
Article:
Azizi, S., Fatahi, A., Bozorgchenani, A. et al. (2026) PACOB: Priority-aware computation offloading in vehicular edge computing based on multi-armed bandit learning. Computer Communications, 257. 108622. ISSN: 0140-3664

https://doi.org/10.1016/j.comcom.2026.108622

PACOB: Priority-Aware Computation Offloading in Vehicular Edge Computing Based on Multi-Armed Bandit Learning

Sadoon Azizi[a,*], Ayub Fatahi[a], Arash Bozorgchenani[b], Mohammad Shojafar[c]

[a] *Department of Computer Engineering and IT, University of Kurdistan, Sanandaj, Iran*
[b] *School of Computer Science, University of Leeds, Leeds, UK*
[c] *5G/6GIC, University of Surrey, Guildford, UK*

Abstract

Vehicular Edge Computing (VEC) has emerged as a transformative paradigm in Intelligent Transportation Systems (ITS), enabling vehicles to offload computationally demanding and latency-sensitive tasks to nearby edge or cloud servers. However, efficient task offloading remains challenging due to the dynamic nature of vehicular networks, heterogeneous edge resources, variable service costs, and diverse task requirements. Traditional approaches often fail to distinguish between delay-sensitive and delay-tolerant tasks, leading to suboptimal trade-offs between response time and monetary cost. To address these limitations, this paper proposes *PACOB*, a priority-aware computation offloading framework based on a multi-armed bandit model enhanced with an Upper Confidence Bound (UCB) mechanism. Unlike conventional methods, PACOB employs dual action-value functions to explicitly distinguish task types and enable task-specific optimization. For delay-sensitive tasks, it prioritizes minimizing response time to meet strict deadlines, whereas for delay-tolerant tasks, it focuses on cost efficiency while preserving service quality. By dynamically learning optimal offloading policies and adapting to real-time network variations, PACOB achieves a balanced optimization of latency and cost across heterogeneous environments. Extensive simulations across diverse VEC scenarios demonstrate PACOB's superiority over four baselines: Random (RND), Priority-Based Heuristic (PBH), ϵ-greedy, and standard UCB. Compared with standard UCB, PACOB reduces average response time by about 35% and improves the percentage of deadline-satisfied tasks (PDST) by 18%, with only a moderate 25% rise in cost for delay-sensitive tasks. For delay-tolerant tasks, PACOB and PBH show comparable cost efficiency, but PACOB achieves slightly better response time and PDST through adaptive redirection of some tasks to underutilized edge servers. These results confirm PACOB's robustness and adaptability in balancing latency, reliability, and cost efficiency for real-time vehicular edge computing.

Keywords: vehicular edge computing, computation offloading, multi-armed bandit, upper confidence bound, response time, and monetary cost.

1. Introduction

The advent of Vehicular Edge Computing (VEC) has revolutionized the landscape of Intelligent Transportation Systems (ITS) by enabling computational offloading to edge and cloud servers [1, 2, 3]. As vehicles become increasingly equipped with sensors, autonomous driving systems, and high-bandwidth connectivity, the demand for real-time processing of delay-sensitive and computation-intensive tasks continues to grow [4, 5]. Applications such as autonomous navigation, real-time traffic management, and in-vehicle entertainment systems require ultra-low latency and high reliability to meet the stringent requirements of modern vehicular networks [6, 7, 8].

VEC has emerged as a transformative paradigm, bridging the computational gap between resource-constrained vehicles and the increasing demand for real-time, high-performance applications. By leveraging edge servers deployed at Road Side Units (RSUs) and remote cloud servers, VEC enables computational offloading to reduce latency and enhance Quality of Service (QoS). However, efficient task offloading in VEC remains a significant challenge [9, 10]. The key difficulties stem from the heterogeneity of edge servers, dynamic traffic variations, limited computational resources, and fluctuating service costs. Moreover, vehicles operate in a highly dynamic environment where task urgency, network conditions, and resource availability are continuously changing. Consequently, real-time, intelligent offloading decisions are essential to maximize computational efficiency, minimize response time, and optimize cost.

To tackle these challenges, Multi-Armed Bandit (MAB)-based learning algorithms have gained attention for adaptive task offloading [11, 12, 13, 14]. MAB frameworks inherently balance exploration (assessing the performance of offloading options) and exploitation (leveraging the best-known options), making them well-suited for dynamic and

*Corresponding author
Email address: s.azizi@uok.ac.ir (Sadoon Azizi)

uncertain environments like VEC. Upper Confidence Bound (UCB)-based approaches, in particular, offer robust theoretical guarantees in decision-making under uncertainty [15]. However, while UCB-based task offloading strategies have been widely used in VEC systems, most existing works fail to address the challenge of task heterogeneity [11, 13, 16]. Tasks vary significantly in terms of delay sensitivity and computational complexity, and a one-size-fits-all offloading policy fails to accommodate their distinct requirements. Without explicitly considering task priority and computational demands, conventional UCB-based methods risk suboptimal resource allocation and inefficient cost management.

In this paper, we mathematically model the computation offloading problem in VEC systems with the dual objective of minimizing response time and monetary cost. To address this challenge, we propose PACOB (**P**riority-**A**ware **C**omputation **O**ffloading in VEC based on multi-armed **B**andit learning), a novel offloading algorithm that leverages an enhanced Upper Confidence Bound (UCB) framework. A key innovation of PACOB is its dual action-value function, which allows the algorithm to distinguish between delay-sensitive and delay-tolerant tasks, enabling task-specific optimization and adaptive decision-making. For delay-sensitive tasks, the proposed algorithm prioritizes response time minimization to meet stringent deadlines, whereas for delay-tolerant tasks, it focuses on cost reduction to ensure economic efficiency. By dynamically learning and adapting to network variations, PACOB intelligently balances performance and cost, significantly improving QoS and resource utilization efficiency in dynamic VEC environments.

The main contributions of this paper are summarized as follows:

- We formulate the task offloading problem in VEC systems as a Mixed Integer Linear Programming (MILP) model, explicitly optimizing response time and monetary cost to enhance both computational efficiency and economic feasibility.
- To address task heterogeneity, we introduce a priority-aware offloading mechanism, distinguishing between delay-sensitive and delay-tolerant tasks by incorporating their unique sensitivities to response time and cost.
- We develop PACOB, a novel UCB-based adaptive task offloading algorithm, leveraging a dual action-value function to intelligently manage task allocation and optimize offloading decisions in real-time.
- We conduct extensive simulation experiments across various realistic VEC scenarios, demonstrating significant improvements in response time, cost efficiency, and task deadline satisfaction. Our results showcase the superiority of PACOB over baseline methods, validating its effectiveness in optimizing task offloading for VEC systems.

The remainder of this paper is structured as follows: Section II reviews the existing literature on task offloading in VEC systems, highlighting current challenges and advancements. Section III presents the system model, detailing the system architecture, the offloading environment, and the problem formulation. Section IV introduces the proposed PACOB algorithm, outlining its core mechanisms and enhancements. Section V describes the performance evaluation setup and provides a comprehensive analysis of the simulation results. Finally, Section VI concludes the paper by summarizing the findings and outlining potential directions for future research.

2. Related Works

Computation offloading in edge and VEC environments has received extensive attention in recent literature [17, 18]. This section aims to review the related works that focus on computation offloading, examining the diverse methodologies that have been employed to tackle this challenge. Each approach is summarized with its specific goals and contributions. We provide a comprehensive categorization of these related works, classifying them into four main categories: heuristic methods, metaheuristic techniques, game theory approaches, and learning-based techniques. This classification helps in understanding the different strategies and their effectiveness in addressing computation offloading problems in VEC and Multi-access Edge Computing (MEC) contexts.

2.1. Heuristic Methods

In [19], Singh et al. present a heuristic algorithm aimed at minimizing the completion time of computational tasks in a multi-user multi-MEC setup. The proposed algorithm allows mobile devices to independently decide whether to offload a task using only local knowledge. Three approaches for offloading decisions and three mechanisms for selecting MEC servers are evaluated. The heuristic algorithm demonstrates the capability to produce solutions that closely approximate the theoretical optimal allocation, validated through extensive simulations. Azizi et al. [20] introduce DECO, a deadline-aware and energy-efficient framework for computation offloading in heterogeneous MEC environments. Unlike many heuristic approaches that independently handle delay or energy optimization, DECO jointly minimizes IoT device energy consumption while ensuring that task deadlines are satisfied through a mixed-integer nonlinear programming (MINLP) formulation. The authors further design heuristic rules for local offloading decisions and propose a priority-aware scheduling algorithm for distributing tasks across heterogeneous edge servers. Although DECO achieves notable improvements in deadline satisfaction and response time, it does not account for monetary costs (e.g., service pricing) or communication overheads, limiting its adaptability in cost-sensitive or bandwidth-constrained MEC systems. Bute et al. in

[21] propose a distributed task offloading scheme designed for VEC networks. The algorithm aims to optimize computational resource utilization and minimize task completion time. By leveraging vehicle-to-vehicle (V2V) communication and distributed decision-making, the scheme effectively manages dynamic network conditions. The proposed algorithm uses a fuzzy logic approach and considers four key metrics: task priority, resource availability, communication delay, and task size. In [22], Ataie et al. present D2FO, a scalable distributed algorithm for offloading time-sensitive tasks in Fog-Cloud IoT environments. The proposed algorithm employs a semi-network-aware distributed scheduling mechanism, focusing on optimizing acceptance rate, response time, and network resource usage. By dynamically adjusting the routing paths and utilizing real-time bandwidth calculations, D2FO outperforms existing methods in handling delay-sensitive applications, as demonstrated through extensive simulations.

Despite the low computational complexity of heuristic algorithms, their solutions often fail to maintain high quality—i.e., near-optimal and stable performance—under heterogeneous and dynamic conditions [23, 24]. These approaches may struggle to cope with diverse task types and the simultaneous consideration of response time and computational cost. Another common limitation is the difficulty in achieving balanced load distribution, which can result in inefficient resource utilization and increased latency under varying network conditions.

2.2. Metaheuristic Methods

In [25], Su et al. propose a novel algorithm for task offloading in VEC, leveraging a multistrategy improved sparrow search algorithm for edge-cloud collaboration. This algorithm optimizes task offloading locations by considering both delay and energy consumption. The proposed method significantly enhances the accuracy and speed of task offloading, as demonstrated through simulations. Materwala et al. in [26] introduce a novel offloading algorithm that leverages an evolutionary genetic algorithm to optimize energy consumption in edge-cloud computing platforms while maintaining service level agreement requirements. The algorithm employs an adaptive penalty function to handle latency and processing time constraints. To minimize the execution time of applications in VEC systems, de Souza et al. [27] introduce a bee colony-based task offloading algorithm. This algorithm reliably reduces application execution time by considering connectivity and energy constraints, along with several contextual parameters, to provide efficient solutions within a feasible time frame for delay-sensitive applications. Long et al. [28] proposed a task offloading algorithm optimized for an edge-cloud collaborative architecture in 6G networks. This algorithm minimizes energy consumption, delay, and multi-node load imbalance as key optimization objectives. Building on the approach introduced in [29], the enhanced algorithm demonstrates fast convergence and effectively balances workloads across edge nodes, reducing delays from overloaded nodes and minimizing energy wastage caused by idle nodes. Luo et al. in [30] focus on a multi-objective optimization approach to minimize delay and cost in VEC systems. The authors establish an offloading framework that considers tasks with varying computation requirements. By formulating a MINLP problem, they aim to optimize offloading decisions and resource allocation. The proposed solution, a particle swarm optimization-based computation offloading algorithm, achieves Pareto-optimal solutions, balancing both delay and cost effectively.

Metaheuristic algorithms offer significant advantages for solving computation offloading problems in VEC environments. Their strong search capabilities enable them to explore the solution space thoroughly and generate near-optimal solutions, making them effective for complex optimization tasks. However, the high execution time associated with these algorithms can be a major drawback. This limitation hinders their applicability in real-time systems and dynamic environments where quick decision-making is crucial.

2.3. Game Theory Methods

A two-stage task offloading mechanism to optimize task completion rates and participant utilities in 5G heterogeneous networks is proposed by Hui et al. [31]. The first stage uses a second price sealed auction model to help macro cell base stations select the optimal small cell base stations based on task requirements and available resources. The second stage involves a bargaining game between the macro cell base stations and autonomous vehicles to finalize task offloading agreements, ensuring efficient resource utilization and improved service performance.

Wang et al. in [32] addresse the problem of task offloading in VEC environments, with a focus on optimizing costs associated with offloading, such as computational and communication expenses. The authors propose a Stackelberg game-based approach to model the interaction between vehicles and edge servers, aiming to achieve an optimal balance that minimizes costs while improving system efficiency. The method considers the interests of both parties (vehicles and edge servers) and provides an efficient resource allocation solution. Chen et al. [33] present a hierarchical framework to address multi-user task offloading and resource purchasing in MEC systems. The two-stage optimization model minimizes user costs and maximizes server profits. The first stage uses a multi-channel access game to achieve Nash equilibrium for user access decisions, while the second stage employs a Stackelberg game to optimize resource pricing and purchasing, ensuring efficient and distributed decision-making. In [34], Cheng et al. tackles the joint task offloading and service caching problem in VEC networks to minimize total task delay, energy consumption, and cost. Their proposed algorithm uses dynamic programming for service caching and a many-to-one matching game for task offloading. This cooperative approach between edge servers and vehicles optimizes

resource utilization and improves QoS, validated through extensive simulations.

Game theoretic algorithms provide robust frameworks for optimizing computation offloading in VEC environments by promoting cooperation among participants and effectively balancing resource allocation. Their strategic models, such as auction-based mechanisms, non-cooperative games, and hierarchical game frameworks, ensure efficient task execution and resource utilization. However, the complexity of these algorithms and the need for extensive computational resources can be challenging in highly dynamic and real-time systems.

2.4. Learning-based Methods

In this subsection, we review several learning-based algorithms, including Reinforcement Learning (RL), Deep Reinforcement Learning (DRL), and MAB theory, proposed for computation or task offloading in VEC environments. In [35], Zhang et al. leverage a deep Q-learning algorithm to optimize task offloading in VEC networks. By using a synchronized random walk model to simulate real traffic conditions, the scheme integrates cloud and edge computing resources to reduce processing delays. This approach effectively addresses the dynamic nature of vehicular networks, ensuring improved adaptability and minimized delays in task processing. Fan et al. in [36] propose a twin delayed deep deterministic policy gradient (TD3) algorithm to optimize task offloading in a VEC scenario. By exploiting flexible cooperation between RSUs, the scheme addresses task result reception failures due to vehicle mobility and load imbalances among RSUs. The TD3-based algorithm minimizes total task processing delay and energy consumption by jointly optimizing task offloading, computing resource allocation, and transmission power.

Sun et al. [11] introduce an adaptive task offloading algorithm (ALTO) based on MAB theory to minimize average offloading delay in VEC environments. The ALTO algorithm allows task vehicles to learn the delay performance of service vehicles dynamically without requiring frequent state information exchange. The algorithm is augmented with input-awareness and occurrence-awareness to adapt to varying workloads and vehicle movements, demonstrating superior performance and significantly reducing delay in both synthetic and realistic highway scenarios. Misra et al. in [37] present a novel approach for optimizing task offloading in IoT environments using a two-tier scheme. Initially, a greedy algorithm selects a nearby fog node for task offloading, reducing initial selection time. Subsequently, an ϵ-greedy nonstationary MAB strategy dynamically allocates subtasks across multiple fog nodes, balancing latency and resource usage. The proposed framework effectively handles dynamic conditions, achieving lower latency and higher speed up in simulations compared to existing methods.

In [38], Xue et al. propose a collaborative partial offloading and resource allocation framework for VEC networks, aiming to minimize the long-term computational overhead—including delay and energy consumption—of vehicles. The system leverages idle vehicles, RSUs, and local nodes as service providers, and models the task offloading problem as a Markov game. A Multi-Agent Deep Reinforcement Learning algorithm, PORA-MATD3, is developed to jointly optimize offloading, task migration, and resource allocation using centralized training and distributed execution. Additionally, a task migration strategy is integrated to ensure reliable delivery of computation results under vehicle mobility constraints. Bozorgchenani et al. [13] address the computation offloading challenge in dynamic VEC environments using MAB theory. The study introduces two approaches: an on-line learning algorithm for real-time network selection to minimize latency, and an off-policy learning algorithm leveraging historical offloading data to detect changes in traffic load and optimize network selection. These methods ensure efficient task offloading by dynamically selecting the least congested networks and reducing task loss in a VEC scenario.

Learning-based algorithms offer robust solutions for task offloading in VEC environments. These algorithms can adapt to dynamic network conditions, optimize resource utilization, and minimize task processing delays. However, they face several delay-sensitive challenges that need to be considered. Firstly, they often require substantial computational resources for execution and training, particularly DRL techniques. Secondly, their convergence time can be challenging, leading to slow adaptation to environmental changes. Lastly, handling the diverse requirements of tasks remains a complex issue in these algorithms.

While existing studies have applied learning-based and MAB-inspired approaches to task offloading in VEC, most overlook the importance of prioritizing heterogeneous task requirements explicitly during decision-making. Notably, Bozorgchenani et al. [13] proposed a learning-based computation offloading framework for VEC environments that dynamically adapts to changing conditions. However, their model does not distinguish between task priorities and lacks an explicit cost-awareness mechanism. In contrast, our proposed framework—PACOB—incorporates both monetary cost and task heterogeneity by leveraging a dual action-value structure within the UCB framework. Unlike traditional UCB variants, PACOB introduces a priority-aware learning paradigm that treats delay-sensitive and delay-tolerant tasks differently, with separate optimization objectives and reward calculations. This design enables PACOB to make fast, lightweight, and context-aware decisions tailored to the specific needs of each task type. Furthermore, by jointly minimizing response time and system cost, our approach achieves superior performance under dynamic and resource-constrained VEC conditions.

3. System Model

In this section, we first describe the system architecture and the offloading environments under consideration.

Following this, we present the mathematical model and formulate the problem.

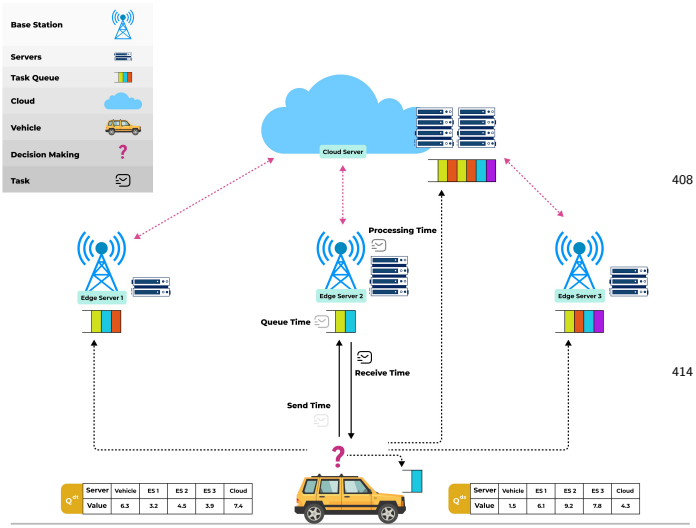


Figure 1: An overview of the VEC architecture and the available task processing options for a smart vehicle, including local execution, offloading to nearby heterogeneous edge servers, and remote cloud servers.

3.1. System Architecture and Offloading Environment

The system architecture of the considered VEC network consists of three main layers: cloud layer, edge layer, and vehicular layer, as illustrated in Fig. 1. In the cloud layer, Cloud Service Providers (CSPs) offer platforms, infrastructure, applications, and storage services via the internet, enabling users to access a wide range of resources on demand. CSPs typically provide these services through a pay-as-you-go model, allowing users to scale their computational resources based on fluctuating needs. Data centers in this layer house powerful physical machines or servers that cater to various customer demands. In the edge layer, edge service providers deploy servers and services close to the data source to reduce latency and enhance performance. Although these edge resources are more limited in capacity compared to cloud resources, they offer advantages in terms of proximity and responsiveness. The vehicular layer comprises smart vehicles equipped with limited computational and storage capabilities. These vehicles outfitted with advanced technologies such as sensors, cameras, and communication devices, can collect and process environmental data and interact with other vehicles, infrastructure, and the internet. This connectivity enables vehicles to share data and collaborate on tasks, enhancing safety, performance, and convenience for drivers and passengers.

While smart vehicles possess limited computational and storage capabilities, these resources are often insufficient for processing computation-intensive tasks. Therefore, in the proposed model, vehicles can either execute tasks locally or offload them to external computing resources. As illustrated in Fig. 1, a generated computational task can be processed locally on the vehicle or offloaded through Vehicle-to-Infrastructure (V2I) or Vehicle-to-Cloud (V2C) communication links. It is worth noting that each vehicle communicates wirelessly with its nearest base station (BS), which serves as a gateway to the VEC infrastructure. The BS maintains high-speed backhaul connections with multiple edge servers and a centralized cloud server, enabling indirect task offloading from vehicles to the cloud through the BS.

The offloading decision for each task is influenced by multiple dynamic factors, including server workload, processing cost, network traffic status, and the task's own requirements such as size and deadline. By continuously collecting and analyzing these environmental data, the proposed model performs data-driven decision-making to determine whether a task should be processed locally or offloaded to edge or cloud resources, thereby achieving a balanced trade-off between response time and cost efficiency.

In practical VEC environments, ensuring security, preserving user privacy, and maintaining fault-tolerant service continuity are fundamental requirements due to the highly dynamic, heterogeneous, and safety-critical nature of vehicular networks [39, 40, 41, 42]. A large body of recent research has focused on developing privacy-preserving reputation management and trust mechanisms to enhance the reliability of cloud-assisted and cooperative vehicular systems. Although these approaches address essential aspects of security and trustworthiness, they operate largely orthogonal to the performance-driven computation offloading model considered in this work and therefore are not explicitly integrated into our formulation. Nevertheless, their relevance to real-world deployments is undeniable, and such mechanisms can be incorporated into future extensions of our framework to support secure, privacy-aware, and resilient computation offloading in next-generation VEC systems.

3.2. Problem Formulation

In this subsection, we formulate the problem of task offloading in VEC environments with the primary objectives of minimizing response time and monetary cost for tasks generated by smart vehicles.

3.2.1. System Overview

We consider a VEC system comprising a smart vehicle equipped with an intelligent agent responsible for real-time task offloading decisions, multiple edge servers, and a powerful cloud server. The system operates using a two-level timing structure consisting of epochs and time-slots, where each epoch is composed of multiple consecutive time-slots. An epoch represents a relatively stable environmental period during which system characteristics, such as task arrival rates, server workloads, background traffic intensity, and service costs, remain unchanged. However, these parameters may vary across different epochs to emulate the

dynamic and non-stationary nature of real-world VEC environments.

At each time slot, computational tasks are generated by the vehicle according to a Poisson arrival process. Consequently, depending on the arrival rate, multiple tasks may be generated during a given time slot. The generated tasks are assumed to be independent of one another [11, 43, 31, 14]. For each generated task, the intelligent agent independently performs an offloading decision by observing the current system state, including server workload, network delay, and service cost parameters. Based on these observations, the agent determines whether the task should be executed locally or offloaded to an edge or cloud server. After task execution, the corresponding system feedback is collected and the associated action-value function is updated accordingly. Since system conditions may change across different epochs, the agent continuously adapts its decision-making policy to evolving environmental dynamics.

Each task generated by the vehicle can be represented as a 4-tuple: $t_i = \{t_i^{\text{size}}, t_i^{\text{deadline}}, t_i^{\text{input}}, t_i^{\text{type}}\}$, where t_i^{size} denotes the task size in Million Instructions (MI), t_i^{deadline} specifies the deadline requirement in milliseconds (ms), t_i^{input} represents the input file size in Kilobytes (KB), and t_i^{type} determines the task category, which in this work is either delay-sensitive or delay-tolerant. It is worth mentioning that while the task size and input file size are often correlated in practice, they describe different aspects of a computational job: t_i^{input} quantifies the amount of data transmitted for offloading (communication workload), whereas t_i^{size} reflects the total computational demand (processing workload). This distinction enables a more accurate modeling of heterogeneous vehicular tasks and aligns with prior studies (e.g., [43, 20]) that separate communication and computation characteristics in distributed computing systems. Delay-sensitive tasks are generated by latency-critical applications, such as autonomous driving control systems and emergency response mechanisms, where timely decision-making is essential. Delay-tolerant tasks arise from computation-intensive applications, including real-time data analytics and in-vehicle entertainment systems, which require substantial processing resources but can tolerate longer response times.

The smart vehicle, denoted as s_0, has limited computational power represented by s_0^{proc}. We assume that local processing of tasks by the vehicle incurs no cost. The set of available servers is denoted as: $S = \{s_1, \ldots, s_j, \ldots, s_m\}$. Each server s_j is characterized by a 4-tuple: $s_j = \{s_j^{\text{proc}}, s_j^{\text{bw}}, s_j^{\text{cost}}, s_j^{\text{type}}\}$ where s_j^{proc} represents the processing power in Million Instructions Per Second (MIPS), s_j^{bw} denotes the bandwidth in Megabits per second (Mbps), s_j^{cost} signifies the processing cost of server s_j in Grid Dollar per second (G$ps) and s_j^{type} indicates the server type, which can be either edge or cloud. In our system, both edge and cloud servers process computational tasks from various vehicles.

To formalize the task offloading decision process, we define two primary decision variables. The binary variable x_i indicates whether task t_i is offloaded ($x_i = 1$) or processed locally by the vehicle ($x_i = 0$). For offloaded tasks, y_i^j denotes whether task t_i is processed by server s_j ($y_i^j = 1$) or not ($y_i^j = 0$). Each task must either be processed locally or offloaded, ensuring that x_i is binary, i.e., $x_i \in \{0, 1\}$. Additionally, if a task is offloaded, it can only be processed by one server, requiring y_i^j to be binary as well, i.e., $y_i^j \in \{0, 1\}$. These constraints ensure the exclusivity of task assignment, guaranteeing that each task is either processed locally or offloaded to a single server. Additionally, each task must be completed within its specified deadline. Let T_i denote the total time taken to complete task t_i. The deadline constraint can be formulated as: $T_i \leq t_i^{\text{deadline}}$, for all generated tasks. This ensures that the total processing time for each task does not exceed its deadline.

3.2.2. Optimization Problem

In VEC systems, minimizing response time and monetary cost is crucial. Response time is particularly important for delay-sensitive tasks to ensure deadlines are met, while minimizing monetary cost is essential for users, especially for computationally intensive tasks that require affordable service providers.

Response Time: The response time for a task depends on whether it is processed locally or offloaded. When a task t_i is processed locally by the vehicle, the response time consists of the local queueing time and the local processing time. Mathematically, the local response time T_i^{local} can be expressed as:

$$T_i^{\text{local}} = T_i^{\text{queue}} + T_i^{\text{process}} \tag{1}$$

where T_i^{queue} is the local queueing time and T_i^{process} is the local processing time. The processing time can be calculated as:

$$T_i^{\text{process}} = \frac{t_i^{\text{size}}}{s_0^{\text{proc}}} \tag{2}$$

where t_i^{size} is the number of instructions of task t_i, and s_0^{proc} is the processing power of the vehicle.

The queueing time depends on the number of tasks already in the queue and their processing times. The offloading response time T_i^{offload} can be expressed as:

$$T_i^{\text{offload}} = T_i^{\text{send}} + T_i^{\text{queue}} + T_i^{\text{process}} + T_i^{\text{receive}} \tag{3}$$

where T_i^{send} is the sending time to the server, which includes the transmission time and the propagation time, T_i^{process} is the processing time at the server, and T_i^{receive} is the time to receive the result.

The sending time is given by:

$$T_i^{\text{send}} = T_i^{\text{trans}} + T_i^{\text{prop}} \tag{4}$$

where the transmission time to the server s_j can be calculated as:

$$T_i^{\text{trans}} = \frac{t_i^{\text{input}}}{s_j^{\text{bw}}} \tag{5}$$

in which t_i^{input} is the input file size of task t_i, and s_j^{bw} is the bandwidth of server s_j. It is worth mentioning that the wireless bandwidth between the vehicle and the base station is not modeled explicitly; instead, its effect is implicitly captured within the overall transmission delay. The propagation time T_i^{prop} depends on the distance between the vehicle and the server. For edge servers, this delay is negligible due to their close proximity to the vehicle [20, 14], while for the cloud server, it is assumed to be a constant value reflecting the longer network distance.

The processing time at server s_j is given by:

$$T_i^{\text{process}} = \frac{t_i^{\text{size}}}{s_j^{\text{proc}}} \tag{6}$$

where s_j^{proc} is the processing power of server s_j.

For the queueing time at the server s_j, we assume an M/M/1 queue model. The receiving time T_i^{receive} is typically negligible due to the smaller size of the results and the high bandwidth, so it is often ignored in similar works [44]. Finally, the response time for task t_i is given by:

$$T_i = x_i \cdot T_i^{\text{offload}} + (1 - x_i) \cdot T_i^{\text{local}} \tag{7}$$

Ensuring task deadlines are met is crucial, and maximizing the percentage of tasks meeting their deadlines is a key performance metric. We define a binary variable δ_i to indicate whether the deadline for task t_i is met. Specifically, $\delta_i = 1$ if the total response time T_i of task t_i is less than or equal to its deadline t_i^{deadline}, and $\delta_i = 0$ otherwise.

The percentage of tasks meeting their deadlines is calculated as:

$$\psi = \frac{\sum_{i=1}^{N} \delta_i}{N} \times 100 \tag{8}$$

where ψ represents the percentage of tasks meeting their deadlines, N is the total number of tasks generated by the vehicle, and δ_i is a binary variable indicating whether task t_i meets its deadline.

Monetary Cost: In our system, the monetary cost of computational servers varies across epochs and is determined by service providers based on factors such as traffic volume, resource availability, energy prices, and other dynamic conditions. The cost for processing a task is calculated using the server's unit processing cost and the time required to complete the task:

$$C_i = T_i^{\text{process}} \cdot s_j^{\text{cost}} \cdot x_i \tag{9}$$

Here, C_i represents the cost of processing task t_i on server s_j, where s_j^{cost} is the unit processing cost of the server, and T_i^{process} is the processing time. Notably, for local processing, the cost is zero, as no external computational resources are utilized.

It is worth noting that the current cost model primarily focuses on server-side processing cost to clearly isolate the impact of offloading decisions on computational resource utilization. Additional factors such as communication energy consumption and vehicle-side energy expenditure can also influence the overall system cost in practical VEC environments. However, incorporating these factors would require detailed physical-layer modeling and application-specific energy profiling, which is beyond the scope of this work. These aspects are therefore left for future extension of the model.

Problem Optimization: The objective of the system is to jointly minimize the average response time and the cumulative monetary cost across all generated tasks. The optimization problem can be formulated as follows:

$$\min \quad \alpha \frac{1}{N} \sum_{i=1}^{N} T_i + \beta \sum_{i=1}^{N} C_i \tag{10a}$$

$$\text{s.t.} \quad T_i \le t_i^{\text{deadline}}, \quad \forall i = 1, \ldots, N \tag{10b}$$

$$x_i \in \{0, 1\}, \quad \forall i = 1, \ldots, N \tag{10c}$$

$$y_i^j \in \{0, 1\}, \quad \forall i = 1, \ldots, N, \forall j = 1, \ldots, m \tag{10d}$$

$$\sum_{j=1}^{m} y_i^j = x_i, \quad \forall i = 1, \ldots, N \tag{10e}$$

where α and β are weighting factors reflecting the relative importance of response time and monetary cost. The response time term is averaged over all tasks to provide a workload-independent performance metric that remains comparable across different traffic conditions and task-generation rates. In contrast, the monetary cost term is modeled as the cumulative system cost, since the total operational expenditure incurred by offloading decisions is itself a key optimization objective in VEC environments.

Constraint (10b) enforces the deadline requirement, ensuring that each task is completed within its specified deadline to satisfy QoS requirements, particularly for delay-sensitive applications. Constraint (10c) defines the binary offloading decision variable, indicating whether a task is processed locally or offloaded to a computational server. Constraints (10d) and (10e) guarantee exclusive task assignment by ensuring that an offloaded task is allocated to exactly one server among the available computational servers.

Given the two types of tasks (delay-sensitive and delay-tolerant), the importance weights associated with response time and monetary cost may differ. At this stage, the optimization problem is formulated in its raw form without normalization to establish a unified and generic optimization framework. The normalization of response time and cost terms, along with adaptive adjustment of the weighting factors α and β, is later incorporated into the PACOB learning framework to enable balanced and task-aware decision making.

Although the MILP formulation does not explicitly

include task-type variables, task heterogeneity is implicitly captured through the adaptive learning mechanism of PACOB. During the learning phase, PACOB dynamically tunes the optimization priorities according to task characteristics, emphasizing response-time minimization for delay-sensitive tasks and cost efficiency for delay-tolerant tasks. This hierarchical integration between the optimization formulation and the PACOB framework enables effective handling of task diversity while preserving tractability and scalability.

4. Proposed Solution

In this section, we develop a learning-based task offloading algorithm based on MAB theory to optimize task processing in VEC systems. Our proposed approach enables the agent to learn the delay and cost performance of various computing resources and make informed decisions on where to offload computational tasks, with the goal of minimizing both expected offloading delay and monetary cost.

4.1. Overview of UCB Algorithm

The MAB problem is a classical model in RL, where an agent needs to choose among a set of actions (or arms) whose expected rewards are initially unknown and must be learned progressively through interaction with the environment. The challenge is to balance exploration (trying out different arms to learn their reward distributions) with exploitation (selecting the arm that currently seems to offer the best reward). In the context of our VEC system, each potential offloading destination (local processing, edge server, cloud server) represents an arm in the MAB. The agent's task is to learn which option (arm) provides the best trade-off between delay and cost for each task.

The UCB algorithm is a widely used approach in MAB problems, particularly in scenarios where balancing exploration and exploitation is crucial. UCB works by maintaining an estimate of the expected reward for each arm and selecting the arm with the highest UCB on the expected reward. This approach ensures that the agent explores different options while also exploiting the best-known option to maximize cumulative rewards.

In the UCB algorithm, the confidence bound for each arm is calculated based on the mean reward observed so far and a term that accounts for the uncertainty or variance in the estimate. The algorithm selects the arm with the highest UCB index, which encourages exploration of less frequently chosen arms, especially in the early stages when the agent has little information about the environment.

Mathematically, the UCB index for an arm a at time t is given by:

$$\mathrm{A}(a,t) = \hat{\mu}_a(t) + c\sqrt{\frac{\ln t}{N_a(t)}} \tag{11}$$

where $\hat{\mu}_a(t)$ is the estimated mean reward for arm a at time t, $N_a(t)$ is the number of times arm a has been selected up to time t, c is a tunable parameter that controls the trade-off between exploration and exploitation, and $\sqrt{\frac{\ln t}{N_a(t)}}$ introduces a logarithmic growth factor, ensuring that all arms continue to be explored over time.

In our proposed solution, the agent will utilize an enhanced version of the UCB algorithm tailored to the VEC environment. The vehicle will continuously update its estimates of delay and cost for each offloading option and adaptively choose the best option for each task, considering the dynamic nature of the network, resource availability, monetary cost, and task type. By improving upon the basic UCB algorithm, we aim to handle the specific challenges of task better offloading in heterogeneous and rapidly changing VEC systems.

4.2. Challenges of Applying UCB in our VEC System

In this subsection, we highlight the key challenges of applying the UCB technique to our VEC system, emphasizing the importance of our proposed algorithm which will be presented in the next subsection.

Diverse Task Types: Our proposed system considers two types of tasks generated by the smart vehicle: (1) delay-sensitive tasks, which lose value if not completed within their deadlines, and (2) delay-tolerant tasks, which retain value even if delayed but typically require substantial computational resources. This diversity in task requirements presents challenges for computing UCB values, as a uniform formula may fail to account for the specific priorities of each task type. For instance, delay-sensitive tasks prioritize response time to meet their strict deadlines, while delay-tolerant tasks focus on cost efficiency without compromising their high computational demands. To effectively manage this diversity, our approach adopts a tailored UCB calculation, ensuring that the unique characteristics and importance of both task types are adequately addressed.

Variable Server Costs: As discussed earlier, the cost charged by servers for executing instructions varies across different time periods. A server might be cost-effective at one epoch but not at another due to changing queue states and resource availability. Thus, the agent must dynamically adjust its offloading decisions to maximize cumulative rewards, which, in this study, represent a balance between response time and cost.

Fluctuating Server Traffic Patterns: Another significant challenge is the variable traffic patterns at servers over different time slices, a reflection of real-world scenarios. The agent must explore the traffic conditions of various servers to offload tasks to those with favorable queue states, ensuring manageable wait times and improved overall system performance in terms of response time.

Heterogeneous Edge Servers: The heterogeneity of edge servers poses another challenge. In the real world, and in

our proposed system, the computational and storage capacities of edge servers vary. For instance, the processing time for a given task may differ between servers, impacting response times. Selecting the appropriate server in each time slice is crucial due to this heterogeneity. Therefore, in the following section, we present a solution to address these challenges.

By identifying and addressing these challenges, our proposed algorithm aims to enhance the performance of task offloading in VEC systems, ensuring both efficiency and effectiveness in dynamic and heterogeneous environments.

4.3. Proposed Solution based on UCB

In this section, we introduce an enhanced UCB-based algorithm tailored to the specific demands of task offloading in VEC systems. To effectively handle task heterogeneity, our method employs separate decision-making strategies for delay-sensitive and delay-tolerant tasks by maintaining two distinct action-value functions (Q-tables). This dual-structure enables the algorithm to optimize decisions based on the unique priorities of each task type. A conceptual overview of this mechanism is illustrated in Fig. 1, where the two Q-tables are explicitly depicted to reflect the system's differentiated learning and offloading behavior.

To effectively manage the diverse task types in VEC systems, where smart vehicles generate both delay-sensitive and delay-tolerant tasks, we introduce a dual action-value table, also referred to as a Q-table. Each Q-table is specialized for one type of task, allowing the agent to learn and optimize the best offloading decisions based on the specific nature of the task at hand.

The objective function we aim to minimize is a weighted sum of the average response time and the total monetary cost, as expressed in Eq.(10a). Given that the units of delay and cost differ, we normalize the rewards for both metrics to ensure a meaningful comparison. The response time for task t_i is normalized as:

$$Z_i^{\text{resp}} = \frac{T_i}{T_i^{\text{deadline}}} \tag{12}$$

The normalized reward for response time is then calculated as:

$$\tilde{R}_i^{\text{RT}} = \begin{cases} 10, & \text{if } Z_i^{\text{resp}} \leq 1 \\ \frac{10}{Z_i^{\text{resp}}}, & \text{if } Z_i^{\text{resp}} > 1 \end{cases} \tag{13}$$

Similarly, the monetary cost for processing task t_i is normalized as:

$$Z_i^C = \frac{C_i}{C_i^{\min}}, \tag{14}$$

where $C_i^{\min}$ represents the minimum possible cost of executing task t_i across all available computational nodes, including edge and cloud servers. This value is determined by evaluating the execution cost of t_i on each server, considering the task's size and the known processing cost at the beginning of each epoch. The agent uses $C_i^{\min}$ to normalize the actual cost C_i of executing the task, ensuring a consistent comparison across different offloading decisions.

The normalized reward for cost is given by:

$$\tilde{R}_i^{\text{C}} = \begin{cases} 10, & \text{if } Z_i^C \leq 1 \\ \frac{10}{Z_i^C}, & \text{if } Z_i^C > 1 \end{cases} \tag{15}$$

For each task generated during time slot t, PACOB updates the Q-value of the selected arm using the following formula:

$$Q(t)_{\text{new}} = \alpha \cdot \tilde{R}_i^{\text{RT}} + \beta \cdot \tilde{R}_i^{\text{C}} \tag{16}$$

where α and β are weighting coefficients used to balance the trade-off between response time and monetary cost, with their values dynamically assigned based on the task type to reflect its specific requirements.

The Q-table is then updated using a weighted moving average, as follows:

$$Q(t)_{\text{updated}} = w_1 \cdot Q(t)_{\text{old}} + w_2 \cdot Q(t)_{\text{new}} \tag{17}$$

where w_1 and w_2 are positive real-valued coefficients that determine the influence of the previous and current Q-values, respectively, in the Q-table update process, with $w_1 + w_2 = 1$. This weighted moving average mechanism enables the Q-values to adapt gradually to new environmental conditions while maintaining stability and preventing abrupt oscillations during online learning.

By implementing two distinct action-value tables—one for delay-sensitive tasks (Q^{ds}) and another for delay-tolerant tasks (Q^{dt})—our algorithm dynamically adjusts its strategy. This structure enables smart vehicles to make optimal offloading decisions that minimize both response time and cost while tailoring the decision-making process to the unique requirements of each task type. This enhanced UCB-based algorithm, therefore, effectively addresses the challenges posed by diverse tasks, variable server costs, fluctuating traffic patterns, and heterogeneous edge servers in VEC systems.

The pseudo-code of the proposed algorithm is presented in **Algorithm 1**. The step-by-step explanation of the algorithm is as follows.

Step 1. Initialization (Line 1): The algorithm starts by initializing the set of available arms (A_t), which includes the vehicle's processor and a subset of the computational servers from the set S. Specifically, $A_t = \{s_0 \cup S\}$, where s_0 represents the vehicle's processor, and S denotes the set of all edge and cloud servers available for task offloading.

Step 2. Q-table Selection (Lines 2-6): The algorithm checks the type of the task t_i. If the task is classified as "delay-sensitive", it reads the Q-table specific to delay-sensitive tasks (Q^{ds}). If the task is "delay-tolerant", it reads the Q-table specific to delay-tolerant tasks (Q^{dt}).

Step 3. UCB Calculation (Lines 7-9): For each arm available in A_t, the algorithm calculates the UCB value using the formula provided in Eq.11.

Step 4. Arm Selection and Task Offloading (Lines 10-11): The arm with the maximum UCB value is selected as the best option for offloading the task t_i. The task is then offloaded to the selected arm, i.e., a.

Step 5. Performance Measurement (Lines 12-14): After offloading the task, the algorithm calculates the response time (T_i) and the monetary cost (C_i) associated with processing the task on the selected arm. These are computed using the respective equations (Eq.7 and Eq.9), respectively. The response time and cost are then normalized using the provided equations (Eq.13 and Eq.15) to standardize the reward values.

Step 6. Q-table Update (Lines 15-18): The algorithm updates the Q-table based on the type of task. For delay-sensitive tasks, it computes a new Q-value ($Q(t)_{\text{new}}$) using a weighted sum of the normalized response time reward (R_i^{RT}) and cost reward (R_i^{C}), with weights α^{ds} and β^{ds}. For delay-tolerant tasks, it similarly computes a new Q-value with weights α^{dt} and β^{dt}. The Q-table is then updated by combining the old Q-value ($Q(t)_{\text{old}}$) with the new Q-value using a weighted moving average, where w_1 and w_2 are the weights.

Step 7. Return the selected arm (Line 19): The algorithm returns the offloading decision and the updated Q-table to be used in future decisions.

This algorithm enables adaptive task offloading in VEC systems by leveraging an enhanced UCB framework that explicitly distinguishes between delay-sensitive and delay-tolerant tasks. By maintaining two lightweight and independent action-value functions, the proposed approach enables task-specific optimization that dynamically adapts to varying network conditions, server costs, and task requirements. Although the algorithm uses a dual Q-function structure, each function is updated only when tasks of the corresponding type are generated. This selective update mechanism ensures minimal computational overhead, making the method highly suitable for real-time VEC environments. The added complexity is linear in the number of servers and remains comparable to traditional UCB implementations. As such, PACOB preserves the fast decision-making and low complexity characteristics essential for latency-sensitive vehicular systems, while offering significantly improved adaptability and performance.

4.4. Computational Complexity and Scalability Analysis

Since PACOB operates as an *online decision-making* algorithm, its computational cost is evaluated on a per-task basis rather than through a separate offline training phase. Unlike deep reinforcement learning approaches that require computationally expensive iterative training procedures, PACOB incrementally updates its action-value estimates online using observed rewards obtained during runtime interactions with the environment.

Algorithm 1 Proposed PACOB Algorithm for Task Offloading in VEC Systems

Input: Task t_i with attributes, Processing power of vehicle s_0^{proc}, Processing cost of edge servers and cloud server s_j^{cost}, UCB parameters $\alpha^{\text{ds}}, \beta^{\text{ds}}, \alpha^{\text{dt}}, \beta^{\text{dt}}, c$, Q-table update parameters w_1, w_2, Q-tables Q^{ds}, Q^{dt}, and Historical data for previous offloading decisions
Output: Offloading decision and Updated Q-table

1: Initialize $A_t = \{s_0\} \cup S$, where $S = \{s_1, s_2, \ldots, s_m\}$ is the set of available computational servers; ▷ Define available arms: vehicle and edge/cloud servers
2: **if** t_i^{type} == "delay-sensitive" **then**
3: Read Q^{ds}; ▷ Use Q-table corresponding to delay-sensitive tasks
4: **else**
5: Read Q^{dt}; ▷ Use Q-table corresponding to delay-tolerant tasks
6: **end if**
7: **for each** arm $a \in A_t$ **do**
8: Compute UCB(a, t) using Eq. 11; ▷ Estimate the upper confidence bound for each arm
9: **end for**
10: Select arm a with maximum UCB value; ▷ Choose server with highest expected reward and confidence
11: Offload task t_i to the selected arm a;
12: Compute response time T_i and monetary cost C_i for task t_i using Eqs. 7 and 9, respectively;
13: Normalize reward components (response time and cost) using Eqs. 13 and 15;
14: **if** t_i^{type} == "delay-sensitive" **then**
15: Compute $Q(t)_{\text{new}} = \alpha^{\text{ds}} \times \tilde{R}_i^{\text{RT}} + \beta^{\text{ds}} \times \tilde{R}_i^{\text{C}}$;
16: Update $Q^{\text{ds}} = w_1 \times Q(t)_{\text{old}} + w_2 \times Q(t)_{\text{new}}$; ▷ Gradually update the delay-sensitive Q-table
17: **else**
18: Compute $Q(t)_{\text{new}} = \alpha^{\text{dt}} \times \tilde{R}_i^{\text{RT}} + \beta^{\text{dt}} \times \tilde{R}_i^{\text{C}}$;
19: Update $Q^{\text{dt}} = w_1 \times Q(t)_{\text{old}} + w_2 \times Q(t)_{\text{new}}$; ▷ Gradually update the delay-tolerant Q-table
20: **end if**
21: Return Offloading decision and Updated Q-table;

For each newly generated task, the algorithm first constructs the set of available offloading candidates $A_t = \{s_0\} \cup S$, where $S = \{s_1, s_2, \ldots, s_m\}$ represents the set of m accessible edge and cloud servers, while s_0 denotes local execution. For every candidate server $a \in A_t$, PACOB computes the corresponding UCB score according to Eq. (11) (Algorithm 1, lines 7–9), and then selects the server with the maximum UCB value (line 10). Since the UCB evaluation requires only a single pass over all candidate servers, the per-task time complexity is $O(|A_t|) = O(m)$, where m denotes the number of available offloading servers. The reward update and Q-table update operations are constant-time procedures and therefore do not affect the dominant complexity term.

In terms of memory overhead, PACOB maintains separate lightweight Q-tables for different task types. However, for each incoming task, only the Q-table associated with the current task type is accessed and updated. Since each Q-table stores one action-value entry per server, the overall space complexity also scales linearly with the number of servers, resulting in $O(m)$.

Furthermore, the proposed framework is inherently scal-

able. If additional edge or cloud servers are introduced into the VEC environment, PACOB only requires extending the action space and adding the corresponding entries to the associated Q-table, while the core learning mechanism, reward formulation, and decision-making process remain unchanged. Since the computational and memory complexities grow linearly with the number of available servers, i.e., $O(m)$, the proposed framework remains computationally efficient even in large-scale VEC environments with numerous edge servers or high task arrival rates. In addition, PACOB does not rely on computationally intensive iterative convergence procedures or deep neural network training, which further enhances its practicality for real-time deployment. These characteristics make PACOB a lightweight, scalable, and efficient solution for dynamic large-scale VEC systems requiring rapid online decision-making.

4.5. Deployment Considerations

The PACOB framework is intended to run directly on the vehicle's onboard computing unit, where offloading decisions need to be generated with minimal processing delay. Due to the linear structure of its learning mechanism and the use of two compact Q-tables whose size grows proportionally with the number of available servers, the required memory and computational load remain very low. These characteristics allow PACOB to be deployed on typical embedded processors commonly used in vehicular platforms, without demanding specialized hardware or complex software environments.

4.6. Extensibility of PACOB

PACOB is designed as a lightweight and modular online learning framework, making it naturally extensible for dynamic and heterogeneous VEC environments. Unlike deep reinforcement learning approaches that rely on fixed-dimensional policy networks and often require costly retraining procedures when the environment changes, PACOB adopts an online tabular UCB-based learning mechanism in which each server is modeled as an independent action (arm) associated with its own action-value statistics. Consequently, if additional edge or cloud servers become available, the framework can seamlessly accommodate them by incrementally extending the action space and adding the corresponding entries to the related action-value function. Previously learned statistics associated with existing servers remain reusable, while newly introduced servers are explored and learned online through the standard UCB exploration–exploitation mechanism. Therefore, PACOB can dynamically adapt to changes in server availability without requiring global retraining, policy redesign, or modifications to the state representation.

Furthermore, PACOB can be readily extended to support additional task categories beyond the two representative task types considered in this work. Specifically, a dedicated action-value function can be introduced for each new task category to capture its specific QoS requirements and optimization objectives. Since the learning processes associated with different task categories are performed independently, extending the framework does not require modifications to the core UCB-based learning structure or the overall decision-making process. Instead, only the reward formulation and the corresponding weighting parameters need to be adjusted according to the characteristics of the newly introduced task type.

Although the current study focuses on delay-sensitive and delay-tolerant tasks to maintain analytical clarity and isolate the impact of the proposed dual-Q-table mechanism, the modular design of PACOB also provides the foundation for supporting more fine-grained priority models and mixed-attribute task profiles in future VEC systems. Owing to its online incremental learning capability and linear computational complexity, PACOB remains computationally efficient and scalable even in large-scale VEC deployments with dynamic server availability and heterogeneous workloads. As part of future work, we plan to further investigate the behavior of PACOB under highly dynamic large-scale scenarios involving fine-grained multi-priority tasks, mixed QoS requirements, and rapidly changing vehicular environments.

Moreover, due to its modular online learning structure, PACOB can be integrated with complementary privacy-preserving, trust-aware, and fault-tolerant mechanisms without modifying its core offloading architecture. For instance, reputation-based server selection, secure task authentication, or reliability-aware reward formulations can be incorporated into the decision-making process to improve service continuity and security in safety-critical VEC environments.

5. Performance Evaluation

To evaluate the performance of the proposed algorithm we design and implement a series of scenarios that simulate various real-world conditions, challenging the algorithm's performance across different metrics.

5.1. Simulation Parameters

The implementation and comparison of the proposed algorithm with baseline algorithms were conducted using Python within PyCharm. To simulate various scenarios, a Poisson distribution was employed to generate both the traffic load on different servers and tasks generated by the vehicle. The specifications of the cloud server, edge servers, and the vehicle are provided in Table 1. Additionally, the task characteristics are summarized in Table 2.

It is important to note that the edge servers are heterogeneous, differing in computational power and bandwidth. However, the cloud server maintains constant computational power and bandwidth, with only the cost varying across different time epochs. The simulation duration is

set to 360 time slots, divided into three equal epochs of 120 slots each. Each slot represents one minute, i.e., 60 seconds. Each experiment is repeated 200 times, and the average of these runs is reported as the final result. The task generation rate by the vehicle, the number of edge servers, the traffic rates for edge and cloud servers, and other parameters are configured based on the designed scenarios, details of which are provided in the subsequent subsection.

It is worth mentioning that the task-generation process in our simulations is modeled as a Poisson arrival process, which is widely adopted in the VEC and MEC literature to capture the stochastic nature of computational workloads [11, 12, 13]. The *task-generation rate of the vehicle* determines how frequently new computational tasks are created and submitted for offloading decisions. In contrast, the *task-generation rate of edge and cloud servers* represent the arrival intensity of background workloads at these servers, reflecting the dynamic and time-varying operational conditions of practical VEC environments. To emulate heterogeneous server utilization levels and varying traffic conditions realistically, the background workload arrival rates are randomly varied between 5 and 25 tasks per time slot, following workload modeling approaches commonly adopted in prior VEC studies [13].

5.2. Scenarios

In this subsection, we describe the experimental scenarios designed to evaluate the performance of the proposed algorithm. Each scenario is tailored to examine specific aspects of the VEC system under various conditions. As detailed in Table 3, we design five scenarios as follows:

Scenario-1 focuses on evaluating the impact of the task-generation rate at the vehicle, while keeping the number of edge servers and the ratio of task types constant. The tasks generated by the vehicle follow a Poisson arrival process with adjustable mean rates, enabling the analysis of PACOB's adaptability to different task arrival intensities.

Scenario-2 investigates the impact of the number of edge servers on system performance, while keeping the task-generation rates and task type proportions fixed. For each simulation run, the number of available edge servers is predetermined and provided to the agent at initialization; hence, the agent does not require retraining when this number varies between scenarios. The background workload on each edge server—representing external vehicular users—is also modeled as a Poisson process with task-generation rates randomly distributed between 5 and 25 tasks per time slot, reflecting heterogeneous and time-varying load conditions.

Scenario-3 examines the effect of varying the proportion of delay-sensitive and delay-tolerant tasks generated by the vehicle, with fixed numbers of edge servers and constant task-generation rates. This scenario highlights PACOB's ability to adaptively balance response time and cost under different workload compositions.

Scenario-4 analyzes the impact of varying edge server costs across epochs while maintaining balanced task type proportions and fixed task-generation rates. The cost of each edge server dynamically changes between epochs according to Table 5, allowing evaluation of PACOB's adaptability under fluctuating pricing conditions.

Scenario-5 evaluates the computational efficiency of the proposed PACOB framework by measuring the average per-task execution time of all compared algorithms. The objective of this experiment is to investigate the computational overhead introduced by the learning and task-type-aware decision mechanisms of PACOB in real-time VEC environments.

These five scenarios collectively form a comprehensive evaluation framework that captures the dynamics of vehicular edge computing environments, including variations in task generation intensity, network size, workload composition, and pricing dynamics. This multi-dimensional setup ensures a robust and realistic assessment of the proposed algorithm's performance under diverse operational conditions.

5.3. Baseline Algorithms

To demonstrate the effectiveness of the proposed PACOB algorithm, we compare its performance with the following baseline algorithms:

- **Random (RND):** For each task t_i generated by the vehicle, this algorithm randomly selects one of the available arms, including the vehicle itself, edge servers, or the cloud server, with a uniform distribution in each round.
- **Priority-based Heuristic (PBH):** This heuristic baseline adopts a deterministic priority-aware offloading policy in which all delay-tolerant tasks are directly offloaded to the cloud server, while delay-sensitive tasks are sequentially distributed among the vehicle itself and the available edge servers in a round-robin manner. This approach reflects a straightforward yet effective rule-based mechanism that captures the basic intuition of priority-driven task assignment and serves as a meaningful benchmark for evaluating the performance of the proposed PACOB algorithm.
- **ϵ-greedy:** This baseline is implemented following the strategy in [13]. For each task in time slot t, the algorithm selects an arm randomly with probability $\epsilon = 1/t$, and with probability $1 - \epsilon$, it selects the arm with the highest Q-value observed so far. The reward function used in this method is identical to that of the proposed PACOB algorithm, jointly considering normalized response time and monetary cost as defined in Eq. 16. However, unlike PACOB, this method maintains a single Q-table shared among all

Table 1: Specifications of computational environment

Resource Type	Computational Power	Bandwidth	Cost	Propagation Time
Edge Servers	[4000 - 10000] MIPS	[70 - 100]% × 10 Mbps	[0.04 - 0.1] G$/s	0 ms
Cloud Server	40000 MIPS	10 Mbps	[0.01 - 0.05] G$/s	200 ms
Vehicle	500 MIPS	-	-	-

Table 2: Specifications of tasks generated by the vehicle

Task Type	Parameter	Value
Delay-sensitive	Task Size	[200 - 4000] MI
	Deadline	[200 - 1000] ms
	Input File Size	[1000 - 5000] KB
Delay-tolerant	Task Size	[2000 - 40000] MI
	Deadline	[1000 - 10000] ms
	Input File Size	[2000 - 10000] KB

tasks, meaning that the agent does not differentiate between delay-sensitive and delay-tolerant task types when making offloading decisions.

- **UCB:** The Upper Confidence Bound (UCB) algorithm is implemented following the standard single Q-table formulation described in [45]. For each task arriving at time slot t, the algorithm selects the arm a with the maximum UCB value $\mathrm{A}(a,t)$, where the exploration coefficient c is set to 2. The reward function employed in this baseline is identical to that used in the proposed PACOB algorithm, jointly considering normalized response time and monetary cost as defined in Eq. 16. Unlike the proposed PACOB framework, which employs separate Q-tables for different task categories (delay-sensitive and delay-tolerant) to capture their distinct optimization objectives, the baseline UCB maintains a single shared Q-table for all tasks. Consequently, it does not distinguish between task types when updating Q-values or selecting actions, treating all tasks uniformly regardless of their latency or cost sensitivity. Therefore, this baseline effectively represents PACOB without the proposed dual Q-table mechanism, enabling a direct ablation analysis of its contribution to the overall system performance.

- **Proposed PACOB:** The proposed PACOB algorithm extends the standard UCB framework by introducing task-type awareness and dual Q-tables for delay-sensitive and delay-tolerant tasks. Similar to the baseline UCB algorithm, the exploration coefficient is set to $c = 2$, which provides a balanced trade-off between exploration and exploitation. Due to the nature of delay-tolerant tasks, the arm corresponding to the vehicle itself is excluded for these tasks, resulting in faster convergence of the learning process. The Q-table is updated using a weighted moving average of the old and new Q-values. Through extensive sensitivity analysis, the best performance was obtained with $w_1 = 0.8$ and $w_2 = 0.2$, where w_1 and w_2 determine the influence of historical and current rewards, respectively. This setting enables smooth convergence and prevents abrupt oscillations during learning. For determining the weighting factors α^{ds} and α^{dt} (and hence β^{ds} and β^{dt}), we performed a sensitivity analysis whose results are shown in Table 4. The experiments were conducted under Scenario-1 (as detailed in Table 3) with a task generation rate of 10, consistent with other scenarios. Based on the results, $\alpha = 0.7$ was selected for both task types, achieving an effective balance between response time and cost optimization.

To ensure a fair and meaningful comparison, the ϵ-greedy and UCB algorithms were implemented under identical parameter configurations and reward formulations as PACOB, differing only in their Q-table structures. Advanced DRL- or game-theoretic-based methods were not included as baselines due to their high computational and memory requirements, slow convergence, and centralized coordination needs, which make them impractical for real-time and distributed vehicular environments. Instead, PACOB emphasizes lightweight, fast-adapting decision-making suitable for on-board deployment in resource-constrained vehicles.

5.4. Metrics

To evaluate the performance of the algorithms, we consider three key metrics: the average response time, the total monetary cost of task execution, and the Percentage of Deadline-Satisfied Tasks (PDST). PDST measures the proportion of tasks that are successfully completed within their specified deadline constraints and is widely used to evaluate QoS satisfaction in VEC environments [46, 47]. These metrics are reported separately for both delay-sensitive and delay-tolerant tasks to clearly demonstrate the behavior and effectiveness of the algorithms for different task categories. This evaluation methodology enables a comprehensive assessment of the proposed framework under diverse QoS requirements and operating conditions.

5.5. Results and Discussion

In this section, we present the results of the designed scenarios and provide an in-depth analysis to highlight the strengths and weaknesses of the proposed approach.

1) Scenario-1 (Impact of Task Generation Rate in the Vehicle): This scenario examines the impact of varying the vehicle's task generation rate, ranging from 5 to 25 tasks per time slot, on system performance. As shown in Fig. 2, increasing the task generation rate leads to higher response time and monetary cost while decreasing PDST across all algorithms due to intensified competition for

Table 3: Details of designed scenarios

Scenario	**# of Edge Servers**	**% of Delay-sensitive Tasks**	**% of Delay-tolerant Tasks**	**Task Generation Rate (Vehicle)**	**Task Generation Rate (Edge Servers)**	**Task Generation Rate (Cloud Server)**	**Objective**
Scenario-1	5	50%	50%	[5-25]	[5-25]	20	Impact of Task Generation Rate in the Vehicle
Scenario-2	[2-8]	50%	50%	10	[5-25]	20	Impact of Number of Edge Servers
Scenario-3	5	[0-100]%	[0-100]%	[5-25]	[5-25]	20	Impact of Task Type Generation Rate in the Vehicle
Scenario-4	5	50%	50%	10	[5-25]	20	Impact of Changing Cost in Edge Servers per Epoch
Scenario-5	5	50%	50%	10	[5-25]	20	Runtime Performance Evaluation Across Algorithms

Table 4: Sensitivity analysis of α across different task categories

	delay-sensitive tasks			**delay-tolerant tasks**		
α	**Response Time** (ms)	**Cost** ($)	**PDST** (%)	**Response Time** (ms)	**Cost** ($)	**PDST** (%)
0.4	745.05	20.47	75.03	1772.51	68.69	89.12
0.5	670.93	22.87	77.45	1510.90	77.52	92.65
0.6	570.30	22.90	79.12	1472.10	85.80	93.46
0.7	**490.52**	**23.40**	**82.70**	**1360.52**	**86.54**	**94.50**
0.8	480.90	24.18	84.00	1220.79	92.32	94.75
0.9	460.45	24.68	84.35	1152.90	100.20	95.04
1.0	450.09	25.05	84.46	1095.12	100.62	95.28

computational resources. Nevertheless, Figs. 2a and 2c demonstrate that the proposed PACOB algorithm consistently achieves superior response time and PDST compared to all baseline algorithms. This improvement stems from PACOB's adaptive learning mechanism, which continuously identifies and selects edge servers with lower latency and lighter workloads for delay-sensitive tasks. Unlike ϵ-greedy and standard UCB approaches that maintain a single shared action-value function for all task categories, PACOB employs a dedicated Q-table for delay-sensitive tasks. This task-specific learning structure enables the algorithm to focus exclusively on latency-oriented reward patterns and rapidly learn more suitable offloading decisions for stringent real-time requirements, resulting in lower response times and higher deadline satisfaction rates. Although the PBH algorithm always offloads delay-sensitive tasks to edge servers, it still exhibits higher response time and lower PDST than PACOB. This is mainly because PBH distributes tasks among edge servers using a deterministic round-robin strategy without considering server occupancy, queue lengths, or instantaneous workload variations. Consequently, some delay-sensitive tasks may be assigned to overloaded edge servers, resulting in increased queuing delay and degraded latency performance. In contrast, PACOB learns to avoid persistently congested servers by favoring offloading actions that historically yield higher rewards. This adaptive learning behavior not only reduces the average response time but also improves deadline satisfaction for delay-sensitive tasks.

Regarding the monetary cost results shown in Fig. 2b, PBH incurs the highest overall cost since all delay-sensitive tasks are executed exclusively on edge servers, whose service cost is generally higher than cloud processing. In comparison, the learning-based approaches (UCB, ϵ-greedy, and PACOB) can flexibly offload a portion of tasks to the cloud whenever such decisions provide a better latency-cost trade-off. Although PACOB incurs a slightly higher cost than UCB and ϵ-greedy due to its stronger preference for low-latency edge execution, this additional cost remains moderate and is justified by the significant gains in response time and deadline satisfaction. Specifically, PACOB achieves approximately 35% lower response time while increasing the monetary cost by only about 25% compared to standard UCB. These results confirm the effectiveness of PACOB in balancing latency and cost for delay-sensitive task management in dynamic VEC environments.

Figs. 2d to 2f present the performance comparison for delay-tolerant tasks under Scenario-1. As observed in the experiments, the proposed PACOB algorithm consistently outperforms both UCB and ϵ-greedy baselines across all metrics, demonstrating markedly superior adaptability to varying task arrival rates. Specifically, PACOB achieves up to 82% improvement in average response time, 75% reduction in monetary cost, and a 30% increase in PDST compared with the second-best performing algorithm. These significant gains are attributed to its dual Q-table learning mechanism, which enables the agent to distinguish between task categories and make type-aware offloading decisions, thereby optimizing both latency and cost simultaneously. Compared with the PBH method, PACOB achieves a closely comparable monetary cost, since both

approaches predominantly offload delay-tolerant tasks to the cloud server to benefit from its lower processing cost. However, under high workload conditions, cloud congestion and queuing delays may increase significantly. In such cases, PACOB adaptively offloads a subset of delay-tolerant tasks to relatively less-loaded edge servers to alleviate cloud-side queuing delay and improve overall responsiveness. Although executing these tasks on edge servers slightly increases the monetary cost due to their higher service prices, this trade-off results in noticeably lower response time and higher PDST. It is important to note that PDST reflects the percentage of tasks completed before their deadlines rather than only the average response time. Therefore, even when PACOB and PBH exhibit relatively similar average latency values, PBH may still experience more deadline violations because its static cloud-oriented policy can create temporary congestion and long queueing delays for a subset of tasks. In contrast, PACOB's adaptive learning mechanism mitigates such congestion effects by dynamically balancing the workload across available servers, thereby improving deadline satisfaction performance.

Overall, the results confirm that PACOB achieves the most balanced trade-off between response time, monetary cost and successful task completion for both delay-sensitive and delay-tolerant workloads. Its task-type-aware learning framework enables adaptive and efficient decision-making across diverse operating conditions, leading to superior scalability and robustness compared with deterministic or single-table learning baselines.

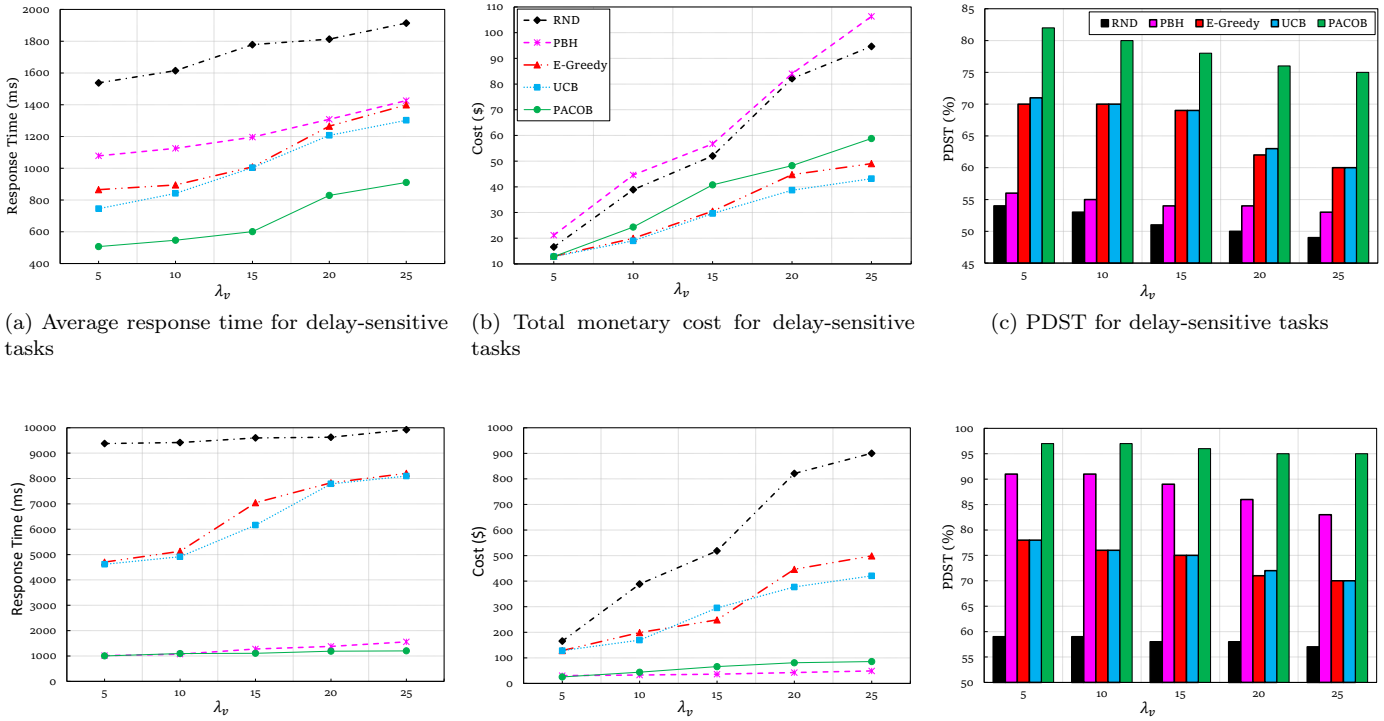
2) Scenario-2 (Impact of Number of Edge Servers): This scenario evaluates how increasing the number of edge servers from 2 to 8 affects the overall system performance. As shown in Figs. 3a–3c, with more available edge servers, all algorithms benefit from reduced response time and improved PDST due to the increased computational capacity. However, the proposed PACOB algorithm consistently outperforms all baselines in terms of latency and PDST, owing to its dual Q-table learning mechanism, which allows more focused and type-specific optimization for delay-sensitive tasks. Unlike UCB and ϵ-greedy, which use a single shared Q-table and occasionally offload some delay-sensitive tasks to the cloud due to limited learning precision, PACOB correctly learns to prioritize nearby edge servers that minimize latency. This targeted offloading slightly increases the monetary cost (Fig. 3b) compared to UCB and ϵ-greedy, as edge processing is inherently more expensive. Nevertheless, this additional cost results in substantial quality-of-service improvement. Compared with the deterministic PBH method, PACOB achieves significantly better performance across all metrics, since PBH lacks adaptive learning and always distributes tasks uniformly across edge servers, whereas PACOB dynamically selects lower-cost and less-loaded servers, ensuring efficient resource utilization and higher service quality.

Figs. 3d–3f illustrate the results for delay-tolerant tasks. The proposed PACOB algorithm again demonstrates superior adaptability compared to learning-based baselines, achieving up to 88% lower response time and 75% cost reduction relative to UCB. These gains are a direct outcome of its dual Q-table structure, which enables separate learning for delay-tolerant tasks and allows the agent to identify optimal offloading choices to minimize cost while maintaining acceptable latency. Compared to PBH, the performance of PACOB is relatively close because both tend to offload most delay-tolerant tasks to the cloud, which offers high computational power at low cost. However, as the number of edge servers increases, PACOB dynamically offloads a small portion of these tasks to low-cost edge servers to prevent potential queuing delays in the cloud. This adaptive behavior leads to slightly higher costs than PBH but results in noticeably better response times and higher PDST, particularly under heavier workloads.

Overall, increasing the number of edge servers enhances performance for all algorithms, but the proposed PACOB achieves the most balanced trade-off between latency, cost, and reliability. Its ability to learn task-type–specific policies enables intelligent adaptation to changing system resources. By prioritizing delay-sensitive tasks for low-latency execution at the edge and flexibly assigning delay-tolerant tasks between the cloud and edge, PACOB maintains efficient operation and consistent service quality. These findings confirm its effectiveness and scalability in dynamic and heterogeneous VEC environments.

3) Scenario-3 (Impact of Task Type Generation Rate in Vehicle): In this scenario, we investigate the impact of varying the proportion of delay-sensitive and delay-tolerant tasks generated by the vehicle. The percentage of delay-sensitive tasks increases from 0% to 100%, with the remaining share assigned to delay-tolerant tasks. As shown in Figs. 4a–4c, increasing the proportion of delay-sensitive tasks leads to higher average response time and monetary cost, while PDST slightly decreases across all algorithms. This behavior occurs because the computational capacity of edge servers remains constant, causing congestion and queuing delays as more delay-sensitive tasks compete for limited low-latency resources. Compared to UCB and ϵ-greedy, the proposed PACOB algorithm achieves substantially lower response time and higher PDST, while its cost increase remains marginal. This demonstrates PACOB's ability to deliver superior QoS for delay-sensitive tasks with only a slight cost overhead. In contrast, the PBH method performs considerably worse in all metrics, as it blindly allocates all delay-sensitive tasks among edge servers without adaptive learning. By intelligently selecting less-loaded and lower-cost servers, PACOB reduces response time and monetary cost by approximately 78% and 65%, respectively, compared to PBH, confirming the clear benefits of its learning-based design.

Figs. 4d–4f present the results for delay-tolerant tasks. The proposed PACOB algorithm again shows a remarkable advantage over the learning-based baselines, achieving consistently lower response time, higher PDST, and lower monetary cost. Compared with PBH, the two meth-



(a) Average response time for delay-sensitive tasks

(b) Total monetary cost for delay-sensitive tasks

(c) PDST for delay-sensitive tasks

(d) Average response time for delay-tolerant tasks

(e) Total monetary cost for delay-tolerant tasks

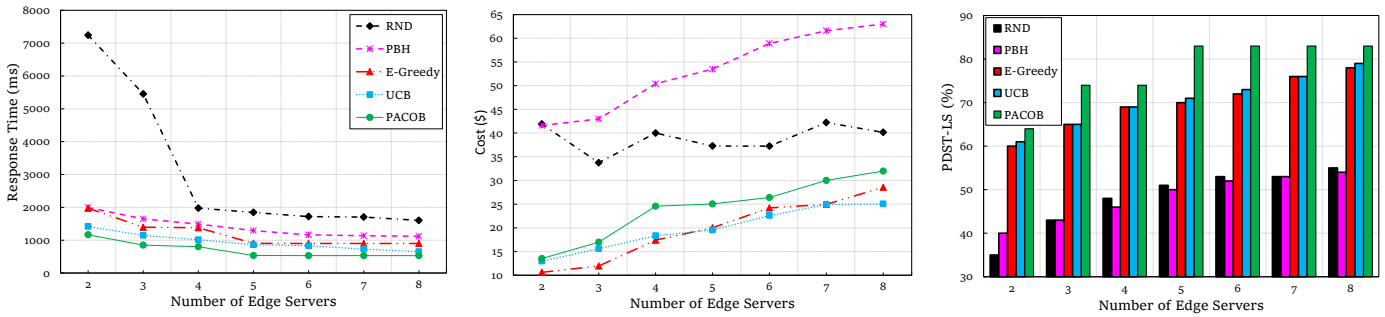
(f) PDST for delay-tolerant tasks

Figure 2: Simulation results for Scenario-1

ods display similar latency behavior since both primarily offload delay-tolerant tasks to the cloud. However, as the proportion of delay-tolerant tasks increases, PBH achieves slightly lower cost because it offloads all such tasks directly to the cloud, which offers lower processing prices. In contrast, PACOB strategically offloads a fraction of delay-tolerant tasks to low-cost edge servers to maintain load balance and meet task deadlines. This adaptive mechanism leads to a minor cost increase but provides superior response time and deadline satisfaction under heavier workloads.

Overall, these results confirm the robustness and adaptability of the proposed PACOB algorithm under varying task-type distributions. Unlike the baselines, which apply uniform or static offloading strategies, PACOB dynamically balances latency and cost objectives across heterogeneous task types. Its dual Q-table framework allows independent learning for delay-sensitive and delay-tolerant tasks, enabling intelligent and context-aware offloading that preserves service quality and cost-efficiency even when task composition changes significantly. This robustness under dynamic task distributions highlights PACOB's suitability for real-world vehicular edge computing environments.

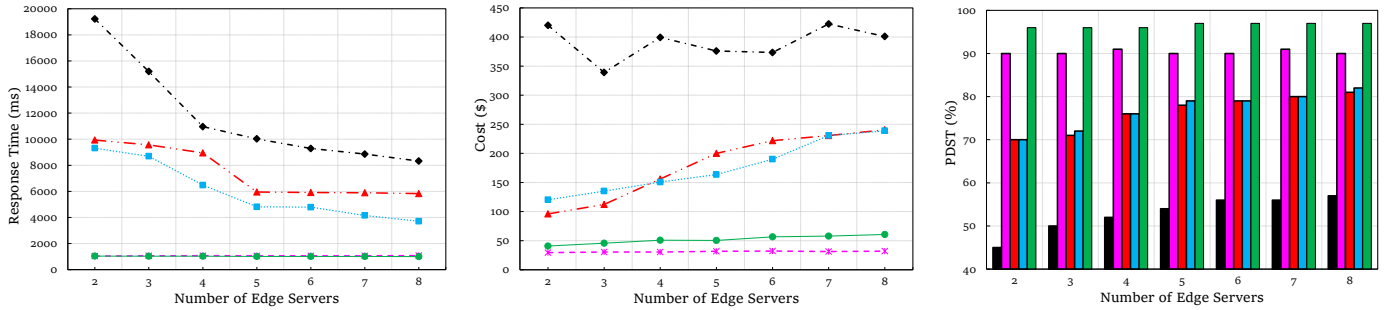
4) Scenario-4 (Impact of Changing Cost in Edge Servers per Epoch): In this scenario, we investigate how dynamic pricing in edge servers across different epochs affects the performance of various task offloading algorithms. Specifically, the processing cost of edge servers fluctuates at the beginning of each epoch, while the cloud server's cost remains constant throughout the simulation. This setup reflects real-world conditions where edge service providers may adjust pricing based on network congestion, resource availability, or demand surges. The specific processing costs of each edge server in different epochs are detailed in Table 5. Fig. 5 shows the behavior of the proposed algorithm compared to the baselines. It is important to note that the results for the Random and PBH algorithms are not included in the presented figures. This exclusion is justified because these algorithms lack learning mechanisms and are inherently unaware of the dynamic changes in edge server costs. Consequently, they cannot adapt their task offloading strategies in response to fluctuating server costs, making their performance less relevant for this analysis. As illustrated in Figs. 5a to 5c, the proposed PACOB algorithm demonstrates a superior capability in identifying and offloading more tasks to the edge server with the lowest processing cost during each epoch. This adaptive behavior stems from the algorithm's design, which emphasizes recent rewards when making offloading decisions. Specifically, according to Equation (17), the algorithm dynamically updates the Q-values by assigning greater importance to the most recent feedback. As a result, when the cost of a particular edge server decreases, its corresponding reward gradually increases, encouraging the algorithm to offload more tasks to that server over time. This mech-



(a) Average response time for delay-sensitive tasks

(b) Total monetary cost for delay-sensitive tasks

(c) PDST for delay-sensitive tasks



(d) Average response time for delay-tolerant tasks

(e) Total monetary cost for delay-tolerant tasks

(f) PDST for delay-tolerant tasks

Figure 3: Simulation results for Scenario-2

anism enables PACOB to effectively adapt to cost variations and exploit cost-efficient edge resources. Another notable observation from the figures is that the proposed PACOB algorithm consistently offloads more tasks to the cloud server compared to the standardϵ-greedy and UCB algorithms. This behavior can be attributed to the fact that for delay-tolerant tasks, the cloud server often offers a more cost-effective and lower-latency offloading option. The algorithm intelligently recognizes this advantage and leverages the cloud infrastructure to optimize overall performance, especially when edge server costs are relatively high or unstable.

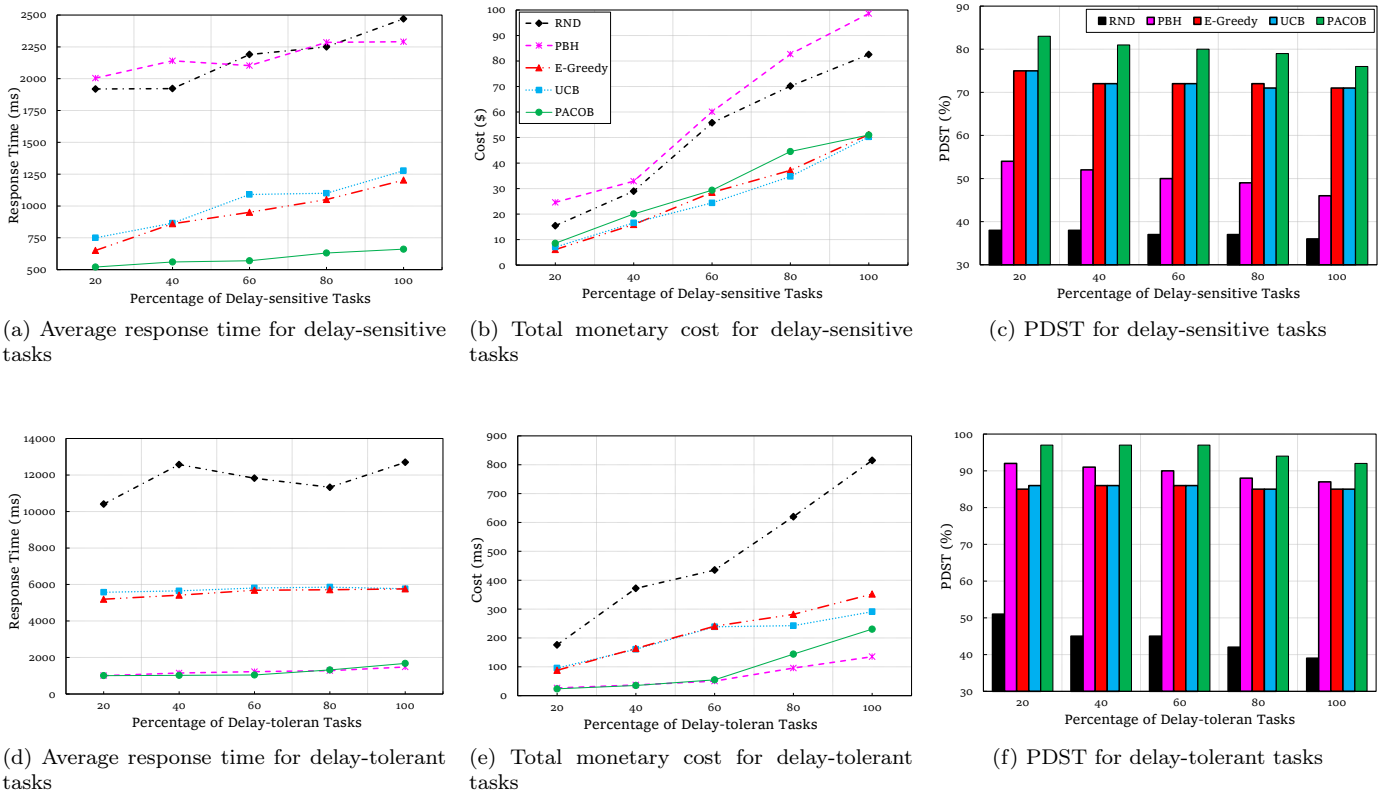
Table 5: The cost of each edge server in each epoch.

	ES1	**ES2**	**ES3**	**ES4**	**ES5**
Epoch 1	**0.04**	0.07	0.08	0.09	0.1
Epoch 2	0.08	0.07	**0.04**	0.09	0.1
Epoch 3	0.1	0.07	0.08	0.09	**0.04**

5) Scenario-5 (Runtime Performance Evaluation Across Algorithms): Fig.6 presents the average per-task runtime of the evaluated algorithms. As expected, the simple heuristic approaches, namely RND and PBH, exhibit the lowest execution overhead due to their minimal decision-making logic and absence of learning operations. In contrast, the learning-based methods incur moderately higher runtime because of the additional reward evaluation and action-selection computations. Among them, PACOB introduces a slightly higher overhead than standard UCB and ϵ-greedy owing to its dual Q-table structure and task-type-aware decision process. Nevertheless, the average runtime of PACOB remains very low (approximately 12.5 ms per task), demonstrating that the proposed framework preserves lightweight online decision-making characteristics suitable for real-time VEC environments. These results confirm that the additional intelligence and adaptability introduced by PACOB are achieved with only a modest computational overhead, making the framework practical even for latency-sensitive vehicular applications requiring rapid offloading decisions.

6. Conclusion and Future Work

In this paper, we introduced the PACOB algorithm, an adaptive task offloading strategy specifically designed for VEC environments. The key innovation of the proposed method lies in its distinct handling of delay-sensitive and delay-tolerant tasks through the use of two separate Q-tables. This structure enables the algorithm to effectively optimize task-specific objectives—minimizing response time for delay-sensitive tasks and reducing monetary costs for computation-intensive delay-tolerant tasks. By incorporating task-specific learning, the algorithm achieves more



(a) Average response time for delay-sensitive tasks

(b) Total monetary cost for delay-sensitive tasks

(c) PDST for delay-sensitive tasks

(d) Average response time for delay-tolerant tasks

(e) Total monetary cost for delay-tolerant tasks

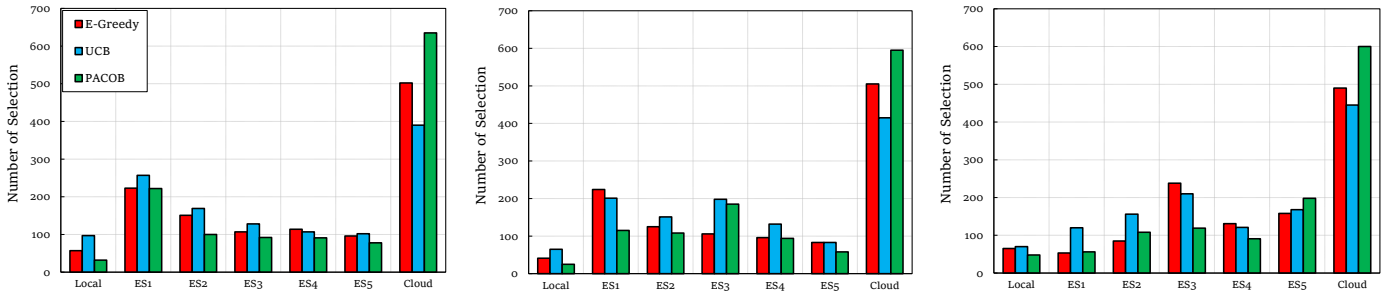
(f) PDST for delay-tolerant tasks

Figure 4: Simulation results for Scenario-3

intelligent and efficient offloading decisions, balancing performance and cost trade-offs. The effectiveness of the proposed PACOB algorithm was rigorously evaluated against four baseline algorithms—Random, PBH, ϵ-greedy, and the standard UCB—across various simulation scenarios. The results demonstrated that the PACOB algorithm significantly improves the QoS for delay-sensitive tasks by reducing response time and increasing the PDST. Although the monetary cost for delay-sensitive tasks was slightly higher compared to the ϵ-greedy and standard UCB algorithms, this increase is justified by the substantial improvement in service quality. For delay-tolerant tasks, the proposed algorithm consistently outperformed all competing methods in terms of response time, cost efficiency, and PDST, showcasing its superior adaptability and effective resource management. Looking ahead, several promising directions can be pursued to further enhance the proposed algorithm. One potential extension of this work is the integration of communication and vehicle-side energy consumption into the decision-making framework. Incorporating these factors would enable a more comprehensive energy-aware optimization that jointly considers latency, monetary cost, and energy efficiency, which is particularly important for battery-constrained vehicular environments and real-world VEC deployments. Additionally, extending PACOB to incorporate security- and privacy-aware decision mechanisms, while enabling collaborative offloading through voluntary computational contributions from nearby vehicles, can substantially enhance system scalability, trustworthiness, and fault tolerance in future VEC deployments. Another promising direction is the adaptation of the PACOB algorithm for deployment in serverless and event-driven computing environments, allowing for dynamic resource provisioning in decentralized systems. Furthermore, the development of self-adaptive mechanisms for tuning key algorithm parameters in real-time could enhance decision-making under dynamic network conditions. Finally, implementing and validating the proposed algorithm in real-world vehicular environments using trace-driven mobility models and realistic communication patterns represents a promising direction for future work. Such an evaluation would offer deeper insights into PACOB's practical effectiveness under dynamic network conditions and help address open challenges related to system scalability, security, and operational robustness. By pursuing these research directions, the proposed PACOB algorithm can be further refined to meet the evolving demands of intelligent transportation systems and next-generation vehicular networks.

7. Acknowledgement

This work was partially supported by Horizon Europe Marie Skłodowska-Curie Actions under the project TRACE-V2X (grant agreement ID 101131204).



(a) Number of node selection for Epoch-1 (b) Number of node selection for Epoch-2 (c) Number of node selection for Epoch-3

Figure 5: Simulation results for Scenario-4

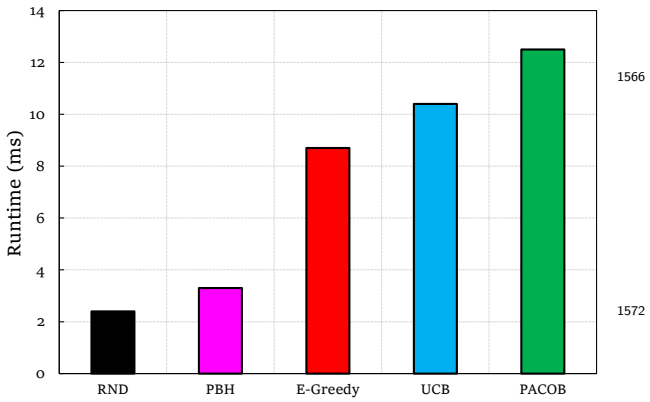


Figure 6: Average per-task runtime comparison of the evaluated offloading algorithms.

References

[1] S. Bitam, A. Mellouk, S. Zeadally, Vanet-cloud: a generic cloud computing model for vehicular ad hoc networks, IEEE Wireless Communications 22 (1) (2015) 96–102.

[2] R. Meneguette, R. De Grande, J. Ueyama, G. P. R. Filho, E. Madeira, Vehicular edge computing: Architecture, resource management, security, and challenges, ACM Computing Surveys (CSUR) 55 (1) (2021) 1–46.

[3] L. Liu, C. Chen, Q. Pei, S. Maharjan, Y. Zhang, Vehicular edge computing and networking: A survey, Mobile networks and applications 26 (2021) 1145–1168.

[4] M. Maurer, J. C. Gerdes, B. Lenz, H. Winner, Autonomous driving: technical, legal and social aspects, Springer Nature, 2016.

[5] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, W. Shi, Computing systems for autonomous driving: State of the art and challenges, IEEE Internet of Things Journal 8 (8) (2021) 6469–6486.

[6] S. A. A. Shah, E. Ahmed, M. Imran, S. Zeadally, 5g for vehicular communications, IEEE Communications Magazine 56 (1) (2018) 111–117.

[7] Z. Ning, J. Huang, X. Wang, Vehicular fog computing: Enabling real-time traffic management for smart cities, IEEE Wireless Communications 26 (1) (2019) 87–93.

[8] L. Yang, A. Zhou, X. Ma, Y. Zhang, S. Wang, Reliability-aware task replication for mobile edge computing, IEEE Internet of Things Journal 11 (14) (2024) 24846–24857.

[9] J. Zhao, Q. Li, Y. Gong, K. Zhang, Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks, IEEE Transactions on Vehicular Technology 68 (8) (2019) 7944–7956.

[10] W. Fan, Y. Su, J. Liu, S. Li, W. Huang, F. Wu, Y. Liu, Joint task offloading and resource allocation for vehicular edge computing based on v2i and v2v modes, IEEE Transactions on Intelligent Transportation Systems 24 (4) (2023) 4277–4292.

[11] Y. Sun, X. Guo, J. Song, S. Zhou, Z. Jiang, X. Liu, Z. Niu, Adaptive learning-based task offloading for vehicular edge computing systems, IEEE Transactions on vehicular technology 68 (4) (2019) 3061–3074.

[12] S. Misra, S. P. Rachuri, P. K. Deb, A. Mukherjee, Multiarmed-bandit-based decentralized computation offloading in fog-enabled iot, IEEE Internet of Things Journal 8 (12) (2021) 10010–10017.

[13] A. Bozorgchenani, S. Maghsudi, D. Tarchi, E. Hossain, Computation offloading in heterogeneous vehicular edge networks: On-line and off-policy bandit solutions, IEEE Transactions on Mobile Computing 21 (12) (2022) 4233–4248.

[14] L. Yin, J. Luo, C. Qiu, C. Wang, Y. Qiao, Joint task offloading and resources allocation for hybrid vehicle edge computing systems, IEEE Transactions on Intelligent Transportation Systems 25 (8) (2024) 10355–10368.

[15] B. Hao, Y. Abbasi Yadkori, Z. Wen, G. Cheng, Bootstrapping upper confidence bound, Advances in neural information processing systems 32 (2019).

[16] P. Qin, Y. Fu, G. Tang, X. Zhao, S. Geng, Learning based energy efficient task offloading for vehicular collaborative edge computing, IEEE Transactions on Vehicular Technology 71 (8) (2022) 8398–8413.

[17] B. Kumar, A. Bhardwaj, D. Prasad Sahu, Task offloading for cavs edge computing environment: Taxonomy, critical review, and future road map, ACM Computing Surveys 58 (8) (2026) 1–35.

[18] A. A. Vali, S. Azizi, M. Shojafar, R. Buyya, Energy-efficient resource management in microservices-based fog and edge computing: State-of-the-art and future directions, ACM Computing Surveys 58 (12) (2026) 1–42.

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

[20] S. Azizi, M. Othman, H. Khamfroush, Deco: A deadline-aware and energy-efficient algorithm for task offloading in mobile edge computing, IEEE Systems Journal 17 (1) (2023) 952–963.

[21] M. S. Bute, P. Fan, L. Zhang, F. Abbas, An efficient distributed task offloading scheme for vehicular edge computing networks, IEEE Transactions on Vehicular Technology 70 (12) (2021) 13149–13161.

[22] I. Ataie, T. Taami, S. Azizi, M. Mainuddin, D. Schwartz, D 2 fo: distributed dynamic offloading mechanism for time-sensitive tasks in fog-cloud iot-based systems, in: 2022 IEEE International Performance, Computing, and Communications Conference (IPCCC), IEEE, 2022, pp. 360–366.

[23] D. Garg, N. C. Narendra, S. Tesfatsion, Heuristic and reinforcement learning algorithms for dynamic service placement on mobile edge cloud, arXiv preprint arXiv:2111.00240 (2021).

[24] Y. Zhang, W. Liang, Y. Yang, Qoe-aware task executions on service models in dt-assisted edge computing, IEEE Transactions on Mobile Computing (2025).

[25] M. Su, G. Wang, J. Chen, Efficient task offloading with swarm intelligence evolution for edge-cloud collaboration in vehicular edge computing, Software: Practice and Experience spe.3125 (2022) 1–28.

[26] H. Materwala, L. Ismail, R. M. Shubair, R. Buyya, Energy-sla-aware genetic algorithm for edge–cloud integrated computation offloading in vehicular networks, Future Generation Computer Systems 135 (2022) 205–222.

[27] A. B. de Souza, P. A. L. Rego, V. Chamola, T. Carneiro, P. H. G. Rocha, J. N. de Souza, A bee colony-based algorithm for task offloading in vehicular edge computing, IEEE Systems Journal 17 (3) (2023) 4165–4176.

[28] S. Long, Y. Zhang, Q. Deng, T. Pei, J. Ouyang, Z. Xia, An efficient task offloading approach based on multi-objective evolutionary algorithm in cloud-edge collaborative environment, IEEE Transactions on Network Science and Engineering 10 (2) (2022) 645–657.

[29] J. Yi, J. Bai, H. He, J. Peng, D. Tang, ar-moea: A novel preference-based dominance relation for evolutionary multiobjective optimization, IEEE Transactions on Evolutionary Computation 23 (5) (2019) 788–802.

[30] Q. Luo, C. Li, T. H. Luan, W. Shi, Minimizing the delay and cost of computation offloading for vehicular edge computing, IEEE Transactions on Services Computing 15 (5) (2022) 2897–2909.

[31] Y. Hui, Z. Su, T. H. Luan, C. Li, G. Mao, W. Wu, A game theoretic scheme for collaborative vehicular task offloading in 5g hetnets, IEEE Transactions on Vehicular Technology 69 (12) (2020) 16044–16056.

[32] S. Wang, D. He, M. Yang, L. Duo, Cost-aware task offloading in vehicular edge computing: A stackelberg game approach, Vehicular Communications 49 (2024) 100807.

[33] Y. Chen, J. Zhao, J. Hu, S. Wan, J. Huang, Distributed task offloading and resource purchasing in noma-enabled mobile edge computing: Hierarchical game theoretical approaches, ACM Transactions on Embedded Computing Systems 23 (1) (2024) 1–28.

[34] C. Cheng, L. Zhai, X. Zhu, Y. Jia, Y. Li, Dynamic task offloading and service caching based on game theory in vehicular edge computing networks, Computer Communications 224 (2024) 29–41.

[35] J. Zhang, H. Guo, J. Liu, Adaptive task offloading in vehicular edge computing networks: a reinforcement learning based scheme, Mobile Networks and Applications 25 (5) (2020) 1736–1745.

[36] W. Fan, Y. Zhang, G. Zhou, Y. Liu, Deep reinforcement learning-based task offloading for vehicular edge computing with flexible rsu-rsu cooperation, IEEE Transactions on Intelligent Transportation Systems 25 (7) (2024) 7712–7725.

[37] S. Misra, S. P. Rachuri, P. K. Deb, A. Mukherjee, Multiarmed-bandit-based decentralized computation offloading in fog-enabled iot, IEEE Internet of Things Journal 8 (12) (2021) 10010–10017.

[38] J. Xue, L. Wang, Q. Yu, P. Mao, Multi-agent deep reinforcement learning-based partial offloading and resource allocation in vehicular edge computing networks, Computer Communications (2025) 108081.

[39] Y. Cheng, J. Ma, Z. Liu, Y. Wu, K. Wei, C. Dong, A lightweight privacy preservation scheme with efficient reputation management for mobile crowdsensing in vehicular networks, IEEE Transactions on Dependable and Secure Computing 20 (3) (2022) 1771–1788.

[40] A. Javed, A. Malhi, K. Främling, Edge computing-based fault-tolerant framework: A case study on vehicular networks, in: 2020 International Wireless Communications and Mobile Computing (IWCMC), IEEE, 2020, pp. 1541–1548.

[41] Z. Liu, L. Wan, J. Guo, F. Huang, X. Feng, L. Wang, J. Ma, Ppru: A privacy-preserving reputation updating scheme for cloud-assisted vehicular networks, IEEE TRANSACTIONS on vehicular technology 74 (2) (2025) 1877–1892.

[42] Y. Sun, Z. Liu, F. Ye, J. Weng, Y. Miao, J. Guo, J. Ma, Pdru: a privacy-preserving dual reputation updating scheme with multi-dimensional feedback scores in vehicle platoon, IEEE Transactions on Dependable and Secure Computing 23 (2) (2026) 4236–4251.

[43] S. Misra, N. Saha, Detour: Dynamic task offloading in software-defined fog for iot applications, IEEE Journal on Selected Areas in Communications 37 (5) (2019) 1159–1166.

[44] A. Khazali, A. Bozorgchenani, D. Tarchi, M. G. Shayesteh, H. Kalbkhani, Joint task assignment, power allocation and node grouping for cooperative computing in noma-mmwave mobile edge computing, IEEE Access 11 (2023) 93664–93678.

[45] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction (2018).

[46] F. Hoseiny, S. Azizi, M. Shojafar, R. Tafazolli, Joint qos-aware and cost-efficient task scheduling for fog-cloud resources in a volunteer computing system, ACM Transactions on Internet Technology (TOIT) 21 (4) (2021) 1–21.

[47] R. Salimi, S. Azizi, M. Shojafar, Dynamic and resource-aware task scheduling in fog-cloud environments via an improved priority-aware genetic algorithm, Journal of Grid Computing 24 (2) (2026) 5.