Adaptive Prioritization and Task Offloading in Vehicular Edge Computing Through Deep Reinforcement Learning

Ashab Uddin, *Graduate Student Member, IEEE*, Ahmed Hamdi Sakr, *Senior Member, IEEE*, and Ning Zhang, *Senior Member, IEEE*

***Abstract*—Vehicular edge computing enables real-time decision-making by offloading vehicular computation tasks to edge servers along roadways. This paper focuses on optimizing offloading and scheduling these tasks, with an emphasis on task prioritization to maximize task completion within deadlines while minimizing latency and energy consumption across all priority levels. We propose a prioritized Deep Q-Network (DQNP) that optimizes long-term rewards through a priority-scaled reward system for each priority level, guiding the deep reinforcement learning (DRL) agent to select optimal actions. The model dynamically adjusts task selection based on environmental conditions, such as prioritizing tasks with higher deadlines in poor channel states, ensuring balanced and efficient offloading across all priority levels. Simulation results demonstrate that DQNP outperforms existing baseline algorithms, increasing task completion by 14%, particularly for high-priority tasks, while reducing energy consumption by 8% and maintaining similar latency. Additionally, the model mitigates resource starvation for lower-priority tasks, achieving task selection rates of 27%, 32%, and 42% for low-, medium-, and high-priority tasks, with completion ratios of 88%, 87%, and 86%, respectively, reflecting balanced resource allocation across priority classes.**

***Index Terms*—Edge computing, prioritized offloading, reinforcement learning, vehicular networks.**

I. Introduction

CONNECTED and Autonomous Vehicles (CAVs) represent the future of transportation, combining advanced sensing technologies, artificial intelligence, and wireless communication systems to enable autonomous driving and smart mobility solutions. These vehicles are designed to navigate complex environments, interact with other road users, and make real-time decisions, all while ensuring safety, efficiency, and comfort. CAVs operate across various levels of autonomy, ranging from basic driver assistance systems like adaptive cruise control, to more advanced systems such as collision avoidance and lane-keeping, and ultimately to fully autonomous driving. To support these capabilities, CAVs are equipped with a vast array of sensors and onboard computing systems that generate and process massive amounts of data. Additionally, CAVs depend heavily on communication technologies, particularly vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) systems, which allow the exchange of critical information, including location, speed, sensor data, and intended maneuvers. Given the dynamic nature of traffic conditions, autonomous systems require real-time analysis of sensor data, such as lidar, radar, cameras, and GPS, which is essential for critical tasks such as obstacle detection and path planning. However, the substantial computational demands of these tasks often exceed the processing capacity of a CAV's onboard hardware, necessitating external support to meet the demands of real-time decision-making.

Edge computing has emerged as a crucial enabler for CAVs to address the limitations of onboard processing capabilities. It involves offloading computational tasks from the vehicle's onboard systems to nearby computing resources, such as roadside units, edge servers, or other connected vehicles [1]. This distributed computing model reduces the latency associated with cloud-based processing by bringing data processing closer to the vehicle. It also helps alleviate the computational burden on CAVs, reducing the energy consumption, and enabling them to run more complex applications that require more processing power than their onboard systems can provide [2]. As the number of connected vehicles continues to grow, the ability to offload tasks to edge servers supports the scalability of autonomous systems, allowing them to handle higher data loads without overwhelming onboard resources [3]. Tasks such as real-time high-definition map updates, machine learning-based object recognition, and traffic management analytics can all significantly benefit from edge computing.

Despite the advantages, the deployment of edge computing for CAVs introduces several challenges [4]. For example, maintaining reliable and high-speed communication between vehicles and edge servers is essential for the timely processing of offloaded tasks. Issues such as network congestion, limited bandwidth, or interference can lead to increased latency and reduce the effectiveness of edge computing. Efficiently distributing tasks among available edge servers, while considering both their computational capacities and network conditions, poses another complex challenge. Ineffective resource allocation can

Received 22 July 2024; revised 2 November 2024; accepted 13 November 2024. Date of publication 18 November 2024; date of current version 5 March 2025. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) . The review of this article was coordinated by Prof. Nan Cheng. *(Corresponding authors: Ning Zhang; Ahmed Hamdi Sakr.)*

The authors are with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, ON N9B 3P4, Canada (e-mail: uddin81@uwindsor.ca; ahmed.sakr@uwindsor.ca; ning.zhang@uwindsor.ca).

Digital Object Identifier 10.1109/TVT.2024.3499962

lead to overloading servers, increased latency, or even task failures. Additionally, the security and privacy of offloaded data must be carefully managed, as transmitting sensitive information to external servers increases vulnerability to cyber threats. The heterogeneity of infrastructure also presents a challenge, as the availability of edge computing resources varies based on geographic location, infrastructure deployment, and network conditions, leading to inconsistent performance in different regions.

Effective offloading strategies, however, present significant opportunities to optimize edge computing systems in vehicular networks by addressing key performance factors. These strategies can be designed to minimize latency, reduce energy consumption, or jointly optimize both, ensuring that the system balances real-time processing demands with energy efficiency [5]. Furthermore, they can help ensuring fairness in service delivery and resource allocation, preventing imbalances where some vehicles or servers are overburdened while others remain underutilized. The subsequent sections of this paper are organized as follows: Section II provides a summary of various techniques in the literature aimed at optimizing edge computing for vehicular networks. Section III presents the system model, while Section IV outlines the problem formulation. The proposed method is detailed in Section V, followed by the simulation and results analysis in Section VI. Finally, Section VII concludes the paper.

II. Related Work

The research community has extensively investigated a range of techniques to optimize edge computing for vehicular networks, which will be outlined in the following subsections.

A. Convex Optimization Approaches

In [6], the authors proposed a low-complexity algorithm for load balancing and offloading in multiuser multiserver Vehicular Edge Computing (VEC) systems. Similarly, [7] introduced a hybrid intelligent optimization algorithm aimed at optimizing task delay and resource consumption. In [8], Optimal Resource Provisioning (ORP) algorithms were developed to enhance system flexibility and cost-efficiency. To minimize delay in heterogeneous cellular networks, [9] proposed an alternating optimization algorithm based on coalitional game theory and convex optimization. Additionally, [10] implemented a logic-based Benders decomposition algorithm and a Multi-Armed Bandit learning-based approach for resource purchasing and task scheduling to maximize the platform profit. In [11], the Share-to-Run IoT Services model was introduced to optimize resource allocation and sharing through dual decomposition.

Moreover, [12] proposed a suboptimal heuristic algorithm for task offloading in mobile edge computing, which was formulated as a two-dimensional knapsack loading problem. The study in [13] investigated the profitability of computation offloading for network operators, utilizing a heuristic-based genetic algorithm. Addressing VEC offloading challenges, [14] introduced the Mobility Prediction Retrieval (MPR) protocol to improve data retrieval efficiency. In [15], an efficient task scheduling algorithm was proposed to minimize the average completion time of multiple applications by prioritizing tasks and applications. Finally, [16] implemented a techno-economic market model for edge computing, employing convex programming to achieve market equilibrium in resource allocation under budget constraints.

B. Game Theory Approaches

Fairness among users and Mobile Edge Computing (MEC) systems has been widely explored through various game-theoretic approaches, covering cooperative, collaborative, and competitive frameworks. For example, in [17], a multiuser non-cooperative computation offloading game was proposed, along with a distributed best-response algorithm aimed at optimizing task offloading strategies in vehicular MEC networks. Similarly, [18] introduced a computation offloading model for 5G networks, employing cooperative game theory and bargaining solutions to optimize dynamic offloading services. Additionally, [19] presented game-theoretic resource management techniques for multi-access MEC, balancing the objectives of service providers, application providers, and network providers.

In [20], a game-theoretical model for computation offloading in edge computing systems was developed, demonstrating the existence of Stackelberg equilibria for both cost-minimizing and time-fair resource allocation policies, supported by two decentralized algorithms. Furthermore, [21] proposed a market-based framework for resource allocation in edge computing, achieving market equilibrium through the Eisenberg-Gale convex program and distributed algorithms, ensuring Pareto-optimal and fair resource distribution among competing services. Finally, [22] formulated a joint resource allocation problem involving uplink, downlink, and computing resources in MEC, transforming it into a Generalized Nash Equilibrium Problem (GNEP) to minimize operational expenditure (OPEX) for Mobile Network Operators (MNOs) and Computing Resource Providers (CRPs), and proposed two decentralized algorithms to efficiently solve this problem.

C. Deep Reinforcement Learning (DRL) Approaches

The integer nature of offloading decision variables makes the optimization problem NP-hard, while the dynamic environment of vehicular networks, characterized by mobility and variability in communication channels, necessitates more robust, real-time decision-making processes. DRL has emerged as a promising solution to address the stochasticity of the vehicular environment and achieve real-time offloading and resource optimization. For instance, [23] formulated the problem as a Semi-Markov process and applied value iteration techniques using Deep Q-Networks (DQN) to determine optimal offloading decisions and resource allocation, focusing primarily on reducing delay. Similarly, [24] and [25] used DQN to minimize both latency and energy consumption, while [26] and [27] demonstrated the effectiveness of DQN-based DRL in resource allocation for both communication and computation tasks.

The Policy Gradient approach has also been explored. For example, [28] proposed a Deep Deterministic Policy Gradient

(DDPG)-based service offloading decision model for vehicular edge computing, effectively managing resource constraints. In [29], Proximal Policy Optimization (PPO) was used to minimize long-term costs by balancing task latency and energy consumption. A hybrid approach combining value iteration and policy gradient, known as Actor-Critic DQN, was introduced by [30] for partial offloading in hybrid NOMA environments. Multi-agent approaches have also been studied; for instance, [31] utilized Multi-Agent DDPG to enable cooperative task completion and cost minimization, with each entity acting as an individual agent. Additionally, Multi-Agent Q-Learning as presented by [32] encouraged reward sharing among users to achieve optimal energy objectives. Furthermore, [33] introduced multi-stack reinforcement learning to prevent agents from revisiting the same state with different actions, optimizing both computational and transmission delays across users. For more comprehensive reviews on offloading and resource allocation, readers are referred to [34].

D. Priority in Computational Offloading

In vehicular environments, task priorities may vary substantially, necessitating diverse strategies to ensure optimal performance and resource utilization. High-priority tasks, such as collision avoidance and autonomous driving assistance, are critical and must be processed with minimal latency to ensure safety and efficiency, employing techniques like edge computing and real-time communication protocols to reduce delays. Conversely, lower-priority tasks like infotainment systems and non-critical data uploads can tolerate higher latency and variability in processing times, allowing for more flexible resource allocation and the use of local processing and cloud computing.

Several studies have integrated prioritization with latency and energy optimization strategies. For example, [35] proposed a dynamic priority-based scheduling system that organizes tasks into a priority queue at the server after transmission or offloading. Similarly, [36] introduced a technique to prioritize tasks based on their deadlines, employing multilevel feedback queues for processing. In another approach, [37] developed a priority-aware offloading and scheduling strategy that processes tasks with strict deadlines first, while using remaining resources to handle tasks with more flexible deadlines, aiming to reduce average response times. Additionally, [38] presented a priority-greedy utility scheduling method that considers both task priority and Quality of Service (QoS). On the same lines, [39] proposed a priority-ordered task offloading scheme where priorities adjust based on increasing waiting and computational latencies, focusing on minimizing the overall delay while ensuring minimal delays for high-priority tasks. Furthermore, [40] introduced an adaptive resource allocation mechanism that follows a priority-sequence approach to optimize allocations across diverse task demands, distinguishing between priority and normal requests.

In [41], the authors explored priority-based offloading to computing nodes using Karush-Kuhn-Tucker (KKT) optimization conditions. In a similar domain, [42] utilized the multi-objective Grey Wolf optimization algorithm to develop a prioritized task offloading strategy, focusing on minimizing latency, social loss rate, and load imbalance. Another variant of the Grey Wolf algorithm was introduced in [43] to address the offloading problem for different priority tasks. Additionally, [44] employed Lagrangian-dual-based decomposition theory for optimal service selection, calculating the gain of offloading tasks based on relative energy and latency compared to local offloading. In terms of mobility considerations, [45] proposed a method for selecting service vehicles with prioritized access to offloading tasks, taking the vehicle's mobility model into account. Specifically, [46] optimized latency using a priority-weighted objective function to ensure that more critical tasks are processed with reduced delays.

Dynamic decision-making has also been a focal point in prioritization research. For instance, [47] introduced a distributed computation offloading algorithm utilizing multi-agent deep reinforcement learning. This algorithm leverages an improved actor-critic network featuring prioritized experience replay and adaptive n-step learning, collectively referred to as JMPA, which enhances both performance and efficiency. The proposed solution adopts a priority-based subtask offloading method using a directed acyclic graph structure, considering internal dependencies in prioritization. Similarly, [48] employed the Soft Actor-Critic (SAC) reinforcement learning method to handle different priority classes within Fog computing networks involving tasks and service vehicles. Moreover, [49] leveraged a Deep Deterministic Policy Gradient (DDPG) approach to optimize latency through partial offloading and resource allocation within the action space. This study applied prioritized TDMA techniques, scheduling tasks sequentially according to their priority. Lastly, [50] also employed a DDPG method within a delay-constrained model designed for three priority classes, focusing on enhancing energy efficiency while minimizing task completion delays by aligning resource allocation with varying priority requirements.

While studies focusing on task offloading prioritize higher-priority tasks, typically using optimization techniques to offload these tasks first, they often consume substantial computational resources. These approaches improve the performance of high-priority tasks but worsen resource imbalances, causing resource starvation and higher failure rates for lower-priority tasks. On the other hand, studies that use DRL often adopt a sequential, first-in-first-out (FIFO) strategy for task offloading, which can lead to delays in processing high-priority tasks as they are handled in the order they arrive. Moreover, much of the existing literature does not provide a detailed performance breakdown by priority class, omitting key metrics like latency, task completion ratios, offloaded task counts, and success rates across different priority levels. As a result, these studies do not fully address issues such as resource starvation or adequately assess the impact on lower-priority tasks.

In this study, we extend our previous work in [5] by introducing a prioritized incentive-based computational offloading scheme that leverages DRL for optimal task selection, MEC server selection, and computational frequency determination. We adopt a model that more accurately captures user and server positional distribution, taking into account user mobility dynamics. This enables a deeper analysis of network performance,

particularly in the presence of spatial randomness, which is essential for evaluating distance-related factors such as path loss. To enhance training efficiency, we refined the state and action spaces, as well as the design of the reward function, to better target primary optimization objectives. Additionally, we introduced variability in resource distribution with episodic periodicity, creating a broader range of scenarios that promote generalization in learning. This ensures the learning process is not biased toward specific resource distribution patterns, resulting in more robust performance across diverse conditions.

Key contributions of our work are the following:

- We propose a DRL-based model, the prioritized DQN (DQNP), which incorporates a fully observable MDP and a priority-based reward system to effectively handle task prioritization in dynamic vehicular edge computing environments.
- Our model efficiently optimizes resource allocation and service quality, achieving higher task completion rates for high-priority tasks while reducing energy consumption, latency, and resource starvation across all priority levels, maintaining fairness in resource distribution.
- We conduct a comprehensive comparative analysis with baseline methods from the literature, including the DRL-based approach in [25] and three other greedy-based approaches. Simulation results demonstrate a significant improvement in overall task completion and higher completion rates for high-priority tasks compared to existing methods.
- We present a distance-segmented analysis, which shows substantial reductions in latency and improved task completion rates, even in the presence of increased path loss. This analysis highlights the effectiveness of adaptive prioritization in addressing offloading challenges in dynamic network scenarios.

III. System Model

A. Environment Model

The proposed system is designed to enable task offloading to any server within the network infrastructure. Servers are strategically positioned at the center of each road segment to ensure a uniform distribution of users around each server. This configuration ensures consistent network coverage and effectively illustrates the impact of path loss when tasks are offloaded to adjacent servers. Users are capable of connecting to a nearby Base Station (BS) and any MEC server located along the road, provided that there is adequate network bandwidth, which is managed through Time Division Multiple Access (TDMA) technology. In this model, network latency, such as delays in accessing the network, is assumed to be negligible.

As depicted in Fig. 1, a road segment of length L meters is divided into K sections, each equipped with a centrally located MEC server. The kth server is denoted as $m_k = (Q_k, f_{\max,k})$, where Q_k describes the current waiting time in the server's queue, and $f_{\max,k}$ represents the maximum CPU frequency of the server. The total number of active users in the environment is denoted as N, while $\mathcal{U}_t^k = u_1^k, u_2^k, \ldots, u_n^k$ represents the set of

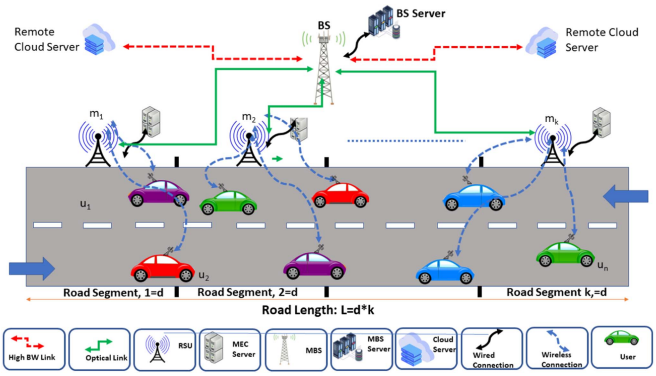


Fig. 1. Environment model.

users associated with the kth server. The distribution of users' locations is assumed to be uniform along the road.

In this system, one of the MEC servers (or a cloud server) is assigned to function as a network controller. The controller acts as a coordinator between MEC servers, gathering key information from other servers about the environment such as task details (e.g., data size, required resources, and deadlines), server status (e.g., utilization and computational capacity), and wireless channel state information (CSI) between users and their respective MEC servers. By analyzing this data, the network controller, also referred to as the agent, employs a learning model to make real-time decisions regarding resource allocation and task offloading. In other words, this agent organizes tasks based on their state information and manages the inflow of tasks from vehicles to the optimal MEC server, without routing the tasks through the controller itself. Additionally, the agent ensures the efficient allocation of computational frequencies to optimize performance.

B. Tasks Priority Model

The process of making an offloading decision by the agent involves the collection of task information from all users. The information collected by the agent at time t are denoted as $\Omega_t = \{\Omega_1, \Omega_2, \ldots, \Omega_j, \ldots, \Omega_M\}$, where Ω_j represents the jth task and M denotes the total number of tasks. Each task is characterized by five parameters, such that $\Omega_j = (u_j, D_j, C_j, \Delta_j, p_j)$. Here, u_j denotes the user index, D_j represents the task data size in bits, C_j is the required CPU cycles to complete the task, Δ_j is the deadline for task completion in seconds, and p_j indicates the priority class, which depends on the application and latency requirements of the task. The number of priority classes is defined as P such that $p_j \in \{1, 2, \ldots, P\}$, where a lower value indicates higher priority due to lower latency requirements.

In practical real-time applications, tasks classified under high priority are relatively scarce compared to regular or lower priority tasks. This scarcity reflects the typical demand for rapid response and the specialized nature of high-priority tasks, which are not as frequently required as standard tasks. Given this understanding, it is logical to model the generation of tasks in a manner that mirrors the natural distribution of task urgency and importance. Consequently, the task generation probability

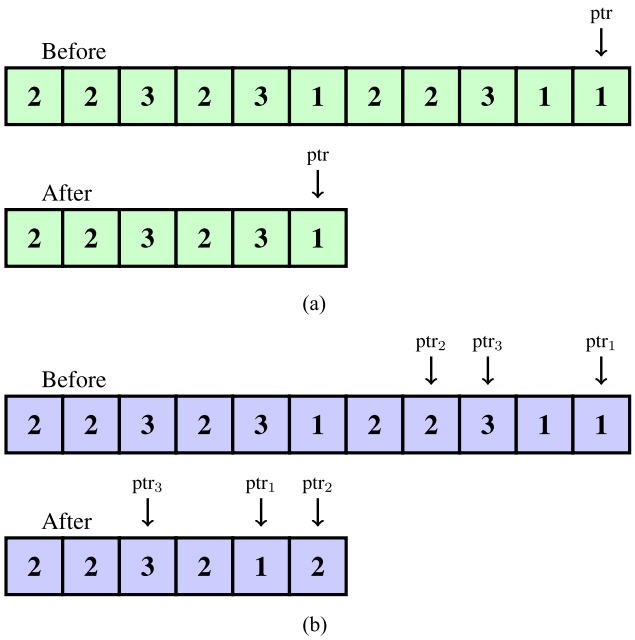


Fig. 2. FIFO queues vs. virtual priority queues. (a) FIFO queue. (b) Virtual priority queue with three priority classes.

follows:

$$\sum_{p=1}^{P} \alpha_p = 1 \quad \text{and} \quad \alpha_p < \alpha_{p-1} \tag{1}$$

where $0 < \alpha_p < 1$ denotes the probability of task generation for priority class p. This probabilistic model ensures that high-priority tasks are generated less frequently, aligning with the real-world distribution of task urgency.

C. Virtual Priority Queues Model

The network controller, also referred to as the agent, manages P virtual priority queues by establishing P pointers, each pointing to the task with the earliest arrival time within each priority class in its respective queue. In contrast to the traditional FIFO method, where tasks are offloaded strictly based on arrival time, the priority queue system offloads tasks according to their designated priority classes. Fig. 2 illustrates the state of the queues before and after offloading five tasks. For instance, while a FIFO queue might offload the first five tasks as $\{2, 2, 3, 1, 1\}$ without considering their priorities, a priority queue would select tasks such as $\{3, 3, 2, 1, 1\}$ based on their priority, ensuring that more critical tasks are processed sooner. This method allows the agent to not only focus on processing tasks in the order of arrival but also to prioritize them, thereby optimizing resource allocation and enhancing the utilization of communication channels for more effective task offloading. Note that once tasks are offloaded to a certain server, they are accumulated in a single unified queue at the server without explicit separation by priority.

D. Communication Model

Users transmit their tasks to any of the accessible MEC servers. To model the communication links between users and these servers, we utilize assumptions based on Rayleigh and block fading channel conditions. A significant improvement in this model, compared to our previous work presented in [5], is the incorporation of actual distances between users and servers, moving away from the previous assumption of equal distances. This update allows for a more precise simulation of the communication environment, particularly enhancing the dynamics of path loss. By considering realistic distances, the model more accurately reflects the variability in signal attenuation experienced over different lengths, leading to a more authentic and effective representation of network behavior.

The transmission rate of task j (i.e., user u_j) to server k is calculated as

$$r_{j,k} = B_j \log_2 \left(1 + \frac{P_{j,k} h_{j,k} L_{j,k}}{N_0}\right), \tag{2}$$

where B_j, $P_{j,k}$, $h_{j,k}$, and $L_{j,k}$ are the allocated bandwidth, the transmit power, the Rayleigh fading coefficient, and the large-scale path loss respectively. The transmission time of the jth task can be calculated as

$$\delta^T_{j,k} = \frac{D_j}{r_{j,k}}. \tag{3}$$

E. Service Time and Energy Models

The agent's main objective is to maximize task completion within specific deadlines while minimizing energy consumption and delays. This is achieved by optimally deciding which tasks to offload and determining the best frequency for each task. Hence The offloading process encompasses three stages: transmitting data, queuing at the MEC server, and processing the task on the server. The agent also focuses on resource allocation to ensure tasks are completed as swiftly as possible within their respective deadlines.

Following this objective, a task Ω_j with priority p_j is considered successfully completed when the total service time $\delta_{j,k}$ is less than or equal to the defined deadline of the task, Δ_j, i.e., $\delta_{j,k} \leq \Delta_j$. The total service time is obtained from

$$\delta_{j,k} = \delta^T_{j,k} + \delta^Q_k + \delta^R_k + \delta^C_{j,k}, \tag{4}$$

where $\delta^T_{j,k}$ is the transmission time of task j to server k, δ^Q_k is the waiting time in the queue of the assigned server, δ^R_j is the residual processing time of the current task running at the assigned server, and $\delta^C_{j,k}$ is the processing time needed to complete task j by the assigned server. In this work, we assume the result of an offloaded task contains minimal data, such as a class ID in an object detection task or a few bytes indicating a decision outcome (e.g., a binary flag for anomaly detection). Given this small data size, the transmission time of results back to the user is considered negligible.

Note that δ^Q_k is calculated by summing up the computation time of all tasks currently waiting in the queue of server k, excluding the currently running task. That is, given M tasks

in the queue of the server, we find

$$\delta_k^Q = \sum_{j=1}^{M} \frac{C_j}{f_{j,k}}, \tag{5}$$

where C_j and $f_{j,k}$ are the designated number of computational cycles and the suggested CPU frequency for task j, respectively. δ_k^R is determined by subtracting the actual runtime of the current task from its total computation time, where the runtime is the elapsed time from when the task began to the current moment on the server. $\delta_{j,k}^C$ is calculated by

$$\delta_{j,k}^C = \frac{C_j}{f_{j,k}}. \tag{6}$$

The total energy consumption $E_{j,k}$ is the sum of the transmission energy, $E_{j,k}^T$, and the computational energy, $E_{j,k}^C$. That is,

$$E_{j,k} = E_{j,k}^T + E_{j,k}^C, \tag{7}$$

where

$$E_{j,k}^T = \frac{D_j \times P_k}{\delta_{j,k}^T} \tag{8}$$

and

$$E_{j,k}^C = c \times C_j \times f_{j,k}^2, \tag{9}$$

with $c = 10^{-26}$ and $f_{j,k}$ being the recommended frequency [51] and [52].

IV. Problem Formulation

This section outlines the problem formulation within the Markov Decision Process (MDP) framework. To simplify the problem, it is assumed that tasks are processed one at a time within each time slot, referred to as t. When the agent receives λ tasks during time slot t, the arrangement of tasks into a sequence is broken down into λ time steps, each denoted by τ.

The state at time slot t can be defined as:

$$s_t = \left\{ s_0, s_1, \ldots, s_\lambda \mid \sum_{\tau=0}^{\lambda} \tau \ll t \right\}. \tag{10}$$

In this model, the state space is composed of the state of the upcoming task from each priority class, the status of the available MEC servers, and the condition of the communication channels between the corresponding users and each server. This representation ensures that the agent gains insights into the current workload, channel conditions, and task urgency across different priority levels. The server waiting time is updated following each action executed by the agent, ensuring that the model adheres to the Markov property for state transitions. This property dictates that the probability of transitioning to a future state, $p(s'|s,a)$, at time step τ is determined based on the current state and the action taken, independent of past states and defined as:

$$p(s'|s,a) = \Pr[s_\tau = s' | s_{\tau-1} = s, a_{\tau-1} = a]. \tag{11}$$

The state at time step τ is represented as

$$s_\tau = [s_\tau^\Omega, s_\tau^T, s_\tau^M], \tag{12}$$

where s_τ^Ω represents the states of the first task waiting in each priority class, and for P number of priority classes and priority queues, $s_\tau^\Omega = [\Omega_\tau^1, \Omega_\tau^2, \ldots, \Omega_\tau^P]$. As described in Section II, $\Omega_j = (u_j, D_j, C_j, \Delta_j, p_j)$ contains task information. The information u_j has no impact on making decisions and transitioning of state. Therefore, for state presentation, we refine this information to $\Omega_j = (D_j, C_j, \Delta_j, p_j)$.

s_τ^T represents the state of the wireless channels and can be represented in terms of transmission rates of all users, $s_\tau^T = [r_1^1, r_1^2, \ldots, r_1^P, r_2^1, r_2^2, \ldots, r_2^P, \ldots, r_K^1, r_K^2, \ldots, r_K^P]$ where r_k^p is the transmission rate of the user that generated task Ω_τ^P for MEC server k which can be calculated using (2). Finally, s_τ^M represents the state of the MEC servers and can be defined as $s_\tau^M = [(\delta_1^Q + \delta_1^R, f_{\max,1}), (\delta_2^Q + \delta_2^R, f_{\max,2}), \ldots, (\delta_K^Q + \delta_K^R, f_{\max,K})]$. Hence, the size of the state vector is $4P + KP + 2K$.

The action taken by the agent can be denoted at time slot t as a series of actions as follows:

$$a_t = \{a_0, a_1, \ldots, a_\lambda \mid \sum_{\tau=0}^{\lambda} \tau \ll t\}. \tag{13}$$

Each action is defined by a combination of the priority class being offloaded, the index of the designated MEC server, and the requested CPU frequency of the edge server. We discretize the requested CPU frequency to limit it to a finite number of levels. Hence, the size of the action space is $P \times K \times f_D$ where f_D is the number of discrete levels of CPU frequency in percentage. That is, an action at time step τ is $a_\tau = [p_\tau, k_\tau, f_\tau]$ which means that the first task waiting in priority queue p_τ will be offloaded with a requested frequency f_τ to MEC server k_τ.

The agent's objective is to maximize the reward by executing all time steps within a time slot. Therefore, the total reward R_t is defined as the sum of all cumulative rewards accumulated over all time steps before reaching the terminating steps in time slot t. That is,

$$R_t(s_t, a_t) = \sum_{\tau=1}^{\lambda} R_\tau(s_\tau, a_\tau). \tag{14}$$

To simultaneously reduce energy consumption and latency while maximizing task completion within an episode, the objective can be expressed as follows:

$$\begin{aligned} \min_{a_\tau} \quad & \sum_{\tau} E_{\chi_p,k} + (\Delta_{\chi_p} - \delta_{\chi_p,k}) \\ \text{s.t.} \quad & \delta_{\chi_p,k} \le \Delta_{\chi_p}, \\ & f_{\chi_p,k} \le f_{\max,k}, \\ & a_\tau = \{p, k, f_{\chi_p,k}\}, \end{aligned} \tag{15}$$

where χ_p is the first task from the pth priority queue, $E_{\chi_p,k}$ is the total energy consumption in (7), and $\delta_{\chi_p,k}$ is the total service time in (4). The first constraint ensures that task completion must occur within the maximum tolerable time, while the second constraint ensures that the server speed does not exceed its maximum frequency.

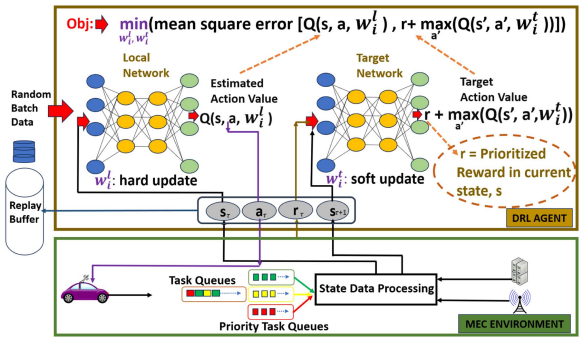


Fig. 3. Proposed DRL model.

A feasible solution to (15) may not always be attainable if the first constraint is not met. Consequently, the agent uses the following reward function to maximize the long-term reward while minimizing energy consumption and latency:

$$R_\tau(s_\tau, a_\tau) = \beta_{T,p} \log_2(1 + (\Delta_{\chi_p} - \delta_{\chi_p,k}))\mathbf{1}_{\delta_{\chi_p,k} \le \Delta_{\chi_p}} - \beta_E \log_2(1 + E_{\chi_p,k}), \tag{16}$$

where $\mathbf{1}_{\{.\}}$ is the indicator function, and β_E and $\beta_{T,p}$ represent the weights of the energy consumption cost and latency cost in the reward function. These weights are carefully selected to ensure that the energy and latency costs are on the same scale as timely task completion. Since our goal is priority-based offloading, $\beta_{T,p}$ varies with priority. Higher task priority corresponds to a higher value of $\beta_{T,p}$. This priority-dependent reward function and joint energy-latency optimization guide the agent to prioritize task selection based on the current state, ensuring timely completion and minimizing resource deprivation for low-priority tasks.

V. Proposed Method

Data preprocessing is crucial for handling the complex and noisy raw state data in our DRL model, especially given the dynamic channel gains from the block fading Rayleigh Channel and varied user distances. Our approach involves normalizing the $P \times K$ transmission speed matrix using the Frobenius norm to maintain the matrix's intrinsic structure and ensure manageable values. The state data is hierarchically structured under a 'states' root node, with subdivisions for Task Data, Task Speed Matrix, and MEC Server Data, each further segmented to reflect distinct task and server specifics. This structured and normalized data setup aids in effectively capturing environmental dynamics and providing clean inputs to the learning model. The task data size, D_τ, and task required CPU cycles, C_τ, are scaled using the max scaling method by dividing by their maximum values. Finally, all normalized and scaled state data are concatenated into a single state.

The method, illustrated in Fig. 3, employs a DRL model Deep Q-Network with a sophisticated prioritized incentive system (DQNP). This model intelligently selects from P priority classes, the appropriate MEC server, and its operating frequency to guarantee the timely completion of tasks. The model's reward function is designed to scale rewards according to the prioritized weight of different $(state, action)$ dynamics. While it might appear that the model mainly emphasizes only one action, i.e task selection, the inclusion of a deadline-bounded task completion variable significantly enhances the function's robustness. This adaptation allows for the optimal coordination of task, server, and resource selection, jointly optimizing latency and energy use. This strategy increases the likelihood of successful task completion by dynamically exploring and exploiting options within the action space, such as adjusting the frequency, selecting servers, and scheduling tasks, even in suboptimal conditions. Simulation results confirm that this method markedly improves task completion rates, reduces latency, and cuts down on energy consumption, effectively responding to the variable demands of state and action conditions to meet real-time requirements efficiently.

To manage challenges like oscillation and divergence in training, the DQNP method utilizes a technique called fixation. Here, the target network, a duplicate of the local/learning network with parameters denoted as w^t, serves as a stable benchmark for updates. Initially, both the learning networks (w^l) and target networks (w^t) share identical parameters. As training progresses, the learning network is refined using the gradient descent method to minimize discrepancies between the predicted Q-values and those of the target network, which are deemed more stable. The Mean Squared Error (MSE) loss, essential for network updates, is influenced by the fixation term, F, which captures the maximum Q-value for the next state, $Q(s', a')$. This fixation helps stabilize the learning process by providing consistent targets for updates, which can be expressed as:

$$F(w_\tau^t) = r_{(s,a)} + \gamma \max_{a'} Q(s', a', w_\tau^t) \tag{17}$$

The MSE can be expressed as:

$$L_\tau(w_\tau^l) = \mathbb{E}_{s,a,r,s',\tau}[F(s, a, w_\tau^t) - Q(s, a, w_\tau^l))^2] \tag{18}$$

At each update step of the learning network, w_τ^l is updated using adam on the gradient of loss minimization, (18), while w_τ^t is updated using soft update rule.

VI. Simulation Results

In this section, we introduce the simulation environment and the performance evaluation results. We conducted simulations of our model utilizing the Python libraries NumPy, Pandas, and PyTorch, and created customized Vehicular and MEC server environments. The local policy network in our DQN maps input states to Q-values across possible actions using a fully connected six-layer architecture with neuron counts of 256, 512, 512, 512, and 256, respectively. Each hidden layer employs a ReLU activation to introduce non-linearity. To stabilize learning, a target network, which is a less frequently updated copy of the policy network, estimates Q-values for the next state, reducing oscillations. Additionally, the DQN uses an experience replay buffer, with a size of 10^6 and batch size of 256. to store and randomly sample experience tuples (state, action, reward, next state, done), improving network generalization and training stability by decorrelating consecutive experiences.

Algorithm 1: DQNP.

Input: epoch, ε_{start}, ε_{end}, ε_{decay}, $\beta_{Time} = \beta(p)$ for $p = 1, 2, \ldots, P$, β_{Energy}
Output: loss, gradients
Initialize Replay Memory D with capacity N
Initialize Learning network weights $w^l = w$
Initialize Target network weights $w^t = w$
Set $\varepsilon = \varepsilon_{start}$
for episode $\leftarrow i = 1$ to epoch **do**
  Receive initial raw data x_1
  Preprocess raw data to state $s = \Phi(x_1)$
  for timesteps $\leftarrow \tau = 1$ to N **do**
    Select action a for state s using ϵ-greedy from $\overline{Q}(s, a, w^l)$
    Execute action a, observe reward $r = r(s, a, \beta_{Energy}, \beta_{Time})$, and next state s'
    Store transition (s, a, r, s') in Replay Memory D
    Set $s \leftarrow s'$ {Update state to the next state}
    if episode terminates **then**
      Start a new episode
    end if
    if batch size is reached and learning network update condition is met **then**
      Sample random batch (s_j, a_j, r_j, s'_j) from D
      Update learning network weights $w^l \leftarrow w^l + \alpha \nabla_{w^l} L(w^l)$ using Adam optimizer
    end if
    if target network update condition is met **then**
      Update target network weights $w^t \leftarrow w^l$
    end if
  end for
  Decay ε using $\varepsilon = \varepsilon \times \varepsilon_{decay}$ if $\varepsilon > \varepsilon_{end}$
  Store score for the current episode
end for

Algorithm 2: Priority Greedy (Reward/Energy/Latency).

Input: Tasks list and action space
Output: Selected action
Initialize $p = P$
while $p \geq 1$ **do**
  Consider task χ_p as a candidate for offloading and initialize $\mathcal{A} = \{\}$
  for each action a in action space **do**
    Apply a and calculate the reward $r_{\chi_p,a}$, energy $E_{\chi_p,a}$, and latency $\delta_{\chi_p,a}$
    if $\delta_{\chi_p} \leq \Delta_{\chi_p}$ **then**
      $\mathcal{A} \leftarrow \mathcal{A} \bigcup \{a\}$
    end if
  end for
  if $\mathcal{A} \neq \emptyset$ **then**
    Select task χ_p and action $a \in \mathcal{A}$ with maximum $r_{\chi_p,a}$ if RGreedy, minimum $E_{\chi_p,a}$ if EGreedy, or minimum $\delta_{\chi_p,a}$ if LGreedy is selected
    break
  else
    $p \leftarrow p - 1$
  end if
end while

To assess the efficacy of the proposed DQNP algorithm, cf., Algorithm 1, we compare our results to four baseline methods:

1) The baseline DRL agent proposed in [25] (DQNB), which considers all tasks to have the same priority, and task offloadeding based on arrival time in sequential manner.
2) A priority greedy algorithm, cf., Algorithm 2 (Priority Reward Greedy), which sorts all tasks based on their priority and maximum reward.
3) Another version of the priority greedy algorithm, cf., Algorithm 2 (Priority Latency Greedy), which sorts all tasks based on their priority and minimum latency.
4) Another version of the priority greedy algorithm, cf., Algorithm 2 (Priority Energy Greedy), which sorts all tasks based on their priority and minimum energy.

Without loss of generalization, we conduct simulations for three distinct priority classes such that $p_j \in \{1, 2, 3\}$. The main simulation parameters are summarized in Table I.

The convergence curves in Fig. 4 demonstrate the progress of learning for proposed DQNP, baseline DQNB, and Priority Greedy. In DQNP, the state-action spaces are more extensive

TABLE I
SIMULATION SETUP PARAMETERS

Parameter	Value
MEC to MEC Distance	300 m
MEC Elevation Height	10 m
Transmission Power	300 mW
SNR and Bandwidth	100 dB, 2×10^7 Hz
Task size ($\times 10^3$ bit)	$\mathcal{U}[10, 20]$
Required CPU cycles ($\times 10^6$ cycle)	$\mathcal{U}[8, 10]$
Server frequency ($\times 10^9$ cycle)	$\mathcal{U}[2, 16]$
Deadline (ms)	$P1 : \mathcal{U}[190, 200]$ $P2 : \mathcal{U}[180, 190]$ $P3 : \mathcal{U}[170, 180]$
Replay buffer size and Batch size	10^6 and 256
Learning rate	5×10^{-4}
Discount factor (γ)	0.9

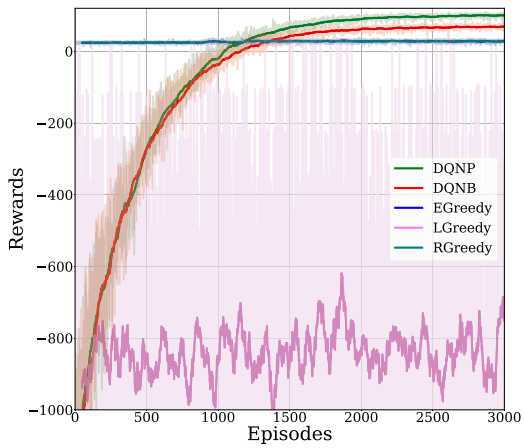


Fig. 4. Convergence of training: DQNP vs. DQNB vs. greedy.

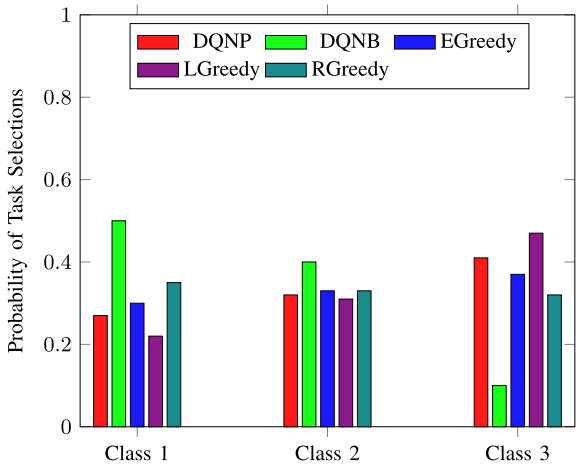


Fig. 5. Probability of task selections: DQNP vs. DQNB vs. greedy.

than in DQNB, which leads to more exploration by the DQNP agent and causes DQNP to converge more slowly compared to DQNB. Meanwhile, in scenarios where the environment changes dynamically at each time step as well as in each episode, the greedy strategies, such as energy greedy and reward greedy, display nearly equal and constant reward gains in each episode. The energy-greedy strategy focuses on minimizing energy consumption, which often results in increased computational latency. To conserve transmission energy, it typically selects the nearest server. However, this server may operate at the lowest frequency to further reduce energy usage, which significantly extends computation time. In the design of the reward functions, the contribution of latency varies from zero to a finite number. More computation time frequently leads to task failures due to boundary of deadlines, ultimately impacting minimum gain for latency in reward while the negative component, energy part, contributes minimally because of lower energy selection, leading to a consistently positive accumulation of rewards and continue reward across all episodes.

Conversely, the reward-greedy strategy aims to maximize the overall reward, typically selecting the nearest server to reduce latency and the lowest frequency to reduce energy use. Similar to the energy-greedy approach, this strategy also experiences higher computational latency due to the selection of lower frequencies, leading to frequent task failures. In contrast, the latency-greedy strategy experiences significant fluctuations in reward gains. This approach focuses on minimizing latency, often by choosing the highest frequency, which, while reducing transmission time by using the nearest server, significantly increases computation energy consumption. As a result, although latency is minimized, the substantial energy costs lead to variable reward gains. These strategies are defined by priority-based heuristics rather than adaptive learning strategies.

Fig. 5 illustrates the task selection distributions, highlighting the advantages of DQNP. This study incorporates dynamic distances, providing a more realistic setting. It is important to note that the task data distribution for transmission and computation parts is independent of priority, while task completion is dependent on the deadline, which serves as the basis for priority classification. As a result, the Reward Greedy system shows a higher probability of selecting tasks from the lower priority class (35%, 33%, and 32%) because the lower priority class has the potential for the highest reward due to its higher deadline. This occurs even though Reward Greedy also checks for the feasibility of task completion from high to low priority order, similar to the energy and latency greedy strategies.

However, Latency Greedy and Energy Greedy strategies select a higher number of tasks from the higher priority class, as they choose actions based on minimum latency and minimum energy, respectively, in priority order. This pattern arises because the Greedy method prioritizes tasks based primarily on the capability to complete tasks and secondarily on the action that provides minimum latency and minimum energy. This contrasts with previous models that assumed uniform distances between servers and users, as cited in previous studies [5]. Since task completion is bounded by the deadline, the task selection for the high priority class is maximum (47%) in Latency Greedy compared to any other methods, while the Energy Greedy strategy selects 37% from the higher priority class.

In contrast, DQNB, which lacks a sophisticated task selection strategy, shows selection probabilities of 50%, 40%, and 10% for classes 1, 2, and 3, respectively. DQNP employs a priority-based DRL strategy that selects tasks according to both priority and current conditions, leading to a priority-weighted distribution where tasks are selected with probabilities of 27%, 32%, and 41% for priority classes 1, 2, and 3, respectively. This approach highlights DQNP's objective to distribute task selections more evenly across varying levels of priority to maximize task completion and optimize latency and energy consumption.

DQNP outperforms other strategies in task selection due to its comprehensive state-action space exploration, adaptive strategy to dynamic environments, and priority-based incentives in its reward function. This ensures an even distribution of tasks across different priority levels. Unlike other strategies that suffer from resource starvation for low-priority classes due to sequential order-based task offloading, DQNP does not prioritize only high-priority tasks. Instead, it adapts its selection policy based on the dynamic state and feedback from the environment to optimize latency, energy, and task completion. This approach ensures that task selections are optimized not only for immediate rewards but also for long-term system performance and efficiency.

Fig. 6 illustrates the task completion ratios for both individual priority classes and the overall completion ratio. DQNB's approach involves considering only one task in its state, mandating offloading irrespective of poor state conditions (e.g., poor channel gain). In [5], when assuming equal distances from each server to users, the task completion ratios across the three classes in both DQNP, DQNB, and Greedy may seem fairly uniform. However, in the current study, the consideration of realistic state dynamics, such as the Euclidean distance from each server to users, provides insights into the impact of dynamic path loss and leads to varying task completion ratios.

For similar reasons, the Greedy algorithm does not perform as demonstrated in [5] due to its priority-based strategy. For example, the Reward Greedy algorithm prioritizes tasks by first attempting to complete the highest priority task, and if that is not

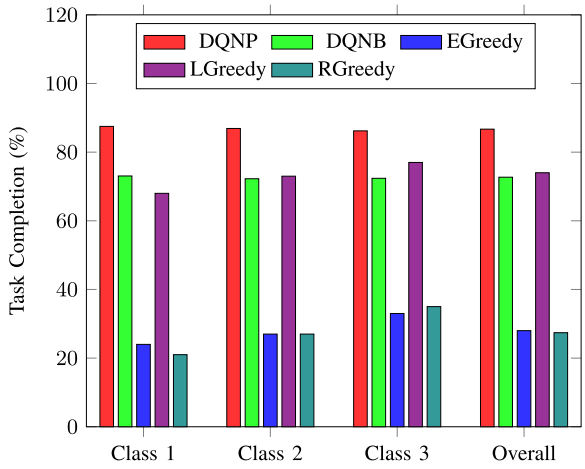


Fig. 6. Average task completion ratio: DQNP vs. DQNB vs. greedy.

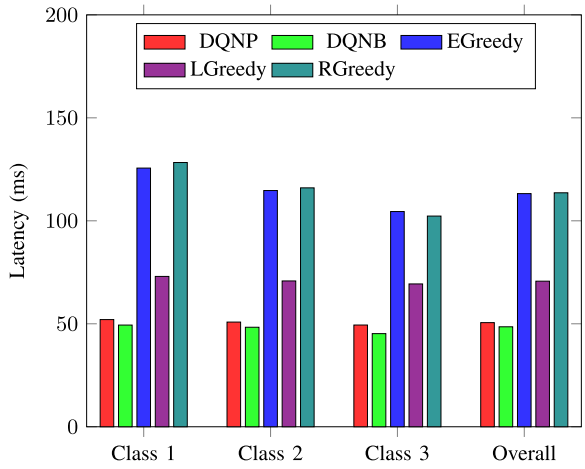


Fig. 7. Latency: DQNP vs. DQNB vs. greedy.

feasible, it considers the lowest priority tasks. If neither can be completed, it opts for the action that yields the highest reward. Since it does not follow any optimization policy like DQNP or DQNB, it selects the minimum server frequency, which causes higher computational latency. Initially, tasks may be completed, but as wait times to the server increase, many tasks ultimately fail.

A similar strategy is followed by the Energy Greedy approach, which, in striving for minimum energy consumption, ignores latency requirements and deadlines—the only bounding conditions for task completion. This priority-based strategy, which neglects learning-based task selection, results in a very low task completion ratio of about 28%. However, latency-greedy approaches select actions that provide minimum latency, helping to maintain acceptable wait times and computational times within deadlines, albeit at the cost of higher computational energy. This approach achieves a high completion rate for high-priority tasks at the expense of higher energy, leading to task completion ratios of 68% for class 1, 73% for class 2, and 77% for class 3.

An important question arises here: if all greedy algorithms share the same reward function and nearly identical algorithms, why is the number of task completions lower for energy and reward greedy strategies? This is because, without following any optimal or learning-based strategy, greedy agents select actions to obtain immediate optimized feedback, such as minimizing energy usage or maximizing rewards. This short-term policy leads to the premature consumption of resources, resulting in the failure of most tasks in future steps. These findings highlight the potential bottleneck of the sequential priority-based or priority-ordered algorithm i.e resource starvation, most of which are discussed in the literature review sections. These findings highlight the limitations of the priority-greedy strategy under the more robust and realistic dynamics of environmental conditions, emphasizing the need for adaptive strategies that consider both task priority and the dynamic conditions of network environments to select tasks optimally, aiming to maximize task completion rates while minimizing latency and energy use.

As expected, DQNP demonstrates superior performance, achieving an overall task completion ratio of approximately 86.7%, a notable increase of 14% and 12.7% compared to DQNB (overall 72.7%) and Latency Greedy (overall 74%) respectively. This enhancement is due not only to DQNP's prioritized task selection mechanism but also to its efficient action mechanism, which reduces resource starvation for low-priority tasks. This strategy achieves a balance between selecting and completing tasks, addressing the key issues with priority-based strategies highlighted in the literature review.

Fig. 7 provides an overview of the performance outcomes concerning latency. As expected, DQNP shows the lowest average latency, approximately 50.56 ms, outperforming the three greedy approaches. It appears that DQNB (48.54 ms) outperforms DQNP in terms of latency. However, the 14% increase in task completion with DQNP compared to DQNB indicates that many tasks that were not finished in DQNB were successfully completed by DQNP, resulting in slightly higher latency for DQNP. DQNP reduces latency by selecting from a broader range of actions focused on prioritized tasks that both maximize task completion and minimize latency, thereby securing a policy for long-term reward maximization. Specifically, latency figures for class 1, class 2, and class 3 in DQNP are 52.04 ms, 50.83 ms, and 49.40 ms, respectively, while DQNB records slightly reduced latencies of 49.38 ms, 48.32 ms, and 45.22 ms, respectively.

In contrast, as expected, only the Latency Greedy approach achieves the lowest overall latencies, nearly 70.69 ms, compared to the two other greedy approaches: Energy Greedy (overall 113.2 ms) and Reward Greedy (overall 113.6 ms). The Energy Greedy approach ignores actions that would result in lower latency, while the Reward Greedy approach favors actions that minimize energy consumption at the expense of latency. Interestingly, the Energy Greedy and Reward Greedy approaches show very similar performance. This similarity arises because both strategies prioritize minimizing energy consumption, albeit through slightly different mechanisms.

Fig. 8 shows the performance in terms of energy consumption. As discussed above, the Energy and Reward Greedy approaches outperform Latency Greedy, DQNB, and DQNP, showcasing a mean energy consumption of 0.397 W. These two algorithms

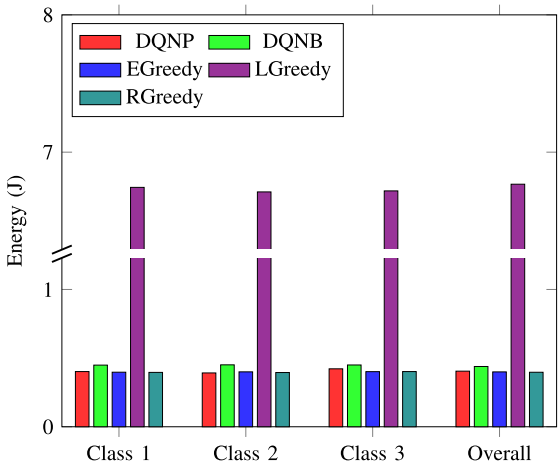


Fig. 8. Energy consumption: DQNP vs. DQNB vs. greedy.

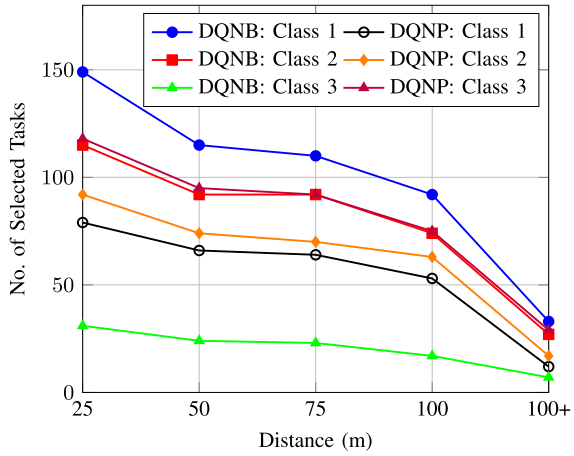


Fig. 9. Number of selected tasks as a function of distance and priority class: DQNP vs. DQNB.

consistently select actions that yield the lowest energy consumption and consequently the highest reward, prioritizing this aspect over latency concerns through their task selection strategy, while the Latency Greedy approach selects the action for minimum latency, causing higher overall energy consumption of 6.74J. Conversely, DQNP and DQNB exhibit overall mean energy consumptions of 0.405 W and 0.449 W, respectively. This indicates that DQNP consumes less transmit power to transmit tasks that are not transmittable in DQNB, while computation power is nearly equal for both DQNP and DQNB (Fig. 14). This outperformance in energy efficiency suggests that DQNP effectively selects appropriate tasks for transmission and offloading according to the dynamics of the transmission state.

This detailed breakdown of the performance metrics emphasizes the consistent superiority of DQNP in jointly optimizing latency and energy, thereby maximizing task completion. This highlights the significance of DQNP's approach in selecting appropriate tasks through an adaptive task selection policy that considers dynamic distance and dynamic channel gain. It indicates that while DQNB has no choice but to accept poor channel gain or longer server distances, DQNP leverages its adaptive strategy to improve returns even in poor states where the value function (how beneficial it is for the agent to be in the state) is very low. By dynamically adapting its task selection, DQNP can offload tasks even in these challenging states, thereby enhancing overall performance.

To more illustratively investigate the performance of DQNP regarding the dynamic distance, we have demonstrated the simulation results across various distance buckets in Fig. 9. Here, distances are categorized by percentile distances between the server and user; for instance, 25 refers to users located within 0 to 25% of the range radius of the server, and so forth, while the term 100+ denotes distances out of the range radius. It is evident from the data that for both DQNP and DQNB, the number of tasks offloaded decreases as the distance increases, indicating that both agents learn to offload tasks to the nearest servers, with offloading rates of 6% for DQNP and 7% for DQNB to the farthest server.

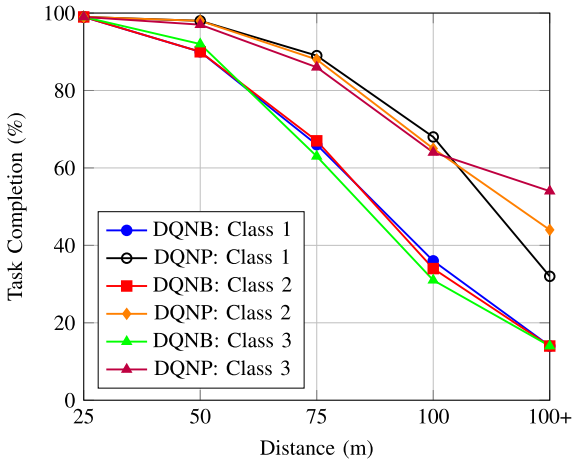


Fig. 10. Average task completion ratio as a function of distance and priority class: DQNP vs. DQNB.

Notably, DQNP's selection strategy shows that tasks with higher priority are offloaded more frequently in each distance bucket compared to DQNB. Moreover, DQNP maintains a balance in task selection from priority 1 to mitigate resource starvation that could result from a sequential priority approach, thus offloading more tasks compared to priority 3 in DQNB. However, this requires a reduction in the number of tasks selected from priority 2 compared to DQNB. This indicates that prioritized task selection in DQNP works regardless of increasing path-loss due to longer distance or poor channel gain. As DQNP employs a priority-weighted approach, the strategy can be adjusted by tuning the weights assigned to the three priority classes.

Fig. 10 demonstrates the superiority of DQNP's task selection strategy, which achieves a higher task completion ratio across all distance buckets for each priority class compared to DQNB. This outcome is crucial as it highlights the importance of prioritizing tasks based on critical state data, such as poor channel gain or limited computational resources at the MEC. The DQNP

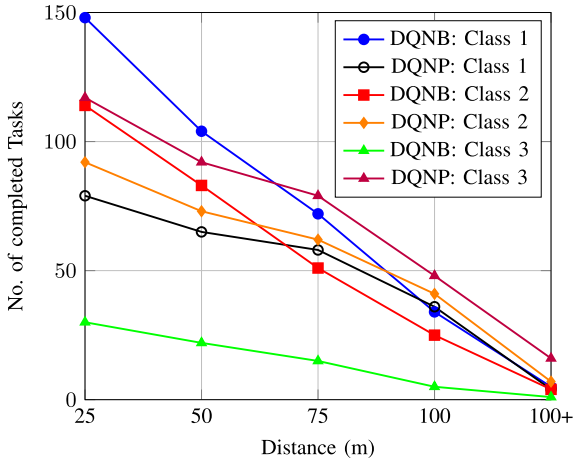


Fig. 11. Number of completed tasks as a function of distance and priority class: DQNP vs. DQNB.

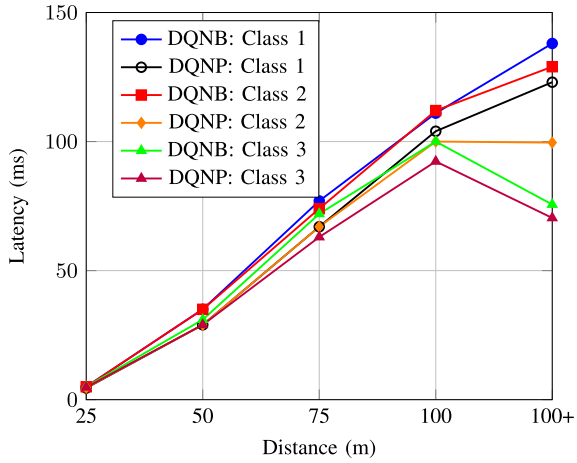


Fig. 12. Transmission latency as a function of distance and priority class: DQNP vs. DQNB.

(Deep Q-Network with Prioritization) demonstrates superior task completion rates across all categories compared to DQNB (Deep Q-Network without Prioritization). Specifically, while DQNB achieves task completion rates of approximately 14% for priority 1, priority 2, and priority 3 tasks at the farthest server, DQNP significantly improves these rates to 32%, 44%, and 54%, respectively. This substantial increase in task completion ratios for marginal distances indicates that DQNP's strategy of prioritized task selection based on state dynamics, such as channel conditions and computational resource availability, results in better performance. The higher task completion rates in DQNP suggest that by prioritizing tasks, the network can more effectively allocate resources and manage transmissions, even under challenging conditions. This improvement in performance is not merely incremental but demonstrates a significant enhancement, underlining the effectiveness of the prioritization strategy implemented in DQNP over the non-prioritized approach in DQNB.

Fig. 11 shows the number of completed tasks. The data clearly demonstrates that DQNB tends to complete a greater number of tasks from class 1, primarily due to its higher selection rate for this class. Conversely, DQNP achieves a superior number of task completions for class 3 compared to DQNB, but slightly fewer completions in class 2 due to its balanced strategic selection. DQNP effectively balances three objectives: reducing latency, minimizing energy, and maximizing task completion, especially when there is significant path loss. On the other hand, DQNB, despite offloading class 1 tasks more often, does not perform as well as DQNP as the distance increases.

Fig. 12 depicts the transmission latency performance across various distance buckets for the tasks that are successfully transmitted, highlighting DQNP's consistently lower latency compared to DQNB. This emphasizes the importance of prioritizing task selection, particularly for servers located at greater distances. DQNP's strategic task selection, driven by the promise of higher rewards as per weighted priority, leads to reduced latency and a higher selection rate across all priority classes compared to DQNB. It is expected that tasks offloaded to the farthest server will experience higher transmission latency in DQNB because the relative weights for the energy and latency components are equal. This balance causes the DQNB agent not to select higher frequencies, as it attempts to optimize both energy and latency simultaneously. Consequently, priorities 2 and 3 for DQNB exhibit higher transmission latency. The exceptional drop in latency for priority 3 in DQNB might be due to a very low distribution of task completions, possibly only one task being completed.

Conversely, for DQNP, the relative weight of the latency component is higher for priorities 2 and 3, which enforces the agent to select actions like task selection according to channel states, server as nearest neighbors, and higher frequency to transmit and compute the tasks faster, resulting in a significant reduction in latency for priority 3 while maintaining the same latency for priority 2. For priority 1, DQNP follows the same trend as DQNB because the relative weights are equal for priority 1 in both DQNP and DQNB.

The latency performance reflects the energy consumption, with the influence of transmission energy being dominant because the computational energy is nearly constant due to less sensitive dynamics of the computational state. For all offloaded tasks, Priority 1 in DQNP is significantly higher than that of DQNB due to its higher transmission latency, as outlined in Fig. 13, which gives equal consideration to energy and latency. However, for Priorities 2 and 3, the higher relative weight on the latency part causes lower transmission latency and thus lower transmission energy, resulting in reduced overall energy consumption. Moreover, the number of non-completed tasks, which also contribute to energy consumption, is higher for Priority 1 than for Priorities 2 and 3. On the other hand, for all priorities in DQNB, latency and energy are equally important due to their equal weights. This causes DQNB to have similar energy consumption to latency, keeping the computational energy requirement constant.

These findings underscore the significance of task prioritization in achieving a balance between the three classes to optimize both latency and task completion, regardless of critical state dynamics such as poor channel gain or higher path

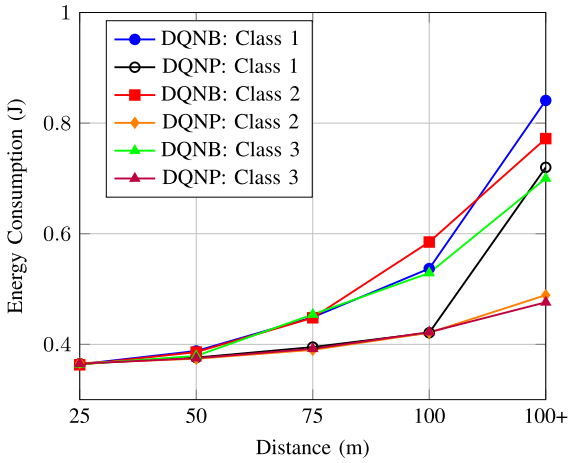


Fig. 13. Energy consumption as a function of distance and priority class: DQNP vs. DQNB.

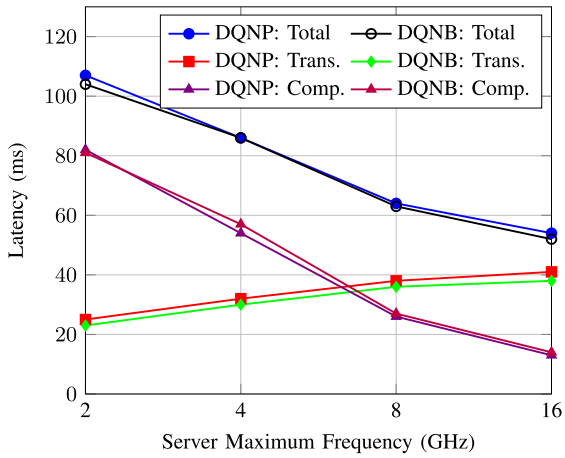


Fig. 14. Computation and communication latency vs. server maximum frequency: DQNP vs. DQNB.

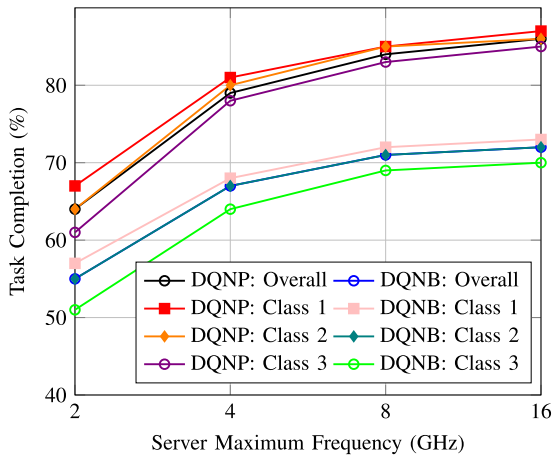


Fig. 15. Average task completion ratio vs. server maximum frequency: DQNP vs. DQNB.

loss. By effectively exploring a broader range of actions and utilizing additional resources, the system can dynamically adapt to varying conditions. Proper task prioritization ensures that high-priority tasks receive the necessary resources and attention, while lower-priority tasks are managed efficiently. This strategic allocation of computational and transmission resources maximizes overall system efficiency. In scenarios with poor channel gain, prioritizing tasks helps decide which tasks to attempt given the limited bandwidth, preventing delays or failures in high-priority tasks. Similarly, higher path loss conditions can be mitigated by selecting actions that adapt to these losses or prioritize less affected tasks. Exploring more actions enhances decision-making by providing more options, leading to better task offloading and completion strategies. This flexibility allows the system to respond effectively to dynamic environments. Utilizing additional resources when necessary ensures tasks are completed efficiently, with dynamic allocation based on task priority and current state conditions maintaining high performance levels. Overall, these findings highlight the importance of a sophisticated task prioritization mechanism that can adapt to critical state dynamics, ensuring optimized latency and task completion for a more efficient and reliable system performance.

To examine the performance of DQNP across varying computational resources, Fig. 14 illustrates the influence of resource constraints on DQNP's performance, simulated at maximum frequencies of 2, 4, 8, and 16 GHz. As expected, at 2 GHz, both DQNP and DQNB systems experience higher computational latency but lower transmission latency. This occurs because task completion is a binary variable constrained by deadlines, meaning higher computational latency reduces the available time slots for transmission latency within the deadline boundary. Conversely, higher frequencies result in decreased computational latency, allowing more time for transmission to meet deadlines. Consequently, overall latency decreases with higher frequencies. Additionally, the task completion ratio increases with frequency, indicating that tasks have more time to complete before deadlines, as shown in Fig. 15. Notably, the transition from 8 GHz to 16 GHz in both Figs. 15 and 14 demonstrates a flatter slope compared to other frequency transitions. This underscores the significant impact of transmission environments on system dynamics and highlights the trade-off in selecting computational frequencies and transmission resources to optimize DQNP performance.

In Fig. 14, the higher transmission latency observed in DQNP compared to DQNB, combined with the higher task completion ratio shown in Fig. 15, indicates that tasks which were not completed by DQNB were successfully completed by DQNP. This differential highlights the effectiveness of DQNP's approach. Specifically, it shows that the tasks which DQNB failed to complete due to timing constraints or resource limitations were efficiently managed and completed by DQNP.

This outcome can be attributed to the prioritized task selection strategy employed by DQNP in the action space. Unlike DQNB, which may follow a sequential or fixed priority-based order for task scheduling, DQNP uses a dynamic scheduling method that evaluates the state of the system and makes informed decisions about which tasks to offload. This dynamic prioritization allows DQNP to optimize the use of available resources, reduce the

impact of transmission latency, and improve overall task completion rates. By dynamically selecting and prioritizing tasks, DQNP is able to adjust to varying conditions and constraints in real-time, ensuring that more tasks are completed within their deadlines. This approach not only maximizes long-term rewards but also significantly enhances the performance and efficiency of the system. The success of DQNP in completing tasks that DQNB could not demonstrate the robustness and effectiveness of its scheduling strategy, providing clear evidence of its superiority in managing computational and transmission resources.

VII. Conclusion

In this study, we presented an innovative DRL-based strategy for task offloading in vehicular edge computing. Our approach leverages a priority-focused reward function, guiding the DQN agent to develop an effective policy that adeptly balances exploration and exploitation. This allows for the optimal selection of priority classes, servers, and frequencies for task completion. Simulation results demonstrate the superiority of our method over existing approaches, particularly in scenarios with stringent deadlines or high-priority tasks. Our strategy not only maximizes task completion rates but also significantly reduces latency and energy consumption. The DQNP (DQN with Prioritization) consistently achieves lower transmission latencies across various distance buckets, highlighting its ability to adapt to dynamic network conditions and manage computational resources efficiently. Furthermore, our approach addresses the common issue of resource starvation in low-priority tasks. By ensuring a weighted distribution of tasks across all priority classes, our method prevents the excessive neglect of less urgent tasks, thereby maintaining a balanced and efficient system performance. The prioritized task selection mechanism in DQNP proves effective in both ideal and challenging channel conditions, such as poor channel gain and high path loss, ensuring that high-priority tasks receive the necessary resources without compromising overall system efficiency.

References

[1] W. Feng, N. Zhang, S. Lin, G. Wang, B. Ai, and L. Cai, "Joint C-V2X based offloading and resource allocation in multi-tier vehicular edge computing system," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 2, pp. 432–445, Feb. 2023.

[2] L. Hou, M. A. Gregory, and S. Li, "A survey of multi-access edge computing and vehicular networking," *IEEE Access*, vol. 10, pp. 123436–123451, 2022.

[3] N. H. Hussein, C. T. Yaw, S. P. Koh, S. K. Tiong, and K. H. Chong, "A comprehensive survey on vehicular networking: Communications, applications, challenges, and upcoming research directions," *IEEE Access*, vol. 10, pp. 86127–86180, 2022.

[4] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 4, pp. 2131–2165, Fourthquarter 2021.

[5] A. Uddin, A. H. Sakr, and N. Zhang, "Prioritized task offloading in vehicular edge computing using deep reinforcement learning," in *Proc. IEEE Veh. Technol. Conf.*, Helsinki, Singapore, Jun. 2024, 1–6.

[6] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4377–4387, Jun. 2019.

[7] J. Sun, Q. Gu, T. Zheng, P. Dong, A. Valera, and Y. Qin, "Joint optimization of computation offloading and task scheduling in vehicular edge computing networks," *IEEE Access*, vol. 8, pp. 10466–10477, 2020.

[8] X. Ma, S. Wang, S. Zhang, P. Yang, C. Lin, and X. Shen, "Cost-efficient resource provisioning for dynamic requests in cloud assisted mobile edge computing," *IEEE Trans. Cloud Comput.*, vol. 9, no. 3, pp. 968–980, Jul.–Sep. 2021.

[9] T. Zhou, Y. Yue, D. Qin, X. Nie, X. Li, and C. Li, "Mobile device association and resource allocation in HCNs with mobile edge computing and caching," *IEEE Syst. J.*, vol. 17, no. 1, pp. 976–987, Mar. 2023.

[10] X. Huang, B. Zhang, and C. Li, "Platform profit maximization on service provisioning in mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13364–13376, Dec. 2021.

[11] C. Pham, D. T. Nguyen, Y. Njah, N. H. Tran, K. K. Nguyen, and M. Cheriet, "Share-to-run IoT services in edge cloud computing," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 497–509, Jan. 2022.

[12] F. Wu et al., "Energy-time efficient task offloading for mobile edge computing in hot-spot scenarios," in *Proc. IEEE Int. Conf. Commun.*, 2021, pp. 1–6.

[13] S. Singh and D. H. Kim, "Profit optimization for mobile edge computing using genetic algorithm," in *Proc. IEEE Region 10 Symp.*, 2021, pp. 1–6.

[14] A. Boukerche and V. Soto, "An efficient mobility-oriented retrieval protocol for computation offloading in vehicular edge multi-access network," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 6, pp. 2675–2688, Jun. 2020.

[15] Y. Liu et al., "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4961–4971 , Jun. 2020.

[16] E. Moro and I. Filippini, "Joint management of compute and radio resources in mobile edge computing: A market equilibrium approach," *IEEE Trans. Mobile Comput.*, vol. 22, no. 2, pp. 983–995, Feb. 2023.

[17] Y. Wang et al., "A game-based computation offloading method in vehicular multiaccess edge computing networks," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4987–4996, Jun. 2020.

[18] S. Kim, "Bargaining game-based offloading service algorithm for edge-assisted distributed computing model," *IEEE Access*, vol. 10, pp. 63648–63657, 2022.

[19] M. Zakarya et al., "epcAware: A game-based, energy, performance and cost-efficient resource management technique for multi-access edge computing," *IEEE Trans. Serv. Comput.*, vol. 15, no. 3, pp. 1634–1648, May/Jun. 2022.

[20] S. Jošilo and G. Dán, "Joint management of wireless and computing resources for computation offloading in mobile edge clouds," *IEEE Trans. Cloud Comput.*, vol. 9, no. 4, pp. 1507–1520, Oct.–Dec. 2021.

[21] D. T. Nguyen, L. B. Le, and V. Bhargava, "Price-based resource allocation for edge computing: A market equilibrium approach," *IEEE Trans. Cloud Comput.*, vol. 9, no. 1, pp. 302–317, Jan.–Mar. 2021.

[22] C. W. Zaw, N. H. Tran, Z. Han, and C. S. Hong, "Radio and computing resource allocation in co-located edge computing: A generalized Nash equilibrium model," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2340–2352, Apr. 2023.

[23] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11158–11168, Nov. 2019.

[24] Z. Wu and D. Yan, "Deep reinforcement learning-based computation offloading for 5G vehicle-aware multi-access edge computing network," *China Commun.*, vol. 18, no. 11, pp. 26–41, Nov. 2021.

[25] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 3, pp. 881–892, Sep. 2021.

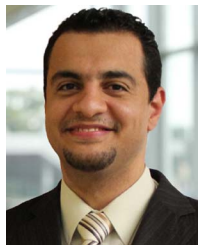
[26] L. Tan, Z. Kuang, L. Zhao, and A. Liu, "Energy-efficient joint task offloading and resource allocation in OFDMA-based collaborative edge computing," *IEEE Trans. Wireless Commun.*, vol. 21, no. 3, pp. 1960–1972, Mar. 2022.

[27] T. Liu, S. Ni, X. Li, Y. Zhu, L. Kong, and Y. Yang, "Deep reinforcement learning based approach for online service placement and computation resource allocation in edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 3870–3881, Jul. 2023.

[28] Y. Ren, X. Yu, X. Chen, S. Guo, and Q. Xue-Song, "Vehicular network edge intelligent management: A deep deterministic policy gradient approach for service offloading decision," in *Proc. 2020 Int. Wireless Commun. Mobile Comput.*, Limassol, Cyprus, 2020, pp. 905–910.

[29] W. Zhan et al., "Deep-reinforcement-learning-based offloading scheduling for vehicular edge computing," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5449–5465, Jun. 2020.
[30] V. D. Tuong, T. P. Truong, T.-V. Nguyen, W. Noh, and S. Cho, "Partial computation offloading in NOMA-assisted mobile-edge computing systems using deep reinforcement learning," *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13196–13208, Sep. 2021.
[31] Z. Liu, Y. Zhao, J. Song, C. Qiu, X. Chen, and X. Wang, "Learn to coordinate for computation offloading and resource allocation in edge computing: A rational-based distributed approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 5, pp. 3136–3151, Sep./Oct. 2022.
[32] X. Liu, J. Yu, Z. Feng, and Y. Gao, "Multi-agent reinforcement learning for resource allocation in IoT networks with edge computing," *China Commun.*, vol. 17, no. 9, pp. 220–236, Sep. 2020.
[33] S. Wang, M. Chen, X. Liu, C. Yin, S. Cui, and H. V. Poor, "A machine learning approach for task and resource allocation in mobile-edge computing-based networks," *IEEE Internet Things J.*, vol. 8, no. 3, pp. 1358–1372, Feb. 2021.
[34] S. Wang, X. Li, and Y. Gong, "Energy-efficient task offloading and resource allocation for delay-constrained edge-cloud computing networks," *IEEE Trans. Green Commun. Netw.*, vol. 8, no. 1, pp. 514–524, Mar. 2024.
[35] L. Gao and M. Moh, "Joint computation offloading and prioritized scheduling in mobile edge computing," in *Proc. 2018 Int. Conf. High- Perform. Comput. Simul.*, Jul. 2018, pp. 1000–1007.
[36] M. Adhikari, M. Mukherjee, and S.N. Srirama, "DPTO: A deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 5773–5782, Jul. 2020.
[37] M. Mukherjee et al., "Delay-sensitive and priority-aware task offloading for edge computing-assisted healthcare services," in *Proc. 2020 IEEE Glob. Commun. Conf.*, Dec. 2020, pp. 1–5.
[38] M. Hosseinzadeh, A. Wachal, H. Khamfroush, and D. E. Lucani, "QoS-aware priority-based task offloading for deep learning services at the edge," in *Proc. 2022 IEEE 19th Annu. Consum. Commun. Netw. Conf.*, Jan. 2022, pp. 319–325.
[39] J. X. Liao and X. W. Wu, "Resource allocation and task scheduling scheme in priority-based hierarchical edge computing system," in *Proc. 2020 19th Int. Symp. Distrib. Comput. Appl. Bus. Eng. Sci.*, Oct. 2020, pp. 46–49.
[40] Z. Sharif, L. T. Jung, I. Razzak, and M. Alazab, "Adaptive and priority-based resource allocation for efficient resources utilization in mobile-edge computing," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 3079–3093, Feb. 2023.
[41] Z. He, Y. Xu, M. Zhao, W. Zhou, and K. Li, "Priority-based offloading optimization in cloud-edge collaborative computing," *IEEE Trans. Serv. Comput.*, vol. 16, no. 6, pp. 3906–3919, Nov./Dec. 2023.
[42] L. Luo, "Priority task offloading decision optimization based on multi-objective grey wolf algorithm in Internet of Things edge computing scenarios," in *Proc. IEEE 3rd Int. Conf. Data Sci. Comput. Appl.*, Dalian, China, 2023, pp. 1574–1577.
[43] Y. Wang, Y. Liu, Z. Sun, L. Liu, J. Li, and G. Sun, "Priority-aware task offloading and resource allocation in vehicular edge computing networks," in *Proc. 18th Int. Conf. Mobility, Sens. Netw.*, Guangzhou, China, 2022, pp. 205–212.
[44] W. Chu, P. Yu, Z. Yu, J. C. S. Lui, and Y. Lin, "Online optimal service selection, resource allocation and task offloading for multi-access edge computing: A utility-based approach," *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 4150–4167, Jul. 2023.
[45] L. Wang, X. Zhu, N. Lit, Y. Liv, S. Ma, and L. Zhai, "Dynamic vehicle aware task offloading based on reinforcement learning in a vehicular edge computing network," in *Proc. 18th Int. Conf. Mobility, Sens. Netw.*, Dec. 2022, pp. 263–270.
[46] M. Mubashir, R. Ahmad, A. Saadat, S. R. Chaudhry, A. K. Kiani, and M. M. Alam, "Task classification for optimal offloading and resource allocation in vehicular edge computing," in *Proc. 8th Int. Conf. Fog Mobile Edge Comput.*, Tartu, Estonia, 2023, pp. 15–21.
[47] L. Geng, H. Zhao, J. Wang, A. Kaushik, S. Yuan, and W. Feng, "Deep-reinforcement-learning-based distributed computation offloading in vehicular edge computing networks," *IEEE Internet Things J.*, vol. 10, no. 14, pp. 12416–12433, Jul. 15, 2023.
[48] J. Shi, J. Du, J. Wang, J. Wang, and J. Yuan, "Priority-aware task offloading in vehicular fog computing based on deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 16067–16081, Dec. 2020.
[49] X. Dai, X. Chen, L. Jiao, Y. Wang, S. Du, and G. Min, "Priority-aware task offloading and resource allocation in satellite and HAP assisted edge-cloud collaborative networks," in *Proc. 15th Int. Conf. Commun. Softw. Netw.*, Shenyang, China, 2023, pp. 166–171.
[50] X. Huang, L. He, and W. Zhang, "Vehicle speed aware computing task offloading and resource allocation based on multi-agent reinforcement learning in a vehicular edge computing network," in *Proc. 2020 IEEE Int. Conf. Edge Comput.*, 2020, pp. 1–8.
[51] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.
[52] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 18, no. 2, pp. 319–333, Feb. 2019.

Ashab Uddin (Graduate Student Member, IEEE) received the B.Sc. degree in electrical & electronics engineering from the Chittagong University of Engineering & Technology, Chittagong, Bangladesh, in 2012. He is currently working toward the Ph.D. degree in electrical engineering with the University of Windsor, Windsor, ON, Canada. His research interests include vehicular edge computing, autonomous vehicles, autonomous system, wireless communication, and deep reinforcement learning, and has a strong background in industrial automation, control, and communication systems, with a focus on implementing autonomous FMCG projects and processes.



Ahmed Hamdi Sakr (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Manitoba, Winnipeg, MB, Canada, in 2017. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, ON, Canada. He was a Principal Researcher with Toyota Motor North America R&D. His research interests include connected and autonomous vehicles, wireless and vehicular networks, edge computing, Internet of Things (IoT), signal processing, and machine learning. He was the recipient of the 2018 IEEE Communications Society Young Author Best Paper Award. He is also an Associate Editor for the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY. He leads several research initiatives on the development of cooperative and automated driving functionalities.

Ning Zhang (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2015. He is currently an Associate Professor and the Canada Research Chair with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, ON. His research interests include connected vehicles, wireless networking, and security. He is also a Distinguished Lecturer of IEEE ComSoc, Highly Cited Researcher (Web of Science), and the Vice Chair of IEEE Technical Committee on Cognitive Networks and IEEE Technical Committee on Big Data. He is/was an Associate Editor for IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, IEEE INTERNET OF THINGS JOURNAL, and IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING. He is /was the TPC/General/Symposium Chair for numerous conferences and workshops, such as IEEE ICC, GLOBECOM, VTC, INFOCOM Workshop, and MOBICOM Workshops.