Journal of Network and Computer Applications 240 (2025) 104211

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Research paper

Distributed deep reinforcement learning for independent task offloading in Mobile Edge Computing

Mohsen Darchini-Tabrizi [a], Amirhossein Roudgar [a], Reza Entezari-Maleki [a,b,c,*], Leonel Sousa [c]

[a] School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran
[b] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
[c] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal

ARTICLE INFO

Keywords:
Mobile Edge Computing
Task offloading
Task size prediction
Deep reinforcement learning

ABSTRACT

Mobile Edge Computing (MEC) has been identified as an innovative paradigm to improve the performance and efficiency of mobile applications by offloading computation-intensive tasks to nearby edge servers. However, the effective implementation of task offloading in MEC systems faces challenges due to uncertainty, heterogeneity, and dynamicity. Deep Reinforcement Learning (DRL) provides a powerful approach for devising optimal task offloading policies in complex and uncertain environments. This paper presents a DRL-based task offloading approach using Deep Deterministic Policy Gradient (DDPG) and Distributed Distributional Deep Deterministic Policy Gradient (D4PG) algorithms. The proposed solution establishes a distributed system, where multiple mobile devices act as Reinforcement Learning (RL) agents to optimize their individual performance. To reduce the computational complexity of the neural networks, Gated Recurrent Units (GRU) are used instead of Long Short-Term Memory (LSTM) units to predict the load of edge nodes within the observed state. In addition, a GRU-based sequencing model is introduced to estimate task sizes in specific scenarios where these sizes are unknown. Finally, a novel scheduling algorithm is proposed that outperforms commonly used approaches by leveraging the estimated task sizes to achieve superior performance. Comprehensive simulations were conducted to evaluate the efficacy of the proposed approach, benchmarking it against multiple baseline and state-of-the-art algorithms. Results show significant improvements in terms of average processing delay and task drop rates, thereby confirming the success of the proposed approach.

1. Introduction

The rise of cloud computing has transformed the utilization of computational services (Gasmi et al., 2022). However, the advent of 5G networks and the widespread adoption of Internet of Things (IoT) applications have created challenges. Routing computational requests to centralized datacenters for a rapidly growing number of devices results in longer delays and reduced Quality of Service (QoS) (Liu et al., 2018; Kazmi et al., 2020).

Many IoT and real-time applications are time-sensitive, requiring fast-paced service delivery. This is one of the problems that the edge computing paradigm seeks to address. Insufficient bandwidth between nodes can negatively impact IoT applications that demand millisecond-level low latency. By offloading tasks to the network edge, end-to-end latency is reduced, resulting in better performance and faster response times (Gasmi et al., 2022). Augmented reality and virtual reality applications are examples of delay-sensitive applications which involve computationally intensive tasks such as video processing (Qiao et al., 2018; Nandi et al., 2024). Another example of applications in this area is Internet of Vehicles (IoV) and autonomous driving, where observing and monitoring vehicle behavior enables solutions for better traffic congestion control and safer transportation. Edge computing is capable of addressing the time-critical needs of smart transportation (Hamdi et al., 2022; Carmo et al., 2024).

The emergence of the edge layer, located between the core network (where the cloud provider is located) and the user device, offers potential benefits of reduced response time and less network congestion. By bringing computational requests closer to the user device, the distance traveled is minimized, resulting in faster response time. Additionally, routing less traffic towards the core network reduces

* Corresponding author at: School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran.
E-mail addresses: mohsen_darchini@comp.iust.ac.ir (M. Darchini-Tabrizi), a_roudgar@comp.iust.ac.ir (A. Roudgar), entezari@iust.ac.ir (R. Entezari-Maleki), las@inesc-id.pt (L. Sousa).

https://doi.org/10.1016/j.jnca.2025.104211
Received 27 December 2024; Received in revised form 26 April 2025; Accepted 30 April 2025
Available online 14 May 2025

congestion and increases effective bandwidth. Mobile Edge Computing (MEC) (Mao et al., 2017a; Feng et al., 2022), also known as Multi-access Edge Computing (Porambage et al., 2018), aims to enhance the computational performance of mobile devices in various architectural setups, including single edge layers, hybrid edge–cloud architectures, and other configurations. In this context, mobile devices are faced with the decision of whether and where to offload computational tasks.

Several solutions have been suggested to address the issue of task offloading in MEC systems. Game theoretic approaches (Liu et al., 2018; Kim, 2022; Xu et al., 2023) have been instrumental in formulating strategies for effective task offloading. In addition, heuristic and meta-heuristic-based methods (Singh et al., 2020; Abbas et al., 2021; Yeganeh et al., 2023) have provided alternative perspectives to address the challenge. Markov-based techniques (Zhou et al., 2019; Yang et al., 2021; Gao et al., 2023) represent another avenue in the search for efficient task offloading solutions. Genetic algorithms (Singh and Kim, 2021; Zhu and Wen, 2021; Li and Zhu, 2020) also offer another likely perspective on similar problems. Deep Reinforcement Learning (DRL) solutions (Zhu and Wen, 2021; Wu et al., 2024; Lim et al., 2022) have recently gained popularity as a promising approach to tackle offloading issues. Despite the variety of methods used to solve the problem, there has been a distinct emphasis on different performance metrics for optimization. Some approaches emphasize the optimization of time and energy consumption (Abbas et al., 2021; Yeganeh et al., 2023; Lim et al., 2022), aiming to enhance the overall performance of the MEC system. Some other strategies concentrate on minimizing the computation delay while accounting for the dynamic mobility patterns of user devices (Lai et al., 2024; Saleem et al., 2022; Hu et al., 2020). Some also focus on optimizing resource allocation in an offloading problem (Kim, 2022; Xu et al., 2023; Luo et al., 2024).

Due to the uncertain and highly dynamic interactions on task offloading in MEC with multiple edge nodes and multiple mobile devices, DRL approaches have gained popularity by their ability to learn near-optimal policies and adapt to these dynamic scenarios (Hortelano et al., 2023). In this paper, we propose two DRL algorithms based on actor–critic models. Similar works (Singh and Kim, 2021; Abbas et al., 2021; Yang et al., 2021) consider a simple First-In-First-Out (FIFO) structure for task queues or do not discuss the queueing structure, which implies the use of FIFO as default in the computation and communication models. Previously presented approaches do not consider advanced queueing structures to enhance offloading performance. In this paper, we introduce a novel algorithm that differs from the traditional FIFO approach by focusing on task sizes. Also, some works assume prior knowledge of task sizes whether represented and referred to as uploading data size as in Yang et al. (2021), or simply input data size as in Abbas et al. (2021) and Penmetcha and Min (2021). In contrast to these works, we presume a more realistic situation where the sizes of tasks are unknown, and make offloading decisions based on an estimate of the size of each arriving task.

The primary novelty of this work lies in the integration of a non-traditional queuing mechanism with DRL, specifically Deep Deterministic Policy Gradient (DDPG) and Distributed Distributional Deep Deterministic Policy Gradient (D4PG). Unlike previous studies that often assume prior knowledge of task sizes, the proposed model dynamically estimates these sizes using a GRU-based predictive model, which significantly enhances decision-making in MEC environments. This approach is particularly suited for dynamic and heterogeneous MEC environments, for which the variability of task sizes and network conditions plays a crucial role. Furthermore, our distributed architecture allows each mobile device to act as an independent RL agent, enabling more adaptive and scalable task offloading decisions. The key contributions of this paper are as follows.

- A novel queueing scheme as an alternative to simple FIFO and a sequence model to predict arriving task sizes. The information obtained from this prediction model for each task arriving to the system is used as an input for the queueing structure at each time step.
- DRL algorithms, specifically DDPG and D4PG, are adopted as core of the offloading models for each mobile device, where in the proposed approach each mobile device acts as an independent RL agent.
- The system recognizes the lack of prior knowledge of task sizes. Our system addresses this limitation by using estimated values, thereby acknowledging the inherent uncertainty in the system. This assumption is essential for addressing the erroneous nature of the system and forms the basis for further research advances.
- The proposed approach is assessed by comparing the obtained numerical results, including average processing delay and drop rate, with state-of-the-art approaches.

The remainder of this paper is structured as follows. Section 2 offers a review of the relevant research conducted in the context of task offloading in different cloud/edge environments. Section 3 presents the system model, outlines the problem statement, formulates the observed state for each mobile device, and Section 4 models the problem as a Markov Decision Process (MDP). Section 5 introduces the proposed DRL algorithm to make the offloading decision based on the observed states at each mobile device. Section 6 details and examines the experimental results. Finally, Section 7 concludes the paper and offers recommendations for future research.

2. Related work

Deep reinforcement learning techniques have been employed to tackle task offloading issues with the goal of improving various system and user-oriented performance metrics, and optimize the resource allocation process. In the following, we introduce some of these approaches that are related to the investigation reported in this paper.

Some authors have used Directed Acyclic Graphs (DAGs) in their task model. Huang et al. (2021) proposed a Deep Q-Network (DQN) based approach to improve latency and energy consumption imposed by local processing and transmission in an Internet of Vehicles (IoV) network, for applications with dependent subtasks. The relationship of the subtasks was modeled through a DAG and priorities were assigned to each subtask using a topological sorting of the DAG. Wang et al. (2022) presented a DRL algorithm for a similar DAG-modeled sequence of dependent tasks to reduce processing delay and energy consumption. In contrast to Huang et al. (2021), DAGs were not only used by Wang et al. (2022) for task prioritization but also for task offloading decisions. Wang et al. (2022) processed the DAG data using a sequence-to-sequence model with an attention-based encoder–decoder network to determine the policy, and define parameters for the policy and network models. They generated a synthetic dataset of DAGs with varied qualities for evaluation purposes. Liu et al. (2024) promotes the utilization of DRL for enhancing task graph offloading processes within MEC environments. The SATA-DRL algorithm marks a notable progression in the field, adeptly adjusting to the complexities and dynamics of MEC environments. By efficiently managing task offloading and scheduling, the algorithm enhances the efficiency and reliability of computational resource allocation and opens up possibilities for further advancements in edge computing technology. Zhao et al. (2024) address the challenges of offloading delay-sensitive, dependency-aware tasks in vehicular edge computing networks. They propose a Stackelberg game-based framework (SDOP) to model the interaction between the SDN controller (leader) and vehicles (followers), aiming to maximize the utilities of both parties while considering task dependencies and resource pricing. The framework accounts for various dependency relationships among subtasks, including sequential and parallel structures, and introduces a Gradient Ascent Plus Genetic Algorithm (GAPG) to solve the optimization problem. The GAPG algorithm combines gradient ascent for pricing and offloading strategies with a genetic algorithm to select optimal subtasks for offloading, ensuring that delay constraints are met. The authors prove the existence and uniqueness of

Nash and Stackelberg equilibria using backward induction. Simulation results show that the proposed GAPG algorithm significantly improves the utility of both the SDN controller and vehicles compared to baseline methods, such as random offloading and non-pricing strategies. This work is particularly notable for its comprehensive consideration of task dependencies and its innovative incentive mechanism to motivate the SDN controller to participate in computation offloading, making it highly relevant for advanced VEC applications in 5G/6G networks.

Some other works focus on optimizing energy jointly with performance parameters and considerations. Yang and Bai (2022) introduced the Delay-Energy Weighted Sum (DEWS) parameter to balance the energy consumption and processing delays of computed tasks in a Mobile Fog Computing (MFC) environment. Their proposed method involves two steps: task assignment to the fog server or local computation, followed by an offloading decision made by the fog server to determine whether to process the offloaded task or further offload it to the cloud server. Qu et al. (2021) aimed to enhance the performance of task offloading decisions in an MEC system under environmental changes by using deep meta reinforcement learning. The system model consists of multiple user devices, one edge server and one cloud server. The proposed meta-learning framework includes the outer and inner models. The outer model focuses on the influence of environmental changes on the offloading performance, i.e., the network parameters of the inner model, while the inner model focuses on offloading decisions. The proposed algorithm aims to reduce energy consumption and processing delay. Merluzzi et al. (2022) investigated a system consisting of multiple User Equipment (UE), one Access Point (AP), and one Edge Server (ES), while taking into account end-to-end service delay and packet error rate. Their objective is to reduce energy consumption by leveraging the concept of putting the AP into sleep mode during times that are not required, thus shifting the system from "always on" to "always available". By applying Little's law, the problem of reducing delay is transformed into a problem of reducing queue length. At the start of every time slot, a short period of time is allocated for control signaling to decide whether or not the AP should be in sleep mode during that time slot. Li et al. (2024) introduced a blockchain-based reliable task offloading framework in edge–cloud cooperative workflows within the Internet of Medical Things (IoMT). This framework combines blockchain and Software Defined Networking (SDN) to optimize offloading decisions dynamically and reliably without latency loss. The system comprises three layers: IoMT device layer, edge layer, and cloud layer, each with specific functions. It addresses data security and privacy challenges in IoMT applications, particularly as the number of IoMT devices and users continues to increase. The framework outperforms others in scalability, robustness, efficiency, and reliability. The completion time and energy consumption for offloading decisions were estimated, and overall workflow completion time and energy use were derived. The authors also analyze the framework's effectiveness in ensuring data security and introduce a reputation assessment model for the blockchain consensus mechanism. Bi et al. (2021) have employed a DRL algorithm combined with a Lyapunov method for making offloading decisions and allocating resources in a system based on Time Division Multiple Access (TDMA). The objective is to increase the amount of processed data while maintaining stability in queue states. Alsadie (2024) propose HybridTO, a hybrid task offloading strategy that combines Grey Wolf Optimizer (GWO) and Particle Swarm Optimization (PSO) to address the challenges of energy efficiency and latency in edge computing. The framework considers capacity and proximity constraints, and latency requirements, leveraging the collaborative capabilities of edge servers to optimize task offloading decisions. The HybridTO algorithm integrates PSO's global exploration capabilities with GWO's local exploitation strengths, enabling efficient resource allocation and task distribution across edge and cloud layers. The algorithm operates in two main phases: (1) PSO is used to explore the solution space and identify promising regions, while (2) GWO refines these solutions by exploiting local optima, ensuring a balance between exploration and exploitation. Extensive simulations demonstrate that HybridTO significantly outperforms baseline methods, such as random offloading and dynamic programming-based approaches, in terms of energy consumption, offloading utility, and response delay, particularly in resource-constrained environments. This work highlights the effectiveness of hybrid optimization techniques in enhancing task offloading performance in EC, offering a robust solution for energy-efficient and latency-sensitive applications. Chen et al. (2022) have proposed a two-step approach using a DRL algorithm and a Lagrangian method to determine bandwidth allocation, energy allocation, and a partial offloading ratio. The objective is to jointly optimize resource allocation and energy allocation. Zhu et al. (2022) aimed to address task offloading in a NOMA-based network to reduce processing and transmission delays, while ensuring workload offloading allocation. In paper (Zhu et al., 2022), task offloading is defined as the allocation of different parts of a task to a particular edge server. The authors proposed a DRL approach to solve the problem by determining the workload offloading portion of tasks and an estimated transmission duration. Chen et al. (2025) address the challenges of task offloading and resource pricing in UAV-assisted MEC systems, emphasizing the optimization of utility for both edge servers and users under constraints, such as limited battery life and computational resources of mobile devices. They propose two game-theoretic algorithms: the Game-based User Allocation (GBUA) algorithm, which minimizes server energy consumption and optimizes user allocation by establishing a Nash Equilibrium (NE), and the Resource Pricing and Task Offloading (RPATO) algorithm, which employs a Stackelberg game model to iteratively optimize offloading and pricing strategies, ensuring convergence to an optimal solution. Extensive experiments demonstrate that the RPATO algorithm outperforms benchmark methods in terms of utility for both servers and EUs, with both algorithms showing significant improvements in utility as iterations progress. This work provides valuable insights into the interplay between resource allocation and pricing in MEC systems, offering a robust framework for future research in this domain.

Some research works present DRL approaches based on actor–critic models. Zhang et al. (2021) proposed a DRL-based algorithm for offloading compute-dependent tasks in a hybrid edge–cloud network to reduce cost, energy consumption and delay. They used a modified version of the DDPG algorithm, where the actor model suggested a continuous computational frequency based on the observed state. A given computational frequency and the observed state were used as inputs for the critic model to compute the Q-values for local computation, cloud offloading, or edge offloading scenarios. Their experiments assumed a uniform distribution of initial user device locations within specific circular regions, which affects the channel gain received from edge servers and energy consumption. Pang et al. (2022) presented an algorithm based on a Twin-Delayed DDPG (TD3) model and Laplace distribution to enhance user privacy and mitigate information leakage related to usage patterns and locations. The TD3 model used the state of the communication channel and the desired level of privacy at each time slot as inputs to determine a coefficient, applied on a range-constrained Laplace distribution to determine an offloading ratio. The effectiveness of the algorithm was evaluated based on the average computational cost and the level of information leakage. Li et al. (2021a) incorporated a TD3 architecture and an LSTM model to increase the number of processed tasks. The proposed algorithm also considers reliable data transmission by including the Signal-to-Noise Ratio (SNR) value in the offloading decision process. The input data for the algorithm includes task size, required CPU cycles, and task deadlines. The results were compared with other methods utilizing DRL algorithms such as DQN, DDPG and Asynchronous Advantage Actor–Critic (A3C).

Approaches for improving computation performance can be applied to a more general scale of MEC environments. Moon and Lim (2021) have addressed the task migration issue in MEC servers of a vehicular network. Their objective was to achieve a balanced load distribution

Table 1
Comparison between related work.

Paper	Objective	Methodology	Model
Huang et al. (2021)	Improve latency and energy in IoV	DQN with DAG modeling	DAG with topological sorting
Wang et al. (2022)	Reduce delay and energy	DRL with DAG and Seq-to-Seq	DAG with Seq-to-Seq
Liu et al. (2024)	Enhance task offloading in MEC	SATA-DRL algorithm	Dynamic task graph offloading
Zhao et al. (2024)	Maximize utility in VEC	Stackelberg game (SDOP) and GAPG	Dependency-aware tasks
Yang and Bai (2022)	Balance energy and delay in MFC	DEWS parameter	Fog/local computation
Qu et al. (2021)	Reduce energy and delay in MEC	Mta-RL with outer/inner models	Environment-aware offloading
Merluzzi et al. (2022)	Reduce energy with AP sleep	AP sleep mode and Little's law	Task scheduling with sleep mode
Li et al. (2024)	Optimize IoMT offloading	Blockchain and SDN framework	Offloading across IoMT layers
Bi et al. (2021)	Increase data and stabilize queues	DRL and Lyapunov method	Offloading in TDMA systems
Alsadie (2024)	Optimize energy and latency	HybridTO (GWO + PSO)	Offloading with constraints
Chen et al. (2022)	Optimize resource and energy	DRL and Lagrangian method	Partial task offloading
Zhu et al. (2022)	Reduce delays in NOMA	DRL for workload offloading	Offloading to edge servers
Chen et al. (2025)	Optimize utility in UAV-assisted MEC	GBUA and RPATO algorithms	Offloading and resource pricing under constraints
Zhang et al. (2021)	Reduce cost, energy and delay	Modified DDPG (actor–critic)	Offloading based on frequency
Pang et al. (2022)	Enhance privacy and reduce leakage	TD3 with Laplace distribution	Privacy-preserving offloading
Li et al. (2021a)	Increase tasks and reliable data	TD3 and LSTM	Reliable data offloading
Moon and Lim (2021)	Balance MEC server load	Controller-based migration	Migration between servers
Penmetcha and Min (2021)	Optimize robot offloading on AWS	DQN-based offloading	Data offloading based on CPU
Tong et al. (2020)	Minimize response time and energy	DRL and CNN	Offloading with user mobility

among the MEC servers. The proposed algorithm in paper (Moon and Lim, 2021) considered a controller that periodically receives load information from each MEC server and performs offloading from a set of overloaded MEC servers to a set of underloaded ones. This allows the computational tasks of each vehicle to be processed by a maximum of two MEC servers, and promotes load balancing between MEC servers. Penmetcha and Min (2021) explored the issue of task offloading for a navigator robot onto a public cloud network, specifically utilizing Amazon Web Services (AWS). A DQN-based offloading approach was proposed, which takes into account CPU availability, data input size of tasks, and latency between the robot and the cloud network. However, the steady network connection between the cloud provider and the simulated robot resulted in a rather stable latency between the cloud network and the robot. Therefore, the impact of the latency parameter on the results is not as impactful as the other two parameters on the offloading decision. By excluding the data input size from the base knowledge and attempting to predict its value, more accurate results can be achieved for cloud robotics applications. Tong et al. (2020) explored the optimization of task offloading and resource allocation within a MEC environment. They proposed a novel DRL-based online algorithm to minimize average task response time and total energy consumption. Their algorithm uses the combination of Q-learning and Convolutional Neural Network (CNN) structure, and considers the mobility of users between base stations, uses the entropy weight method for objective weighting, and aims to improve the users' Quality of Experience (QoE) and reduce power consumption to simulate a realistic application environment. Experimental results demonstrate that the algorithm surpasses other methods in terms of system utility, indicating the overall effectiveness of the optimization.

The above-mentioned research highlights the potential of using DRL algorithms to address task offloading challenges in various settings and environments. To the best of our knowledge, no prior research has fully explored the integration of recent advances to improve task offloading procedures. In this regard, we propose an innovative approach based on DRL algorithms. This approach considers task size as an important factor in offloading decision, and tackles the uncertain and unknown nature of this parameter. Thus, the proposed approach is based on a scheduling algorithm that operates based on task size estimation. The primary objective is to improve the overall performance of the MEC system by minimizing the average processing delay and the drop rate.

The selection of DDPG and D4PG as the core algorithms in our study is motivated by their unique advantages in handling continuous action spaces and uncertainty in dynamic environments. DDPG is particularly effective in managing continuous decision-making processes, which is crucial for fine-grained control over task offloading and resource allocation in MEC scenarios. D4PG, on the other hand, extends DDPG by incorporating a distributional perspective, enabling more robust learning under high variability in task execution times and network conditions. Compared to alternative DRL methods, such as Proximal Policy Optimization (PPO) or Soft Actor–Critic (SAC), DDPG and D4PG offer a better balance between computational efficiency and convergence speed, making them well-suited for real-time MEC decision-making. This is particularly important in environments where rapid adaptation to changing conditions is required. Table 1 provides an overview of the relevant studies, outlining their optimization objectives, methodology, task model, and contributions.

3. System model

This section describes the general structure of the system explored in this study and offers a detailed mathematical characterization of the system model. It also introduces the task size prediction model, the queuing model and algorithm, the processing and transmission processes, and the MDP for the problem. Table 2 provides a summary of the notation used in this paper.

The MEC system studied in this paper consists of M mobile devices and N edge nodes. Computational tasks are considered indivisible, i.e., they are either completely processed on the local device or offloaded to an edge node. The offloading decision for each mobile device is made at the device independently from other devices (Abu-Taleb et al., 2022). We assume that tasks arrive at each time slot with a specific probability like the assumption made in Tang and Wong (2022) and Liu et al. (2016), which is called the task arrival probability in this paper. The size of each task is considered as the main property describing a computational task. We use a pretrained sequence model based on Gated Recurrent Units (GRU) to predict size of incoming tasks at the moment of arrival. The precise computational size in bits can never really be known as this value is used to simulate the computation process. In other words, we can never be sure how many bits the CPU will process to complete a task. Thus, no actual task when arrived has a precise known size. There can be rough estimations based on a task's certain features such as sequential behavior which is the one addressed in this paper. Subsequently, an offloading decision is made based on the predicted task size and other information extracted from the observed environmental state, which will be discussed in the following sections.

Table 2
Description of notations used in this paper.

Symbol	Definition
T	Number of time slots
M	Set of mobile devices
N	Set of edge nodes
$u_m(t)$	A task that arrived in mobile device m during time slot $t \in T$
$\sigma_m(t)$	Computational size of task $u_m(t)$
$\acute{\sigma}_m(t)$	Estimated computational size of task $u_m(t)$
$\sigma^e_{m,n}(t)$	Computational size of a task sent from mobile device m to edge node n, which is received by edge node e at time slot t
f^{device}_m	Processing capacity of mobile device $m \in M$ (in CPU cycles per second)
f^e_n	Processing capacity of edge node $n \in N$ (in CPU cycles per second)
$o^c_m(t)$	Number of time slots any newly arrived task must wait before being processed locally
$\upsilon^c_m(t)$	Time slot in which task $u_m(t)$ is either processed or dropped
Δ	Duration of each time slot
ρ_m	Processing density needed by task $u_m(t)$ (in CPU cycles per bit)
$q^e_{m,n}(t)$	Queue length of tasks offloaded from $m \in M$ to $n \in N$ in time slot $t \in T$ (in bits)
$b^{edge}_{m,n}(t)$	The amount of task data, in bits, that is dropped from the queue for mobile device m by end of time slot t at edge node n
$\chi_n(t)$	The collection of active queues at edge node $n \in N$ during time slot $t \in T$
$s_m(t)$	Observed state by mobile device m at time slot t
$a_m(t)$	Selected action as offloading decision by mobile device m at time slot t
$c_m(t)$	Observed cost by mobile device m for action taken in time slot t
$r_m(t)$	Observed reward by mobile device m for action taken in time slot t

The offloading decision is considered to be binary, it accommodates the non-divisible nature of arriving tasks. The DRL model makes an offloading decision whether to handle the task locally or to offload it to a specific edge node. Based on this decision, a task will be put on either the computation queue or transmission queue of the device when it waits for its turn to be either processed or transmitted in each queue. This process is repeated for each time slot t when a task arrives.

3.1. Task size prediction

The mobile device is unaware of the size of task $u_m(t)$ when it arrives at time slot t. To estimate the size of the new task, information about the sizes of past tasks is utilized. The estimation process takes L time slots, during which the mobile device has the ability to independently ascertain the size of the task. Consequently, at time slot t, tasks that arrived at time slots $t-L$ and earlier are known to the device. Therefore, the size of these tasks is used to estimate the size of the task arrived at time slot t. In addition, we take Γ as the number of task size records used in the estimation procedure. Hence, the sizes of tasks at time slots $t-L-\Gamma+1$, $t-L-\Gamma+2$, …, $t-L$ are used as inputs to the sequence model for predicting the size of the task arriving at time slot t. Furthermore, at time slot $t+L$, the size of $u_m(t)$ becomes known to the device. During this time window of L time slots, the actual size of a task can be inferred by other metrics, such as transmission or processing delay or through direct computations at the device. We denote the estimated size of task $u_m(t)$ as $\acute{\sigma}_m(t)$. The sequence model is based on a GRU layer network, which is trained using a mean squared error loss function prior to running the training episodes of the MEC task offloading problem. GRU is recommended for this use case due to its lower computational complexity. The value of $\acute{\sigma}_m(t+1)$ is required as part of the observed state by mobile device m at time slot $t+1$ $s_m(t+1)$. In fact, the task size prediction model predicts two values, $\acute{\sigma}_m(t)$ and $\acute{\sigma}_m(t+1)$, based on the aforementioned input information.

3.2. Queue structure

The size of tasks arriving at each time slot has a direct effect on the processing delay of each task (Wei and Liang, 2024). Thus, we propose a queueing structure that considers task size to schedule tasks in each queue accordingly.

The structure for both the transmission and computation queues consists of multiple inner queues operating in a FIFO manner. We assume three inner queues for the computation queue, denoted as $q^c_{m,0}$, $q^c_{m,1}$, $q^c_{m,2}$. Similarly, the transmission queue has inner queues denoted as $q^t_{m,0}$, $q^t_{m,1}$, $q^t_{m,2}$. Although we only mention the inner queues of the computation queue for brevity, the explanation also applies to the inner queue structures of the transmission queue. Assuming that we have a known range of task sizes for each task arriving at each time slot, we describe this range as $[j_1, j_2]$. Subsequently, this range is divided into three sub-ranges, $[j_1, j_1+\frac{j_2-j_1}{3})$, $[j_1+\frac{j_2-j_1}{3}, j_1+2\frac{j_2-j_1}{3})$ and $[j_1+2\frac{j_2-j_1}{3}, j_2)$, categorizing the tasks into small, medium, and large classes. These sub-ranges are referred to as the first–third, second–third and third–third ranges, respectively.

Upon the arrival of task $u_m(t)$, an estimated size $\acute{\sigma}_m(t)$ is assigned to it. If no task is generated, $u_m(t)$ is set to zero for presentation simplicity. Once the offloading decision is determined for the task $u_m(t)$, it is assigned to the appropriate queue. For instance, if it is decided that task $u_m(t)$ should be processed locally, it will be placed in the computation queue. Depending on the value of $\acute{\sigma}_m(t)$ and the sub-range it falls into, the task $u_m(t)$ is assigned to one of the inner queues. For example, if $\acute{\sigma}_m(t)$ belongs to the first-third sub-range, the corresponding task will be placed in $q^c_{m,0}$.

When a mobile device wants to select a task from the computation queue, it does so by alternately selecting from the sub-queues $q^c_{m,0}$, $q^c_{m,1}$ and $q^c_{m,2}$ in a round-robin manner. This selection procedure provides a balance between processing tasks with low and high processing requirements. Our queuing mechanism is designed to efficiently manage heterogeneous task arrivals by employing a priority-based multi-queue structure. This approach dynamically classifies tasks into different queues based on their estimated execution time and priority, optimizing task scheduling and resource utilization. Specifically, tasks are categorized into small, medium, and large classes, each assigned to a separate inner queue. This classification allows for more balanced processing and reduces the likelihood of task starvation, particularly for smaller tasks that might otherwise be delayed by larger and more resource-intensive tasks. All the behaviors discussed regarding the queue structure for the computation queue are also applied to the transmission queue structure as depicted in Fig. 1.

3.3. Computational model

The system model addressed in this paper includes two key parameters that affect the local computation of tasks. The first parameter, denoted by $o^c_m(t)$, represents the number of time slots that each newly arrived task must wait before being processed locally if it is to be included in the computation. The second parameter is the processing capacity of the mobile device denoted by f^{device}_m (in CPU cycles per second), a constant value (Tang and Wong, 2022). Additionally, we define $\upsilon^c_m(t) \in T$ as the time slot in which task $u_m(t)$ is either processed or dropped. If no task has arrived at time slot $t \in T$, the value of $\upsilon^c_m(t)$ is set to zero. In order to compute the value of $o^c_m(t)$, we use the values of $\upsilon^c_m(\acute{t})$ for $\acute{t} < t$ by applying Eq. (1).

$$o^c_m = \max\{0, \max_{\acute{t}\in\{0,1,\ldots,t-1\}} \upsilon^c_m(t') - t + 1\},\ m \in |M|,\ t \in T\ . \tag{1}$$

The value of $\upsilon^c_m(t)$, as determined by Eq. (2). This is calculated if task $u_m(t)$ is set to be processed locally and is added to the processing queue at the start of the time slot $t \in T$.

$$\upsilon^c_m(t) = \min\{t + o^c_m(t) + \lceil \frac{\acute{\sigma}_m(t)}{\frac{f^{device}_m \Delta}{\rho_m}} \rceil - 1,\ t + \zeta_m - 1\}, \tag{2}$$

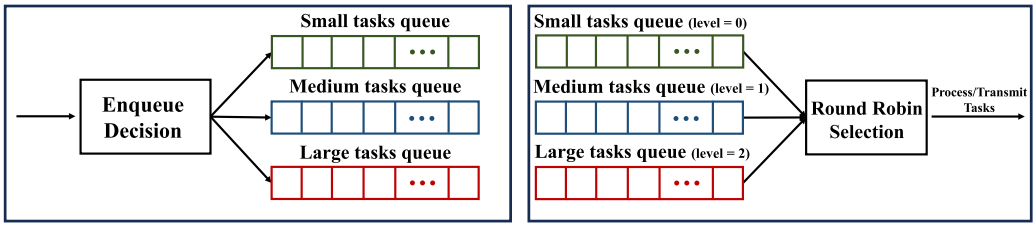

Fig. 1. Enqueue and Dequeue: queueing structure of each computation and transmission queues in a single mobile device.

where Δ is the constant duration of each time slot and ζ_m is task deadline for each task. The value of $\frac{f_m^{device}\Delta}{\rho_m}$ refers to the number of bits that can be processed per time slot and ρ_m denotes the processing density needed by task $u_m(t)$.

It should be noted that the value of $o_m^c(t)$ is computed before the offloading decision is taken, and it is indeed a value that affects the offloading decision. Also, the computation of $v_m^c(t)$ is affected by both $o_m^c(t)$ and the estimated value of $\acute{\sigma}_m(t)$ for task $u_m(t)$. Hence, the values of $v_m^c(t)$ and $o_m^c(t)$ are essentially estimations. Therefore, the values of $o_m^c(t)$ and $v_m^c(t)$ represent an approximate maximum and minimum thresholds, respectively. The estimated values of upper and lower bounds are provided due to the round robin selection procedure of inner queue structure discussed in Section 3.2.

The inner structure of each queue, dedicated to a mobile device at the edge nodes, mirrors the structure previously described for the mobile device queue models. For each time slot t, the value $q_{m,n}^e(t)$ denotes the queue length (in bits) corresponding to tasks offloaded from mobile device m to edge node n. This value reflects the queue state at the end of time slot t and remains constant throughout the time slot, for the purpose of real decision-making.

The computational model at the edge node is defined according to Tang and Wong (2022). For any given edge node n, the queue for a mobile device m at time slot $t \in T$ is identified as an active queue if either a task has arrived at edge node n during time slot t ($q_{m,n}^e(t)$) or the corresponding queue is still occupied at the termination of the previous time slot $t-1$ ($q_{m,n}^e(t-1) > 0$). The variable $X_n(t) = |\chi_n(t)|$ represents the count of operational queues at edge node n during time slot t. Each edge node is assumed to have a single CPU dedicated to task processing. According to the generalized processor capacity sharing model (Parekh and Gallager, 1993), the CPU's processing power at edge node n is distributed among the active queues within the set $\chi_n(t)$ during each time slot t. The processing capacity assigned to each queue at an edge node can vary from one time slot to another and is unpredictable, due to the changing number of active queues at each edge node. Let f_n^e denote the processing capacity of edge node $n \in N$ in CPU cycles per second. When the time slot t ended, $b_{m,n}^{edge}(t)$ bits of tasks are dropped from the computation queue assigned to mobile device m. Consequently, the queue length for mobile device m at edge node n is updated based on the bits that have arrived, been processed, and been dropped, as shown in Eq. (3). f_n^{device} is assumed to be known by mobile device m for each $n \in N$.

$$q_{m,n}^e(t) = \max\{0,\ q_{m,n}^e(t-1) + \sigma_{m,n}^e(t) - \frac{f_n^e \Delta}{\rho_m X_n(t)}(m \in \chi_n(t)) - b_{m,n}^{edge}(t)\}. \quad (3)$$

3.4. Transmission model

Tasks that require offloading to edge nodes are chosen from the transmission queue and transmitted over a wireless network link. The transmission is subject to path loss and small-scale fading (Li et al., 2021b; Mao et al., 2017b), and the channel gain between the mobile device and the edge node is represented by $|h_{m,n}|^2$ and assumed to be constant. The primary reason for adopting this assumption in our work was to simplify the problem and align with common practices in state-of-the-art research, where many studies assume constant channel gain to facilitate tractable analysis and focus on other aspects of MEC, such as computation offloading and resource allocation. This assumption enables us to derive closed-form solutions and gain initial insights into the behavior of system. The transmission power (P), bandwidth (W), and received noise power (ς^2) – all assumed to be constant – affect the data transfer rate $r_{m,n}^t$ between mobile device m and edge node n. Thus, the transmission rate is invariant and constant (Wang et al., 2022; Tang and Wong, 2022), and computed as Eq. (4) (Zhao et al., 2020).

$$r_{m,n}^t = W \log_2 (1 + \frac{|h_{m,n}|^2 P}{\varsigma^2}),\ m \in |M|,\ n \in |N|. \quad (4)$$

The transmission queue in the wireless network operates similarly to the computation queue. Similar to Tang and Wong (2022) and Zhao et al. (2020), we assume that a task can be completed if it is allocated the least amount of required processing capacity. However, only the number of CPU cycles needed to complete a task is taken as the unit of processing capacity in Limin and Ke (2023). Herein, the computational size of a task in bits is an influencing factor. In other words, the total processing capacity allocated to mobile device m by edge node n is expected to be less than the size of task $u_{m,n}^e(t)$ from the start of processing to just one time slot before processing is completed or the task is dropped. This is greater than the size of task $u_{m,n}^e(t)$ from the start of processing to the time slot when processing is completed or the task is dropped.

To evaluate the robustness of the proposed approach under variational network conditions, we conducted simulations in highly variable environments, where latency and other system conditions were subject to significant variations. Our results show that the proposed system dynamically adapts to these changes by continuously monitoring network metrics and adjusting task offloading decisions accordingly. The reward function, which incorporates network variability, ensures that long-term performance is optimized even in the presence of unpredictable network conditions. This adaptability is particularly crucial in real-world MEC environments, where network stability cannot always be guaranteed.

4. System architecture

At the start of each time slot $t \in T$, mobile device $m \in M$ assesses its current condition. The specific parameters defining the state will be discussed in Section 4.1. For each newly arrived task, an offloading decision must be made as an action, as describe in Section 4.2. The observed state and the chosen action of the mobile device result in a specific cost or reward, which is inversely related to the cost. The cost depends on the task processing delay according to Tang and Wong (2022) if the task is completed, otherwise on a constant penalty value as introduce in Section 4.3. The primary objective of the mobile device is to optimize the policy that maps observed states to chosen actions to minimize the expected long-term cost, i.e., maximize the expected long-term reward.

4.1. State definition

The proposed system presumes that every edge node announces the count of its active queues to all mobile devices during each time slot. Each mobile device keeps track of the load history of all edge nodes within a specified range of time slots. This load history is represented by a matrix called $H(t)$, where each element represents the load level of an edge node at a specific time slot. This matrix can be constructed locally by each mobile device using the broadcasted information, and the signaling overhead is small. The edge node has all M queues operational, this information can be represented using $\lfloor \log_2 M \rfloor + 1$ bits. $s_m(t)$ is the current observed state by each mobile device at the start of each time slot. $s_m(t)$ is consisted of the estimated tasks size ($\acute{\sigma}_m(t)$), potential queuing delays ($o^c_m(t)$ and $o^t_m(t)$), the queue length at the edge node in the previous time slot ($q^e_m(t-1) = (q^e_{m,n}(t-1), n \in N)$), and the load history matrix. The queue length at the edge node can be computed locally based on the constraint shown in Eq. (3), and the actual size of a transmitted task is inferred from the transmission delay and the transmission rate of the mobile device. The system also keeps track of dropped tasks based on arrival times and deadline constraints. Hence, the value of $b^{edge}_{m,n}(t)$ can be inferred locally by the device.

4.2. Action definition for offloading decision

In each time slot t, a mobile device m must make an offloading decision for the newly arrived task $u_m(t)$. This decision involves selecting between local processing and transmitting the task to an edge node. If offloading is chosen, the device also determines the specific edge node for task execution. We denote the action vector by Eq. (5).

$$a_m(t) = \{a | a \in \{0, 1, 2, \dots, |N|\}\}. \tag{5}$$

4.3. Cost calculation based on task processing delay

When a task is executed, its processing delay is calculated as the time between its arrival and the moment it is processed. We denote the processing delay of task $u_m(t)$ for mobile device m as $g_m(s_m(t), a_m(t))$. This delay is quantified by the number of time slots and is determined based on the observed state $s_m(t)$ and the action $a_m(t)$ taken by the device. Additionally, the cost variable associated with the processed task is defined in accordance with Eq. (6).

$$c_m(s_m(t), a_m(t)) = g_m(s_m(t), a_m(t)). \tag{6}$$

Additionally, the cost associated with the dropped task is defined as Eq. (7)

$$c_m(s_m(t), a_m(t)) = C, \tag{7}$$

where $C > 0$ represents a penalty with a constant value. For brevity, $c_m(t)$ is used to represent the cost variable $c_m(s_m(t), a_m(t))$, for the rest of this paper. It is trivial that a reward function such as $r_m(t)$ is defined as $r_m(t) = -c_m(t)$ which is the actual value used in the implementations for computing the loss function. The computational size of each task directly affects both the computation time and the transmission time, thereby influencing the overall computation delay. An increase in delay also reflects the impact of network congestion on queueing. Given this model, a straightforward approach to defining both reward and cost can be simple while still providing meaningful insights into the nature of each observed state.

5. Proposed offloading approach

In this section, we present a DRL offloading approach based on two actor–critic deterministic policy gradient algorithms. Each mobile device, denoted as m, independently applies these algorithms to make an offloading decision in a distributed manner. Herein, we illustrate the implementation details and internal neural network models of the proposed DDPG and D4PG algorithms. Furthermore, we highlight the improvements of the D4PG algorithm compared to the DDPG algorithm.

5.1. Proposed DDPG-based Algorithm

DDPG is an off-policy algorithm that uses an actor–critic approach to learn a Q-function and policy. Our proposed DDPG algorithm consists of an actor model, a critic model, two target networks, one for the actor and one for the critic, and a replay buffer. Fig. 2 shows the flow of proposed DDPG algorithm. The actor and actor target networks are denoted as μ_θ and $\mu_{\theta_{Tgt}}$, respectively. Similarly, the critic and critic networks are represented as Q_ϕ and $Q_{\phi_{Tgt}}$, respectively. Thus, the two parameters θ and ϕ correspond to the weights of the actor and critic models, respectively. The policy determined by the actor networks are converted to integers to comply with the denotation in Eq. (5). Given a state $s_m(t)$ at mobile device m during time slot t, the DDPG algorithm aims to learn a deterministic policy $\mu_\theta(s_m(t))$ that maximizes the $Q_\phi(s_m(t), a_m(t))$. For brevity, we denote $\mu_\theta(s_m(t))$ and $Q_\phi(s_m(t), a_m(t))$ as $\mu_\theta(s)$ and $Q_\phi(s, a)$, respectively.

In order to alleviate the computational load at mobile devices, the proposed algorithm leverages edge nodes to assist in the training process. Each mobile device maintains a copy of the actor network, which is responsible for offloading decision for each task arriving at each time slot $t \in T$. The weights of this network are learned through the training process performed at the designated edge node. The edge node can transmit the learned weights to the mobile device at constant intervals through a wireless network interface.

Every mobile device $m \in M$ is assigned an edge node $n_m \in N$ for support during training. Furthermore, $M_n \subset M$ represents the set of mobile devices being trained at edge node n, defined as $M_n = \{m \in M \mid n_m = n\}$.

In the proposed algorithm, every edge node n maintains a replay memory R_m for each mobile device m, where it keeps track of the observed transitions, i.e. the experience, at each time slot t as $(s_m(t), a_m(t), c_m(t), s_m(t+1), d_m(t+1))$ where $d_m(t)$ indicates whether the state $s_m(t+1)$ is a final state or not. In the proposed approach, the final state is defined as the last time slot in each training episode.

Each edge server n_m manages four neural networks for each mobile device $m \in M_n$ including an actor network $Actor_m$, a critic network $Critic_m$, an actor target network $ActorTgt_m$, and a critic target network $CriticTgt_m$. It is worth noting that $Actor_m$ and $ActorTgt_m$ have similar neural network structures but have different network parameter vectors θ_m and $\theta_{m_{Tgt}}$. Similarly, this distinction applies to $Critic_m$ and $CriticTgt_m$ networks and their corresponding network parameter vectors ϕ_m and $\phi_{m_{Tgt}}$.

5.2. Proposed D4PG-based Algorithm

D4PG is an off-policy algorithm that builds on the DDPG. It introduces a distributional critic network, Prioritized Experience Replay (PER), and multiple distributed actors for independent learning. PER assigns non-uniform priorities to each experience in the replay buffer. The idea is to give higher priority to those experiences that have resulted in worse value loss during training. The intuition behind this approach is that experiences leading to higher value loss indicate unlearned or poorly learned aspects of a specific state space by the agent. Consequently, these experiences are given higher priority, increasing the likelihood that they will be sampled in the next round of training, thereby facilitating further learning. Naturally, experiences that have not yet been sampled for training are also given the highest priority.

The D4PG algorithm incorporates a critic layer in the network, which produces a distribution of possible Q-values for each input state–action pair. A distributional operator is employed to convert the mapping from state–action pairs to distributions, resulting in another distributional Q-function of the same form. The detailed definitions of these operators and their impact on the loss functions in our proposed approach are discussed in the following sections.

The definition of the observed states and their specifications is similar to that of the DDPG discussed earlier. Furthermore, similar

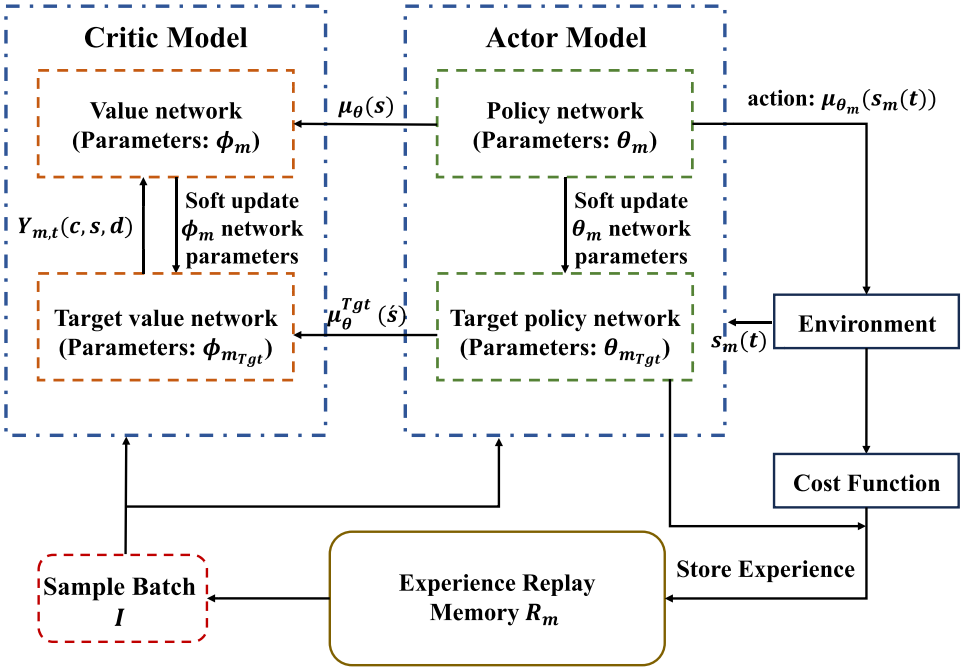

Fig. 2. Schematic relation of the actor and critic models in the DDPG algorithm.

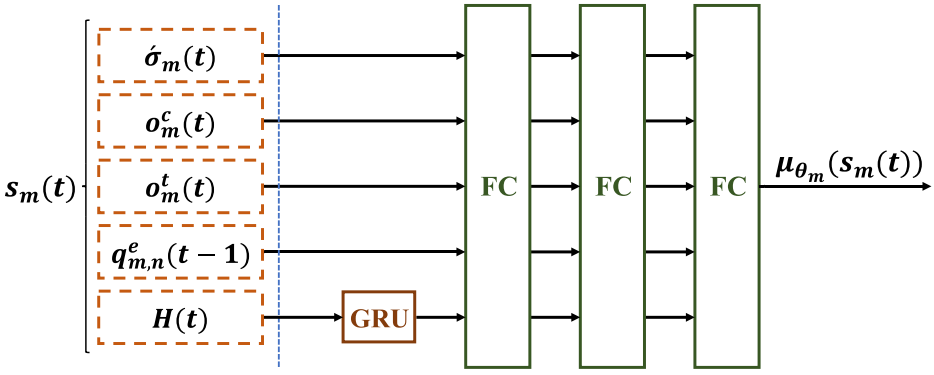

Fig. 3. Actor and actor target internal network structures for both DDPG and D4PG algorithms.

to the DDPG model, the learning of D4PG is assisted by edge nodes. Similarly to what discussed for the DDPG structure, the actor and actor target networks use the similar neural network structures, and their parameter vectors are denoted as θ_m^δ and $\theta_{m_{Tgt}}^\delta$. The critic and critic target networks use the same neural network structure and their parameter vectors are denoted as ϕ_m^δ and $\phi_{m_{Tgt}}^\delta$.

5.3. Depiction of neural network structures in DDPG and D4PG

The policy neural network is designed to map each state to a particular action or policy. The different variables that make up each state are input into the network individually. The load level matrix for edge nodes goes through a GRU layer, which is utilized to forecast the future load levels of edge nodes. GRU is a commonly used approach for learning sequential series behaviors, which is preferred over other sequence models such as LSTM due to its lower computational complexity. Fig. 3 shows the neural network structure used for the actor and actor target networks for both DDPG and D4PG algorithms.

As mentioned earlier, the neural network architecture of the actor and actor target networks are identical, differing only in their network parameter vectors. The predicted load levels and other state values then pass through three Fully Connected (FC) layers. The Rectified Linear Unit (ReLU) activation function is applied to the first two FC layers, and a sigmoid activation function is employed for the third FC layer. The network parameters for this structure, associated with mobile device m are denoted by θ_m and $\theta_{m_{Tgt}}$ for its corresponding actor target network. Thus, receiving $s_m(t)$ as an input, $Actor_m$ produces an output denoted by $\mu_{\theta_m}(s_m(t))$, or for brevity $\mu_{\theta_m}(t)$.

The critic network used in DDPG has a similar structure to the actor network in terms of the number of layers and their purpose (Fig. 4). However, there are two main differences: an action $a_m(t)$ is included as one of the inputs to the neural network structures; and the last FC layer does not use any activation functions, as it directly computes the Q-value at this step. For clarity, we refer to the last FC layer in actor and critic networks as the action layer and the value layer, respectively.

In comparison to the DDPG, the critic network for D4PG has a different structure in its last FC layer. It gives more than one value as output of the Q-function, representing a distribution of possible values. The distributional operator takes the distribution generated by the target critic network as input and projects it onto another set of distribution, referred to as the target distributions.

All explanations of the internal neural network structures of the actor and critic networks apply to their respective target networks as well.

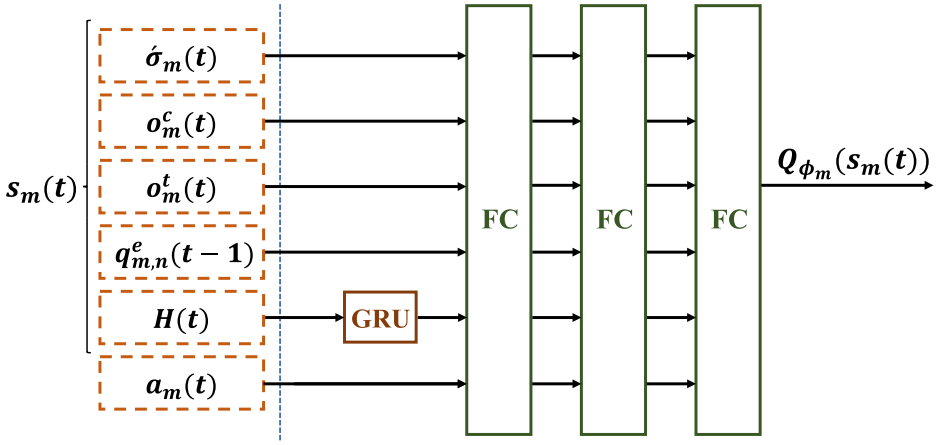

Fig. 4. Critic and critic target network structure for the DDPG-based algorithm.

5.4. Action selection algorithm

The action selection algorithm employed at each mobile device described in this section applies to both DDPG and D4PG algorithms. However, the learning algorithms executed at the edge nodes follow different steps, as discussed in Section 5.5. We examine numerous training episodes, with each episode comprising a defined number of time slots. The overall number of episodes is represented by E. At the beginning of each episode, the following initialization is applied.

$$s_m(1) = (\acute{\sigma}_m(1), o^c_m(1), o^t_m(1), q^e_{m,n}(0), H(1)), \tag{8}$$

where $s_m(1)$ is the initial state of each episode, namely the state of the first time slot in each episode. The value of $q^e_{m,n}(0)$ is set to 0, and the matrix $H(1)$ is initially set as a zero matrix with dimensions $T^{step} \times N$. Additionally, $\acute{\sigma}_m(1)$ is set to 0.

Algorithm 1: Action Selection Policy for Each Mobile Device

1. Repeat the following steps for each episode $= 1, 2, ..., E$
2. Set up $s_m(1)$ according to Eq. (8)
3. **for** the time slot $t \in T$ **do**
4. **if** the device receives new task $u_m(t)$ **then**
5. Request the actor parameters from edge node n_m
6. Receive the request parameter vector θ_m
7. Observe $H(t)$ and compute $s_m(t)$
8. Select action $a_m(t)$ according to Eq. (9)
9. **end if**
10. Observe and compute the next state $s_m(t+1)$
11. Observe costs $c_m(\acute{t})$ for any time slot $\acute{t} \in \tau_{m,t}$
12. **for** every task $u_m(\acute{t})$ that $\acute{t} \in \tau_{m,t}$ **do**
13. Transmit experience $(s_m(\acute{t}), a_m(\acute{t}), c_m(\acute{t}), s_m(\acute{t}+1), d_m(\acute{t}))$ to n_m
14. **end for**
15. **end for**

The pseudocode of the action selection policy for mobile device m is represented in Algorithm 1. Upon arrival of a task, mobile device m requests updated network parameter weights for the actor (policy) network from the corresponding edge node n_m. The signaling overhead of parameter vectors is considered negligible in this study. Although the parameter request step has no direct impact on the implementation, the concept is discussed for clarity. While update requests generate communication overhead, they are only required at fixed intervals because the learning algorithm at the edge node operates periodically instead of at every time slot. This approach ensures that unnecessary communication is avoided by synchronizing the update intervals with both the learning frequency and the parameter transmission frequency. As a result, the system achieves a balance between maintaining effective updates and minimizing communication overhead, enhancing overall efficiency.

Algorithm 2: DDPG-based Learning Algorithm at Edge Nodes

1. Initialize replay memory R_m for each $m \in M_n$
2. Initialize parameter vector θ_m for $Actor_m$ for each $m \in M_n$
3. Initialize parameter vector ϕ_m for $Critic_m$ for each $m \in M_n$
4. Set actor target parameter vector $\theta_{m_{Tgt}} \leftarrow \theta_m$ for $m \in M_n$
5. Set critic target parameter vector $\phi_{m_{Tgt}} \leftarrow \phi_m$ for $m \in M_n$
6. **while** each episode in $1, 2, ..., E$ **do**
7. **if** receive an experience $(s_m(t), a_m(t), c_m(t), s_m(t+1), d_m(t+1))$ from $m \in M_n$ **then**
8. Store $(s_m(t), a_m(t), c_m(t), s_m(t+1), d_m(t+1))$ in R_m
9. Select a set of experiences I from R_m
10. **for** every experience $i \in I$ **do**
11. Acquire experience $(s_m(i), a_m(i), c_m(i), s_m(i+1), d_m(i+1))$ and compute target value according to Eq. (10)
12. **end for**
13. Set vector $Q^{Tgt}_{\phi_m} := (Q^{Tgt}_{\phi_{m,i}}, i \in I)$
14. Update Q-function by one step of gradient descent according to Eq. (11)
15. Update policy by one step of gradient ascent according to Eq. (12)
16. Update target networks with Eq. (13)
17. **end if**
18. **end while**

In the $Actor_m$ network, mobile device m utilizes parameter vector θ_m to choose the action $a_m(t)$ for $u_m(t)$ as described in Eq. (9).

$$a_m(t) = \begin{cases} \mu_{\theta_m}(s_m(t)), & probability\ of\ \epsilon \\ a\ randomly\ selected\ action\ from\ A, & probability\ of\ 1-\epsilon \end{cases}, \tag{9}$$

where $1-\epsilon$ represents the probability of random exploration. When the next time slot begins, the mobile device m perceives the state $s_m(t+1)$. It is important to note that $u_m(t)$ may require several additional time slots for processing, transmission, or dropping. Consequently, $c_m(t)$ will not be observed in line 11 of Algorithm 1 until $u_m(t)$ is either processed or dropped. Additionally, at time slot $t+1$, the mobile device might observe costs $c_m(\acute{t})$ for certain tasks $u_m(\acute{t})$ that arrived at $\acute{t} < t$. Let $\tau_{m,t} \subset T$ be the set of time slots where tasks $u_m(\acute{t})$ for some $\acute{t} < t$ have either been processed or dropped.

It should be noted that for certain $m \in M$ and $t \in T$, the set $\tau_{m,t}$ might be empty. Additionally, it is important to mention that the transmission of $s_m(\acute{t})$ is carried out by mobile device m and $s_m(\acute{t}+1)$ for each $\acute{t} \in \tau_{m,t}$ as part of the captured experience $(s_m(\acute{t}), a_m(\acute{t}), c_m(\acute{t}), s_m(\acute{t}+1), d_m(\acute{t}+1))$ in line 13 of Algorithm 1, it is not required to send $H(\acute{t})$ and $H(\acute{t}+1)$. These two values are already known since the count of active queues is broadcast by every edge node at each time slot. Thus, each edge node is informed about the historical load levels of the other edge nodes. Although the parameter vector notations in Algorithm 1 are the same as those defined for DDPG, the logic remains consistent and the corresponding parameter vectors could be modified within the context of D4PG.

5.5. The learning algorithms

In this section, we introduce the learning algorithm based on DDPG and D4PG, that is executed at edge node n for mobile device m. The pseudocode of the learning algorithms for DDPG and D4PG is provided in Algorithm 2 and Algorithm 3, respectively.

In Algorithm 2, after initializing $Actor_m$, $Actor_{m_Tgt}$, $Critic_m$, $Critic_{m_Tgt}$ and R_m for mobile device m at edge node n, the procedure waits for a new experience to arrive. The learning procedure is only triggered if new experiences are received from at least one mobile device $m \in M_n$. Hence, it is obvious that triggering the learning procedure in every time slot would be redundant. In practice, this learning procedure can also be performed at specific intervals to align with the update parameters request interval for better consistency, for which more details were discussed in Section 5.4.

The edge node initially samples a random set of experiences symbolized by I. Let $|I|$ represent the size of the experience set I, which is the number of experiences randomly selected from R_m. For each experience $i \in I$, the edge node computes a target value as represented in Eq. (10).

$$Y(c_m(t), s_m(t+1), d_m(t+1)) = c_m(t) + \gamma(1-d_m(t+1))Q^{Tgt}_{\phi_m}(s_m(t+1), \mu^{Tgt}_{\theta_m}(s_m(t+1))). \tag{10}$$

For brevity, we express Eq. (10) as $Y_{m,t}(c,s,d) = c + \gamma(1-d)Q^{Tgt}_{\phi}(\acute{s}, \mu^{Tgt}_{\theta}(\acute{s}))$. This target value is used to compute the value of loss function and update the Q-function by one step of gradient descent. The value (critic) loss function is computed as Eq. (11).

$$L(Q_{\phi_m}, Y(c,s,d)) = \nabla_{\phi} \frac{1}{|I|} \sum_{(s,a,c,\acute{s},d)} (Q_{\phi_m}(s,a) - Y(r,\acute{s},d))^2. \tag{11}$$

Subsequently, the parameters of the policy network are adjusted by one step of gradient ascent as Eq. (12).

$$\nabla_{\theta_m} \frac{1}{|I|} \sum_{s \in I} (Q_{\phi}(s, \mu_{\theta}(s))). \tag{12}$$

The parameters of the actor and critic networks are updated by learning rate parameters α and β. Consequently, the target network parameters are updated using a Polyak value denoted by P as described in Eq. (13).

$$\begin{aligned} \theta_{Tgt} &\longleftarrow P\theta_{Tgt} + (1-P)\theta \\ \phi_{Tgt} &\longleftarrow P\phi_{Tgt} + (1-P)\phi \end{aligned} \tag{13}$$

The overall structure of Algorithm 3 closely mirrors that of Algorithm 2. However, the computation of loss functions for the actor and critic networks leverages a different approach. Additionally, the sampling of batches is influenced by the priorities assigned to each experience in the replay memory. To offer a more detailed explanation of the procedure and to illustrate the differences between these two approaches, we review the general steps while providing relevant details.

Algorithm 3: D4PG-based Learning Algorithm at Edge Nodes

1. Initialize replay memory R^{δ}_m for $m \in M_n$
2. Initialize $Actor_{\delta,m}$ parameter vector θ^{δ}_m for $m \in M_n$
3. Initialize $Critic_{\delta,m}$ parameter vector ϕ^{δ}_m for $m \in M_n$
4. Set actor target parameter vector $\theta^{\delta}_{m_{Tgt}} \leftarrow \theta^{\delta}_m$ for $m \in M_n$
5. Set critic target parameter vector $\phi^{\delta}_{m_{Tgt}} \leftarrow \phi^{\delta}_m$ for $m \in M_n$
6. **while** each episode in $1, 2, ..., E$ **do**
   if receive an experience
7. $(s_m(t), a_m(t), c_m(t), s_m(t+1), d_m(t+1))$ from $m \in M_n$ **then**
8. Retain $(s_m(t), a_m(t), c_m(t), s_m(t+1), d_m(t+1))$ in R^{δ}_m
9. Sample a set of η experiences of length λ I^{δ} from R^{δ}_m
10. Compute the target distributions according to Eq. (14)
11. Update Q-function by one step of gradient descent according to Eq. (15)
12. Update policy by one step of gradient ascent according to Eq. (16)
13. Update target networks with Eq. (17)
14. **end if**
15. **end while**

First, the actor, actor target, critic, critic target networks and replay memory, designated to each m, are initialized. The process of requesting a parameter vector update from mobile device m to its corresponding edge node n is similar to that discussed in the DDPG algorithm. In other words, the architectural specifications considering the information flows between mobile devices and their designated edge nodes remain unchanged, while the utilization of this information for learning and training at each edge node is different. η transitions of length λ can be sampled as I^{δ} where the implementation in this work considers $\lambda = 1$. The conventional representation for such sampling is in the form of $(s^{i:i+\lambda}_m, a^{i:i+\lambda-1}_m, r^{i:i+\lambda-1}_m)$. The target distribution is also conventionally referred to as represented in Eq. (14), where m refers to a mobile device.

$$Y^i_m = \sum_{n=0}^{\lambda-1} (\gamma^n_{\delta} r^{i+n}_m) + \gamma^{\lambda}_{\delta} Q_{\phi_{Tgt}}(s_m(i+\lambda), \theta^{\delta}_m(s_m(i+\lambda))). \tag{14}$$

Subsequently, the value (critic) loss function is defined as Eq. (15).

$$L(Q^{\delta}_{\phi_m}, Y^i_m) = \nabla_{\phi} \frac{1}{|I^{\delta}|} \sum_i G(Y^i_m, Q_{\phi_m}(s_m(i), a_m(i))), \tag{15}$$

where $G(.)$ is a Binary Cross Entropy (BCE) function for computing the distance between two distributions. As discussed earlier, the priority of a learned experience is updated in alignment with this loss value. The parameters of the actor network are computed by Eq. (16).

$$\nabla^{\delta}_{\theta_m} \frac{1}{|I^{\delta}|} \sum_i (\theta^{\delta}_m(s_m(i))) \nabla^{\delta}_a Q_{\phi}(s_m(i), \theta^{\delta}_m(s_m(i))). \tag{16}$$

Consequently, the parameter vectors of the respective actor and target networks of a mobile device m are updated as Eq. (17).

$$\begin{aligned} \theta^{\delta}_{m_{Tgt}} &\longleftarrow P^{\delta}\theta^{\delta}_{m_{Tgt}} + (1-P^{\delta})\theta^{\delta}_m \\ \phi^{\delta}_{m_{Tgt}} &\longleftarrow P^{\delta}\phi^{\delta}_{m_{Tgt}} + (1-P^{\delta})\phi^{\delta}_m \end{aligned} \tag{17}$$

Table 3
Value of the parameters used for simulation.

Parameters/symbols	Default values/ranges
Δ	0.1 s
$f_m^{device}(t)$	2.5 GHz
$f_n^e(t)$	41.8 GHz
$r_{m,n}(t)$	14 Mbps
ρ_m	0.297 Gigacycles per Mbits
$Task\ size$	Varied from 2 to 5 Mbits
$Task\ arrival\ probability$	0.6
M	50
N	5
ζ_m	10 time slots

6. Experimental results

The simulated environment to assess the performance of the proposed approach consists of 50 mobile devices and 5 edge nodes. Each device has a task arrival probability of 0.6 with a deadline of 10 time slots. The length for each time slot is considered to be 0.1 s. The Adam optimizer was employed to update the parameter networks of both algorithms (Tang and Wong, 2022). Additional parameter settings for the environment are detailed in Table 3. The parameters are aligned with Tang and Wong (2022) except for the task arrival probability. The goal is to evaluate the proposed approach in a scenario where tasks arrive frequently enough to challenge the network, but not so frequently that it becomes overwhelmed. This allows us to provide a clear comparison of how the proposed method handles task allocation efficiently compared to state-of-the-art approaches. By striking this balance, we can effectively demonstrate our approach's strengths in managing network resources, highlighting its advantages in a meaningful and relevant way. The simulation code was written in Python 3, and the PyTorch library was used to implement the proposed DRL approach. Furthermore, Based on Fig. 5, it is evident that both proposed DDPG and D4PG-based algorithms show convergence over 1000 episodes of experimentation on a sample of devices, with similar behavior expected across the remaining devices. The D4PG algorithm achieves faster convergence and attains lower loss values compared to the DDPG implementation throughout the training process. This indicates that our D4PG exhibits superior optimization and learning capabilities, particularly in complex environments. While both algorithms eventually approach a similar point of convergence, D4PG does so more efficiently, highlighting its potential for enhanced performance in reinforcement learning tasks. These results, obtained over 1000 episodes, underscore the effectiveness of the proposed methods, with D4PG offering significant advantages in terms of convergence speed and stability across the sampled devices, suggesting comparable performance for the broader system.

In order to evaluate the proposed approach, we compute the average processing delay and drop rates of tasks in three distinct scenarios: (1) different task arrival probabilities, (2) different number of mobile devices, and (3) different task deadlines. In each experiment, two parameters are kept constants, while the remaining one serves as the variable under investigation for measuring the drop rate and the average processing delay. The average processing delay is determined by averaging the processing delays of all tasks handled by all mobile devices throughout the entire experiment. The drop rate is defined as the proportion of tasks that were dropped to the total number of tasks across all devices during the experiment.

The results obtained from simulations allow a comparison between the two proposed DDPG and D4PG approaches with the DRL algorithm presented based on double dueling DQN in Tang and Wong (2022) paper. Two baseline algorithms, namely Local Computing and Random Offloading, are also used for comparison. Local Computing only computes tasks on the mobile device and Random Offloading makes a random decision about the tasks from available resources which are the mobile device and the edge nodes. The DRL algorithm proposed by Tang and Wong (2022) and the baseline algorithms use a simple FIFO queueing structure and assume prior knowledge of the task size upon their arrival, unlike our work. As mentioned earlier, this assumption is not realistic and we have no insight about the task size upon arrival.

Figs. 6 and 7 illustrate the average processing delays and task drop rates, respectively, under varying task arrival probabilities. The proposed algorithms demonstrate superior system performance compared to the DQN-based approach in Tang and Wong (2022) and the two baseline methods. Notably, under the local and random policies, the average processing delays decline beyond a task arrival probability around 0.6, due to the increased task drop rate. This reduction in delays stems from fewer tasks being successfully processed.

Figs. 8 and 9 present the sensitivity of key performance metrics – average processing delay and task drop rate – by varying numbers of mobile devices. Both proposed algorithms outperform the DQN-based approach in Tang and Wong (2022), achieving significantly lower delays and drop rates. Notably, the D4PG-based algorithm exhibits further improvements in both metrics compared to its DDPG-based counterpart. The declining drop rate trend in these algorithms indicates a convergence in performance.

For random offloading, the delay initially increases marginally but reduces beyond approximately 70 devices. This behavior stems from the elevated task load on edge nodes and the system as a whole, which leads to higher task drop rates and consequently reduces the average delay for completed tasks (Fig. 9). The relatively high task arrival probability (0.6) in these experiments further accelerates the observed reduction in processing delays.

Figs. 10 and 11 evaluate the proposed algorithms in the third scenario, which considers varying task deadlines (measured in time slots). As anticipated, the results reveal a clear trend: average processing delays increase while drop rates decrease with longer task deadlines. These extended deadlines enable more time for task processing, preventing drops that would occur under stricter time constraints. The proposed algorithms consistently outperform the DQN-based approach from Tang and Wong (2022), demonstrating superior performance in both drop rates and average delays, with particularly notable improvements in processing delay. Furthermore, the D4PG-based algorithm achieves better average delay performance than the DDPG-based variant.

We further analyze the relative drop rate and average processing delay for locally computed versus offloaded tasks. The relative drop rate for local tasks is calculated as the ratio of dropped tasks to the total number selected for local processing, with an analogous definition for offloaded tasks. This granular comparison provides novel insights into algorithmic performance by revealing different effects on distinct task groups. A comprehensive evaluation of any algorithm's effectiveness requires analysis of its specific impacts across these categories.

Fig. 13 demonstrates that relative drop rates for both local and offloaded tasks follow comparable trends varying task arrival probabilities. However, as observed in Fig. 12, offloaded tasks experience significantly higher average processing delays than their locally computed counterparts. This disparity highlights the substantial impact of system load on transmission latency and computational overhead at edge nodes.

Figs. 14 and 15 present the performance metrics by varying the number of mobile devices. The results reveal two key findings: (1) offloaded tasks consistently exhibit higher average processing delays than locally computed tasks, and (2) the proposed D4PG algorithm shows superior performance in comparison to DDPG in both task categories. Specifically, D4PG achieves lower processing delays across all tasks (Fig. 8), indicating more efficient overall task processing. Notably, under high-load conditions in dense networks (i.e., with increased number of mobile devices), the D4PG algorithm maintains a significantly lower drop rate for offloaded tasks compared to DDPG, as evidenced in Fig.

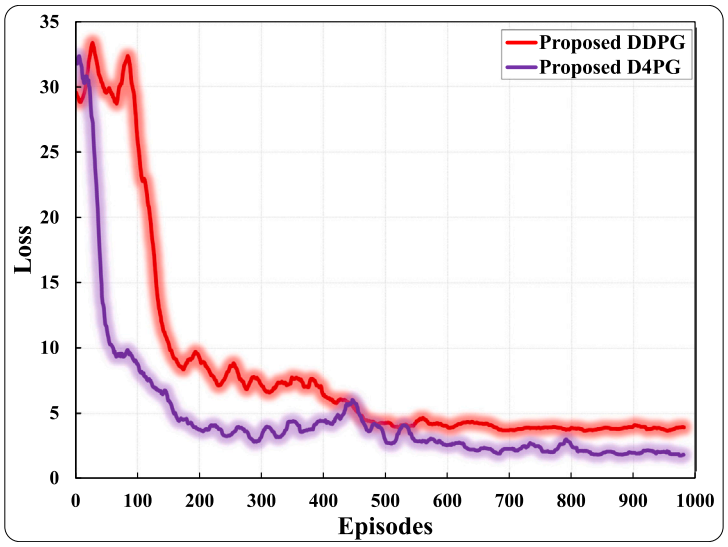

Fig. 5. Convergence analysis of proposed methods.

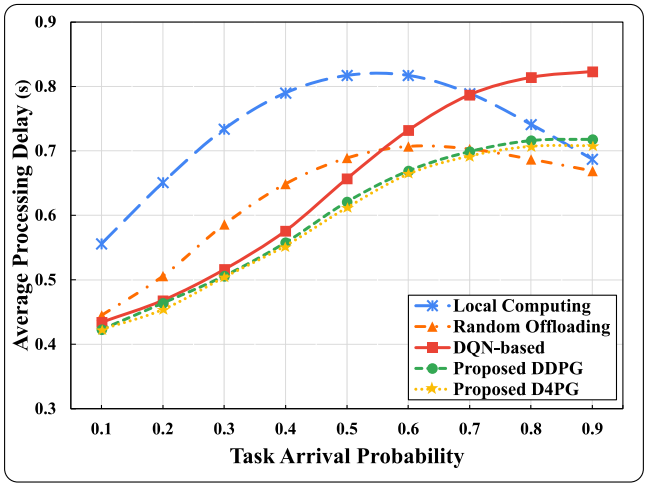

Fig. 6. Average processing delay for different task arrival probabilities.

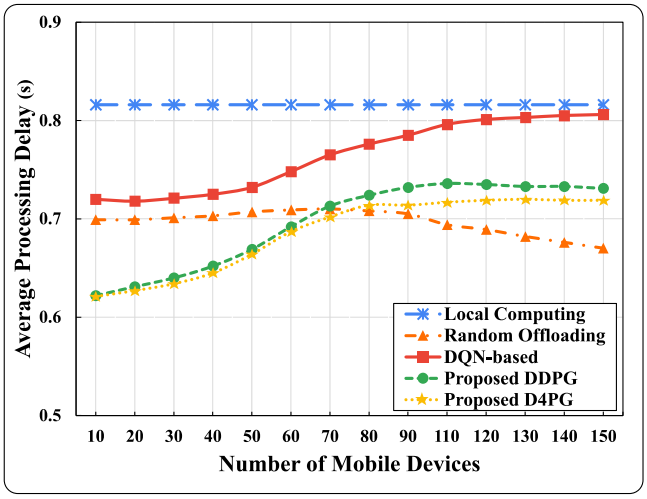

Fig. 8. Average processing delay for different number of mobile devices.

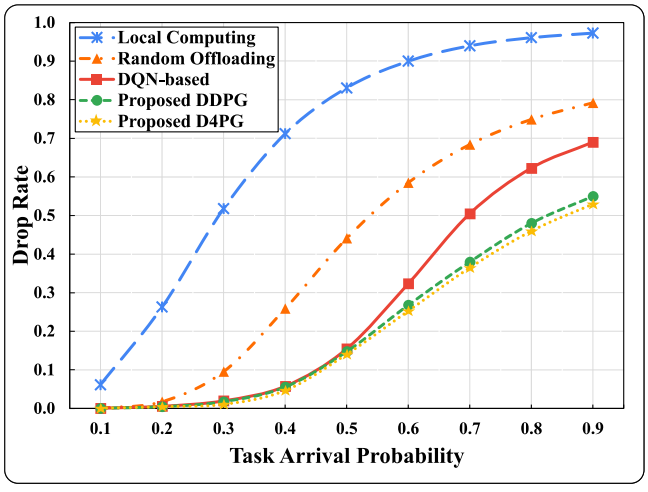

Fig. 7. Drop rate for different task arrival probabilities.

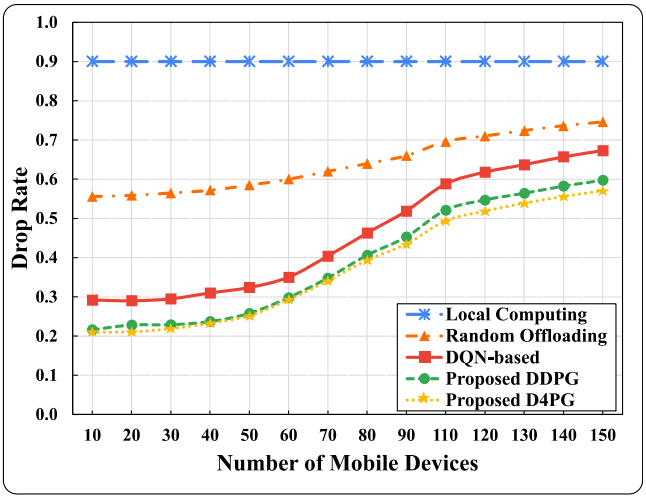

Fig. 9. Drop rate for different number of mobile devices.

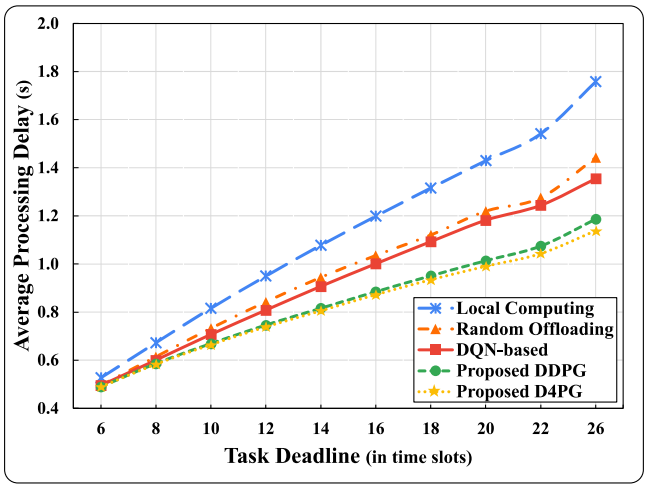

Fig. 10. Average processing delay for different task deadline (in time slots).

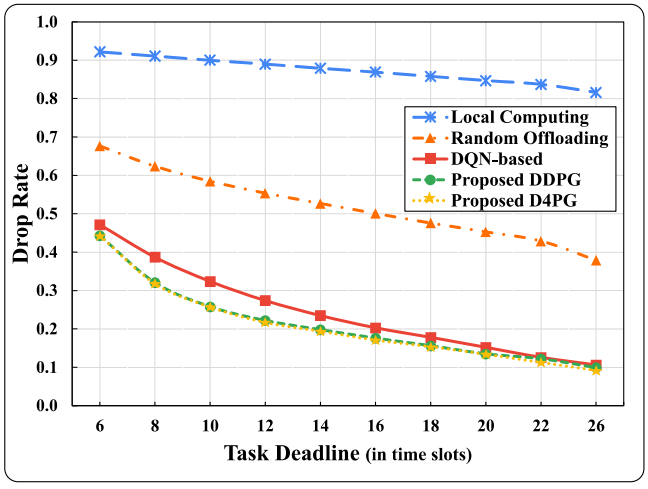

Fig. 11. Drop rate for different task deadline (in time slots).

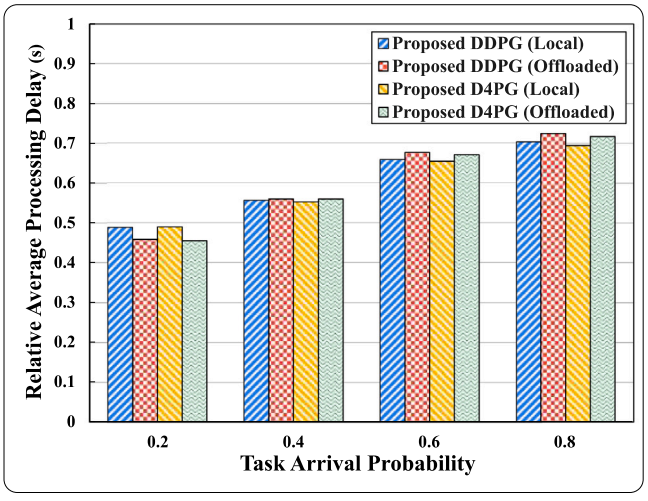

Fig. 12. Relative average processing delay per different task arrival probabilities.

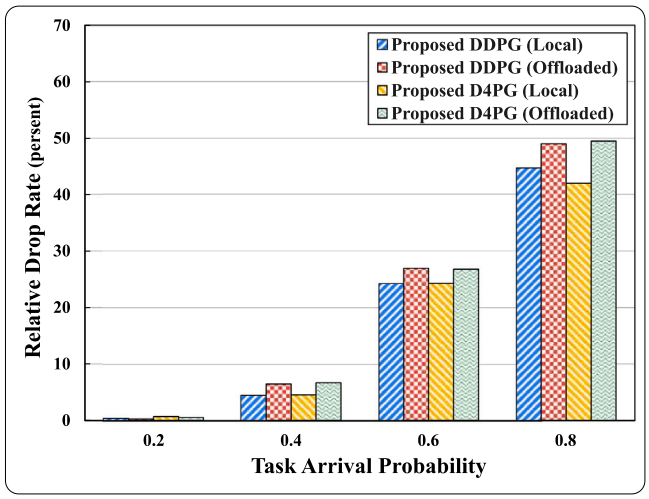

Fig. 13. Relative drop rate per different task arrival probabilities.

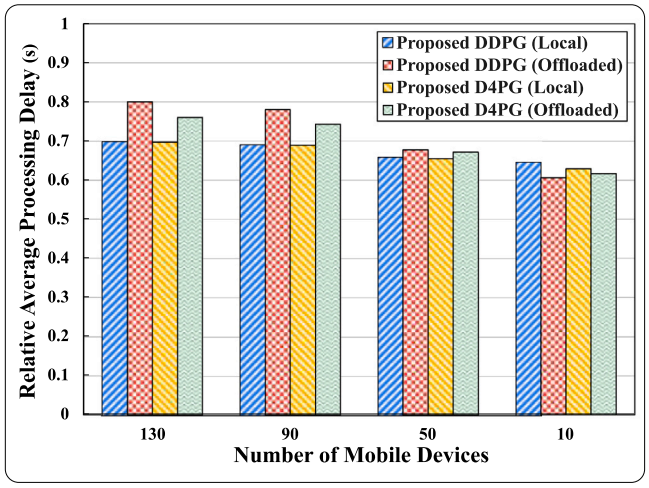

Fig. 14. Relative average processing delay per different number of mobile devices.

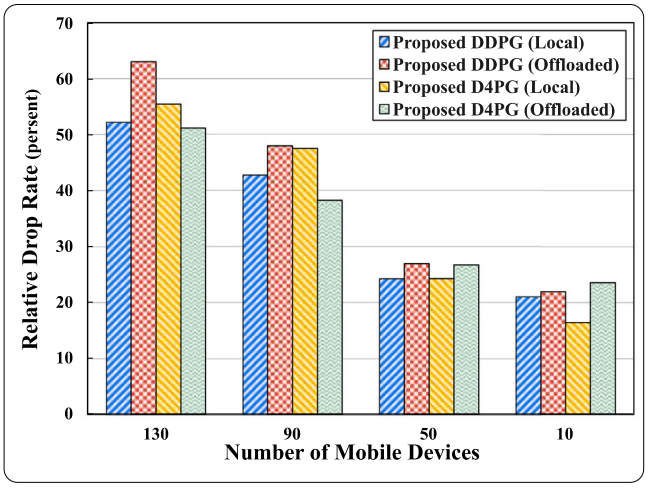

Fig. 15. Relative drop rate per different number of mobile devices.

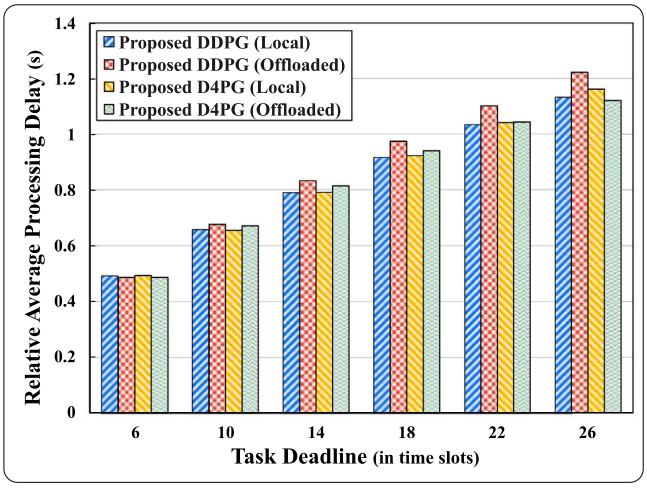

Fig. 16. Relative average processing delay per different task deadlines (in time slots).

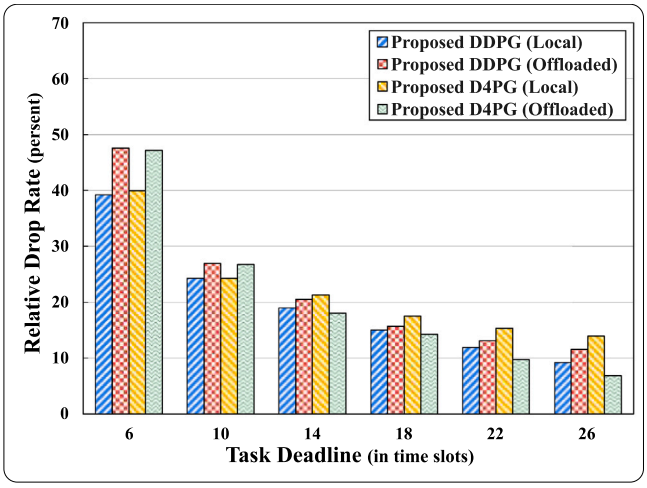

Fig. 17. Relative drop rate per different task deadlines (in time slots).

15. This performance advantage becomes particularly pronounced in scenarios with larger network congestion.

Figs. 16 and 17 illustrate how varying task deadlines affect relative average processing delays and drop rates across different task categories. The results demonstrate that the D4PG algorithm consistently outperforms DDPG, yielding lower processing delays for offloaded tasks. Furthermore, increasing task deadlines leads to significantly greater reductions in drop rates for offloaded tasks in both algorithms compared to their local counterparts. This performance improvement occurs because extended deadlines provide more flexibility in task scheduling and execution. The particularly strong effect observed for offloaded tasks reflects the greater processing capacity and resource availability at edge nodes under identical system conditions, allowing them to more effectively utilize the extended time provided by longer deadlines.

7. Conclusions and future work

This paper explored the task offloading issue within a MEC system, and presented an offloading approach based on DDPG and D4PG algorithms considering a decentralized system structure. Additionally, a size-based scheduling algorithm was introduced to handle the processing or transmission of queued tasks while taking into account the lack of information about task sizes upon arrival, and consequently the need for size estimation. Although this assumption inevitably introduces small errors in the computations, it provides a more realistic representation of the problem. The proposed approach exhibited robust performance despite the erroneous assumptions. The decentralized nature of the considered architecture allows each mobile device to make independent offloading decisions based on local observations of states and load levels at edge nodes. Additionally, we have conducted a detailed analysis by comparing the proposed approach to standard baselines. The results indicate that the proposed algorithms achieve a favorable balance between computational efficiency and performance, making them suitable for deployment in resource-constrained MEC environments. These findings further validate the practicality of our approach and its potential for real-world applications.

Several directions for future research and extension of the current work can be identified. The scheduling algorithm based on task size can be further optimized for efficiency by considering additional factors that influence task prioritization in real-world applications, such as the degree of dependence between subtasks and different deadline constraints. However, a careful trade-off should be managed between the time complexity of the scheduling algorithm and its impact on system performance. Furthermore, future studies can explore the inclusion of task types and applications, incorporating the diverse nature and sources of individual tasks. Building upon the task offloading strategies explored in this work, future research can extend these efforts by further optimizing the trade-off between energy efficiency and computational performance through adaptive resource allocation and intelligent scheduling. While the current work provides a strong foundation for efficient task execution in dynamic MEC environments, integrating energy-aware task migration and dynamic workload balancing can further enhance system sustainability, especially in scenarios with stricter energy constraints.

Evaluating the proposed algorithms on real computational tasks in a pre-configured demo system would provide practical insights and a more pragmatic understanding of mobile device behavior. Another possible extension is to consider a communication model to accurately study the effect of signaling overhead in the transmitting parameter vectors and the number of active queues broadcasting by the edge nodes. In addition, to reduce the dependency of the mobile devices action selection policy on data sent by edge nodes, each mobile device could predict a load coefficient per edge node based on its past interactions. This would allow mobile devices to make even more independent offloading decisions while still adhering to the state model presented in this work. Furthermore, the communication overhead between mobile devices and edge nodes can be reduced by utilizing techniques such as deep compression and pruning, which have shown potential in reducing the computational complexity and communication overhead of Deep Neural Networks (DNNs) used by DRL algorithms. Utilizing these methods would increase the efficiency of algorithms deployed at the edge nodes. Finally, while the presented work assumes a uniform distribution for task sizes to provide a foundational and tractable analysis, future work could explore more complex real-world distributions, such as normal, exponential, or Pareto, to further refine task size estimation and enhance the robustness of offloading strategies. This extension would build upon the current framework and provide deeper insights into system performance under diverse workload patterns.

CRediT authorship contribution statement

Mohsen Darchini-Tabrizi: Writing – review & editing, Writing – original draft, Validation, Methodology, Data curation, Conceptualization. **Amirhossein Roudgar:** Writing – original draft, Visualization, Software, Methodology, Formal analysis, Data curation. **Reza Entezari-Maleki:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Methodology. **Leonel Sousa:** Writing – review & editing, Validation, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Abbas, A., Raza, A., Aadil, F., Maqsood, M., 2021. Meta-heuristic-based offloading task optimization in mobile edge computing. Int. J. Distrib. Sens. Networks 17 (6).

Abu-Taleb, N.A., Abdulrazzak, F.H., Zahary, A.T., Al-Mqdashi, A.M., 2022. Offloading decision making in mobile edge computing: A survey. In: International Conference on Emerging Smart Technologies and Applications (ESmarTA). Ibb, Yemen, pp. 1–8.

Alsadie, D., 2024. Efficient task offloading strategy for energy-constrained edge computing environments: A hybrid optimization approach. IEEE Access 12, 85089–85102.

Bi, S., Huang, L., Wang, H., Zhang, Y.-J.A., 2021. Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks. IEEE Trans. Wirel. Commun. 20 (11), 7519–7537.

Carmo, P.R.X., Bezerra, D.F., Filho, A.T.O., Freitas, E., Silva, M., Dantas, M., Oliveira, B., Kelner, J., Sadok, D.F., Souza, R., 2024. Living on the edge: A survey of digital twin-assisted task offloading in safety-critical environments. J. Netw. Comput. Appl. 232 (104024).

Chen, W., Shen, G., Chi, K., Zhang, S., Chen, X., 2022. DRL based partial offloading for maximizing sum computation rate of FDMA-based wireless powered mobile edge computing. Int. J. Comput. Telecommun. Netw. 214 (109158).

Chen, Z., Yang, Y., Xu, J., Chen, Y., Huang, J., 2025. Task offloading and resource pricing based on game theory in UAV-assisted edge computing. IEEE Trans. Serv. Comput. 18 (01), 440–452.

Feng, C., Han, P., Zhang, X., Yang, B., Liu, Y., Guo, L., 2022. Computation offloading in mobile edge computing networks: A survey. J. Netw. Comput. Appl. 202 (103366).

Gao, X., Ang, M.C., Althubiti, S.A., 2023. Deep reinforcement learning and Markov decision problem for task offloading in mobile edge computing. J. Grid Comput. 21 (78).

Gasmi, K., Dilek, S., Tosun, S., Ozdemir, S., 2022. A survey on computation offloading and service placement in fog computing-based IoT. J. Supercomput. 78 (2), 1983–2014.

Hamdi, A.M.A., Hussain, F.K., Hussain, O.K., 2022. Task offloading in vehicular fog computing: State-of-the-art and open issues. Future Gener. Comput. Syst. 123, 201–212.

Hortelano, D., de Miguel, I., Barroso, R.J.D., Aguado, J.C., Merayo, N., Ruiz, L., Asensio, A., Masip-Bruin, X., Fernández, P., Lorenzo, R.M., Abril, E.J., 2023. A comprehensive survey on reinforcement learning-based computation offloading techniques in edge computing systems. J. Netw. Comput. Appl. 216 (103669).

Hu, H., Song, W., Wang, Q., Zhou, F., Hu, R.Q., 2020. Mobility-aware offloading and resource allocation in MEC-enabled IoT networks. In: International Conference on Mobility, Sensing and Networking. MSN, Tokyo, Japan, pp. 554–560.

Huang, Q., Xu, X., Chen, J., 2021. Learning-aided fine-grained offloading for real-time applications in edge-cloud computing. Wirel. Netw. 30 (5), 3805–3820.

Kazmi, S., Iqbal, M., Coleri, S., 2020. Total transmission time minimization through relay selection for full-duplex wireless powered cooperative communication networks. In: Ad-Hoc, Mobile, and Wireless Networks. Cham, pp. 257–268.

Kim, S., 2022. Bargaining game based offloading service algorithm for edge-assisted distributed computing model. IEEE Access 10, 63648–63657.

Lai, S., Huang, L., Ning, Q., Zhao, C., 2024. Mobility-aware task offloading in MEC with task migration and result caching. Ad Hoc Netw. 156, 103411.

Li, J., Chen, Z., Liu, X., 2021a. Deep reinforcement learning for partial offloading with reliability guarantees. In: IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking. New York City, NY, USA, pp. 1027–1034.

Li, L., Quek, T.Q.S., Ren, J., Yang, H.H., Chen, Z., Zhang, Y., 2021b. An incentive-aware job offloading control framework for multi-access edge computing. IEEE Trans. Mob. Comput. 20 (1), 63–75.

Li, Z., Zhu, Q., 2020. Genetic algorithm-based optimization of offloading and resource allocation in mobile-edge computing. Inform. 11 (2).

Li, J., Zhu, M., Liu, J., Liu, W., Huang, B., Liu, R., 2024. Blockchain-based reliable task offloading framework for edge-cloud cooperative workflows in iomt. Inform. Sci. 668 (120530).

Lim, D., Lee, W., Kim, W.-T., Joe, I., 2022. DRL-OS: A deep reinforcement learning-based offloading scheduler in mobile edge computing. Sensors 22 (23).

Limin, J., Ke, Z., 2023. Enhanced DQN in task offloading across multi-tier computing networks. In: 20^{Th} International Computer Conference on Wavelet Active Media Technology and Information Processing. ICCWAMTIP, pp. 1–6.

Liu, J., Mao, Y., Zhang, J., Letaief, K.B., 2016. Delay-optimal computation task scheduling for mobile-edge computing systems. In: IEEE International Symposium on Information Theory. ISIT, Barcelona, Spain, pp. 1451–1455.

Liu, J., Mi, Y., Zhang, X., Li, X., 2024. Task graph offloading via deep reinforcement learning in mobile edge computing. Future Gener. Comput. Syst. 158, 545–555.

Liu, Y., Wang, S., Huang, J., Yang, F., 2018. A computation offloading algorithm based on game theory for vehicular edge networks. In: IEEE International Conference on Communications. Kansas City, MO, USA, pp. 1–6.

Luo, C., Zhang, J., Cheng, X., Hong, Y., Chen, Z., Xing, X., 2024. Computation offloading in resource-constrained edge computing systems based on deep reinforcement learning. IEEE Trans. Comput. 73 (1), 109–122.

Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B., 2017a. A survey on mobile edge computing: The communication perspective. IEEE Commun. Surv. Tutorials 19 (4), 2322–2358.

Mao, Y., Zhang, J., Song, S.H., Letaief, K.B., 2017b. Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems. IEEE Trans. Wirel. Commun. 16 (9), 5994–6009.

Merluzzi, M., di Pietro, N., Di Lorenzo, P., Strinati, E.C., Barbarossa, S., 2022. Discontinuous computation offloading for energy-efficient mobile edge computing. IEEE Trans. Green Commun. Netw. 6 (2), 1242–1257.

Moon, S., Lim, Y., 2021. Task partitioning for migration with collaborative edge computing in vehicular networks. In: IEEE 3rd Eurasia Conference on IOT, Communication and Engineering. ECICE, Yunlin, Taiwan, pp. 102–107.

Nandi, P.K., Reaj, M.R.I., Sarker, S., Razzaque, M.A., or Rashid, M.M., Roy, P., 2024. Task offloading to edge cloud balancing utility and cost for energy harvesting internet of things. J. Netw. Comput. Appl. 221 (103766).

Pang, X., Wang, Z., Li, J., Zhou, R., Ren, J., Li, Z., 2022. Towards online privacy-preserving computation offloading in mobile edge computing. In: IEEE Conference on Computer Communications. London, United Kingdom, pp. 1179–1188.

Parekh, A.K., Gallager, R.G., 1993. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. IEEE/ACM Trans. Netw. 1 (3), 344–357.

Penmetcha, M., Min, B., 2021. A deep reinforcement learning-based dynamic computational offloading method for cloud robotics. IEEE Access 9, 60265–60279.

Porambage, P., Okwuibe, J., Liyanage, M., Ylianttila, M., Taleb, T., 2018. Survey on multi-access edge computing for internet of things realization. IEEE Commun. Surv. Tutorials 20 (4), 2961–2991.

Qiao, X., Ren, P., Dustdar, S., Chen, J., 2018. A new era for web AR with mobile edge computing. IEEE Internet Comput. 22 (4), 46–55.

Qu, G., Wu, H., Li, R., Jiao, P., 2021. DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing. IEEE Trans. Netw. Serv. Manag. 18 (3), 3448–3459.

Saleem, O., Munawar, S., Tu, S., Ali, Z., Waqas, M., Abbas, G., 2022. Intelligent task offloading for smart devices in mobile edge computing. Int. Wirel. Commun. Mob. Comput. (IWCMC) 312–317.

Singh, R., Armour, S., Khan, A., Sooriyabandara, M., Oikonomou, G., 2020. Heuristic approaches for computational offloading in multi-access edge computing networks. In: IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications. London, UK, pp. 1–7.

Singh, S., Kim, D.H., 2021. Profit optimization for mobile edge computing using genetic algorithm. In: IEEE Region 10 Symposium. TENSYMP, Jeju, Korea, Republic of, pp. 1–6.

Tang, M., Wong, V.W.S., 2022. Deep reinforcement learning for task offloading in mobile edge computing systems. IEEE Trans. Mob. Comput. 21 (6), 1985–1997.

Tong, Z., Deng, X., Ye, F., Basodi, S., Xiao, X., Pan, Y., 2020. Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment. Inform. Sci. 537, 116–131.

Wang, J., Hu, J., Min, G., Zhan, W., Zomaya, A.Y., Georgalas, N., 2022. Dependent task offloading for edge computing based on deep reinforcement learning. IEEE Trans. Comput. 71 (10), 2449–2461.

Wei, J., Liang, X., 2024. Research on task offloading delay in the IoV based on a queuing network. IEEE Access 12, 31324–31333.

Wu, H., Geng, J., Bai, X., Jin, S., 2024. Deep reinforcement learning-based online task offloading in mobile edge computing networks. Inform. Sci. 654 (119849).

Xu, X., Liu, K., Dai, P., Jin, F., Ren, H., Zhan, C., Guo, S., 2023. Joint task offloading and resource optimization in NOMA-based vehicular edge computing: A game-theoretic DRL approach. J. Syst. Archit. 134, 102780.

Yang, Z., Bai, W., 2022. Distributed computation offloading in mobile fog computing: A deep neural network approach. IEEE Commun. Lett. 26 (3), 696–700.

Yang, G., Hou, L., He, X., He, D., Chan, S., Guizani, M., 2021. Offloading time optimization via Markov decision process in mobile-edge computing. IEEE Internet Things J. 8 (4), 2483–2493.

Yeganeh, S., Sangar, A.B., Azizi, S., 2023. A novel Q-learning-based hybrid algorithm for the optimal offloading and scheduling in mobile edge computing environments. J. Netw. Comput. Appl. 214 (103617).

Zhang, Q., Gui, L., Zhu, S., Lang, X., 2021. Task offloading and resource scheduling in hybrid edge-cloud networks. IEEE Access 9, 85350–85366.

Zhao, L., Huang, S., Meng, D., Liu, B., Zuo, Q., Leung, V.C.M., 2024. Stackelberg-game-based dependency-aware task offloading and resource pricing in vehicular edge networks. IEEE Internet Things J. 11 (19), 32337–32349.

Zhao, R., Wang, X., Xia, J., Fan, L., 2020. Deep reinforcement learning based mobile edge computing for intelligent internet of things. Phys. Commun. 43 (101184).

Zhou, W., Fang, W., Li, Y., Yuan, B., Li, Y., Wang, T., 2019. Markov approximation for task offloading and computation scaling in mobile edge computing. Mob. Inf. Syst. 1–12.

Zhu, B., Chi, K., Liu, J., Yu, K., Mumtaz, S., 2022. Efficient offloading for minimizing task computation delay of NOMA-based multiaccess edge computing. IEEE Trans. Commun. 70 (5), 3186–3203.

Zhu, A., Wen, Y., 2021. Computing offloading strategy using improved genetic algorithm in mobile edge computing system. J. Grid Comput. 19 (38).