marrakesh, Morocco, 2019, pp. 1–6.

[28] H. Huang, Q. Ye, and H. Du, "Reinforcement learning based offloading for realtime applications in mobile edge computing," in *Proc. IEEE Int. Conf. Commun.*, Dublin, Ireland, 2020, pp. 1–6.

[29] M. Kumar, G. K. Walia, H. Shingare, S. Singh, and S. S. Gill, "Ai-based sustainable and intelligent offloading framework for IIoT in collaborative cloud-Fog environments," *IEEE Trans. Consum. Electron.*, early access, Sep. 29, 2023, doi: 10.1109/TCE.2023.3320673.

[30] X. Yuan et al., "A DQN-based frame aggregation and task offloading approach for edge-enabled IOMT," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 3, pp. 1339–1351, Jun. 2023.

[31] S. E. Mahmoodi, R. Uma, and K. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 301–313, Jun. 2016.

[32] J. Wang, J. Hu, G. Min, W. Zhan, A. Y. Zomaya, and N. Georgalas, "Dependent task offloading for edge computing based on deep reinforcement learning," *IEEE Trans. Comput.*, vol. 71, no. 10, pp. 2449–2461, Oct. 2022.

[33] V. D. Tuong, T. P. Truong, T.-V. Nguyen, W. Noh, and S. Cho, "Partial computation offloading in NOMA-assisted mobile-edge computing systems using deep reinforcement learning," *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13196–13208, Sep. 2021.

[34] N. Cheng et al., "Space/aerial-assisted computing offloading for IoT applications: A learning-based approach," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1117–1129, May 2019.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[36] H. Xu, Z. Cai, R. Li, and W. Li, "Efficient citycam-to-edge cooperative learning for vehicle counting in ITS," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 9, pp. 16600–16611, Sep. 2022.

[37] W. Li, X. Cheng, Z. Tian, S. Wang, R. Bie, and J. Yu, "Truthful auction analysis and design in multiunit heterogenous spectrum markets with reserve prices," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 1, pp. 157–170, Mar. 2021.

[38] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[39] T. Haarnoja et al., "Soft actor-critic algorithms and applications," 2018, *arXiv:1812.05905*.