Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Optimizing task offloading and resource allocation in edge-cloud networks: a DRL approach

Ihsan Ullah[1], Hyun-Kyo Lim[2], Yeong-Jun Seok[2] and Youn-Hee Han[2*]

Abstract

Edge-cloud computing is an emerging approach in which tasks are offloaded from mobile devices to edge or cloud servers. However, Task offloading may result in increased energy consumption and delays, and the decision to offload the task is dependent on various factors such as time-varying radio channels, available computation resources, and the location of devices. As edge-cloud computing is a dynamic and resource-constrained environment, making optimal offloading decisions is a challenging task. This paper aims to optimize offloading and resource allocation to minimize delay and meet computation and communication needs in edge-cloud computing. The problem of optimizing task offloading in the edge-cloud computing environment is a multi-objective problem, for which we employ deep reinforcement learning to find the optimal solution. To accomplish this, we formulate the problem as a Markov decision process and use a Double Deep Q-Network (DDQN) algorithm. Our DDQN-edge-cloud (DDQNEC) scheme dynamically makes offloading decisions by analyzing resource utilization, task constraints, and the current status of the edge-cloud network. Simulation results demonstrate that DDQNEC outperforms heuristic approaches in terms of resource utilization, task offloading, and task rejection.

Keywords Edge-cloud computing, Task offloading, Resource allocation, Deep Reinforcement learning, Markov decision process (MDP)

Introduction

The advent of IoT and 5G has enabled the development of new applications in areas such as surveillance, augmented/virtual reality, and facial recognition, which heavily depend on both computational resources and data storage for optimal performance. IoT and mobile devices have limited resources so it is difficult for them to support such intelligent, delay-sensitive applications [1]. Hence, edge computing can help these devices by offloading computation-intensive tasks to the cloud. The cloud can cause delays in communication and data transfer as a result of network congestion and high usage, this can limit their use in real-time applications such as autonomous driving, advanced navigation, augmented reality (AR), and virtual reality (VR). Overall, edge-cloud computing and 5G can work together to enable new types of applications and services that require low-latency, real-time processing. The use of edge computing can reduce latency, enhance performance, and reduce costs associated with cloud computing. The edge servers can process computation-intensive tasks instead of sending them to the cloud [2].

Edge-cloud computing is a distributed computing model that utilizes both cloud computing resources and edge devices, such as servers and IoT devices, to process and transmit data. This model aims to improve

*Correspondence:
Youn-Hee Han
yhhan@koreatech.ac.kr
[1] Advanced Technology Research Center, Korea University of Technology and Education, Cheonan, Republic of Korea
[2] Future Convergence Engineering, Department of Computer Science and Engineering, Korea University of Technology and Education, Cheonan, Republic of Korea

processing speed decision-making and reduce latency and bandwidth usage by bringing the cloud's processing power closer to the edge where data is generated and collected. Edge and cloud computing can coexist and work together to provide the necessary resources for task execution. The integration of AI with edge-cloud computing allows for the deployment of machine learning algorithms on edge devices, providing a high level of intelligence to the edge-cloud network and enabling it to understand its environment [3, 4]. Before offloading tasks to the cloud or edge, it is essential to carefully consider the requirements and available resources. However, edge-cloud computing is a dynamic and resource-constrained environment. Hence making an optimal decision for task offloading based on the available resource is a critical issue.

Task offloading is the process of transferring a task or workload from a local device to a remote device, such as a server or cloud resource, to improve the local device's performance and efficiency. Offloading tasks can result in increased delay and energy consumption while edge servers may have limited computational capacity, which can lead to increased computing latency. It is important to consider these trade-offs before making a decision. 5G networks with high densities may also experience higher transmission delays. Cooperative task offloading is a technique used in edge-cloud networks to improve the performance of distributed systems. In the distributed approach, tasks are split among devices in the network, such as edge devices and cloud servers, to optimize resource usage and reduce the workload on individual devices. In an edge-cloud computing environment, it can be challenging to determine the optimal location for task offloading, as there are many factors to consider, including the computational capacity of edge servers, the transmission delays of networks, and the diverse requirements of end devices. Numerous research has been conducted on the topic of computation offloading in edge-cloud networks [5–9]. However, due to the diverse requirements of end devices and the limited information available about wireless channels, bandwidth, and computing resources in edge-cloud networks, it is challenging to design an optimal offloading strategy.

Deep Reinforcement Learning is a subset of machine learning that combines reinforcement learning and deep learning techniques to handle high-dimensional state-action spaces and accelerate training for complex decision-making tasks. Many successful applications of DRL have been demonstrated in a wide range of fields, including gaming [7], robotics [8], and networking. The application of DRL to edge-cloud-assisted networks can optimize system performance by perceiving users' mobility [10, 11]. Task offloading problems have been addressed with DRL in [9, 12, 13], and [14]. It can be able to find the best solutions to the optimization problems of time-varying and dynamic network environments. In the context of task offloading at the edge, Dueling-DQN could potentially be used to learn the optimal decision-making policy for offloading tasks to different locations in a dynamic and resource-constrained edge-cloud environment. The DDQN architecture separates the estimator into two streams, value, and advantage, and then aggregates them to make the final estimation of the Q-value. This allows for better generalization of the Q-value estimation by decoupling the estimation of the value of a state from the estimation of the advantage of taking a specific action in that state. This could involve training the DDQN to learn the optimal trade-offs between different factors, such as the task requirements, available resources, latencies, and costs, to make informed decisions about the best location for task offloading.

In this research, we propose an advanced edge-cloud computing scheme that leverages the power of Double Deep Q-Network (DDQN) to optimize the offloading of computations and the allocation of resources for the offloaded tasks. Our proposed scheme is an extension of our previous work that employed DQN [9], but it goes one step further by utilizing the DDQN algorithm to improve the decision-making process and achieve a more efficient and optimized solution for the edge-cloud computing task offloading and resource allocation problem. The goal of the proposed scheme is to minimize energy consumption while satisfying the computation and communication requirements of the offloading tasks at edge-cloud computing. DDQN enables efficient and effective offloading decisions in dynamic and resource-constrained environments, such as edge-cloud, by adapting to changes and learning from past experiences. In terms of training, DDQN separates the Q-value estimator into two streams, value and advantage, and then aggregates which improve the generalization. We present the task offloading and resource allocation problem as an MDP and use the DDQN algorithm to identify the optimal policy. Our simulation results show that the proposed method outperforms heuristic schemes and DQNEC in terms of resource utilization, maximum task offloading, and task rejection. The main contributions of our work are:

- To optimize resource allocation for compute-intensive and delay-sensitive tasks in the edge cloud computing environment, we developed a scheme based on the double DQN. Our scheme determines the best actions to take in the current system state, ultimately improving the overall performance of the edge cloud

- Considering the dynamics and unpredictability of edge device environments, we modeled the task offloading problem as a Markov Decision Process (MDP) and applied Double-DQN to maximize the long-term cumulative discounted reward. Our optimization objective was to maximize resource utilization, minimize task rejection, and minimize the round-trip time to complete the task within the deadline
- By using the proposed Double-DQNEC scheme, we can reduce idle time and balance the workload of the edge-cloud computing system. The simulation results show that the DDQN-based algorithm significantly outperforms the heuristic algorithm.

The rest of the paper is organized as follows: In Sect. 2 the work-related are reviewed, especially relative to task offloading and resources allocation in the edge-cloud computing environment; Sect. 3 provides the system model and formulation of the proposed DDQNEC scheme in detail; in Sect. 4 the simulation result and comparison are discussed that validate our approach; finally, the conclusion is made in Sect. 5.

Related work

DRL learns optimal policies and makes quick decisions by interacting with a time-varying environment, which is suitable for dynamic MEC systems. There are mainly two research streams, including value-based and policy-based methods.

Value-based DRL methods

Huang et al. introduced DROO, an online offloading algorithm for wireless-powered mobile edge computing (MEC) networks, in their work published in [12]. To solve the optimization problem, the algorithm splits it into two sub-problems, namely resource allocation and offloading decision. Through evaluation and simulation, the authors observed that DROO surpassed existing methods in terms of energy efficiency, task completion time, and network congestion. This makes DROO a promising solution for enhancing the performance of wireless-powered MEC networks. Li et al. [13] applied the deep Q-network (DQN) to jointly handle computation offloading and resource allocation in multiuser MEC, minimizing the sum cost of delay and energy consumption. In [14], an improved resource allocation policy was proposed for IoT edge computing based on the deep Q-network (DQN) approach. This policy aims to improve the efficiency of resource utilization and reduce task completion delays in the system. The DQN algorithm allows the system to learn from the experience and make better decisions for resource allocation in real-time.

Chen et al. [15] proposed the Deep-SARL algorithm to address the computation offloading problem in a mobile edge computing system by formulating it as a Markov decision process. The algorithm uses deep reinforcement learning with a self-attention mechanism to learn an optimal offloading policy that maximizes the long-term utility performance. The Deep-SARL algorithm was evaluated through simulations, and the results showed that it outperformed existing offloading methods in terms of average reward and energy efficiency. The proposed algorithm can help improve the performance of mobile edge computing systems, especially for applications that require real-time processing and low latency. Lu et al. [16] proposed a DQN algorithm for large-scale heterogeneous MEC, achieving good performance without prior knowledge of environment statistics. However, value-based methods are limited in handling continuous action space, such as in the dynamic JCORA problem. In [10], a DQN-based approach was introduced to solve the challenging problem of jointly optimizing task offloading and bandwidth allocation in MEC networks. The proposed solution effectively balances trade-offs between quality of service, energy efficiency, and network congestion.

In [11], a comprehensive review was provided of the current state-of-the-art techniques and computational resources used for partitioning and offloading in MEC networks. The challenges and opportunities of various approaches were discussed and their pros and cons were highlighted. In [17], a task offloading scheme based on DQN was proposed to select the optimal edge server and transmission mode for maximizing task offloading utility. The proposed scheme achieved high performance in terms of task completion time, energy consumption, and network congestion. Liu et al. [18] proposed a novel approach for allocating resources in a mobile edge computing system, where vehicles were used as edge devices to provide computational services to nearby users. The proposed algorithm was based on deep reinforcement learning (DRL) and aimed to maximize the overall system utility by efficiently allocating resources among the vehicles. In [19], an intelligent DRL-based resource allocation scheme for wireless networks was proposed to minimize service time and balance resources.

In [20], a migration algorithm was proposed to optimize task migration using a multi-agent reinforcement learning approach. The algorithm leveraged the collective intelligence of multiple agents to make optimal migration decisions, taking into account various factors such as network conditions, task requirements, and system resources.

In [21], the authors introduced a DQN-based computation offloading scheme for mobile edge computing (MEC) networks. The proposed scheme aimed to minimize the

long-term cost of the system while ensuring a satisfactory level of quality of service for the end-users. By leveraging DQN, the scheme was able to learn an optimal offloading decision policy in a data-driven manner, which could adapt to the dynamic and uncertain network conditions in MEC environments. [22] introduced a DDQN-based backscatter-aided hybrid data offloading scheme that significantly improved energy efficiency while maintaining transmission rate and reliability. In [23], an approach was proposed for making offloading decisions in a mobile edge computing (MEC) environment by jointly optimizing CPU frequencies and transmit powers. This approach aimed to minimize the energy consumption of mobile devices while maintaining the quality of service for offloaded tasks. The objective of [24] was to minimize the costs associated with energy consumption and computation delay in mobile edge computing networks, which are critical factors for their performance. To achieve this, the authors presented the RL-SARSA algorithm as a solution for resource management and optimal offloading decisions. In [25] the authors proposed DDQNL-IST, a game-learning algorithm that combined DDQN and distributed LSTM with intermediate state transition to lower the complexity of offloading computation under time-varying conditions. DDQNL-IST used distributed LSTM and double-Q learning to improve processing and predicting time intervals and delays. The devices could exploit information asymmetry to obtain a better game learning outcome.

Policy-based DRLs methods

Lu et al. [26] presented the double-dueling deterministic policy gradient (D3PG) algorithm for edge computing, capable of optimizing three critical performance metrics: service latency, energy consumption, and task success rate. The simultaneous optimization of these metrics can lead to efficient resource allocation in dynamic edge computing environments. The D3PG algorithm can improve the overall performance of edge computing systems by reducing service latency, decreasing energy consumption, and improving the task success rate.

Zhang et al. [27] proposed two DRL algorithms for dynamic computation offloading in edge computing to minimize service latency. The hybrid-AC algorithm optimized resource allocation in single-device scenarios using a decision-based actor-critic approach. The md-Hybrid-AC algorithm achieved efficient resource allocation in multi-device scenarios using the multidevice actor-critic approach. These algorithms are significant contributions to the field of edge computing, as they reduce service latency and energy consumption, and improve the task success rate. They have the potential to enhance the performance of various edge computing applications.

Chen and Wang [28] proposed a decentralized DDPG-based mechanism called JCORA. It addresses the challenge of resource allocation in a decentralized MEC system, where multiple users compete for limited computing resources. By using DDPG agents, the JCORA mechanism can learn the optimal computation offloading policies for each user without relying on centralized control or communication. This approach offers several advantages, including improved scalability and reduced overhead [29], presented a trustworthy DRL strategy for computation offloading in IoT edge networks, designed to handle selfish or forgery attacks in intelligent vehicle networks. This strategy employs an intelligent system model and a DPGAQ scheme to anticipate untrusted vehicle attacks during IoT offloading. It evaluates device trustworthiness for vehicle networks in mobile edge networks and uses the intelligent trusted system model and DPGAQ scheme to prevent malicious vehicle attacks. The strategy also uses a quantization algorithm to simplify offloading decisions in high-dimensional action spaces. In [30], a new policy-based multi-agent deep reinforcement learning algorithm known as post-decision state (GPDS) is introduced to address malicious interference in wireless networks. By assessing the communication quality, spectrum availability, and jammer's strategy from the post-decision state, the mobile users can optimize their transmission power and frequency to increase their SINR and channel throughput.

Liu and Liao [31] designed actor–critic-based approach to optimize resource allocation and offloading decisions. In their approach, the authors used a hybrid policy that can handle both discrete and continuous actions. This approach allows for efficient resource allocation and computation offloading decisions in edge computing systems. The DAC algorithm is capable of dynamically adjusting the decision-making process according to the system's needs, ensuring that optimal resource allocation is achieved at all times. Moreover, the DAC approach is scalable and can be applied to various edge computing scenarios, making it a valuable tool for addressing the challenges of resource allocation and computation offloading in edge environments.

Qiu et al. [32] developed a distributed and collective deep reinforcement learning (DRL) algorithm, called DC-DRL, to address the challenges of computation offloading in resource-intensive and deadline-sensitive applications. The algorithm is capable of optimizing resource allocation and task scheduling in a distributed manner, enabling efficient and scalable processing of complex tasks. By leveraging the collective intelligence of multiple agents, DC-DRL can achieve higher performance and

better task completion rates compared to traditional centralized approaches.

In [33], a DDPG-based is algorithm is proposed for collaborative computation offloading in heterogeneous edge computing. The offloading algorithm they propose is designed to work across all three edge networks, despite their heterogeneity. By doing so, they aim to improve the overall efficiency and performance of the network, by dynamically routing computation tasks to the most appropriate network based on factors such as network load, latency, and available resources.

In [34] a joint multi-task offloading and resource allocation scheme is suggested in satellite IoT. It involved modeling tasks with dependencies as DAGs and using A-PPO to optimize offloading strategy. The proposed approach aimed to improve offloading efficiency and could have led to better performance and faster task completion.

In [35] A DDPG-based scheme was proposed that considered energy consumption and task completion for a multiuser scenario, utilizing simultaneous wireless information and power transfer technology. It formulated an optimization problem that jointly optimized task offloading ratio, uplink channel bandwidth, power split ratio, and computing resource allocation. The proposed algorithm achieved optimal energy consumption and delay and utilized an inverting gradient updating-based dual actor-critic neural network design to improve the convergence and stability of the training process.

System model

In this section, we present the system model and the problem formulation for task offloading and resource allocation. The network model with the edge-cloud system of the DDQNEC scheme is shown in Fig. 1. Our scheme involves connecting end devices such as sensors, mobile devices, and IoT devices to base stations through wireless links. The edge computing system is connected to the core cloud via the backbone network, allowing for the offloading of tasks and the utilization of available resources in the public cloud. This batch processing approach waits for a predefined number (N) of task requests before determining the optimal location

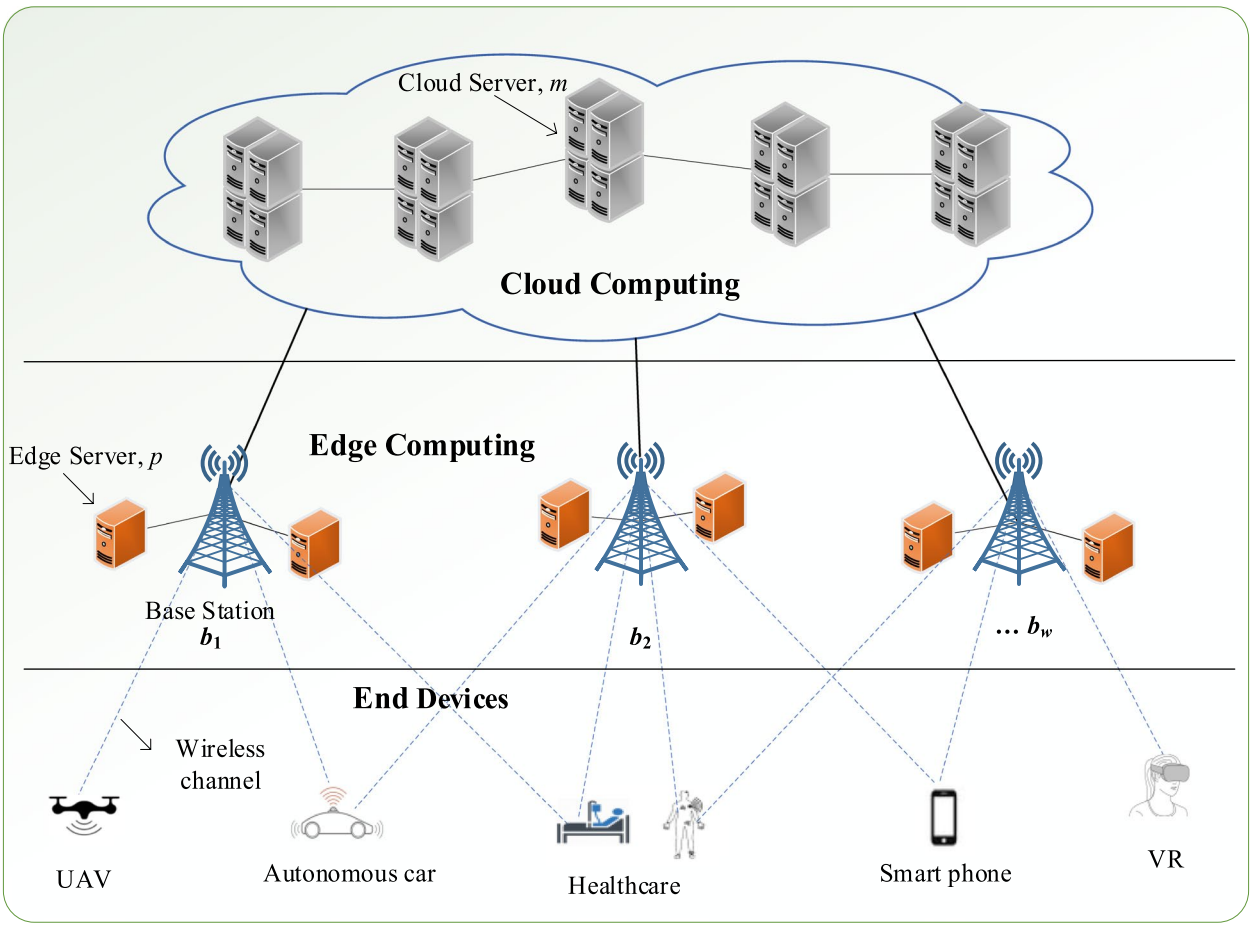


Fig. 1 The system model and structure of edge-cloud computing

for each task, whether it be the edge or the cloud, taking into consideration the availability of resources and the deadline. By evaluating a batch of task requests at a time, this approach allows for better resource utilization and decision-making. Both bandwidth and computing resources are considered when making offloading decisions, to optimize resource usage, minimize delay, and reduce energy consumption. In the following section, we provide a detailed description of the system model, including the task, communication, and computation offloading models. Table 1 provides a list of the notations used in our models.

Task model

A task t_n is represented as a tuple of four variables, $(\ddagger_n, \dagger_n, c_n, \tau_n), (1 \leq n \leq N)$ where $\ddagger_n$ is the input data size in bytes, $\dagger_n$ is the resultant data size, c_n is the required computational resource in CPU units, and τ_n is the task latency requirement. The value of x_n is either 0 or 1, representing a binary decision on whether to assign a task to the edge or the cloud.

$$x_n = \begin{cases} 0\, task\ \ t_n\, is\, executed\, at\, edge, \\ 1\, task\ \ t_n\, is\, executed\, at\, cloud, \end{cases} \tag{1}$$

Typically, multiple resources are required for offloading tasks; however, our scheme considers only CPU resources required for the task [36–38].

$$c_n = z_n \times \varsigma \tag{2}$$

where c_n represent the total CPU units required to process the task t_n, $\ddagger_n$ represents the total size of input data, while ς represents the computational resources required to process a single unit of data in bytes.

Wireless bandwidth model

To offload the task from the end device to the edge or cloud, the device must be connected to the nearest base station by a wireless channel. Let's $\mathcal{B}$ the set of all base stations $\mathcal{B} = \{b_1, b_2, \ldots, b_W\}$, and each base station b_w has a set of wireless channels that provides different data rates as $\beta_h^w \in \left\{\beta_1^w, \beta_2^w, \beta_3^w, \ldots, \beta_{H_w}^w\right\}$. Each channel serves different tasks and σ_h^w represents the remaining bandwidth of each channel as $\{\sigma_1^w, \sigma_2^w, \sigma_3^w, \ldots, \sigma_{H_w}^w\}$. Then at time step t, the bandwidth utilization $\mathcal{U}_W(t)$ of all the base stations can be formulated as

$$\mathcal{U}_W(t) = \frac{\sum_{w=1}^{W}(\sum_{h=1}^{H}\beta_h^w)}{B} \tag{3}$$

where B represents the bandwidth of all base stations

Table 1 Notation and description

Symbol	Definitions
$\mathcal{T}$	A set of tasks generated by edge devices
t_n	Each task generated by the end device n
z_n	The input data size of the task t_n
c_n	Computation resource size required for the task t_n
τ_n	Maximum tolerable latency of the task t_n
y_n	The resultant data of the task t_n
x_n	A binary variable to indicate whether the task t_n is assigned to edge or cloud, (0 indicates edge and 1 indicates cloud)
ς	CPU unit to process the one byte of data
$\mathcal{B}$	Set of base stations (BS),$\mathcal{B} = \{b_1, b_2, \ldots, b_W\}$
$\mathcal{U}_B(t)$	The bandwidth utilization of all BSs at time step t
H_w	Set of wireless channels related to the BS b_w
β_h^w	The bandwidth of each channel h b_w
σ_h^w	Remaining bandwidth of the BS b_w
$\mathcal{P}$	The set of computing servers at the edge
p	Computing server at the edge ($p \in \mathcal{P}$)
c_p	The available computing capacity of server p
$T_n^{proc_e}$	The processing time for the task t_n at the edge
$\mathcal{M}$	The set of computing servers in the cloud
m	Computing server at the edge ($m \in \mathcal{M}$)
$T_n^{proc_c}$	The processing time for the task t_n at the cloud server
c_m	T he computing capacity of a cloud server m
C_c	T he total computing capacity of cloud servers
$T_n^{proc_c}$	The total computation time for the task t_n at cloud
$\mathcal{U}_M(t)$	The computational resources utilization of cloud servers at time step t
$T_n^{trans_e}$	Transmission time for the task t_n data sent to the edge server
$T_n^{trans_c}$	Transmission time for the task t_n data to the cloud server
$T_n^{prop_e}$	The propagation time of the link between the nodes and edge servers
$T_n^{prop_c}$	The propagation delay of the link between the edge and cloud
rtt_n^e	The total round-trip time to the edge for a task t_n
rtt_n^c	The total round-trip time to the cloud for a task t_n

Computational model

i. *Edge computing:*

In our scheme, the set of edge servers is denoted as $\mathcal{P} = \{1, 2, 3 \ldots P\}$, and c_p denote the available computational capacity of edge server $p, (p \in \mathcal{P})$. The computation time $T_n^{Proc_e}$ for task t_n to compute at edge server p is given by

$$T_n^{proc_e} = \frac{c_n}{c_p} \tag{4}$$

The utilization of the computational resources of the edge server at time t is represented as

$$\mathcal{U}_P(t) = \frac{\sum_{p=1}^{P}(c_p(t))}{C_e} \tag{5}$$

where C_e denotes the total available computing capacity of all servers at the edge.

ii. *Cloud computing:*
The set of cloud servers is denoted as $\mathcal{M} = \{1, 2, 3 \ldots M\}$, and c_m denote the available computational capacity of edge server $m, (m \in \mathcal{M})$. The processing time $T_n^{proc_c}$ for task t_n to compute it at the cloud server m is given by

$$T_n^{proc_c} = \frac{c_n}{c_m} \tag{6}$$

The utilization of the computational resources of the cloud server at time t is represented as

$$\mathcal{U}_M(t) = \frac{\sum_{m=1}^{M}(c_m(t))}{C_c} \tag{7}$$

where C_c denotes the total available computing capacity of all servers in the cloud.

Delay model

In computation offloading, tasks are sent to an edge or cloud server for processing. The process involves three types of delays: transmission delay, propagation delay, and processing delay.

i. Transmission Time

For task t_n, data transmission is required in both directions: from the end device to the edge/cloud server with a data size of $\ddagger_n$, and from the edge/cloud server back to the end device with a resultant data size of $\dagger_n$.

Hence, a specific amount of bandwidth $\beta_h^w(edge)$ or $\beta(cloud)$ is required to fulfill the minimum latency τ_n of task t_n. Transmission time which needs to send data of task t_n to the edge $T_n^{trans_e}$ and cloud $T_n^{trans_c}$ can be formulated as

$$T_n^{trans_e} = \frac{z_n}{\beta_h^w} + \frac{y_n}{\beta_h^w} \tag{8}$$

$$T_n^{trans_c} = T_n^{trans_e} + \frac{z_n}{\beta} + \frac{y_n}{\beta} \tag{9}$$

ii. Propagation Time
In the given model, the propagation delay is assumed to be constant, with a value of $T_n^{prop_e} = 5ms$ for edge server and $T_n^{prop_c} = 50ms$ for the cloud server. This simplifying assumption is made for ease of calculation and analysis. The actual propagation delay may vary depending on the location of the resource.

iii. *Processing delay:*
Processing delay for the task t_n edge server $T_n^{proc_c}$ and cloud server $T_n^{proc_c}$ can be obtained from Eq. (4) and (6).
Therefore, the overall time for a task to be completed by an edge rtt_n^e or cloud rtt_n^c is the sum of the delay caused by data transmission, propagation, and processing which is represented as

$$rtt_n^e = T_n^{trans_e} + T_n^{prop_e} + T_n^{proc_e} \tag{10}$$

$$rtt_n^c = T_n^{trans_c} + T_n^{prop_c} + T_n^{proc_c} \tag{11}$$

The total resources cost CO_{total} can be obtained by adding the total utilization of bandwidth CO_W, edge server CPU CO_P, and cloud server CPU CO_M for total task offloading as follows:

$$\begin{aligned} CO_W &= W_W \bullet \mathcal{U}_W \\ CO_P &= W_P \bullet \mathcal{U}_P \\ CO_M &= W_M \bullet \mathcal{U}_M \end{aligned}$$

$$CO_{total} = CO_W + CO_P + CO_M \tag{12}$$

where each resource (bandwidth, edge, and cloud computational resources) has been assigned a cost weight, with $W_w = 1$ being assigned to bandwidth, $W_p = 5$ for edge resources and $W_M = 10$ for cloud computational resources. The agent learns to pick the cheapest resource for a task based on cost weights. It assigns tasks to the best location (edge or cloud) accordingly. If both resources are available, the agent assigns the task to the edge due to lower cost.

Formal problem formulation

The multi-objective problem solved in this paper is described formally as follows:

Optimization:

$$maximize\ (\mathcal{U}_W + \mathcal{U}_P + \mathcal{U}_M) \tag{13}$$

$$minimize\ (CO_W + CO_P + CO_M) \tag{14}$$

Equation (13) aims to achieve the optimization objective of maximizing resource utilization in both the edge and cloud, while minimizing the cost of task offloading specified in Eq. (14). For more details on resource utilization and cost, please refer to Eqs. (3), (5), (7), and (12).

Subject to the constraints:

$$\sum_{w=1}^{W}\sum_{h=1}^{H} ch_h^w \bullet \mu_h^w \leq B \tag{15}$$

$$\sum_{n=1}^{N} c_n \bullet (1 - x_n) \leq C_e \tag{16}$$

$$\sum_{n=1}^{N} c_n \bullet x_n \leq C_c \tag{17}$$

$$rtt_n^e \bullet (1 - x_n) + rtt_n^c \bullet x_n \leq \tau_n \tag{18}$$

DDQN-based task offloading and resource allocation

In this section, we introduce DQNEC, a proposed scheme that utilizes the DDQN algorithm to make optimal decisions and select the best location for task execution by analyzing the current state of the edge-cloud environment. It aims to improve resource utilization and balance the trade-off between delay and resource cost, to maximize the performance of edge-cloud computing systems. This is achieved by maximizing task offloading while minimizing delay and cost as defined in Eq. (12), (13), and (14). We formulate this multi-objective problem using a Markov Decision Process (MDP).

Markov decision process

A Markov decision process (MDP) models sequential decision-making problems where an agent makes decisions to maximize reward. It includes elements such as agent, state, action, policy, and reward. We formulate task offloading and resource optimization problems as an MDP to find the optimal policy π^*. The policy is a mapping of states to action probabilities, represented by $\pi(a|s)$ for all possible actions a for each state s. RL algorithms are often used to solve MDPs, as they allow an agent to learn the optimal behavior for a given MDP through trial and error. In the DDQN-based framework, the agent observes the state s_t by attracting to the edge-cloud environment and taking an action a_t as computing server selection via a deterministic policy and receives an immediate reward r_t. The agent uses the action-value function $Q(s_t, a_t)$ to update the agent policy. The goal of the agent is to maximize the long-term reward by finding an optimal resource allocation policy. In the following section, the state, action, and reward of the proposed scheme are explained in more detail.

State

The state s_t includes full information on the edge-cloud network. It includes the number of remaining tasks (N^t), I^t is from 1 to N (that is $I^t = t$), which specifies the task which should be currently determined by the agent, the total remaining computational capacity of edge servers and cloud servers ($C_c + C_e$), total remaining bandwidth at edge and cloud ($B_e + B_c$), the number of cloud-server (N^c), the remaining CPU of each server (α_m). In addition, information on edge such as the number of edge servers (N^e), and the remaining total CPU of the edge server (α_{p_w}) exists. CPU (U_m, U_p) and bandwidth information allocated to each cloud and edge is added. Finally, each task's information $t_1, t_2, \ldots, t_N$ is included. State $s_t \in \mathcal{S}$ can be defined as

$$\begin{gathered} s_t = \{N^t, I^t, C_c + C_e, B_c + B_e, \\ N^c, C_c, B_c, \alpha_1, \alpha_2, \ldots, \alpha_m, \ldots, \alpha_M, \mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_m, \ldots \mathcal{U}_M, \\ N^e, C_e, B_e, \alpha_1, \alpha_2, \ldots, \alpha_p, \ldots, \alpha_P, \mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_p, \ldots \mathcal{U}_P, \\ t_1, t_2, \ldots, t_N\} \end{gathered}$$

Action

In our model, the agent takes action by observing the current state of the environment. The goal of the agent is to make the optimal decision to maximize resource utilization and minimize the overall average service delay with the minimum rejection of tasks. Action $a_t \in \mathcal{A}$ at each time step t can be defined as the action to offload the t-th task ($1 \leq t \leq N$) and allocate the resources (Bandwidth and CPU) to the task for execution within the task deadline. Action a_t can be defined as

$$a_t = \{\eta, x_n\} \tag{19}$$

where η represents the computation server, and x_n selects the edge or cloud location for a task s_n, with η belonging to {1,2,...,P} (edge server) when $x_n = 0$, and

$\eta \in \{1, 2, \ldots, M\}$ (cloud server) $x_n = 1$. The agent will take actions based on the task offloading strategy in each time step and receive rewards from the environment in the following time step.

Reward

In RL, the agent's objective is to maximize the sum of rewards from good actions. Our reward function is designed to optimize resource utilization, minimize cost, and satisfy delay constraints. The reward r_t can be calculated by the total resource utilization $\rho(t)$ at time step t in Eq. (20), the total cost $\sigma(t)$ at time step t in Eq. (21) and delay constraint satisfaction for the task s_t at time step t in Eq. (22).

$$\rho(t) = \mathcal{U}_W(t) + \mathcal{U}_P(t) + \mathcal{U}_M(t) \tag{20}$$

$$\sigma(t) = CO_W(t) + CO_P(t) + CO_M(t) \tag{21}$$

$$r_t(s_t, a_t) = \frac{\rho(t)}{\sigma(t)} \left[\tau_t - (rtt_t^e \bullet (1 - x_t) + rtt_t^c \bullet x_t)\right] \tag{22}$$

DDQN Framework for task offloading

In our model, we used the DDQN algorithm for the learning process. The DDQN algorithm is an off-policy algorithm and is applied to environments with discrete action spaces. The learning process for DDQN is described in Algorithm 1 and also depicted in Fig. 2.

As shown in Fig. 2, the proposed learning process based on DDQN applies replay memory M, which can store a set of recent experience (s_i, a_i, r_i, s_{i+1}) which an agent gathers by interacting with the environment, and then uses for DDQN learning. In particular, the system records the experience for every step. During the network training, a mini-batch (size: b) is extracted from the replay memory M, and the Q network can learn from the previous experience. DDQN uses two neural networks, i) *the prediction network* $Q_\pi(s, a|\theta)$ as a function approximator to estimate the action-value function Eq. (15), where θ is the weight of the neural network, ii) the target network $\overline{Q}_\pi(s, a; \overline{\theta})$ to estimate the target value y_i. The target network has the same structure as the prediction network. However, its weights $\overline{\theta}$ are copied from θ every fixed number of iterations (K) instead of every training epoch. The following Eq. (23), (24), and (25) are the main equations for calculating the loss value.

$$q_i \approx Q_\pi(s_i, a_i|\theta) \tag{23}$$

$$y_t = r + \gamma \times \max_a \overline{Q}_\pi(s_{t+1}, a|\overline{\theta}) \tag{24}$$

$$loss = \left(y_t - Q_\pi(s, a)\right)^2 \tag{25}$$

DDQN updates the Q-function network's parameters, θ, using the loss value and stochastic gradient descent (SGD) with each mini-batch.

$$\theta = \theta - \alpha \Delta_\theta \sum_{i=1}^{b} \frac{\left(q_i - y_i\right)^2}{b} \tag{26}$$

where α is the learning rate.

Input: the full information of the edge-cloud network and the tasks to be offloaded.
Output: Selecting a server at the edge or cloud for offloading tasks, taking into consideration the constraints on resources and the requirements of the tasks.
Initialize two neural networks as Q-networks Q_π and $\overline{Q}_\pi$, with random weight (and bias) or parameters θ and $\overline{\theta}$
Initialize replay memory M

for episode = 1 to e **do**
  Initialize the first state s_0
  for $t = 1$ *to* T **do**
    Gather the state s_t from the edge-cloud environment
    if random value $< \epsilon$ **than**
      select a random action a_t
    else
      select $a_t = \underset{a}{\text{argmax}}\, Q(s_t, a|\theta)$
    Execute action a_t, received reward r_t and next state s_{t+1}
    Store (s_t, a_t, r_t, s_{t+1}) into M
    Get the mini-batch of (size: b) (s_i, a_i, r_i, s_{i+1}) from M
    for $i = 1$ to b **do**
      if $s_{i+1} = ternimal$
        $y_i = r_i$
      else
        $y_i = r_i + \gamma \times \max_a \overline{Q}_\pi(s_{i+1}, a|\theta)$
      $q_i = Q_\pi(s_i, a_i|\theta)$
    end for
    $\theta = \theta - \alpha\Delta_\theta \sum_{i=1}^{b} \frac{(q_i - y_i)^2}{b}$ //Gradient descent
    Every K episode, copy θ to $\overline{\theta}$
  end for
end for

Algorithm 1. Training Stage of the DDQNEC algorithm

Performance evaluation

This section evaluates the DDQNEC scheme's performance through computer simulation. The focus is on resource utilization, task acceptance ratio, task rejection ratio, and cost ratio using a simulation environment based on i9-10900 K CPU, 64 GB RAM, RTX 3090 GPU, Linux Ubuntu 20.04.02 LTS, Python 3.8, and PyTorch

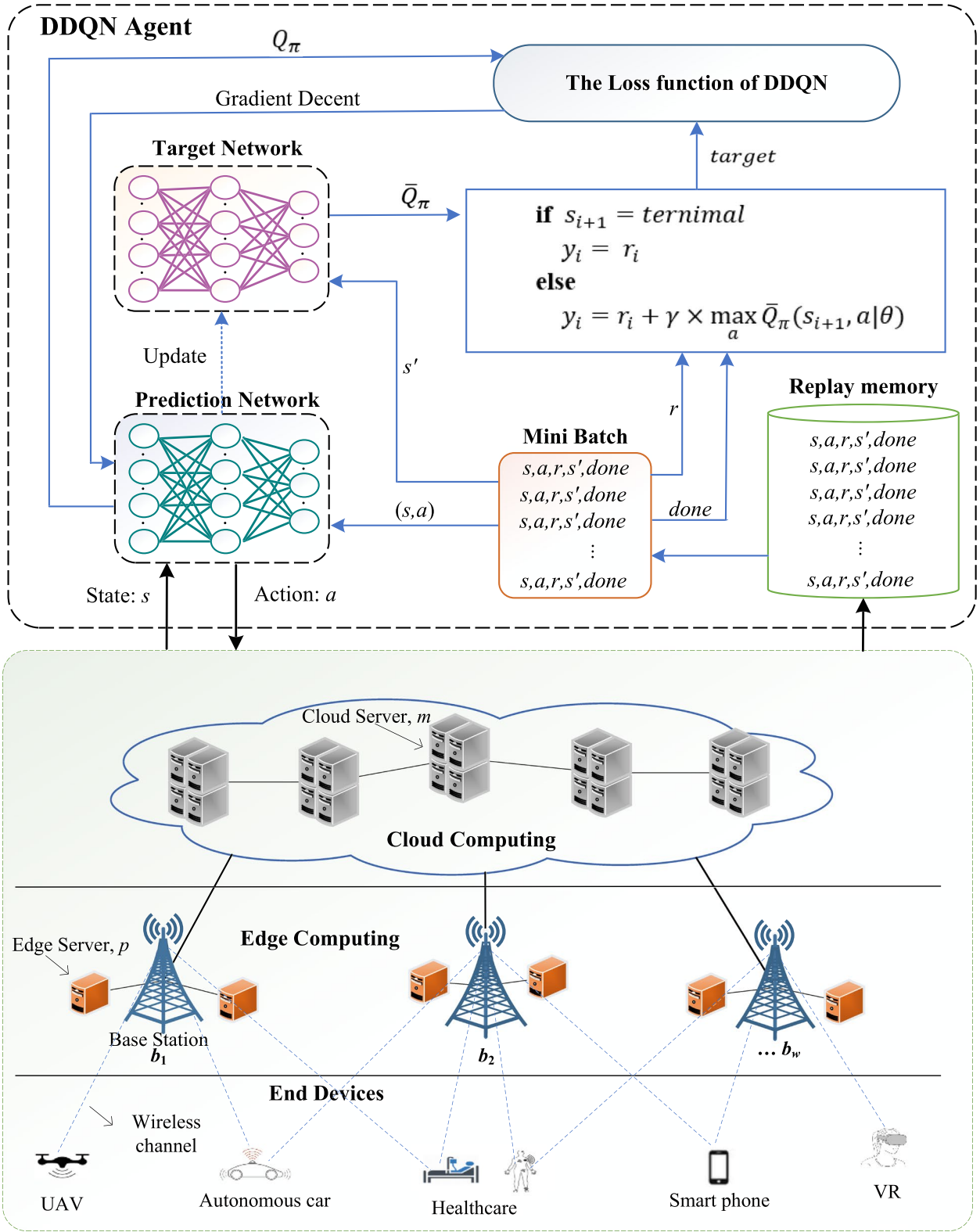


Fig. 2 The architecture of the DDQNEC scheme for task offloading and resource allocation

1.9.0 to reflect real-world edge-cloud computing environments and analyze and compare the results to existing methods. The DDQNEC is evaluated and compared to three heuristic methods (heuristic1, heuristic2, heuristic3) and DQNEC [9] using a simulated edge-cloud environment (as shown in Fig. 2) to measure its efficiency. Heuristic1 uses FIFO for tasks and prefers edge resources if available, Heuristic2 prioritizes tasks with high resource demands, and Heuristic3 uses the 0/1 knapsack algorithm to maximize utilization as profit. We conducted tests in two different types of environments, (small and large). In both the small and large environments, the number of tasks is distributed as: 50, 100, 150, 200, 250, and 300. However, the available resources at the edge and cloud, and task requirements are different in both environments. The small environment has fewer resources, whereas the large environment has more resources.

- In the small environments, the task parameters are as CPU requirement:10~20, data size:10~20, bandwidth:100*15, deadline:5~10 ms, and the available resources in the small environments are as the number of edge servers:30 with CPU capacity of 40~60, cloud servers:20 with CPU capacity of 60~80, and bandwidth: 100 Mbps.
- In large environments, the task requirements are higher than the small environments as CPU:20~30, data size:20~30, bandwidth:100*30, and deadline:10~15 ms, and the edge-cloud resources are, edge-servers:50 with CPU capacity of 50~80, cloud servers:30 with CPU capacity 60~100 and available bandwidth: 100 Mbps.

The simulation parameters used in our study are presented in Table 2, while Table 3 shows the configuration of the environment in which we conducted our experiments. Our scheme uses DDQN to decide whether to offload or reject a task based on resource availability and waiting tasks. It considers network information when selecting the computing server for offloading to optimize network performance. We compare our scheme's performance with heuristic algorithms in both environments in the following section.

Figure 3 presents a comprehensive comparison of the task rejection ratios of five different schemes. As the number of tasks increases, it can be observed that the rejection ratio for all five schemes also increases. However, the DDQNEC scheme exhibits a significantly lower increase when compared to the other four schemes, thereby indicating a superior performance in terms of task acceptance ratio and a lower rejection rate. The ability of DDQNEC to accept more tasks is a result of its ability to intelligently assign tasks to servers that are optimally matched in terms of resource requirements and availability. This not only saves and preserves resources for future utilization but also allows for more tasks to be accepted. A high acceptance rate is beneficial as it leads to higher resource utilization and reduces idle time in the system. As a result, DDQNEC outperforms other methods in terms of resource utilization, indicating its effectiveness in improving the proposed edge-cloud system.

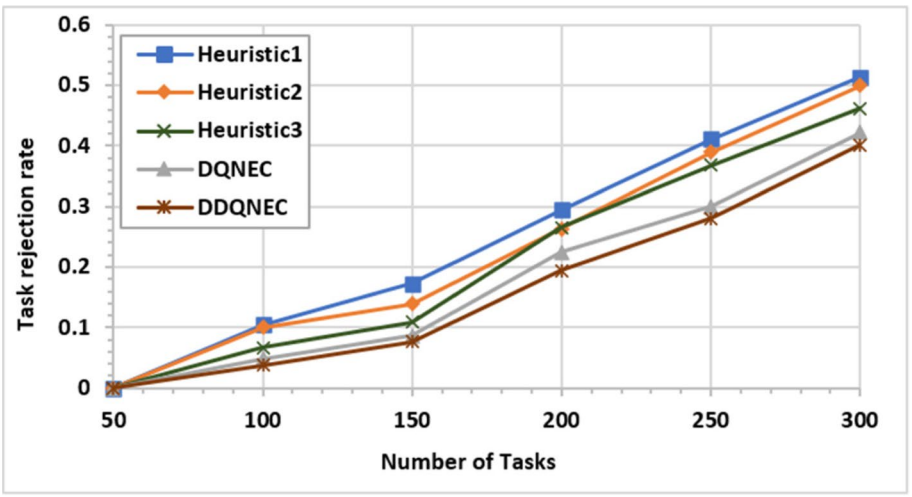
Figure 4 shows the comparison of the average utilization of the proposed scheme DDQNEC with four other heuristic schemes: heuristic1, heuristic2, heuristic3, and DQNEC. As the number of tasks increases,

Table 2 Simulation environment configuration

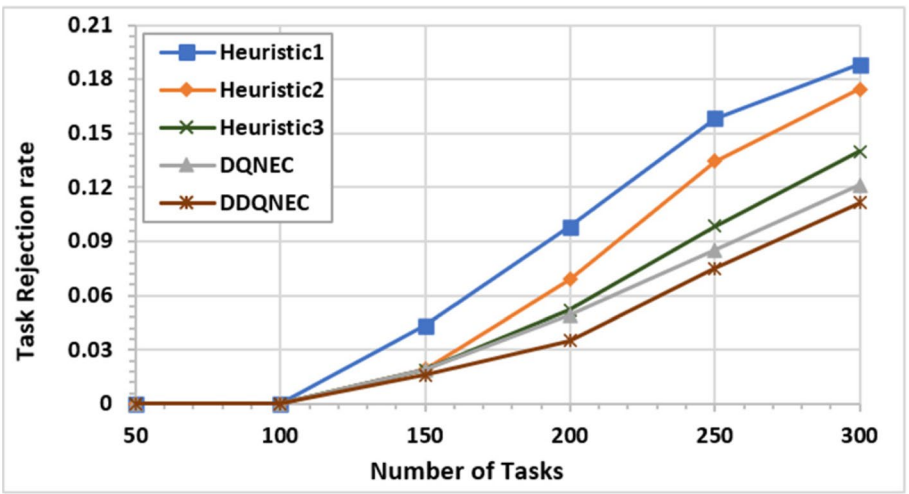
Small environments	
Number of cloud server	20
Cloud server CPU capacity	60 ~ 80 (uniform distribution)
Edge link bandwidth	100 (Mbps)
Cloud link bandwidth	200 (Mbps)
Number of the edge server	30
Edge server CPU capacity	40 ~ 60 (uniform distribution)
Task CPU request	10 ~ 20
Task data size	10 ~ 20 MB
Task tolerable delay	10 ~ 15 ms
Number of offloading tasks	50, 100, 150, 200, 250, 300
Large environments	
Number of cloud server	30
Cloud server CPU capacity	60 ~ 100 (uniform distribution)
Edge link bandwidth	200 (Mbps)
Cloud link bandwidth	300 (Mbps)
Number of the edge server	50
Edge server CPU capacity	50 ~ 80 (uniform distribution)
Task CPU request	20 ~ 30
Task data size	20 ~ 30 MB
Task tolerable delay	5 ~ 10 ms
Number of offloading tasks	50, 100, 150, 200, 250, 300

Table 3 Learning parameters of the DDQNEC algorithm

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



(a): Small Environment

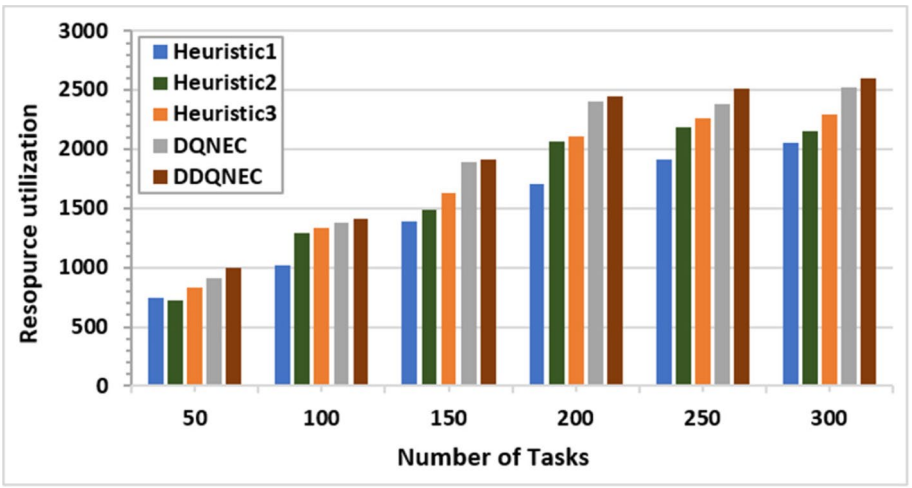


(b): Large Environment

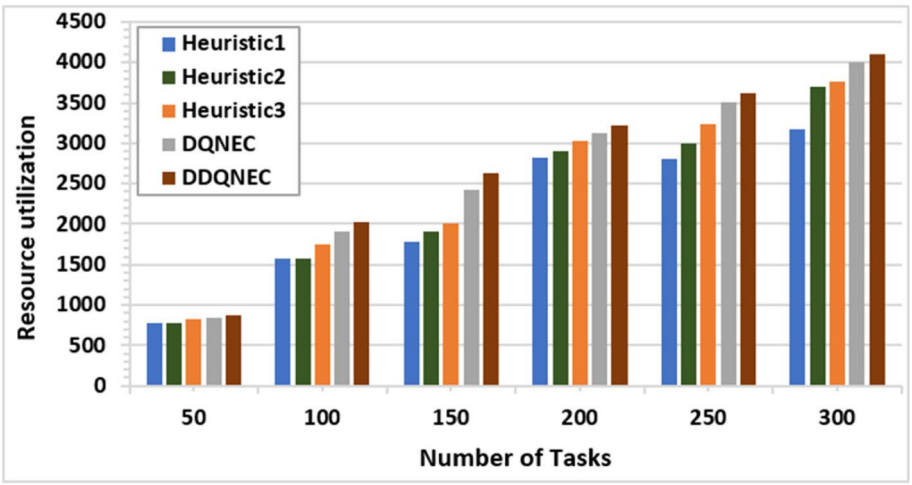
Fig. 3 Task rejection comparison **a**: Small Environment **b**: Large Environment

the average resource utilization for all five schemes also increases. However, it is observed that DDQNEC consistently demonstrates a higher utilization rate when compared to the other four algorithms in both environments, as depicted in Figs. 5(a) and (b). The task rejection ratio is a crucial metric that has a direct impact on resource utilization. A low task rejection ratio implies high resource utilization. DDQNEC employs a robust mechanism for selecting the best servers based on task requirements, thereby improving the efficiency of the edge-cloud system. Additionally, DDQNEC makes use of intelligent resource allocation strategies, resulting in an increased acceptance rate of tasks while maintaining resource utilization. A high acceptance rate generally leads to a higher average utilization compared to cost. The results demonstrate that DDQNEC achieves a higher utilization rate than the other algorithms, thus highlighting the effectiveness of the DDQN approach in enhancing the performance of the edge-cloud system.

Figure 5 presents a comprehensive comparison of the cost ratios of five different schemes as the number of tasks increases. As the number of tasks increases, it can be observed that the cost ratio for all five schemes also



(a): Small Environment



(b): Large Environment

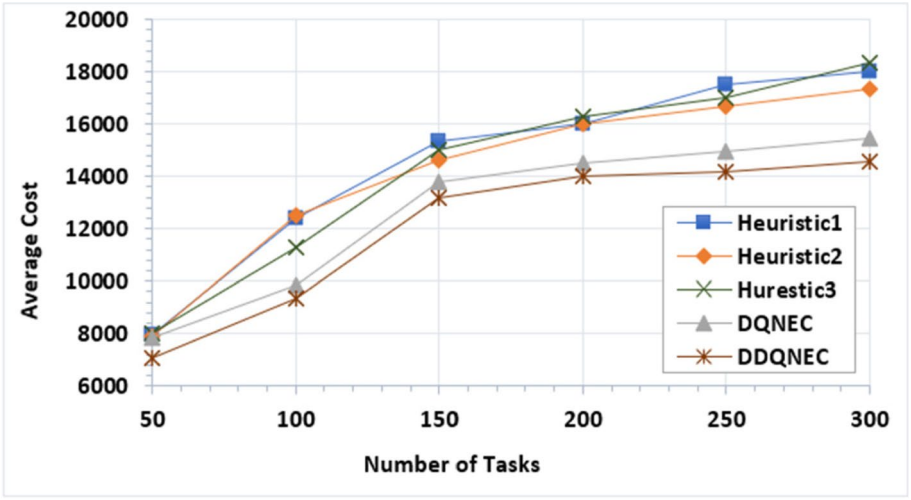
Fig. 4 Resource utilization comparison **a**: Small Environment **b**: Large Environment

increases. However, the proposed scheme DDQNEC exhibits a significantly lower increase in comparison to the other four schemes, both in small and large environments, thereby indicating a superior performance in terms of cost ratio. Additionally, DDQNEC has a significantly lower task rejection rate when compared to the three heuristics and DQNEC, which implies that it accepts more tasks for offloading and increases the utilization of the edge-cloud system. The key factor that enables DDQNEC to achieve this is its ability to intelligently assign tasks to servers that are optimally matched in terms of resource requirements and availability, thus minimizing the overall cost and maximizing the utilization of the edge-cloud system.

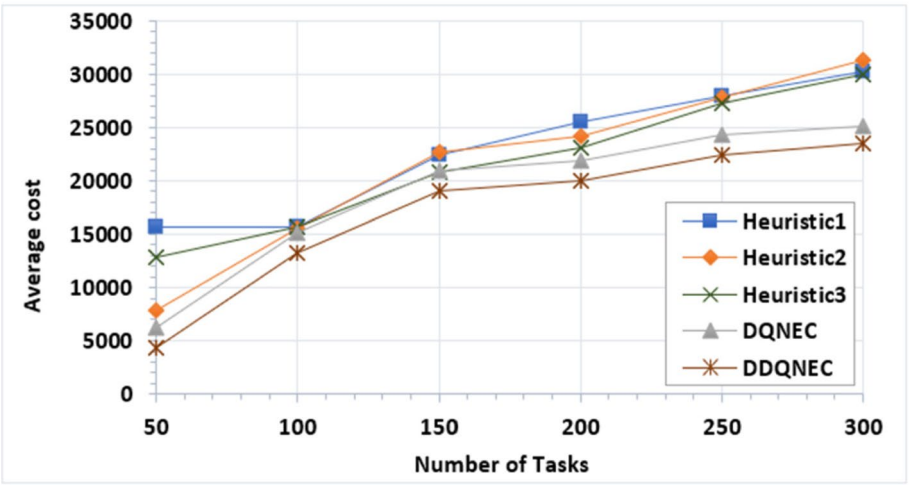
Conclusion

The task offloading and resource allocation in edge-cloud dynamic environments is a difficult problem. A solution is proposed by formulating it as an MDP optimization problem and using the DDQN algorithm to find an optimal solution for task offloading. The DDQNEC model uses an agent to make better decisions for end devices and offload their computation-intensive and low-latency task to the edge or cloud server. This improves the performance in terms of average cost, average utilization, and task rejection rate, and also improves resource utilization compared to other algorithms.

In the future, we aim to improve the DDQNEC scheme for task offloading and resource allocation by



(a): Small Environment



(b): Large Environment

Fig. 5 Average cost comparison **a**: small environment **b**: large environment

using advanced machine learning and AI algorithms. We will analyze the edge-cloud network by considering various factors, such as the characteristics and capabilities of end devices, to optimize task offloading. Furthermore, we will explore the use of reinforcement learning techniques for managing a significant number of IoT devices with varying task requirements, with a focus on techniques suitable for continuous action spaces.

Acknowledgements
This research was supported by Basic Science Research Programs through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. NRF-2023R1A2C1003143 and NRF-2018R1A6A1A03025526).

Authors' contributions
Methodology: Ihsan Ullah; Resources: Ihsan Ullah and Hyun-Kyo Lim; Software: Hyun-Kyo Lim, Yeong-Jun Seok; Supervision: Youn-Hee Han; Writing original draft: Ihsan Ullah; Writing review editing: Ihsan Ullah, Hyun-Kyo Lim; All authors read and approved the final manuscript.

Funding
This study was supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant No. NRF-2023R1A2C1003143 and NRF-2018R1A6A1A03025526.

Availability of data and materials
The data used to support the findings of this study are available from the corresponding author upon request.

Declarations

Competing interests
The authors declare no competing interests.

Received: 28 January 2023 Accepted: 23 May 2023
Published online: 26 July 2023

References

1. Singh A, Satapathy SC, Roy A, Gutub A (2022) AI-based mobile edge computing for IoT: applications, challenges, and future scope. Arabian J Sci Engin (AJSE) 47(8):9801–9831. https://doi.org/10.1007/s13369-021-06348-2
2. Dai B, Niu J, Ren T, Atiquzzaman M (2022) Towards mobility-aware computation offloading and resource allocation in end-edge-cloud orchestrated computing. IEEE Internet Things J 9(19):19450–62
3. Dai Y, Xu D, Maharjan S, Qiao G, Zhang Y (2019) Artificial intelligence empowered edge computing and caching for internet of vehicles. IEEE Wirel Commun 26(3):12–18
4. Rodrigues TK, Suto K, Nishiyama H, Liu J, Kato N (2019) Machine learning meets computation and communication control in evolving edge and cloud: challenges and future perspective. IEEE Commun Surv Tutor 22(1):38–67
5. Rodrigues TG, Suto K, Nishiyama H, Kato N, Temma K (2018) Cloudlets activation scheme for scalable mobile edge computing with transmission power control and virtual machine migration. IEEE Trans Comput 67(9):1287–1300
6. Zhao J, Li Q, Gong Y, Zhang K (2019) Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks. IEEE Trans Veh Technol 68(8):7944–7956
7. Nguyen TT, Le LB, Le-Trung Q (2019) Computation offloading in MIMO based mobile edge computing systems under perfect and imperfect CSI estimation. IEEE Trans Serv Comput 14(6):2011–2025
8. Dai Y, Xu D, Maharjan S, Zhang Y (2018) Joint computation offloading and user association in multi-task mobile edge computing. IEEE Trans Veh Technol 67(12):12313–12325
9. Ullah I, Lim H.-K., Seok Y.-J., and Han Y.-H (2022) "Optimal task offloading with deep Q-network for edge-cloud computing environment," presented at the 2022 13th International Conference on Information and Communication Technology Convergence (ICTC), IEEE, pp. 406–411
10. Huang L, Feng X, Zhang C, Qian L, Wu Y (2019) Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. Digit Commun Netw 5(1):10–17
11. Gu F, Niu J, Qi Z, Atiquzzaman M (2018) Partitioning and offloading in smart mobile devices for mobile cloud computing: State of the art and future directions. J Netw Comput Appl 119:83–96
12. Huang L, Bi S, Zhang Y-JA (2019) Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Trans Mob Comput 19(11):2581–2593
13. Li J, Gao H, Lv T, Lu Y (2018) "Deep reinforcement learning based computation offloading and resource allocation for MEC", presented at the, 2018 IEEE wireless communications and networking conference (WCNC) IEEE, pp. 1–6
14. Xiong X, Zheng K, Lei L, Hou L (2020) Resource allocation based on deep reinforcement learning in IoT edge computing. IEEE J Sel Areas Commun 38(6):1133–1146
15. Chen X, Zhang H, Wu C, Mao S, Ji Y, Bennis M (2018) Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. IEEE Internet Things J 6(3):4005–4018
16. Lu H, Gu C, Luo F, Ding W, Liu X (2020) Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. Future Gener Comput Syst 102:847–861
17. Zhang K, Zhu Y, Leng S, He Y, Maharjan S, Zhang Y (2019) Deep learning empowered task offloading for mobile edge computing in urban informatics. IEEE Internet Things J 6(5):7635–7647
18. Liu Y, Yu H, Xie S, Zhang Y (2019) Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks. IEEE Trans Veh Technol 68(11):11158–11168
19. Wang J, Zhao L, Liu J, Kato N (2019) Smart resource allocation for mobile edge computing: a deep reinforcement learning approach. IEEE Trans Emerg Top Comput 9(3):1529–1541
20. Liu C, Tang F, Hu Y, Li K, Tang Z, Li K (2020) Distributed task migration optimization in MEC by extending multi-agent deep reinforcement learning approach. IEEE Trans Parallel Distrib Syst 32(7):1603–1614
21. Chen X, Zhang H, Wu C, Mao S, Ji Y, Bennis M (2018) "Performance optimization in mobile-edge computing via deep reinforcement learning", presented at the, 2018 IEEE 88th Vehicular Technology Conference (VTC-Fall) IEEE , pp. 1–6
22. Xie Y, Xu Z, Xu J, Gong S, and Wang Y (2019) "Backscatter-aided hybrid data offloading for mobile edge computing via deep reinforcement learning," presented at the International Conference on Machine Learning and Intelligent Communications, Springer, pp. 525–537
23. Tian K, Chai H, Liu Y, Liu B (2022) Edge Intelligence empowered dynamic offloading and resource management of MEC for Smart City internet of things. Electronics 11(6):879
24. Alfakih T, Hassan MM, Gumaei A, Savaglio C, Fortino G (2020) Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA. IEEE Access 8:54074–54084
25. Chen M, Liu W, Wang T, Zhang S, Liu A (2022) A game-based deep reinforcement learning approach for energy-efficient computation in MEC systems. Knowl.-Based Syst 235:107660
26. Lu H, He X, Du M, Ruan X, Sun Y, Wang K (2020) Edge QoE: Computation offloading with deep reinforcement learning for Internet of Things. IEEE Internet Things J 7(10):9255–9265
27. Chen J, Wu Z (2021) Dynamic computation offloading with energy harvesting devices: a graph-based deep reinforcement learning approach. IEEE Commun Lett 25(9):2968–2972
28. Chen Z, Wang X (2020) Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach. EURASIP J Wirel Commun Netw 2020(1):1–21
29. Chen M, Yi M, Huang M, Huang G, Ren Y, Liu A (2023) A novel deep policy gradient action quantization for trusted collaborative computation in intelligent vehicle networks. Expert Syst Appl 221:119743
30. Chen M et al (2022) GPDS: a multi-agent deep reinforcement learning game for anti-jamming secure computing in MEC network. Expert Syst Appl 210:118394
31. Liu K.-H, and Liao W (2020) "Intelligent offloading for multi-access edge computing: A new actor-critic approach," presented at the ICC 2020–2020 IEEE International Conference on Communications (ICC), IEEE, pp. 1–6
32. Qiu X, Zhang W, Chen W, Zheng Z (2020) Distributed and collective deep reinforcement learning for computation offloading: a practical perspective. IEEE Trans Parallel Distrib Syst 32(5):1085–1101
33. Li Y, Qi F, Wang Z, Yu X, Shao S (2020) Distributed edge computing offloading algorithm based on deep reinforcement learning. IEEE Access 8:85204–85215
34. Chai F, Zhang Q, Yao H, Xin X, Gao R, Guizani, M (2023) Joint multi-task offloading and resource allocation for mobile edge computing systems in satellite IoT. IEEE Trans Veh Technol 1–15
35. Chen S, Ge X, Wang Q, Miao Y, Ruan X (2022) DDPG-based intelligent rechargeable fog computation offloading for IoT. Wirel Netw 28(7):3293–3304
36. Cheng M, Li J, Nazarian S (2018) "DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers", presented at the, 2018 23rd Asia and South pacific design automation conference (ASP-DAC) IEEE ,129:134
37. Nath S, and Wu J (2020) "Dynamic Computation Offloading and Resource Allocation for Multi-user Mobile Edge Computing," presented at the GLOBECOM 2020–2020 IEEE Global Communications Conference, IEEE, pp. 1–6
38. Chen J, Xing H, Xiao Z, Xu L, Tao T (2021) A DRL agent for jointly optimizing computation offloading and resource allocation in MEC. IEEE Internet Things J 8(24):17508–17524

Publisher's Note