Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.074154

ARTICLE

A Multi-Agent Deep Reinforcement Learning-Based Task Offloading Method for 6G-Enabled Internet of Vehicles with Cloud-Edge-Device Collaboration

Fangxiang Hu[1], **Qi Fu**[1,2,*], **Shiwen Zhang**[1] **and Jing Huang**[1]

[1]School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, China
[2]School of Electronic Information Engineering, Changsha Institute of Technology, Changsha, China

*Corresponding Author: Qi Fu. Email: fuqi@hnust.edu.cn

Received: 03 October 2025; Accepted: 06 February 2026; Published: 09 April 2026

ABSTRACT: In the Internet of Vehicles (IoV) environment, the growing demand for computational resources from diverse vehicular applications often exceeds the capabilities of intelligent connected vehicles. Traditional approaches, which rely on one or more computational resources within the cloud-edge-device computing model, struggle to ensure overall service quality when handling high-density traffic flows and large-scale tasks. To address this issue, we propose a computational offloading scheme based on a cloud-edge-device collaborative 6G IoV edge computing model, namely, Multi-Agent Deep Reinforcement Learning-based and Server-weighted scoring Selection (MADRLSS), which aims to optimize dynamic offloading decisions and resource allocation. The scheme first designs an improved multi-agent proximal policy optimization (MAPPO) algorithm, decoupling centralized training from distributed execution for multiple terminal vehicle agents. Specifically, the centralized training of terminal vehicles is migrated to the high-performance edge layer, while lightweight decision-making networks are retained at the terminal vehicles to enable efficient and dynamic task offloading decisions. Additionally, a server-weighted scoring selection (SS) algorithm is proposed, which integrates two key metrics—short-term server load and geographical proximity—to select the optimal server and allocate communication resources. The proposed scheme improves the quality of experience (QoE) while balancing energy consumption. Simulation results demonstrate that the MADRLSS scheme significantly outperforms existing benchmark methods in terms of task offloading efficiency and stability, maintaining QoE consistently above 82% and effectively enhancing service quality in complex vehicular scenarios.

KEYWORDS: Vehicular edge computing; cloud-edge-device collaboration; multi-agent proximal policy optimization; high-density vehicle; task offloading; resource allocation; 6G

1 Introduction

With the rapid advancement of wireless network communication technology, the Internet of Vehicles (IoV) has laid the foundation for Intelligent Transportation Systems (ITS). The core of IoV lies in establishing an integrated network connecting diverse entities—vehicles, roadside infrastructure, pedestrians, and cloud services—to effectively enhance road safety and improve traffic efficiency [1]. As a key component of ITS, Intelligent Connected Vehicles (ICVs) have gained enhanced computing, storage, sensing, and communication capabilities, enabling numerous new applications such as vehicle control, real-time route planning, traffic condition alerts, and voice recognition. However, the tasks generated by these applications often exhibit characteristics of high real-time requirements, low latency, and high computational intensity, placing significant strain on onboard computers. For instance, real-time safety decisions in ICVs rely on processing sensor data streams of higher than 8 Gbps [1], which exceeds the limited computational capacity of On-Board

Units (OBUs). This directly impacts the operational efficiency of vehicle applications and user experience. To alleviate the issue of insufficient computational power in vehicles, computational offloading offers a viable solution. This approach involves transferring computational tasks to other computing nodes with sufficient resources—such as edge servers or cloud servers—for execution [2].

Early computational offloading approaches involved delegating computational tasks to centralized processing on remote cloud servers. While the cloud possesses the most powerful computational resources and storage capabilities, this approach incurs significant communication latency [3]. To reduce transmission latency, Vehicular Edge Computing (VEC) has emerged. VEC deploys servers near terminal vehicles, shifting computational tasks from the cloud to locations closer to the vehicles themselves. This approach meets the demand for low communication latency, though computational resources remain limited. Furthermore, idle ICVs serve as dynamic "edge" computing nodes. However, the high mobility of vehicles often results in unstable communication connections, potentially causing offloading failures. Existing solutions predominantly employ single-layer or dual-layer computing resources [4,5]. However, in highly dense real-world traffic environments, the computational offloading demands of different vehicle applications vary significantly. Therefore, designing a computational offloading scheme that coordinates multi-tier computing resources to meet the diverse offloading requirements of various vehicle applications represents a vital challenge.

Meanwhile, as the next generation of wireless networks, 6G is expected to deliver extremely high data rates (with peak rates up to 1 Tbps), support massive-scale and ultra-high-speed wireless access (i.e., sub-millisecond latency while connecting tens of billions of communication devices), and provide smarter, more sustainable, and seamless 3-D coverage [6]. Prathiba et al. demonstrated the robustness and reliability of 6G technology by deploying a hybrid algorithm that integrates multi-agent reinforcement learning with Maximum Entropy Inverse Reinforcement Learning (MaxEntIRL) within the sensor network of a 6G Autonomous Vehicle Network (AVN). This approach significantly enhances the efficiency and accuracy of detecting malicious and anomalous behaviors by reducing transmission latency [7]. The enhanced data transmission capabilities of 6G technology will create favorable conditions for cross-tier computing resource coordination, thereby improving the efficiency of offloading computational tasks in complex scenarios.

In recent years, research on computational offloading in VEC environments has proliferated. Traditional approaches typically employ heuristics [8], game theory [9], and Machine Learning (ML) [10] to explore computational offloading performance, yielding favorable results. However, both heuristic and game-theoretic algorithms require meticulously designed computational offloading and resource allocation schemes tailored to specific offloading scenarios, often necessitating extensive iterations to converge on optimal solutions. Traditional ML methods are constrained by the need for manually designed data features and feature extraction. These approaches lack scalability and flexibility when applied to large-scale, dynamic IoV environments. Offloading algorithms based on deep reinforcement learning are currently employed to overcome the limitations of traditional approaches. Deep Reinforcement Learning (DRL) is an artificial intelligence approach that integrates the perceptual capabilities of deep learning with the decision-making abilities of reinforcement learning. Through repeated interactions between an agent and its environment, where actions yield rewards or penalties, DRL learns strategies that maximize benefits [11]. Consequently, DRL methods can adapt to complex offloading scenarios.

However, in the distributed real-world environment of IoV, a single agent must acquire global environmental information to make decisions for all vehicle application tasks. This incurs significant communication overhead, impacting decision-making efficiency. As an augmentation to single-agent DRL, Multi-Agent Deep Reinforcement Learning (MADRL) provides an effective solution for distributed decision-making in dynamic and decentralized environments [12,13]. In the learning paradigms of MADRL algorithms, the centralized training and distributed execution (CTDE) paradigm is generally adopted as the mainstream

approach [12]. The centralized training phase in existing approaches typically deploys agents on distributed devices. In real-world IoV distributed environments, it is impossible to gather information from all agents for centralized training. This approach also neglects the training overhead associated with deploying agents to terminal vehicles.

To address these challenges, we adopt a cloud-edge-device collaborative [3] 6G-enabled Internet of Vehicles edge computing system model, which has three layers: the vehicle layer, the edge layer, and the cloud layer. As illustrated in Fig. 1, ICVs reside in the first layer. These vehicles not only generate data but also possess computational resources provided by their OBUs. Vehicles with idle resources can handle tasks with low to moderate complexity, such as those generated by navigation applications. The second layer comprises Roadside Units (RSUs) equipped with servers and edge system. This edge layer is capable of processing tasks with low-latency requirements, such as those arising from risk assessment applications. The third layer consists of a remote cloud configured with server clusters, offering the most powerful computational resources and storage capacity. It supports computationally intensive tasks that are not highly latency-sensitive, such as those generated by infotainment applications. This collaborative architecture, which integrates computational resources across all three layers, is well-suited for high-density urban road traffic scenarios. Terminal vehicles generate various tasks during operation. Due to limited onboard computational resources or insufficient battery power, all computational tasks are fully offloaded via Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communications to idle vehicles, edge servers, or the remote cloud, thereby achieving balanced utilization of system resources. On this basis, this paper proposes a computational offloading solution named Multi-Agent Reinforcement Learning with Server Selection (MADRLSS). This framework integrates MADRL with CTDE, alongside Server weighted scoring-based Selection (SS). By jointly optimizing Quality of Experience (QoE) and the energy consumption of tasks, MADRLSS addresses the diverse requirements of vehicle application tasks while enhancing the success rate of task offloading. The significant contributions of this paper are as follows:

1. Optimizing the 6G IoV architecture enhances modular management by dividing the edge system into the edge management module and the agent management module. This architecture achieves distributed coordination of cloud-edge-device computing and storage resources through exploring task offloading methods suitable for 6G networks, significantly improving the system's overall computational offloading performance.
2. To address real-time offload decisions for each task, we design an improved MADRL algorithm by decoupling CTDE. The paper models a fully cooperative multi-agent reinforcement learning problem, designing a joint optimization objective to maximize the balance between QoE and energy consumption for tasks. Each terminal vehicle is treated as an independently executing agent that collaborates with other terminal vehicle agents. The edge system centrally trains each agent, guided by the global joint optimization objective, to refine each agent's offload strategy.
3. A flexible SS algorithm is proposed. By incorporating weighted scores of two critical factors in the offloading process—the short-term load and the geographical distance of servers—the algorithm determines the optimal server and communication resource allocation for task offloading.

The rest of this paper is ordered as follows. Section 2 introduces related work on task offloading in VEC. Section 3 describes the system model and issue formulation for computational task offloading. Section 4 designs the MADRLSS approach. In Section 5, experiments are reasonably designed, and the results are analyzed. Finally, Section 6 provides a conclusion.

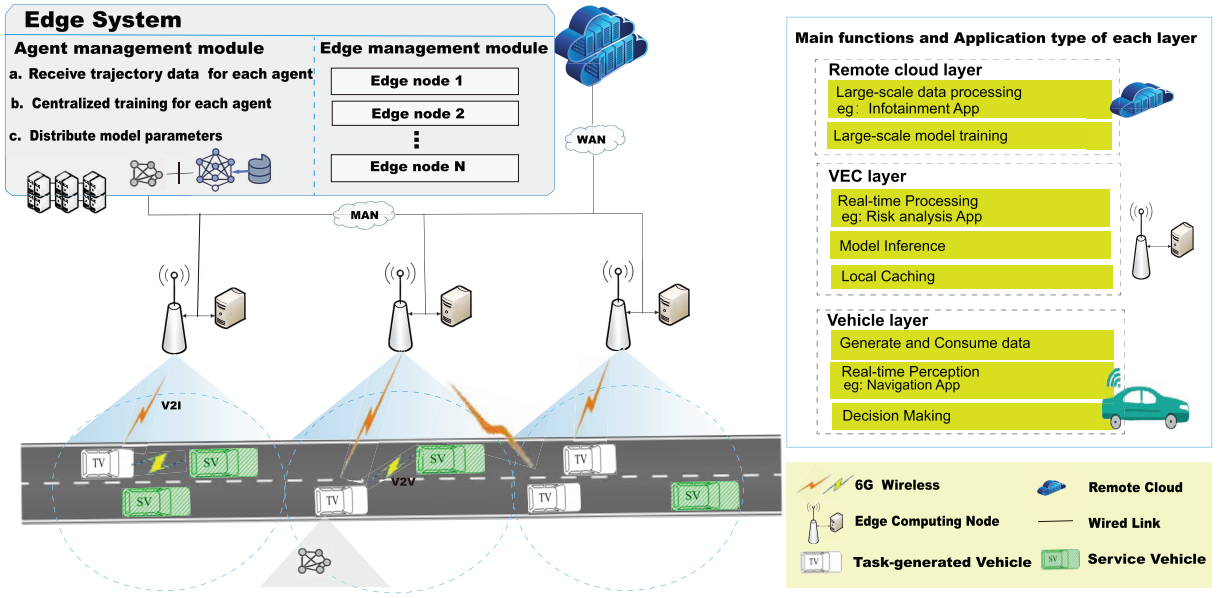


Figure 1: Task offloading system in the cloud-edge-device collaborative architecture for 6G IoV

2 Related Work

In recent years, the task computation offloading problem in VEC environments has gradually emerged as a research hotspot attracting joint attention from both academia and industry.

2.1 Computational Offloading Architecture for IoV

The computing offloading architecture for IoV is becoming increasingly adaptable to diverse requirements as it continues to evolve.

Bitam et al. [14] designed a cloud computing model named VANET-Cloud for vehicular ad hoc networks (VANETs), which centrally processes task requirements (e.g., computational demands for safety and non-safety applications) through cloud servers. To address the high latency and inefficiency issues associated with cloud computing's centralized task processing, Zhang et al. [15] proposed a collaborative fog architecture for IoV composed of a fog layer and an edge layer. This architecture enables computational resources to be located closer to the vehicles, thereby reducing latency. To minimize latency and enhance reliability in computational task completion for mobile devices, Lin et al. [13] proposed a two-tier edge architecture that integrates mobile vehicles—treated as mobile edge servers—and RSUs equipped with servers as fixed edge servers. By offloading computational tasks to either mobile or fixed edge servers, their approach aims to reduce latency and improve the quality of service for these tasks. Literature [16] proposes a three-tier collaborative computing architecture with cloud-edge-device coordination. Tasks are executed locally or offloaded to idle vehicles, edge servers, and cloud servers. By reducing energy consumption and task latency while improving task completion rates, this approach achieves 85% resource utilization.

2.2 Optimization Objective of Computational Offloading

Currently, research on computational offloading primarily focuses on optimization objectives such as minimizing task offloading latency, minimizing energy consumption, optimizing task dependencies, and improving application result quality [3,17].

Zhou et al. [18] proposed a computational offloading scheme with demand forecasting and Reinforcement Learning (RL), named CODR, which effectively decreases latency. In [19], an efficient task offloading based on traffic prediction in IoV-Enabled edge computing aims to resolve the computational load imbalance in edge servers resulting from irregular spatiotemporal traffic flow, thereby improving service response speed. In [20], battery-powered autonomous vehicles and edge devices consume significant amounts of energy during transmission and computation tasks. To reduce energy consumption costs, He and Li [21] proposed an offloading scheduling strategy with minimized power overhead for IoV based on Mobile Edge Computing (MEC). They designed a Simulated Annealing (SA) algorithm to solve the offloading optimization model. In [22], a method employing DRL and an edge computing architecture for vehicle task scheduling decisions is proposed to reduce computational completion delays and energy consumption in multi-task scheduling. Chen et al. [23] proposed a real-time, dependency-aware task offloading method (DODQ) based on Deep Q-Networks (DQN). By modeling task dependencies within vehicle applications as Directed Acyclic Graphs (DAGs), the DODQ enables real-time dynamic task scheduling to rapidly generate offloading plans within milliseconds. The trade-off among various optimization objectives in computation offloading is also a critical issue. Li et al. [24] identified a new trade-off between the Quality of Result (QoR) and service response time in MEC. By relaxing the requirement for the highest QoR, they managed to reduce both the service response time and application energy consumption.

2.3 Optimization Strategies of Computational Offloading

In the IoV environment, optimization strategies for computational offloading have undergone significant evolution.

The initial offloading strategies employed heuristic [8,25], game-theoretic [9], and contract-theoretic algorithms [26], tailored for vehicular networks. To address latency and quality-balanced task allocation in Vehicular Fog Computing (VFC), Zhu et al. utilized Mixed Integer Linear Programming (MILP) [27] to solve NP-hard problems. The approach achieved improved task allocation performance by balancing service latency and quality degradation. However, MILP relies on expert-designed heuristics, demanding specialized knowledge and limiting extensibility to non-binary integer variables. Meanwhile, traditional methods are prone to local optima, exhibit slow convergence, and often fail to attain global optimal solutions.

Subsequently, to adapt to the real-time, dynamic, and complex IoV environment, ML [10] and DRL [28] were introduced to address offloading decisions and resource allocation for vehicle computing tasks in cloud or edge environments. In [29], the researchers employed Dropout regularization and Double Deep Q-Network (DDQN) to mitigate the overestimation bias of DQN. They modeled communication and computational states using finite Markov chains to achieve resource allocation. However, this value-based DRL is well-suited for handling discrete action spaces but struggles with complex task offloading problems involving infinite states.

To extend Markov decision processes with infinite states and continuous action spaces, researchers have actively explored hybrid algorithms based on policy or policy-value functions. Zhou et al. [18] proposed a computational offloading method based on twin delayed deterministic policy gradient (TD3). The system's intelligent center offloads vehicle application tasks to four distinct locations (local computation, edge servers on RSUs, edge servers on Base Station (BS), and idle vehicles within the same region) at varying proportions, thereby enhancing the overall performance of 6G IoV systems. Considering the partially observable characteristics of the distributed nature and environmental state in IoV networks, literature [30] designed an efficient algorithm based on Multi-Agent Deep Deterministic Policy Gradient (MADDPG). Each agent observes the local state of the environment to make computational offloading decisions, thereby

alleviating the communication costs associated with a single centralized agent acquiring global information. This approach reduces the latency and energy consumption of computational offloading.

We briefly compared recent relevant studies in Table 1. It is found that although existing research has provided reasonable computation offloading strategies for in-vehicle application computing tasks under the VEC paradigm, the MADRL algorithm—which employs terminal vehicles as agents—still faces significantly increased computational burdens in high-density traffic and task scenarios. This primarily stems from existing methods that typically execute policies and conduct centralized training simultaneously on local vehicles, leading to resource constraints at the terminal level. To continuously optimize computation offloading strategies, this paper proposes a MADRL training approach distinct from existing solutions. We first enhance the modular management capabilities of the edge layer within the cloud-edge-device integrated computing model. Building upon this foundation, we investigate task offloading and resource allocation mechanisms, subsequently proposing a computational offloading method named MADRLSS. This approach decouples centralized training from distributed execution by migrating the data-driven training process of vehicle agents to edge servers, while vehicles solely execute policy actions. This design substantially reduces computational pressure on terminal vehicles, comprehensively optimizing the QoE of tasks and system energy consumption while significantly enhancing overall computational offloading efficiency in VEC environments.

Table 1: Comparison of recent task offloading strategies in VEC environment (2023–2025).

Study (Year)	Offloading mode	Proposed algorithm	Learning paradigm	Offloading decision location	Training location	Offloading architecture	Objectives	Handover*
[28] (2023)	Binary	PPO[1]-based	Centralized	Base station	Base station	Edge-device	Completion rate of key tasks	×
[13] (2023)	Binary	MADDPG[2]-based	Centralized training and distributed execution	Vehicles/ edge servers	Vehicles/ edge servers	Edge-device	QoE	×
[23] (2024)	Binary	Deep Q-Networks -based	Centralized	System center	System center	Cloud-edge	Delay	×
[18] (2024)	Partial	TD3[3]-based	Centralized	Intelligence center	Intelligence center	Edge-device	Delay	×
[30] (2024)	Partial	MADDPG[2]-based	Centralized training and distributed execution	Vehicles /edge servers/ cloud servers	Vehicles /edge servers/ cloud servers	Cloud-edge-device	Delay, energy	×
[19] (2025)	Binary	DRL-based	Centralized	System center	System center	Edge-device	Response speed of vehicle services	×
[22] (2025)	Binary	MAPPO[4]-based	Decentralized	Vehicles	Vehicles	Edge-device	Delay, energy	×

(Continued)

Table 1 (continued)

Study (Year)	Offloading mode	Proposed algorithm	Learning paradigm	Offloading decision location	Training location	Offloading architecture	Objectives	Handover*
[16] (2025)	Binary	MADDPG[2]-based	Centralized training and distributed execution	Vehicles edge servers/ cloud servers	Vehicles/ edge servers/ cloud servers	Cloud-edge-device	Computational efficiency, energy, latency	×
Our	Binary	MAPPO[4]-based	Centralized training and distributed execution	Vehicles	Edge servers	Cloud-edge-device	QoE, energy	√

Note: *Whether to handle RSU communication coverage handover caused by vehicle mobility; [1]Proximal Policy Optimization; [2]Multi-Agent Deep Deterministic Policy Gradient; [3]Twin Delayed Deterministic Policy Gradient; [4]Multi-Agent Proximal Policy Optimization.

3 System Model and Issue Statement

This section highlights the task offloading system within the cloud-edge-device collaborative architecture for 6G IoV, as shown in Fig. 1. It depicts the system modeling process, encompassing system model, mobility model, task model, communication model, computation model, and task utility model. The key symbols used in this paper and their descriptions are detailed in Table 2.

Table 2: Symbols and their descriptions.

Symbol	Description
$Datacenter, host_h, vm_g, f^{vm}$	A cluster equipped with multiple servers, compute host h, virtual machine g and CPU computational capacity of a virtual machine.
$F_{cloud}, H_{cloud}, G_{cloud}, f_{cloud}^{vm}$	Computational capacity, the number of hosts of the remote cloud servers, the number of virtual machines for each of the remote cloud hosts, and the CPU computational capacity of each of the remote cloud virtual machines.
N, Rs	The number of edge computing nodes and the set of edge servers.
F_n, H_n, G_n, f_n^{vm}	Computational capability, the number of hosts of the edge server n, the number of virtual machines of each host for the edge server n, and CPU computational capacity of each virtual machine for the edge server n.
SV, O, SV_s, SV_o	Service vehicle, the number of service vehicles, the set of service vehicles, and the service vehicle o.

(Continued)

Table 2 (continued)

Symbol	Description
TV, M, TV_s, TV_m	Task-generated vehicle, the number of task-generated vehicles, the set of task-generated vehicles, and the task-generated vehicle m.
$p_m(\tau), p_{max}, p_{fiber}(\tau), \sigma$	The transmission power of the TV_m during time slot τ, the maximum transmission power of a vehicle, the transmission power over the time-slotted τ optical fiber between the RSU and the remote cloud, and the Gaussian noise in the environment.
$F_i, w_i, Datacenter_i, v_i, (x_i, y_i)$	The local server's computational capacity, allocated bandwidth, cluster, speed, and positional coordinate of the vehicle i.
$D_{m,s}, D_{max}$	Distance between vehicle m and node s, the maximum communication distance between vehicles.
$inData_m, outData_m, c_m,$ $type_m, maxE_m, sen_m$	The input data volume, output data volume, total quantity of instructions required for the task execution, task type, maximum expected delay requirement, and delay sensitivity of $Task_m$.
$r_m(\tau)$	The uplink transmission rate between vehicle m and other vehicles or edge computing nodes.
w_s^{total}, n_s	The total channel bandwidth of server s and the number of TVs that offload tasks to server s.
r_{MAN}, T_{MAN}^{pro}	The transmission rate of the Metropolitan Area Network and the propagation delay between RSUs.
η^{sv}, η^{Edge}	Energy consumption coefficients of a vehicle and an edge server.
r_{WAN}, T_{WAN}^{pro}	The transmission rate and the propagation delay of the Wide Area Network.
T_m, QoE_m, E_m, B_m	The total service delay, QoE, energy consumption, and total utility for the offloading task m.
ω_{QoE}	The weighting factor for balancing QoE and energy consumption.

3.1 Cloud-Edge-Device Collaborative 6G VEC System Model

The system model described in this paper is made up of three layers: the remote cloud layer, the edge layer, and the vehicle layer. The system time is divided into multiple time slots, denoted $\mathbb{T} = \{1, \ldots, \tau, \ldots, \tau_{end}\}$, with each time slot duration being Δ, each time slot representing one time step.

The remote cloud layer consists of a cluster equipped with multiple servers, offering abundant computing resources. The cluster is denoted as $Datacenter = \{host_1, \ldots, host_h, \ldots, host_H\}$, encapsulating a set of compute hosts (each host is a server). These hosts may feature either homogeneous or heterogeneous hardware configurations. Each host contains G virtual machines (VMs), denoted as $host_h = \{vm_1, \ldots, vm_g, \ldots, vm_G\}$. The CPU computational capacity of each virtual machine vm_g is denoted as f^{vm}, with units in GIPS. The remote cloud servers is represented as $Cloud = \{Datacenter_{cloud}\}$, and its computational capacity is $F_{cloud} = H_{cloud} \cdot G_{cloud} \cdot f^{vm}_{cloud}$.

The edge layer comprises N 6G edge computing nodes and an edge system, with each node consisting of an RSU and a single edge server connected via fiber optic links. The set of edge servers is represented as $Rs = \{R_1, \ldots, R_n, \ldots, R_N\}$. Each edge server is described by $R_n = \{F_n, x_n, y_n, Datacenter_n\}$ represents a single edge server, where F_n denotes the computational capability of edge server n, calculated as $Fn = H_n \cdot G_n \cdot f^{vm}_n$. x_n and y_n indicate the location of edge computing node n. The edge system comprises an agent management module and an edge management module, communicating in real-time with each edge node to transmit and receive information. It is equipped with high-performance servers.

The vehicle layer encompasses all ICVs on the road, comprising M task-generated vehicles (TVs) and O service vehicles (SVs) with idle resources. TVs locally execute computational tasks related to safety-critical applications, such as those generated by vehicle control and accident prevention systems. These types of tasks are not allowed to be offloaded. Computing tasks generated by other applications are fully offloaded to servers with abundant computing resources for processing. This paper discusses the scenario of complete task offloading. Each vehicle is represented as a seven-tuple, denoted as $EV_i = \{p_i, w_i, F_i, Datacenter_i, v_i, x_i, y_i\}$, where p_i represents the signal transmission power of vehicle i, w_i denotes the channel bandwidth allocated to the vehicle during upload tasks, each vehicle is configured with a single server, each server has one VM, and $F_i = f^{vm}_i$ indicates the computational capacity of the vehicle's local server. v_i is the vehicle's speed, and x_i and y_i are the vehicle's positional coordinates. At a certain moment, a vehicle enters a road segment within the communication coverage of an RSU. The set of TVs is denoted as $TVs = \{TV_1, \ldots, TV_m, \ldots, TV_M\}$. The set of SV servers is represented as $SVs = \{SV_1, \ldots, SV_o, \ldots, SV_O\}$. The components of each TV_m and each SV_o are the same as those of an EV_i.

When the TV enters the coverage area of the RSU signal, it observes local information and formulates its own offloading decision—that is, offloading tasks to cloud servers, edge servers, or SVs. Based on this offloading decision, the TV selects an appropriate target server and sends the requested task data to it. The target server executes the task and returns the computation results to the TV.

3.2 Mobility Model

Based on the mobility models from references [4,28], we assume vehicles travel on conventional one-way dual-lane ordinary highways. The road represents the x-axis, with the y-axis perpendicular to the road. The direction of vehicle travel from left to right corresponds to the positive direction of the x-axis. Therefore, the real-time positions of vehicles and RSUs can be represented by two-dimensional coordinates. The distances between TVs and SVs, as well as between TVs and edge computing nodes, can be calculated as follows:

$$D_{m,s} = \sqrt{(x_m - x_s)^2 + (y_m - y_s)^2} \tag{1}$$

where m is an index of TV, $\forall m \in \{1, 2, \ldots, M\}$. s is the subscript of the server, $\forall s \in \{1, 2, \ldots, O + N\}$. It is noted that whether a communication link can be established between two vehicles relies on the distance between them. Let D_{max} denotes the maximum communication distance between vehicles, which is the sum of the

communication range radius of both vehicles. If $D_{m,s} \leq D_{max}$, the two vehicles establish a communication connection. Otherwise, the communication connection fails.

3.3 Task Model

The system includes three types of application tasks: navigation applications, risk assessment applications, and infotainment applications. Tasks of different application types exhibit significant differences in terms of data size, computational power requirements, and latency tolerance. This paper adopts the identifier variable $Types = \{0, 1, 2\}$ to denote the three application task types: 0 represents navigation applications, 1 represents risk assessment applications, and 2 represents infotainment applications.

Assuming each TV generates one request task per time slot, the set of all request tasks within the system's time slot τ is denoted as $Tasks(\tau) = \{Task_1, \ldots, Task_m, \ldots, Task_M\}$, where M represents the total number of request tasks. An offloading request task is represented by a seven-tuple of the form $Task_m = \{tv_m, inData_m, outData_m, c_m, type_m, maxE_m, sen_m\}$, where tv_m is the vehicle ID generating task m, $inData_m$ denotes the task input data volume, $outData_m$ denotes the task output data volume, c_m refers to the total quantity of instructions required for the task execution (in GI units), and $type_m$ represents the task type. Each vehicle in the system continuously generates tasks of the same type. $maxE_m$ denotes the maximum expected delay requirement for completing $Task_m$, while $sen_m \in [0, 1]$ represents the delay sensitivity of $Task_m$. Delay sensitivity indicates delay tolerance; tasks with zero tolerance exhibit higher values.

3.4 Communication Model

The IoV communication system primarily consists of two communication modes: V2V and V2I. The two parties establish a high-bandwidth communication connection via a 6G wireless channel. The OBUs equipped in vehicles utilize activated PC5 interfaces to enable V2V and V2R links, providing communication services between vehicles and between vehicles and RSUs, respectively [4,31]. Vehicles can detect nearby RSUs and other vehicles in real time through sidelink communication on PC5 [32]. RSUs and surrounding vehicles periodically broadcast collaborative perception information specified in the C-V2X standard—such as their geographic location, computational capabilities, and available communication resources—and transmit these data to perceiving vehicles at a 10 Hz frequency. This ensures a balance between timely discovery and network overhead. Given that edge computing nodes are fixed along roadside locations with proximity and a fixed network topology, fiber-optic networks connect these nodes to the metropolitan area network (MAN) [10]. This MAN connection enables edge computing nodes to share computational resources via task migration. To ensure continuity of computation offloading services, if a TV leaves the coverage area of the serving edge computing node before completing an offloaded task request, the computation results will be transmitted to the TV via other edge computing nodes using a multi-hop approach. The handover process between edge computing nodes will only fail if a TV leaves the coverage area of the serving edge computing node while sending or receiving data. The MAN connecting RSUs is linked to the wide area network (WAN) via fiber optic cables. Consequently, vehicles can connect to remote cloud servers through nearby RSUs [10].

In the communication system described in this paper, we employ frequency division multiple access (FDMA) technology to evenly distribute the server's total channel bandwidth among all TVs [28,33]. It is assumed that communication interference between individual vehicles and between vehicles and edge computing nodes within a given road segment is negligible [28,34]. By Shannon's theorem, the uplink transmission rate for V2V and V2I communications can be computed as:

$$r_m(\tau) = \frac{w_s^{total}}{n_s} \cdot \log_2\left(1 + \frac{p_m(\tau)}{\sigma^2}\right) \tag{2}$$

where w_s^{total} is the total channel bandwidth of server s, $\forall s \in \{1, 2, ..., O + N\}$, n_s represents the number of TVs that offload tasks to server s, $p_m(\tau) \in [0, p_{max}]$ is the transmission power of the TV m during time slot τ. p_{max} is the maximum transmission power of a vehicle. σ is the Gaussian noise in the environment.

3.5 Computational Model

In the IoV offloading system, the TV typically offloads requested tasks to other servers to alleviate its own computational limitations. To fully leverage the heterogeneous resources of the cloud-edge-end computing model and meet the computational demands of different task types, this paper categorizes task offloading into three modes. Since the data volume of task computation results is significantly smaller than the input data volume, the communication latency and energy consumption associated with the SV or the edge computing node returning data to the TV can be considered negligible. Assume that each server utilizes all of its computational resources when executing tasks. Latency and energy consumption are detailed as follows.

3.5.1 Offload to SV for Processing

The service time T_m^{SV} that the TV sends the generated task m to the SV o for executing includes uploading delay and processing delay. T_m^{SV} is calculated as follows:

$$T_m^{SV} = T_{TV,SV}^{up} + T_{TV,SV}^{process} = \frac{inData_m}{r_m} + \frac{c_m}{F_o} \tag{3}$$

where F_o represents the available computing capacity of the SV o.

The energy consumption E_m^{SV} that the TV sends the generated task m to the SV o for executing includes uploading energy consumption and processing energy consumption. E_m^{SV} is calculated as follows [30,35]:

$$E_m^{SV} = E_{TV,SV}^{up} + E_{TV,SV}^{process} = p_m(\tau) \cdot \frac{inData_m}{r_m} + \eta^{sv} \cdot \frac{c_m}{F_o} \tag{4}$$

here, η^{sv} refers to a vehicle's energy consumption coefficient (energy consumption per second) [16].

3.5.2 Offload to Edge Server for Processing

The service time T_m^{Edge} that the TV sends the generated task m to the edge server n for executing includes uploading delay and processing delay. Specifically, the delay for task migration among RSUs becomes a critical issue when the computational capacity of the edge server n covering the TV is inadequate, T_m^{Edge} is calculated as follows:

$$T_m^{Edge} = \begin{cases} T_{TV,Edge}^{up} + T_{TV,Edge}^{process}, & RSU_c \\ T_{TV,Edge}^{up} + T_{RSU_c,RSU_t}^{migrate} + T_{TV,Edge}^{process}, & RSU_c \to RSU_t \end{cases} \tag{5}$$

where $T_{TV,Edge}^{up} = \frac{inData_m}{r_m}$ and $T_{TV,Edge}^{process} = \frac{c_m}{F_n}$, F_n represents available computing capacity of the edge server n, RSU_c is the RSU within communication range covering the TV m, RSU_t is the target RSU for task migration, and $T_{RSU_c,RSU_t}^{migrate}$ represents the communication delay between RSU for task m.

$$T_{RSU_c,RSU_t}^{migrate} = \frac{inData_m}{r_{MAN}} + T_{MAN}^{pro} \tag{6}$$

here, r_{MAN} represents the transmission rate of task m within MAN, while T_{MAN}^{pro} denotes the propagation delay between RSUs. Both are constants.

Corresponding to the service time, the energy consumption E_m^{Edge} that the TV sends the generated task m to the edge server o for executing includes energy consumption of uploading and task migration, processing energy consumption. E_m^{Edge} are calculated as follows [30,35]:

$$E_m^{Edge} = \begin{cases} E_{TV,Edge}^{up} + E_{TV,Edge}^{process}, & RSU_c \\ E_{TV,Edge}^{up} + E_{RSU_c,RSU_t}^{migrate} + E_{TV,Edge}^{process}, & RSU_c \to RSU_t \end{cases} \tag{7}$$

where $E_{TV,Edge}^{up} = p_m(\tau) \cdot \frac{inData_m}{r_m}$ and $E_{TV,Edge}^{process} = \eta^{Edge} \cdot \frac{c_m}{F_n}$, η^{Edge} refers to an edge server's energy consumption coefficient [16,30], $T_{RSU_c,RSU_t}^{migrate} = p_{fiber}(\tau) \cdot T_{RSU_c,RSU_t}^{migrate}$ represents the transmission energy consumption between RSU for the task m. $p_{fiber}(\tau)$ is the transmission power over the time-slotted τ optical fiber between the RSU and the remote cloud.

3.5.3 Offload to Remote Cloud Server for Processing

Due to the high computational power of the servers deployed in the remote cloud, processing delay and processing energy consumption are negligible compared to transmission latency and energy consumption. Therefore, the service time T_m^{Cloud} and energy consumption E_m^{Cloud} involved in offloading the $Task_m$ from the TV to the cloud server comprise only the upload transmission from the TV to the RSU and the round-trip communication between the RSU and the remote cloud.

$$T_m^{Cloud} = T_{TV,RSU}^{up} + T_{RSU,Cloud}^{upload} + T_{Cloud,RSU}^{down} \tag{8}$$

where $T_{TV,RSU}^{up} = \frac{inData_m}{r_m}$, $T_{RSU,Cloud}^{upload} = \frac{inData_m}{r_{WAN}} + T_{WAN}^{pro}$ and $T_{Cloud,RSU}^{down} = \frac{outData_m}{r_{WAN}} + T_{WAN}^{pro}$, r_{WAN} is the transmission rate of the WAN, T_{WAN}^{pro} is the propagation delay of the WAN. Both are constants.

Similar to T_m^{Cloud}, E_m^{Cloud} is calculated as follows [16]:

$$E_m^{Cloud} = E_{TV,RSU}^{up} + E_{RSU,Cloud}^{upload} + E_{Cloud,RSU}^{down} \tag{9}$$

where $E_{TV,RSU}^{up} = p_m(\tau) \cdot \frac{inData_m}{r_m}$, $E_{RSU,Cloud}^{upload} = p_{fiber}(\tau) \cdot T_{RSU,Cloud}^{upload}$ and $E_{Cloud,RSU}^{down} = p_{fiber}(\tau) \cdot T_{Cloud,RSU}^{down}$.

Based on the above, the total service delay for the offloading $Task_m$ is:

$$T_m = \phi_{sv} \cdot T_m^{SV} + \phi_{edge} \cdot T_m^{Edge} + \phi_{cloud} \cdot T_m^{Cloud} \tag{10}$$

and the energy consumption is:

$$E_m = \phi_{sv} \cdot E_m^{SV} + \phi_{edge} \cdot E_m^{Edge} + \phi_{cloud} \cdot E_m^{Cloud} \tag{11}$$

where $\phi_{sv}, \phi_{edge}, \phi_{cloud} \in \{0,1\}$, $\phi_{sv} + \phi_{edge} + \phi_{cloud} = 1$ indicates three distinct offload decision modes. ϕ_{sv}, ϕ_{edge} and ϕ_{cloud} indicate offloading task m to the SV server, the edge server, and the remote cloud server, respectively. Since each task can ultimately only be offloaded to and executed on a single server, only one of these offloading decision modes can be selected.

3.6 Task Utility Model

In IoV offloading systems, evaluating offloading benefits solely through the service time assessment system of successfully offloaded tasks is rather one-sided. For instance, if the offloading decision provides sufficient service time for a small fraction of successfully offloaded tasks while most offloading attempts fail, this does not demonstrate robust system performance. Therefore, we employ a QoE formula that simultaneously considers the service time and failure status of each task. The QoE formula is defined as follows:

$$QoE_m = \begin{cases} 0, & T_m \geq 2maxE_m \\ (1 - \frac{T_m - maxE_m}{maxE_m}) \cdot (1 - sen_m), & maxE_m < T_m < 2maxE_m \\ 1, & T_m \leq maxE_m \end{cases} \tag{12}$$

here, T_m denotes the actual service delay. QoE_m is the QoE of the task m. When the actual service time of task m exceeds $maxE_m$, the value of QoE_m for task m decreases. If the service time exceeds twice $maxE_m$, QoE_m is set to 0.

We aim to maximize the QoE for all tasks while minimizing energy consumption as much as possible. To accurately reflect the offloading utility of a task, we comprehensively consider both the task's QoE and energy consumption as the total utility B_m for task m, calculated as follows:

$$B_m = \omega_{QoE} \cdot QoE_m - (1 - \omega_{QoE}) E_m \tag{13}$$

where ω_{QoE} denotes the QoE weighting coefficient, used to balance the weighting of the task's QoE and energy consumption. $\omega_{QoE} \in [0, 1]$.

3.7 Issue Statement

Our objective is to enable vehicles to make effective offloading decisions for requested tasks based on current environmental conditions, thereby reducing task failure rates, minimizing service latency, and enhancing system performance. This is achieved through continuous training of decision-making network models under the assisted control of edge systems. Within the constraints of limited resources across all SVs, edge computing nodes, and remote clouds, to maximize the average total utility of all tasks. The optimization problem is defined as follows:

$$obj: \quad \max \sum_{\tau=1}^{\tau_{end}} \frac{\sum_{m=1}^{M} B_m}{M} \tag{14}$$

$$\begin{aligned} s.t. \quad & C_1 : \phi_{cloud}, \phi_{edge}, \phi_{sv} \in \{0, 1\} \\ & C_2 : (\phi_{cloud} + \phi_{edge} + \phi_{sv}) = 1 \\ & C_3 : 0 \leq \frac{w_s^{total}}{n_s} \leq w_s^{total} \\ & C_4 : 0 \leq p_m(\tau) \leq p_{max} \end{aligned}$$

Constraint C_1 specifies that request tasks must employ full offloading; Constraint C_2 ensures that any given request task can select only one offloading mode; Constraint C_3 ensures that the communication channel bandwidth allocated to a request task cannot exceed the total channel bandwidth of the target server; Constraint C_4 limits the vehicle's transmission power range between 0 and p_{max}.

4 MADRLSS Scheme

When servers and vehicles in the IoV are integrated into the environment at once, particularly when the number of vehicles reaches a medium or large scale, or when there are a large number of servers, the state space and action space can become excessively high-dimensional, leading to a significant increase in neural network parameters. This may prolong training time and even make the network hard to train [4]. To address this issue, we decompose the MADRLSS scheme into two processes: task offloading decision and resource allocation, as shown in Fig. 2. First, a MADRL-based algorithm that decouples centralized training from distributed execution, MAPPO generates an offloading decision scheme for each request task, preliminarily determining its offloading destination. Next, the TV selects the optimal server as the target server using the SS algorithm based on the decision scheme and allocates appropriate channel bandwidth to the request task. Upon acquiring these resources, the request task is immediately offloaded by the TV to the physical target server. Assign a virtual machine to execute the request task based on the sequence of virtual machines of the physical target server. Repeat the two processes until the system reaches its final state τ_{end} and all requested tasks have been completed. The MADRLSS method decomposes the task offloading process, reducing the dimensionality of the action space in reinforcement learning and making the network easier to train.

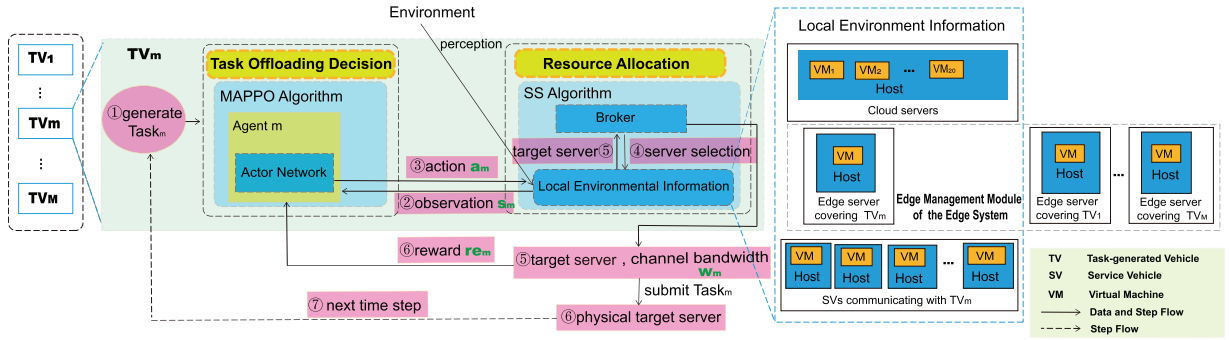


Figure 2: Task offloading full workflow for the MADRLSS scheme

4.1 Task Offloading Decision Based on MAPPO Algorithm with Decoupling Centralized Training from Distributed Execution

The task offloading process in the VEC environment exhibits Markovian properties and can be abstracted as a Markov decision process (MDP). In this paper, task offloading decisions are formulated as cooperative multi-agent reinforcement learning (MARL). Each TV is treated as an agent that continuously interacts with the environment, attempting actions and receiving rewards or penalties. Agents can collaboratively learn offloading strategies from others to achieve optimal task offloading decisions. First, we model the task offloading decision using a Markov game, which can be viewed as an extension of MDP to multi-agent settings [36]. Next, we outline the MADRL algorithm—MAPPO [37]—which offers stability advantages for finding optimal offloading decisions across multiple agents.

4.1.1 Markov Game Modeling

Task offloading decisions are modeled as a Markov game using the tuple $G = \{M, S, \{A_m | m \in M\}, \{RE_m | m \in M\}, P\}$, where M denotes the number of agents and S represents the global state space. At each time step, each agent $m \in M$ observes only the local state space: $S \to S_m$, with the agent's local state denoted as S_m. The private action spaces of each agent form the joint action space A, defined as $A := \times_{(m \in M)} A_m$. Based on S and A, the reward function $RE_m : S \times A \to RE$ represents the reward function for

agent m, while $P : S \times A \times S \to [0,1]$ denotes the global state transition function. These symbols will be elaborated upon subsequently.

Local State Space

At time step τ, the agent m can observe local information $s_m(\tau)$, including key details of the task $Task_m$ generated by the TV m, such as task size, required computational resources, sensitivity, and maximum tolerated delay. The cloud's available computational capacity, average virtual machine utilization, and WAN rate. Available computational capacity of the server of the RSU n, which covers the TV m, average virtual machine utilization, and the upload rate from the TV m to the RSU n. The average computational capacity and average virtual machine utilization of SVs establishing communication connections with the TV m, along with the average upload rate between the TV m and these SVs.

$$s_m(\tau) = \{Task_m(\tau), F_{cloud}(\tau), vm_{cloud}^{avg}(\tau), r_{WAN}(\tau), F_n(\tau), vm_n^{avg}(\tau), r_m^n, F_{SVs}^{avg}(\tau), vm_{SVs}^{avg}(\tau), r_m^{sv}\} \quad (15)$$

Global State Space

The global state of the system is the combination of the local state spaces of all agents at time step τ, expressed as:

$$s(\tau) = \{s_m(\tau)\}_m^M \quad (16)$$

Action Space

Each agent's action space consists of the available offloading decision modes for the requested task (remote cloud server, edge server, and SVs). $A_m(\tau)$ is represented by a binary vector. Positions with a value of 1 in this vector indicate the offload decision mode.

$$A_m(\tau) = (\phi_{cloud}, \phi_{edge}, \phi_{sv}) \quad (17)$$

where $\phi_{cloud}, \phi_{edge}, \phi_{sv} \in \{0,1\}$, $\phi_{cloud} + \phi_{edge} + \phi_{sv} = 1$. The action taken by the agent m at time step τ is denoted as $a_m(\tau) \in A_m(\tau)$.

Then, the action space of the entire system can be represented as:

$$A(\tau) = (A_1, \cdots, A_m, \cdots, A_M) \quad (18)$$

Reward Function

The Agent m executes offloading decisions $a_m(\tau)$ based on observed states $s_m(\tau)$. The environment provides the immediate reward $re_m(\tau)$ following execution of the MADRLSS method. Only when the reward inherently aligns with task utility can the agent learn offloading strategies that maximize overall system task utility. To accommodate different resource utilization levels across task types, a probability vector is designed for selecting distinct decision-making offloading methods, denoted as $Prob^{type} = (prob_{type}^{\phi^{cloud}}, prob_{type}^{\phi^{egde}}, prob_{type}^{\phi^{sv}})$, it represents the probability that a request task of type $type$ selects one of three offloading decision methods: remote cloud server, an edge server, or a SV. Here, $\forall type \in Types$.

Therefore, the reward function for the $Task_m$ at time step τ is calculated as follows:

$$re_m(\tau) = \begin{cases} -C_{type}, & fails\ to\ V2V \\ prob_{type}^{\phi} \cdot \alpha^{\phi} \cdot B_m = prob_{type}^{\phi} \cdot \left(\frac{success_{\phi}}{success_{\phi} + fail_{\phi}} + \frac{1}{upload_{\phi}}\right) \cdot B_m, & other \end{cases} \quad (19)$$

where $\phi \in \{\phi^{cloud}, \phi^{edge}, \phi^{sv}\}$. $prob^{\phi}_{type} \in Prob^{type}$ is the probability of selecting the offloading decision method ϕ for the $Task_m$ of type $type$. Since different types of request tasks cannot be completed within a single time step, the rewards provided by the environment to the agent exhibit significant delay. Therefore, variable α^{ϕ} is set as a dynamic feedback factor for agent-environment interaction, enabling real-time perception of the execution status of recent tasks under the offloading decision method ϕ. This enhances the environmental relevance of the instant reward $re_m(\tau)$, α^{ϕ} incorporates the number $upload_{\phi}$ of request tasks in the uplink of the offloading decision method ϕ, the number $success_{\phi}$ of historically successful tasks, and the number $fail_{\phi}$ of historically failed tasks. When the offload decision is ϕ^{sv}, if no communication link is established between vehicles, the environment imposes penalties C_{type} for different task types, which are constant values.

Since each agent is fully cooperative, at time step τ, all agents share a computable join reward such as:

$$RE(\tau) = \sum_{m=1}^{M} re_m(\tau) \tag{20}$$

To standardize the reward scale across different task types and prevent excessively large or small $re_m(\tau)$ values from negatively impacting the training of the value function. We adopt reward scaling, which dynamically calculates the standard deviation of a rolling discounted sum of rewards, then divides the current reward by this standard deviation [38]. The final combined reward is recorded as $RE^{scal}(\tau)$.

The objective of the MARL algorithm is for each agent to find a policy $\pi_m : S_m \times A_m \rightarrow [0,1]$ that maximizes the expected cumulative discounted reward $Rewards = \sum_{\tau=0}^{\tau_{end}} \gamma^{\tau} RE(\tau)$ obtained throughout the entire task offloading process, where τ_{end} is the time step at which the system reaches its final state and $\gamma \in [0,1]$ represents the reward discount factor for each time step.

4.1.2 Analysis of the MAPPO Algorithm

MAPPO is a multi-agent reinforcement learning framework based on Proximal Policy Optimization (PPO) [39]. PPO is an Actor-Critic (AC) architecture reinforcement learning algorithm comprising two actor networks (π_{θ} and $\pi_{\theta^{old}}$) with identical structures and a critic network V_{ω}; the network parameters are θ, θ^{old}, and ω, respectively. As an extension of PPO for multi-agent systems, MAPPO equips each agent with its own AC architecture, typically enhancing multi-agent coordination efficiency through CTDE. The actor network outputs action probability distributions. The old network $\pi_{\theta^{old}_m}$ generates sampled data across multiple time steps, while the new network π_{θ_m} learns network parameters using this sampled data, periodically synchronizing with the old network's parameters. The critic network V_{ω_m} estimates the value of the current global state, addressing the non-stationary nature of MARL environments and facilitating cooperative policy learning among agents.

The MAPPO algorithm framework presented in this paper is illustrated in Fig. 3. All agents are homogeneous, and each agent can only observe local information when executing actions. We configure each agent with its own actor network, while all agents share a common critic network. To reduce learning pressure on individual agents, each agent m deploys only its old actor network $\pi_{\theta^{old}_m}$ and executes actions independently. The new actor networks π_{θ_m} for all agents and the common critic network V_{ω} are deployed on the agent management module within the edge system, where centralized training is conducted for all agents. As the local computational capabilities of each agent are comparable and incur no training overhead, the primary communication overhead in centralized training stems from data transmission between agents and the agent management module at the edge system. Leveraging the low-latency, high-reliability, and massive-scale high-speed terminal access capabilities of 6G networks, the system achieves efficient, real-time wireless

data transmission and control capabilities. Based on these characteristics, we can adopt a fully synchronized mechanism to perform centralized training on all agents at a unified periodic frequency.

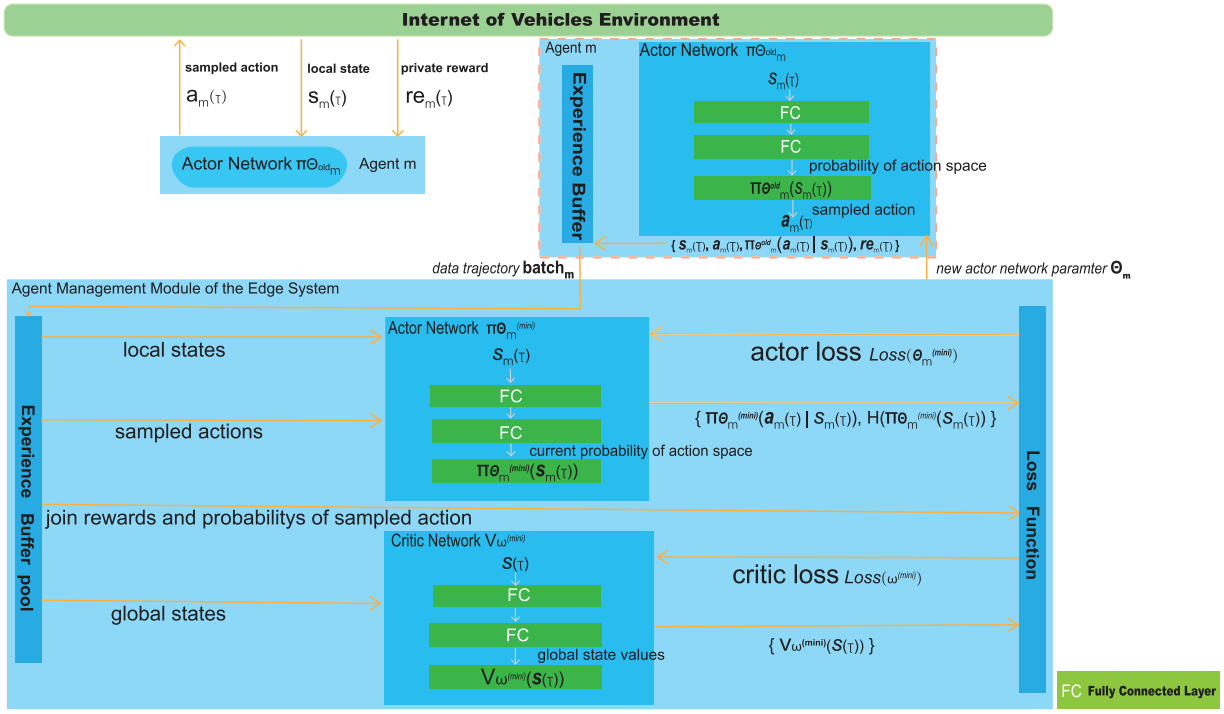


Figure 3: Framework of MAPPO.

During the MAPPO algorithm training, the agent management module of the edge system first forwards the initialized model parameters of the new actor network π_{θ_m} to each agent m as the model parameters of its local old actor network $\pi_{\theta_m^{old}}$. each agent m employs the $\pi_{\theta_m^{old}}$ to observe the current local state $s_m(\tau)$ and generate an offloading decision action $a_m(\tau)$. After interacting with the environment, it receives a reward $re_m(\tau)$. Collecting multiple data trajectories to form a batch $batch_m$. This $batch_m$ is uploaded at the same frequency to the agent management module of the edge system. At this stage, each agent m remains unaware of the global state and does not need to consider the unloading decisions made by other agents.

$$batch_m = \{s_m(\tau), a_m(\tau), \pi_{\theta_m^{old}}(a_m(\tau)|s_m(\tau)), re_m(\tau)|\tau \in T_{batch}\} \tag{21}$$

here, T_{batch} represents the number of time steps within $batch_m$. $a_m(\tau)$ is sampling action at the time step τ, $\pi_{\theta_m^{old}}(a_m(\tau)|s_m(\tau))$ refers to probability of the sampling action $a_m(\tau)$.

On the agent management module of the edge system, the data trajectory $batch_m$ received from agent m is stored in the module's experience buffer pool. Once data trajectories $\{batch_m|m \in M\}$ for all agents have been fully collected, each agent's new actor network $\{\pi_{\theta_m}|m \in M\}$ is updated centrally, integrating the data trajectories to update the critic network V_ω. Each update employs a mini-batch approach, denoted as $mini$. The new actor network $\pi_{\theta_m^{(mini)}}$ aims to maximize the expected value of accumulated discounted rewards combined with the new actor network's entropy. The loss function is the negative of this composite:

$$Loss(\theta_m^{(mini)}) = -(Loss^{CLIP}(\theta_m^{(mini)}) + c_{ent}Loss^{ENT}(\theta_m^{(mini)})) \tag{22}$$

then, $Loss^{CLIP}(\theta_m^{(mini)})$ is calculated as follows:

$$Loss^{CLIP}(\theta_m^{(mini)}) = E[\min(ratio_{\theta_m^{(mini)}}(\tau)\hat{A}^{(mini)}(\tau), clipratio_{\theta_m^{(mini)}}(\tau)1-\epsilon 1+\epsilon\hat{A}^{(mini)}(\tau)] \tag{23}$$

which $ratio_{\theta_m^{(mini)}} = \frac{\pi_{\theta_m^{(mini)}}(a_m(\tau)|s_m(\tau))}{\pi_{\theta_m^{old(mini)}}(a_m(\tau)|s_m(\tau))}$ refers to the ratio of the probabilities of selecting an action under the policies π_{θ_m} and $\pi_{\theta_m^{old}}$ in the same local state, describing the difference between the two, also known as the importance sampling weight. To prevent excessive divergence between the two policies and ensure the probability ratio remains close to 1, a clipping function is applied to constrain it. The clip function restricts out-of-bound values to the range $[1-\varepsilon, 1+\varepsilon]$, while leaving other values unchanged. Additionally, the min function(min) is used to take the minimum value between the clipped and unclipped results, further limiting the magnitude of policy updates [22]. $E[\cdot]$ denotes the expected value.

The generalized advantage estimation (GAE) method is employed to estimate the expected advantage function at each time step τ, denoted as $\hat{A}(\tau)$, expressed as follows:

$$\hat{A}(\tau) = \delta(\tau) + (\gamma\lambda)\delta(\tau+1) + \cdots + (\gamma\lambda)^{T_{batch}-\tau+1}\delta_{T_{batch}-1} \tag{24}$$

where $\delta_\tau = RE^{scal}(\tau) + \gamma V_\omega(s(\tau+1)) - V_\omega(s(\tau)$, λ is the weight coefficient controlling the multi-step sampling advantage function.

To encourage exploration of actions of the actor π_{θ_m}, the MAPPO algorithm introduces an entropy regularization term:

$$Loss^{ENT}(\theta_m^{(mini)}) = E[H(\pi_{\theta_m^{(mini)}}(s_m(\tau)))] \tag{25}$$

here, the function H refers to the entropy of the actor π_{θ_m}, which represents the uncertainty of the action distribution. Increasing entropy prevents the actor network from prematurely converging to local optima. c_{ent} is the entropy coefficient to balance the intensity of action exploration.

The optimization objective of the critic network V_ω is to minimize the estimation error of global state values, so ω is updated via the mean squared error loss. The loss function of the critic network is as follows:

$$Loss(\omega^{(mini)}) = E[V_{\omega^{(mini)}}(s(\tau)) - V_{\omega^{(mini)}}^{return}(s(\tau))]^2 \tag{26}$$

where $V_\omega^{return}(s(\tau)) = \hat{A}(\tau) + V_\omega(s(\tau))$ represents the cumulative discount reward from the time step τ to T_{batch}.

After completing each batch update, the parameters of the new actor network $\{\theta_m | m \in M\}$ are forwarded to each agent to update their old actor network $\{\theta_m^{old} | m \in M\}$ parameters. Repeat the two processes—local trajectory data collection and upload by each agent, and centralized training and parameter forwarding by the edge system—until the MAPPO algorithm converges and training concludes. The specific process is detailed in Algorithm 1. It is worth noting that during training, the data volume of each agent m uploading its trajectory $batch_m$ (approximately 10 KB) to the agent management module of the edge system, as well as the newly received actor network parameters (approximately 25 KB), is small and negligible [33,40].

After the MAPPO algorithm training concluded, the fully trained new actor network $\{\pi_{\theta_m} | m \in M\}$ is deployed to each agent $\{m | m \in M\}$, retaining only independent execution without requiring the critic network V_ω.

Algorithm 1: Task offloading decisions based on MAPPO algorithm with decoupling centralized training from distributed execution

Input: Num of episode *Epi*, maximum number of steps per episode T_{steps}, epochs *EK*, batch size T_{batch}, mini-batch size *B*, the IoV Environment

Output: $\{\theta_m | m \in M\}, \omega$

On the agent management module of the edge system, randomly initialize the parameters of the actor networks $\{\pi_{\theta_m} | m \in M\}$ and the critic network V_ω, $\{\theta_m | m \in M\}$ and ω ;

Initialize the experience buffer pool D ;

Forward the parameters $\{\theta_m | m \in M\}$ to all task vehicle agents, initialize their local relevant actor networks $\{\pi_{\theta_m^{old}} | m \in M\}$;

while $i < Epi$ **do**

  Reset rewards scaling function;

  for τ in 1, ..., T_{steps} **do**

    for *each agent* $m \in M$ **do**

      Monitor local state $s_m(\tau)$;

      Sampled the action $a_m(\tau)$ and obtained the probability of action $\pi_{\theta_m^{old}}(a_m(\tau)|s_m(\tau))$ uses $\pi_{\theta_m^{old}}$;

      Interact with the environment using the action $a_m(\tau)$ and receive the private reward $re_m(\tau)$ (after the environment executes all agent actions at time step τ) ;

      Store $\left\{s_m(\tau), a_m(\tau), \pi_{\theta_m^{old}}(a_m(\tau)|s_m(\tau)), re_m(\tau)\right\}$ in the experience buffer of agent m ;

      if $Epi \mod Freq == 0$ **then**

        Upload the sampled data trajectory $batch_m$ to the agent management module ;

        Empty $batch_m$ in the experience buffer of agent m ;

      end

    end

  end

  The agent management module receives data trajectory $batch_m$ of each task vehicle agent m, stores it in the experience buffer pool D, until the data trajectories $\{batch_m | m \in M\}$ from all agents have been fully collected ;

  for τ in 1,..., T_{batch} **do**

    Join reward $RE(\tau) \leftarrow \sum_{m=1}^{M} re_m(\tau)$ and calculate dynamically reward scaling $RE^{scal}(\tau)$ with $RE(\tau)$;

    Integrate the local states of all agents into a global state $s(\tau) \leftarrow \{s_m(\tau)\}_m^M$ and obtain $V_\omega(s(\tau))$;

    Store $\left\{s(\tau), RE^{scal}(\tau), V_\omega(s(\tau))\right\}$ in the experience buffer pool D ;

  end

  Compute GAE with (24) and $batchs_{mini} \leftarrow \frac{|D|}{B}$;

  for *mini* in 1, ..., $batchs_{mini}$ **do**

    Update each actor network $\pi_{\theta_m^{(mini)}}, m \in M$ with (22) ;

    Update the critic network $V_{\omega^{(mini)}}$ with (26) ;

  end

  The agent management module forwards the new parameters $\{\theta_m | m \in M\}$ to all agents, synchronize the parameters $\{\theta_m^{old} | m \in M\}$ for their local relevant actor networks ;

end

return $\{\theta_m | m \in M\}, \omega$;

4.2 Resource Allocation Based on the SS Algorithm

After obtaining an offloading decision method, the task $Task_m$ preliminarily identifies potential offloading locations. It is necessary to further determine the specific server for $Task_m$'s offloading and allocate the channel bandwidth resource w_m for the upload task. We designed the SS algorithm to determine the target server.

4.2.1 Short-Term Load and Distance

There are two influencing factors involved in the SS algorithm: short-term load (STL) of a server and distance between the TV and a server (the SV or the edge server) calculated using Eq. (1) The STL of a server indicates the total number of tasks offloaded to the relevant server within a short period (e.g., the last 0.5 s). In this paper, task offloading decisions are made on a per-time-step basis. Within each time step, generated request tasks are immediately sent by the TV to the target server for execution based on the offloading decision. During this period, the server's load state is considered instantaneously constant. Considering that the maximum expected service time for the in-vehicle application tasks involved in this paper is an integer multiple of 0.5 s, a single task can be completed in approximately 0.5 s at its fastest, thereby causing changes in server load. Therefore, this paper sets the server load update cycle to 0.5 s. Based on this update frequency, the STL value can be used to predict the future load state of the server [10]. It is denoted as STL_s, where s is the server index, $\forall s \in \{1, 2, ..., O + N\}$.

4.2.2 Weighted-Scoring Mechanism Based on the SS Algorithm

1) The request task $Task_m$ is offloaded to the remote cloud server. $Task_m$ receives the channel bandwidth evenly allocated by the nearby edge server R_c, represented as $w_m = \frac{w_c^{total}}{n_c}$. TV_m forwards $Task_m$ to the cloud server via R_c.

2) When task $Task_m$ is offloaded to an edge server, TV_m will prioritize choosing the edge server R_c whose communication range covers TV_m. Once the average CPU utilization of virtual machines on the edge server R_c exceeds 70%, it faces a risk of insufficient computational resources. At this point, TV_m selects the nearest edge server with the lowest load as the migration target, relocating the task from R_c to target edge server R_{target}. Similarly, $Task_m$ obtains R_c's average channel bandwidth, $w_m = \frac{w_c^{total}}{n_c}$. Inter-edge server transmission is achieved via the MAN. Let $vm_{threshold}$ denote the average CPU utilization threshold for edge server virtual machines. Optimal edge server selection is achieved by maximizing the function, calculated as follows:

$$n^{target} = \begin{cases} n_c^m, & vm_c \le vm_{threshold} \\ \arg\max\limits_{n \in N} \left\{ \frac{\omega_{stl}^{edge}}{STL_n} + \frac{\omega_d^{edge}}{D_{m,n}} \right\}, & vm_c > vm_{threshold} \end{cases} \tag{27}$$

where n^{target} is the index of the target edge server, m is the index of the TV, n_c^m is the index of the edge server whose communication range covers TV_m, ω_{stl}^{edge} is the weight coefficient for STL of the edge server, ω_d^{edge} is the weight coefficient for the distance between the TV and the edge server, and vm_c is the average CPU utilization threshold of the virtual machines in the edge server whose communication range covers TV_m.

3) When the request task $Task_m$ is offloaded to the SV server, the TV_m selects the SV server with the closest distance and the smallest load from the SVs that can establish a communication connection. The set $O_{connect}$ is used to represent the SV servers with communication connections. The optimal SV server

selection is achieved by maximizing the function, calculated as follows:

$$o^{target} = \arg \max_{o \in |O_{connect}|} \left\{ \frac{\omega_{stl}^{O_{connect}}}{STL_o} + \frac{\omega_d^{O_{connect}}}{D_{m,o}} \right\} \tag{28}$$

here, o^{target} denotes the target SV server subscript, $\omega_{stl}^{O_{connect}}$ represents the weight coefficient for the SV server's STL, and $\omega_d^{O_{connect}}$ denotes the weight coefficient for the distance between the TV and a SV server. $Task_m$ will obtain the mean channel bandwidth of the target SV server, $w_m = \frac{w_{o_{target}}^{total}}{n_{o_{target}}}$.

Notably, when offloading the request task $Task_m$ to an edge server or an SV server, different offloading decision modes assign varying weights to two factors: the server's STL and the current distance between TV_m and the server. These weights are determined through iterative trial and adjustment, as detailed in Algorithm 2.

Algorithm 2: Resource allocation based on the SS algorithm

Input: Task vehicle TV_m, action(offloading decision) A, server set of A $SERVER_A$, the edge server R_c whose communication range covers TV_m

Output: Target server $Server_{target}$, channel bandwidth w_m

```
// Number of server set of A and channel bandwidth for the upload task
```

$S_A \leftarrow |SERVER_A|$;

$w_m = 0$;

switch A **do**

  Case 1 **do**

    `// Offloading decision on edge servers`

    if $vm_c \le vm_{threshold}$ **then**

      $n_{target} \leftarrow n_c^m$;

    else

      $n_{target} \leftarrow \arg \max_{n \in S_A} \left\{ \frac{\omega_{stl}^{edge}}{STL_n} + \frac{\omega_d^{edge}}{D_{m,n}} \right\}$;

    end

    $Server_{target} \leftarrow SERVER_A[n_{target}]$;

    $w_m = \frac{w_c^{total}}{n_c}$;

    break ;

  case 2 **do**

    `// Offloading decision on SV servers`

    $SERVER_{connect} \leftarrow \varnothing$;

    for o *in* S_A **do**

      if $D_{m,o} \le D_{max} m, o$ **then**

        $SERVER_{connect} \leftarrow SERVER_{connect} \cup \{SERVER_A[o]\}$;

      end

    end

    $o_{target} \leftarrow \arg \max_{o \in |SERVER_{connect}|} \left\{ \frac{\omega_{stl}^{O_{connect}}}{STL_o} + \frac{\omega_d^{O_{connect}}}{D_{m,o}} \right\}$;

(Continued)

Algorithm 2 (continued)

$Server_{target} \leftarrow SERVER_{connect}[o_{target}]$;

$w_m = \frac{w_{o_{target}}^{total}}{n_{o_{target}}}$;

break ;

case *default* **do**

// Offloading decision on cloud servers

$Server_{target} \leftarrow Server_{cloud}$;

$w_m = \frac{w_c^{total}}{n_c}$;

end

end

return $Server_{target}, w_m$;

4.3 Complexity Analysis

This section will provide a detailed analysis of the time complexity of the algorithms involved.

The time complexity of the MAPPO algorithm requires distinguishing between the time complexity during training and after training completion. During training, the time complexity of the MAPPO algorithm primarily consists of the independent execution of the actor network $\{\pi_{\theta_m^{old}} | m \in M\}$ and the number of centralized training executions for the actor network $\{\pi_{\theta_m} | m \in M\}$ and critic network V_ω. Assuming the actor network and critic network have Lay_a and Lay_c fully connected layers, respectively, the time complexity during training can be expressed as $O\left(M \cdot Epi \cdot T_{steps} \sum_{l=1}^{Lay_a - 1} n_l^a n_{l+1}^a + \left\lfloor \frac{Epi}{Freq} \right\rfloor \cdot EK \cdot |T_{batch}| \left(M \cdot \sum_{l=1}^{Lay_a - 1} n_l^a n_{l+1}^a + \sum_{l=1}^{Lay_c - 1} n_l^c n_{l+1}^c\right)\right)$, where M denotes the number of agents, Epi represents the number of training episodes, $Freq$ indicates the training frequency, and EK signifies the number of update rounds. T_{steps} corresponds to the number of time steps per episode. n_l^a and n_l^c represent the number of neurons in the actor network and the critic network at the l-th layer, respectively. The time complexity after training does not include the centralized training of actor and critic networks for all agents that do not require edge systems. Only the actor network executed independently by each agent is considered. At this time, the time complexity can be expressed as $O\left(M \cdot Epi \cdot T_{steps} \sum_{l=1}^{Lay_a - 1} n_l^a n_{l+1}^a\right)$.

The time complexity of the SS algorithm primarily depends on the number of iterations over the set of edge servers Rs or the set of idle vehicles SV_s, it is $O(\max(N, 2 \cdot O))$. This exhibits linear scaling, resulting in extremely low time complexity.

Therefore, the total time complexity for each vehicle agent in the MADRLSS method is the number of operations performed independently by each agent $O(Epi \cdot T_{steps} \sum_{l=1}^{Lay_a - 1} n_l^a n_{l+1}^a + \max(N, 2 \cdot O))$.

5 Simulation Experiment

5.1 Experimental Setup

This paper utilizes the EdgeCloudSim [41] experimental platform as the edge computing simulation software to emulate cloud-edge-device computing scenarios, enabling the modeling of computational and network resources as well as mobile vehicles. All algorithms are implemented using both JAVA and Python 3.10, along with PyTorch 2.5.1. All experiments are conducted on the same machine (13th Gen Intel(R) Core(TM) i5-13500H 2.60 GHz, 32 GB RAM, Windows 11 operating system).

The duration of each time slot Δ is set to 0.5 s. We simulate a 10 km-long one-way dual-lane urban traffic scenario for performance analysis. 10 RSUs with a communication radius of 500 m are deployed along the road, providing full coverage. To better align with real-world vehicle mobility scenarios, the road is divided into multiple sections, with each section corresponding to the coverage area of an RSU. p_{max} for each vehicle is randomly sampled from the range $[1, 2]$ W. Vehicles travel on each segment at non-constant speeds, moving at one of three speeds—20 km/h, 40 km/h, or 60 km/h—to reflect different traffic densities. The low-speed segment represents a hotspot that appears during severe traffic congestion. Configure one host for each $Datacenter$. In the experiment, three different applications running on TVs generate tasks at varying rates, each with distinct characteristics in Table 3, while other experimental parameters are listed in Table 4. The experimental parameters are set with reference to literature [10,13,16,28,30]. After the simulation starts, 5 to 20 vehicles are randomly selected on the road as TVs, and 7 vehicles are designated as SVs. These vehicles are randomly assigned to road segments and then move in the same direction at predetermined speeds. To maintain a constant number of vehicles throughout the simulation, the road is designed as a loop route.

Table 3: Application characteristics.

	Navigation Application	Risk Analysis Application	Entertainment and Information Application
Proportion* (%)	30	35	35
Maximum Latency Demand (sec)	0.5	1	1.5
Latency Sensitiveness (sec)	0.5	0.8	0.25
Input/Out Data (KB)	100/10	150/10	105/420
Task Execution Instructions (Gl)	3	10	20

Note: *Percentage of the vehicles generating related application tasks.

Table 4: Parameters setting.

Parameter	Value
Mock Time/Preheating Period	360/3 min
Number of TVs	5–20
Number of RSUs	10
Computing Capacity of an Edge Server VM	40 GIPS
Total Channel Bandwidth of an Edge Server	100 MHz
Communication Radius of an RSU	500 m
Energy Consumption Coefficient of an Edge Server	$3 * 10^{-27}$
Number of SVs	7
Computing Capacity of an SV VM	7 GIPS
Total Channel Bandwidth of an SV	100 MHz
Communication Radius of a Vehicle	200 m
Transmission Power of a Vehicle	[1, 2] W
Energy Consumption Coefficient of a Vehicle	10^{-27}
Number of VMs per Cloud/Edge Server/SV	20/1/1
Capacity of a Cloud VM	150 GIPS
WAN/MAN Bandwidth	50/1000 Mpbs

(Continued)

Table 4 (continued)

Parameter	Value
WAN Propagation Delay/MAN Propagation Delay	150/10 ms
Long Distance Optical Fiber Transmission Power	[0.00625, 0.01] W
Environmental Noise	$1.22 * 10^{-4}$ W
QoE Weighting Coefficient	0.8
Fully-Connected Layers of an Actor/Critic Network	2
Number Neurons of per Hidden Layer	64
Training Frequency	1 episode
Time Steps per Episode	256
Number of Update Rounds	4
Time Steps of a Batch	256

5.2 Benchmark Methods

To verify the effectiveness of the presented task offloading scheme, MADRLSS, we picked the following three benchmark algorithms for comparison with the method proposed in this paper.

1. **Multi-Agent Dueling Double Deep Q-Network (MAD3QN) and SS Algorithm-Based Offloading Method (MAD3QNSS):** The MADRL algorithm MAD3QN [42] employs an epsilon-greedy strategy for discrete action exploration to obtain task offloading decisions. Resource allocation utilizes the SS algorithm presented in this paper.
2. **Random and SS Algorithm-Based Offloading Method (RandomSS):** Task offloading decisions are generated probabilistically using a random approach, where tasks are assigned with equal probability to execute on remote cloud servers, edge servers, or SV servers. Resource allocation employs the SS algorithm described in this paper.
3. **MADRL and Default-Based Offloading Method (MADRLDefault):** MADRLDefault employs the MAPPO algorithm described in this paper for task offloading decisions, with the system defaulting to selecting the target server instead of using the SS algorithm. That is, when the offloading decision method is set to remote cloud servers, it remains consistent with MADRLSS. When the offloading decision method is an edge server, tasks are only offloaded to edge servers within the communication coverage area of TV, without considering task migration. When tasks are offloaded to an SV server, the system randomly selects one SV to execute the task.

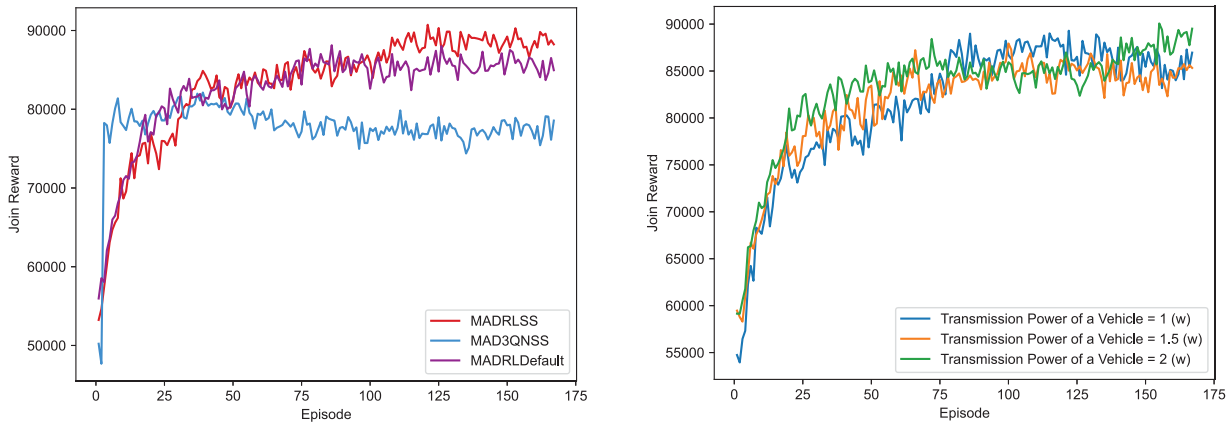
5.3 Performance Evaluation

To completely analyze the performance of the proposed solution, we incorporate five key performance indicators into the comparative algorithm performance assessment: algorithm convergence, service time, task failure rate, energy consumption, and QoE.

5.3.1 Algorithmic Convergence

Fig. 4 illustrates the convergence performance of the task offloading schemes. The learning curves of MADRLSS, MAD3QNSS, and MADRLDefault during training are shown in (a), demonstrating convergent performance across the same number of tasks. The proposed MADRLSS exhibits poor reward performance within the first 50 episodes but enters a faster, smoother convergence phase with higher joint rewards

after exceeding 50 episodes. In contrast, the MAD3QNSS learning curve initially increases, likely due to favorable state-action sequences encountered during multi-agent learning that temporarily yield higher rewards. However, the joint reward ceases to rise thereafter. MADRLDefault exhibits reward patterns similar to MADRLSS for the first 120 episodes, but the joint reward also stops increasing after 120 episodes. The primary reason is that the MAPPO algorithm employed in MADRLSS incorporates a pruning mechanism to limit the magnitude of policy updates. While ensuring the scale of policy updates, it introduces policy entropy to enhance exploration capabilities among agents, enabling cooperative learning of optimal offloading strategies. This also demonstrates that the SS algorithm in MADRLSS achieves higher rewards through rational resource allocation. Since RandomSS does not involve a reward function, its reward behavior was not analyzed.



(**a**) Convergence performance of different offloading schemes

(**b**) Convergence performance of different maximum transmission power of a vehicle (the MADRLSS scheme)

Figure 4: Comparison of convergence performance.

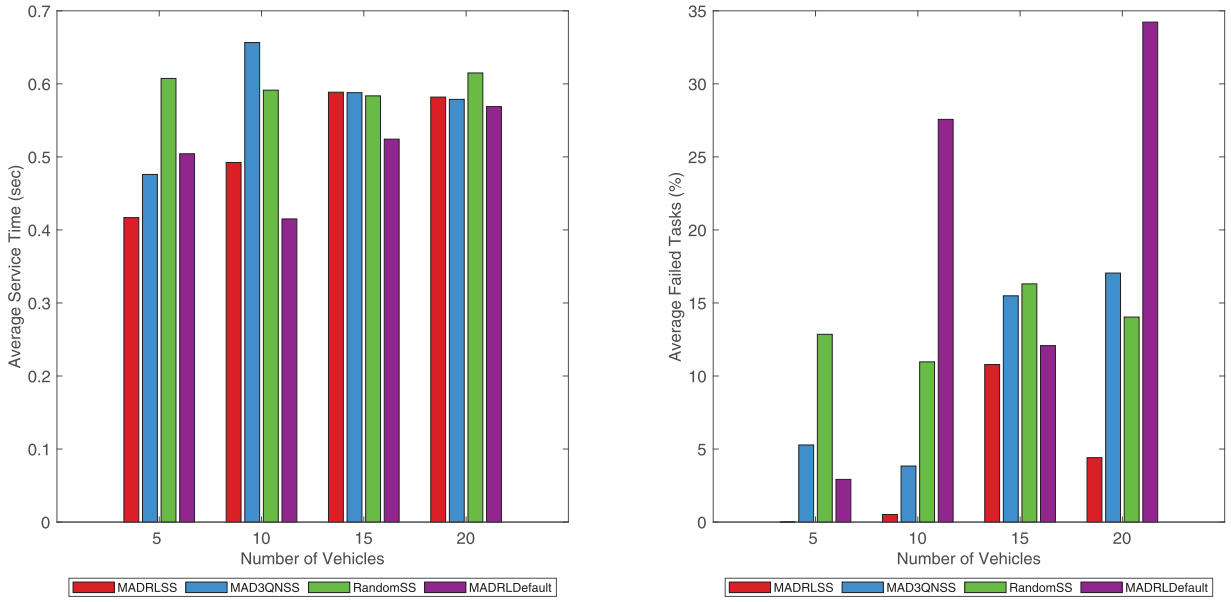
The impact of the proposed MADRLSS method on the joint reward at different maximum transmission powers of a vehicle is demonstrated in (b). As the maximum transmission power of a vehicle increases, the joint reward also increases. Additionally, under the condition of maximum transmission power of a vehicle, the proposed MADRLSS method exhibits lower reward volatility and more stable convergence. The primary reason is that increasing the maximum transmission power enables higher V2V and V2I transmission rates and shorter transmission times. This improves the service time and QoE for delay-sensitive tasks. Since QoE carries a higher weighting, the energy consumption increase resulting from the power boost is smaller than the corresponding QoE improvement. This enhances the stability of the agent's collaborative learning optimization for offloading decision strategies.

5.3.2 Service Time and Task Failure Rate

Fig. 5 illustrates service time and the task failure rates of different offloading schemes.

Service time refers to the time from when a service task is requested to when it is successfully executed and returns. A comparison of the service time for different task offloading schemes with the same quantity of vehicles is demonstrated in (a). Along with the rising number of vehicles, the quantity of generated tasks also rises. Due to intensified resource competition and network congestion, the average service time of all offloading schemes increases. The average service time of MADRLSS is at a moderate level under the same quantity of vehicles. The average service time for MAD3QNSS and RandomSS is relatively high.

MADRLDefault exhibits the lowest overall average service time, but it also has a high task failure rate. This indicates that only a small fraction of tasks are successfully executed. The primary reasons are the failure to migrate tasks to other edge servers with sufficient resources when the current edge server becomes locally overloaded during task offloading, coupled with a random selection of SVs. This demonstrates the effectiveness of the SS algorithm.



(**a**) Average service time

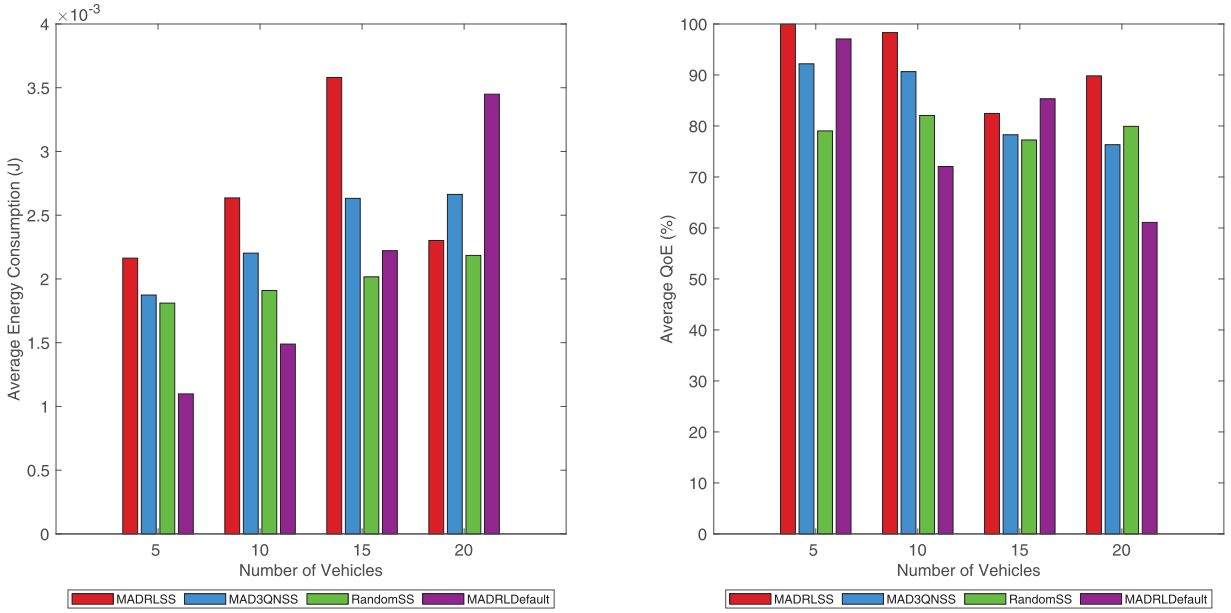
(**b**) Average task failure rates

Figure 5: Comparison of average task failure rates and average service time of different offloading schemes.

The relationship between the average number of failed tasks and the number of vehicles is shown in (b). As the number of task-generated vehicles increases, greater pressure is placed on the system's various resources. Consequently, the average task failure rate rises for all schemes. The MADRLSS scheme proposed in this paper consistently maintains the lowest average task failure rate, remaining below 11%. The average task failure rate for MAD3QNSS and RandomSS is relatively high. MADRLDefault initially did not have the highest average task failure rate. However, as the number of vehicles increased, the average task failure rate suddenly surged, reaching its peak at 10 and 20 vehicles. The primary reason is that MADRLSS can synergistically utilize computational resources across cloud, edge, and idle vehicles to reduce task failure rates. Specifically, MADRLSS offloads most small-to-medium-sized request tasks to edge servers near TVs. When an edge server nears local overload, it migrates the request tasks to nearby alternative edge servers for execution. Large-scale, less time-sensitive requests are offloaded to cloud servers. Simple requests are offloaded when SVs with favorable conditions appear near the TV. This demonstrates MADRLSS's robust capability to allocate computational resources reasonably even under resource-constrained and high-network-load conditions.

5.3.3 Energy Consumption and QoE

Fig. 6 illustrates the average energy consumption per task offloading and the average QoE per request task.



(**a**) Average energy consumption (**b**) Average QoE

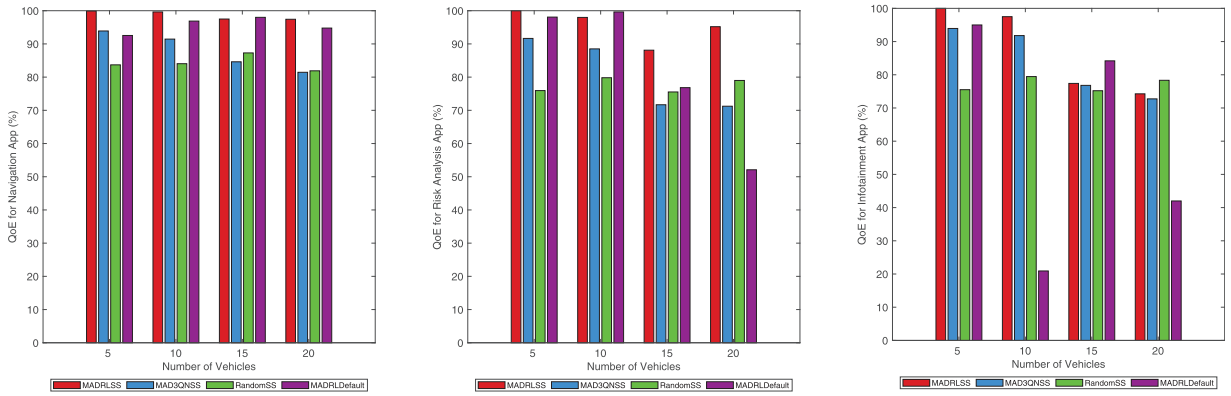
Figure 6: Comparison of average energy consumption and average QoE.

A comparison of the average energy consumption for different task offloading schemes with the same quantity of vehicles is demonstrated in (a). Along with the increase in the number of vehicles generated for tasks, the pressure on the system's resources has grown accordingly. The average energy consumption of all offloading schemes increases. The proposed method, MADRLSS, exhibits relatively high overall average energy consumption, while MAD3QN and RandomSS demonstrate comparatively lower average energy consumption. MADRLDefault typically achieves the lowest average energy consumption in the early stages. However, when the number of vehicles reaches 20, it exhibits the highest average energy consumption, indicating task offloading instability.

The tendency of the average QoE of tasks to change with the quantity of vehicles is shown in (b). The increasing number of vehicles generated for tasks places greater strain on system resources, resulting in a continuous decline in the average QoE for all solutions. Under different vehicle densities, the MADRLSS scheme proposed in this paper achieves the highest average QoE overall compared to other schemes, consistently maintaining a level above 82%. When the number of vehicles is 5, the MADRLSS scheme can supply the highest QoE (100%). MAD3QNSS and RandomSS typically exhibit relatively low average QoE. Due to instability in the resource allocation process, MADRLDefault exhibits significant fluctuations in average QoE. Specifically, it achieves a higher average QoE when the number of vehicles is 5 or 15, but the average QoE is lowest for other vehicle counts.

Fig. 7 illustrates the average QoE comparison across different application types for various task offloading schemes under identical vehicle counts. In the navigation application (a) and the risk assessment application (b), the proposed MADRLSS method achieves the highest overall average QoE. In entertainment and information application (c), MADRLSS delivers the highest average QoE when vehicle counts are 5 and 10. Subsequently, MADRLSS's average QoE was slightly lower than RandomSS and MADRLDefault, which incorporate random strategies, possibly due to encountering favorable conditions by chance. However, compared to the stable MAD3QNSS, MADRLSS consistently achieved the highest average QoE. Therefore,

compared to the baseline methods, MADRLSS demonstrates optimal overall task offloading performance, ensuring the QoE for computational tasks across different application types.



(**a**) Navigation application (**b**) Risk analysis application (**c**) Entertainment and information application

Figure 7: Comparison of average QoE of different application types.

Overall, while MADRLDefault achieves low energy consumption by avoiding frequent task transmission and processing, this comes at a high cost, such as higher task failure rates and lower, unstable QoE, severely impacting the overall performance of the IoV offloading system. In real-world IoV scenarios, the ability to guarantee task completion latency and service experience quality is often more critical than minimizing energy consumption. For instance, delays in processing critical tasks like navigation applications update or risk analysis applications may introduce safety hazards, potentially negating the benefits of reduced energy consumption. Furthermore, high task failure rates can result in the loss of critical tasks, undermining the emotional experience of vehicle users.

5.4 Experimental Summary and Implications

Experimental results demonstrate that the proposed MADRLSS scheme achieves superior performance across all evaluation metrics compared to existing benchmark approaches. Its advantages become particularly pronounced in scenarios with increased request task volumes, indicating not only its feasibility in small-scale environments but also its effective adaptation to complex scenarios involving high-density vehicles and large-scale tasks. This is primarily attributed to the CTDE architecture adopted in the method. This design transfers the computationally intensive model training process to edge systems, substantially reducing computational pressure on terminal vehicles. Consequently, it enables scaling to larger-scale distributed IoV environments. This characteristic positions MADRLSS as a robust task offloading solution with strong scalability, tailored for real-world complex scenarios.

6 Conclusion

This paper addresses the challenge of coordinating heterogeneous resources across cloud, edge, and terminal vehicles in high-density, large-scale task-based vehicle networking scenarios. By setting the optimization objective as maximizing the total utility of all tasks (i.e., the weighted sum of QoE and energy consumption of tasks), it proposes a MADRL method based on a centralized training and distributed execution decoupling architecture, termed MADRLSS, to achieve efficient task offloading and rational allocation of heterogeneous resources in distributed environments.

First, the MAPPO algorithm in the MADRLSS approach employs a Markov game to model the multi-vehicle, multi-server task offloading decision process. Through multi-agent collaborative learning, it shifts centralized evaluation and training to edge systems, enabling each agent to solve dynamic task offloading decisions in a lightweight manner. Then, considering edge server load balancing and unstable V2V communication connections, this paper designs the SS algorithm based on weighted server scoring. By incorporating two key factors—server load and distance between servers and TVs—it implements a weighted scoring mechanism, identifies the optimal target server, and achieves efficient communication resource allocation.

Experimental results indicate that at the same environmental conditions, the proposed MADRLSS scheme outperforms other baseline schemes. While it incurs a slight increase in energy consumption, it demonstrates significant advantages in most indicators, including algorithm convergence, service time, task failure rate, and QoE. This scheme coordinates multi-vehicle collaborative learning of respective offloading strategies through joint incentives, effectively integrating multi-level computational resources across cloud, edge, and terminal vehicles. Consequently, it significantly enhances the overall efficiency of task offloading and execution in IoV environments.

In future work, we will focus on task offloading in dynamic and open vehicle networking environments, extending existing algorithms to ensure their security and protect user privacy.

Acknowledgement: Not applicable.

Funding Statement: This work is supported in part by the Scientific Research Fund of Hunan Provincial Education Department (24A0337), and the Natural Science Foundation of Hunan Province (2025JJ50348).

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, methodology, software, and writing, Fangxiang Hu; validation and formal analysis, Qi Fu and Jing Huang; supervision and project administration, Shiwen Zhang. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: Due to the nature of this research, participants of this study did not agree for their data to be shared publicly, so supporting data is not available.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Li C, Dong M, Fu Y, Richard Yu F, Cheng N. Integrated sensing, communication, and computation for IoV: challenges and opportunities. IEEE Commun Surv Tutor. 2026;2025(1):1. doi:10.1109/COMST.2025.3612388.
2. Zhang S, Yi N, Ma Y. A survey of computation offloading with task types. IEEE Trans Intell Transp Syst. 2024;25(8):8313–33. doi:10.1109/TITS.2024.3410896.
3. Liu J, Du Y, Yang K, Wu J, Wang Y, Hu X, et al. Edge-cloud collaborative computing on distributed intelligence and model optimization: a survey [Internet]. 2025 [cited 2025 Oct 3]. Available from: https://arxiv.org/abs/2505.01821.
4. Chen Q, Song X, Song T, Yang Y. Vehicular edge computing networks optimization via DRL-based communication resource allocation and load balancing. IEEE Trans Mob Comput. 2025;24(9):9222–37. doi:10.1109/TMC.2025.3559707.
5. Ju T, Zhang W, Yang Y, Huo J. A second order decision-making dynamic offloading method for vehicle edge computing tasks. J Jilin Univ. 2025; 1–13. (In Chinese). doi:10.13229/j.cnki.jdxbgxb.20241283.
6. Wang CX, You X, Gao X, Zhu X, Li Z, Zhang C, et al. On the road to 6G: visions, requirements, key technologies, and testbeds. IEEE Commun Surv Tutor. 2023;25(2):905–74. doi:10.1109/COMST.2023.3249835.

7. Prathiba SB, Raja G, Anbalagan S, AK S, Gurumoorthy S, Dev K. A hybrid deep sensor anomaly detection for autonomous vehicles in 6G-V2X environment. IEEE Trans Netw Sci Eng. 2023;10(3):1246–55. doi:10.1109/TNSE.2022.3188304.
8. Xu B, Deng T, Liu Y, Zhao Y, Xu J, QI J, et al. Optimization of cooperative offloading model with cost consideration in mobile edge computing. Soft Comput. 2023;27(12):8233–43. doi:10.1007/s00500-022-07733-1.
9. Xia Y, Tian J, Zhang H, Yuan D. Joint task offloading and pricing strategy for multi-tier vehicular edge computing networks: a multi-leader multi-follower stackelberg game approach. IEEE Trans Cogn Commun Netw. 2026;12:1877–91. doi:10.1109/TCCN.2025.3600993.
10. Sonmez C, Tunca C, Ozgovde A, Ersoy C. Machine learning-based workload orchestrator for vehicular edge computing. IEEE Trans Intell Transp Syst. 2021;22(4):2239–51. doi:10.1109/TITS.2020.3024233.
11. Lei J, Liu H, Zhang J, Li S. Review of vehicular edge computing and task offloading based on deep reinforcement learning. Comput Syst Appl. 2025;34(11):1–19. (In Chinese). doi:10.15888/j.cnki.csa.010002.
12. Mohamad A, Hady SH, Mahardhika Pratama ZC, Kowalczyk R. Multi-agent reinforcement learning for resources allocation optimization: a survey. Artif Intell Rev. 2025;58(11):354. doi:10.1007/s10462-025-11340-5.
13. Lin J, Huang S, Zhang H, Yang X, Zhao P. A deep-reinforcement-learning-based computation offloading with mobile vehicles in vehicular edge computing. IEEE Internet Things J. 2023;10(17):15501–14. doi:10.1109/JIOT.2023.3264281.
14. Bitam S, Mellouk A, Zeadally S. VANET-cloud: a generic cloud computing model for vehicular Ad Hoc networks. IEEE Wirel Commun. 2015;22(1):96–102. doi:10.1109/MWC.2015.7054724.
15. Zhang W, Zhang Z, Chao HC. Cooperative fog computing for dealing with big data in the internet of vehicles: architecture and hierarchical resource management. IEEE Commun Mag. 2017;55(12):60–7. doi:10.1109/MCOM.2017.1700208.
16. Zhang P, Wang E, Tan L, Kumar N, Wang J, Liu K. Enhancing task offloading in vehicular networks: a multi-agent cloud-edge-device framework. Veh Commun. 2025;53:100898. doi:10.1016/j.vehcom.2025.100898.
17. Li ZY, Wang Q, Chen YF, Xie GQ, Li RF. A survey on task offloading research in vehicular edge computing. Chin J Comput. 2021;44(5):963–82. (In Chinese). doi:10.11897/SP.J.1016.2021.00963.
18. Zhou X, Bilal M, Dou R, Rodrigues JJPC, Zhao Q, Dai J, et al. Edge computation offloading with content caching in 6G-Enabled IoV. IEEE Trans Intell Transp Syst. 2024;25(3):2733–47. doi:10.1109/TITS.2023.3239599.
19. Xu XL, Yang W, Yang CY, Cheng Y, Qi LY, Xiang HL, et al. Efficient task offloading based on traffic prediction in IoV-enabled edge computing. Acta Electron Sin. 2025;53(2):329–43. (In Chinese).
20. Zhang Q, Wang Y, Zhang X, Liu L, Wu X, Shi W, et al. OpenVDAP: an open vehicular data analytics platform for CAVs. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS); Piscataway, NJ, USA: IEEE; 2018. p. 1310–20. doi:10.1109/ICDCS.2018.00131.
21. He B, Li T. An offloading scheduling strategy with minimized power overhead for internet of vehicles based on mobile edge computing. J Inf Process Syst. 2021;17(3):489–504. doi:10.3745/JIPS.01.0077.
22. Li P, Xiao Z, Gao H, Wang X, Wang Y. Reinforcement learning based edge-end collaboration for multi-task scheduling in 6G enabled intelligent autonomous transport systems. IEEE Trans Intell Transp Syst. 2025;26(10):1–14. doi:10.1109/TITS.2024.3525356.
23. Chen X, Hu S, Yu C, Chen Z, Min G. Real-time offloading for dependent and parallel tasks in cloud-edge environments using deep reinforcement learning. IEEE Trans Parallel Distrib Syst. 2024;35(3):391–404. doi:10.1109/TPDS.2023.3349177.
24. Li Y, Chen Y, Lan T, Venkataramani G. MobiQoR: pushing the envelope of mobile edge computing via quality-of-result optimization. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS); Piscataway, NJ, USA: IEEE; 2017. p. 1261–70. doi:10.1109/ICDCS.2017.54.
25. Zhu C, Tao J, Pastor G, Xiao Y, Ji Y, Zhou Q, et al. Folo: latency and quality optimized task allocation in vehicular fog computing. IEEE Internet Things J. 2019;6(3):4150–61. doi:10.1109/JIOT.2018.2875520.

26. Zhao N, Pei Y, Niyato D. Incentive mechanism for task offloading and resource cooperation in vehicular edge computing networks: a deep reinforcement learning-assisted contract approach. IEEE Internet Things J. 2024;11(24):41098–109. doi:10.1109/JIOT.2024.3457592.
27. Zhu C, Pastor G, Xiao Y, Li Y, Ylae-Jaeaeski A. Fog following me: latency and quality balanced task allocation in vehicular fog computing. In: 2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON); Piscataway, NJ, USA: IEEE; 2018. p. 1–9. doi:10.1109/SAHCN.2018.8397129.
28. Lv P, Xu W, Nie J, Yuan Y, Cai C, Chen Z, et al. Edge computing task offloading for environmental perception of autonomous vehicles in 6G networks. IEEE Trans Netw Sci Eng. 2023;10(3):1228–45. doi:10.1109/TNSE.2022.3211193.
29. Ning Z, Dong P, Wang X, Rodrigues JJPC, Xia F. Deep reinforcement learning for vehicular edge computing: an intelligent offloading system. ACM Trans Intell Syst Technol. 2019;10(6):1–24. doi:10.1145/3317572.
30. She H, Yan L, Guo Y. Efficient end-edge–cloud task offloading in 6G networks based on multiagent deep reinforcement learning. IEEE Internet Things J. 2024;11(11):20260–70. doi:10.1109/JIOT.2024.3372614.
31. Annu, Rajalakshmi P. Towards 6G V2X sidelink: survey of resource allocation—mathematical formulations, challenges, and proposed solutions. IEEE Open J Veh Technol. 2024;5(22):344–83. doi:10.1109/OJVT.2024.3368240.
32. Noor-A-Rahim M, Liu Z, Lee H, Khyam MO, He J, Pesch D, et al. 6G for vehicle-to-everything (V2X) communications: enabling technologies, challenges, and opportunities. Proc IEEE. 2022;110(6):712–34. doi:10.1109/JPROC.2022.3173031.
33. Li H, Meng S, Sun J, Cai Z, Li Q, Zhang X. Multi-agent deep reinforcement learning based multi-task partial computation offloading in mobile edge computing. Future Gener Comput Syst. 2025;172:107861. doi:10.1016/j.future.2025.107861.
34. Wang J, Lv T, Huang P, Mathiopoulos PT. Mobility-aware partial computation offloading in vehicular networks: a deep reinforcement learning based scheme. China Commun. 2020;17(10):31–49. doi:10.23919/JCC.2020.10.003.
35. Yue K, Peng K, Lin Y, Zhao X, Xu X, Leung VCM. PORPRS: priority-aware task offloading in HAP-aided internet of vehicles via GRPO with dynamic residual shrinkage networks. Ad Hoc Netw. 2025;179:104004. doi:10.1016/j.adhoc.2025.104004.
36. Liu Y, Wang W, Hu Y, Hao J, Chen X, Gao Y. Multi-agent game abstraction via graph attention neural network. Proc AAAI Conf Artif Intell. 2020;34(5):7211–8. doi:10.1609/aaai.v34i05.6211.
37. Yu C, Velu A, Vinitsky E, Gao J, Wang Y, Bayen A, et al. The surprising effectiveness of PPO in cooperative multi-agent games. arXiv:2103.01955. 2022. doi:10.48550/arXiv.2103.01955.
38. Engstrom L, Ilyas A, Santurkar S, Tsipras D, Janoos F, Rudolph L, et al. Implementation matters in deep policy gradients: a case study on PPO and TRPO. arXiv:2005.12729. 2020.
39. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. arXiv:1707.06347. 2017.
40. Shuai K, Miao Y, Hwang K, Li Z. Transfer reinforcement learning for adaptive task offloading over distributed edge clouds. IEEE Trans Cloud Comput. 2023;11(2):2175–87. doi:10.1109/TCC.2022.3192560.
41. Sonmez C. EdgeCloudSim. 2018 [cited 2025 Oct 3]. Available from: https://github.com/CagataySonmez/EdgeCloudSim/tree/master?tab=readme-ov-file.
42. Ji Y, Wang Y, Zhao H, Gui G, Gacanin H, Sari H, et al. Multi-agent reinforcement learning resources allocation method using dueling double deep Q-network in vehicular networks. IEEE Trans Veh Technol. 2023;72(10):13447–60. doi:10.1109/TVT.2023.3275546.