MVPOA: A Learning-Based Vehicle Proposal Offloading for Cloud–Edge–Vehicle Networks

Wenjing Xiao, Xin Ling, Miaojiang Chen, Junbin Liang, Salman A. Alqahtani, and Min Chen, *Fellow, IEEE*

***Abstract*—Vehicular edge computing (VEC) is an emerging computing paradigm that is rapidly advancing the development of the Internet of Vehicles (IoV). However, edge server has limited data storage capacity and computing resource, making it difficult to handle the massive offloading requests from IoV applications. Moreover, the mobility of vehicles and dynamic data traffic make it highly challenging to design optimal offloading and resource allocation strategies. To address the challenges mentioned above, we design a cloud–edge–vehicle hierarchical architecture for IoV task offloading, introducing a cloud server to assist in computation and alleviate the overload pressure on edge server. Considering the impact of vehicle mobility on task offloading, we propose a mobility detection method to predict which vehicles might leave the communication range of the base station, thereby preventing task offloading failures. Additionally, to achieve efficient task offloading and resource allocation in this complex IoV system, we propose a multiagent-reinforcement-learning-based vehicle proposal offloading algorithm (MVPOA). This algorithm enables vehicles to autonomously decide whether to process tasks locally or propose offloading to edge server. The edge server then decides whether to accept offloading requests based on task priority and sends rejected tasks to cloud server for processing, thereby maximizing the utilization of resources at each layer of the system. Simulation results demonstrate that MVPOA outperforms other baseline approaches in optimizing system delay and energy consumption.**

***Index Terms*—Cloud-assisted computing, mobility, multiagent reinforcement learning, resource allocation, task offloading, vehicular edge computing (VEC).**

I. Introduction

With the advancement of artificial intelligence technology, an increasing number of smart vehicles are connecting to the Internet of Vehicles (IoV), leading to explosive growth in computation-intensive and delay-sensitive applications, such as image-assisted navigation, dynamic video processing, and safety-enhanced autonomous driving [1], [2]. These applications require real-time processing and analysis of vast amounts of sensor data, often demanding substantial computing resources. However, the limited computing capability of vehicles can result in significant delay and increased energy consumption, adversely affecting user experience. Vehicle edge computing (VEC), as an application of mobile-edge computing (MEC) in vehicular scenarios, alleviates computation delay and reduces energy consumption by offloading vehicular tasks to roadside edge servers with greater computational capabilities, which makes it an effective solution for meeting the high computational demands of connected vehicles [3], [4]. Nevertheless, the storage capacity and computational power of edge server are limited, making it unrealistic to efficiently meet the offloading demands of all vehicles in a timely manner.

The computation and data storage capabilities of the cloud server are several orders of magnitude greater than those of edge servers, enabling it to provide sufficient resources for vehicular tasks. However, previous research on task offloading in MEC has rarely considered offloading tasks to the cloud server. This is primarily because the significant data transmission delay due to the greater distance between users and the cloud server makes it less feasible. However, the advent of the 6G network, with its high bandwidth and low delay [5], presents new opportunities for incorporating cloud computing into MEC systems [6], [7]. Thus, an increasing number of studies explore the integration of cloud computing with MEC. Wu et al. [8] and Ding et al. [9] employed a game theory approach to minimize delay and energy consumption in cloud–edge collaborative computing. Additionally, Pang et al. [10] and Chen et al. [11] focused on the task offloading problem with dependency constraints in cloud-edge collaboration. These studies demonstrate that combining the strong computing and storage power of cloud server with the proximity advantage of MEC server can better meet the diverse demands of applications regarding delay and energy consumption.

Though remarkable progress has been made, the dynamics of tasks and the heterogeneity of resources in the cloud–edge–vehicle architecture still pose three major challenges for existing methods, limiting further improvements in the efficiency of task offloading and resource allocation for VEC.

1) *High Mobility of Vehicle Users:* In previous research on task offloading, it is generally assumed that users remain

Received 31 August 2024; revised 20 November 2024 and 19 December 2024; accepted 26 December 2024. Date of publication 31 December 2024; date of current version 21 February 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62462002, and in part by the Research Supporting Project, King Saud University, Riyadh, Saudi Arabia, under Grant RSPD2025R585. *(Corresponding author: Min Chen.)*

Wenjing Xiao, Miaojiang Chen, and Junbin Liang are with the School of Computer, Electronics and Information and the Guangxi Key Laboratory of Multimedia Communications and Network Technology, Guangxi University, Nanning 530004, China (e-mail: wenjingx@gxu.edu.cn; mjchen_cs@gxu.edu.cn; liangjb@gxu.edu.cn).

Xin Ling is with the School of Computer, Electronics and Information, Guangxi University, Nanning 530004, China (e-mail: lingxin@st.gxu.edu.cn).

Salman A. Alqahtani is with the Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Riyadh 11574, Saudi Arabia (e-mail: salmanq@ksu.edu.sa).

Min Chen is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, and also with Pazhou Laboratory, Guangzhou 510330, China (e-mail: minchen@ieee.org).

Digital Object Identifier 10.1109/JIOT.2024.3524469

within the communication range of the base station. However, in IoV scenarios, intelligent vehicles travel at high speeds on highways, and the communication state between the vehicle and the base station constantly changes [12]. If the offloading or execution time of the vehicular task is too long, vehicles may leave the communication range of edge servers before completing their offloading tasks, leading to offloading failures.

2) *Multilayered Resource Architecture:* In the IoV system, edge server is generally located near vehicles, offering low communication delay and relatively strong computation capability, though with limited storage capability. Although cloud server has powerful storage and computation capabilities, it suffer from longer data transmission delay. Due to the heterogeneity of resources across the cloud, edge, and vehicles, maximizing the utilization of each layer's resources is a critical issue [13].

3) *Real-Time Decision-Making Requirement:* Since applications, such as autonomous driving and collision warning in the IoV, often involve the safety of vehicles and passengers, real-time decision-making is generally required. However, traditional machine learning decision algorithms often require a large number of iterations, resulting in high time complexity, which makes them unsuitable for real-time decision-making. Thus, to achieve low delay task offloading, it is necessary to adopt an online offloading and resource allocation algorithm for VEC.

To address the above challenges, we investigate task offloading and resource allocation for IoV, and propose an efficient and feasible solution. First, considering the mobility of vehicles, we propose a mobility-aware vehicle offloading architecture with cloud assistance, which adopts a vehicle movement detection method to predict which vehicles are likely to leave the communication range of the base station, thereby reducing the likelihood of task offloading failures. Second, we propose a layered proposal offloading strategy that allows each vehicle to autonomously decide whether to process tasks locally or propose offloading them to the edge server. For tasks proposed for offloading, the edge server prioritizes them based on predefined priority levels, selecting tasks sequentially until its storage capacity is fully utilized, the remaining tasks proposed for offloading will then be transmitted to the cloud server for processing. This strategy fundamentally differs from traditional vehicular network offloading frameworks, which typically rely on static, rule-based approaches, such as only based on the task size threshold or the distance between the vehicle and the server to determine whether to offload the task. Instead, our approach emphasizes dynamic and distributed decision-making, giving vehicles more flexibility and enabling vehicles to adjust their offloading behavior autonomously according to real-time conditions. By fully utilizing the computational resources at the local, edge, and cloud layers, this layered approach achieves a more balanced workload distribution, ultimately minimizing system delay and energy consumption. Finally, given the exceptional performance of multiagent deep reinforcement learning (MADRL) in real-time decision-making, we design a multiagent vehicle proposal offloading algorithm (MVPOA). The main contributions of this article are summarized as follows.

1) We design a cloud–edge–vehicle three-layer offloading architecture and model vehicle mobility. Additionally, we propose a mobility detection method to prevent task offloading failures caused by vehicles moving out of the communication range of the base station.
2) We propose a layered proposal offloading strategy that enables vehicles to fully utilize resources available at each layer of the cloud–edge–vehicle architecture.
3) We design a MVPOA that enables real-time decision-making and efficient resource allocation in dynamic and complex IoV systems.
4) We conduct extensive simulation experiments across various scenarios, and the results demonstrate that our proposed MVPOA exhibits outstanding performance in reducing system delay and energy consumption.

II. Related Work

A. Task Offloading for MEC

In recent years, task offloading in edge computing has garnered widespread attention and research. Li et al. [14] proposed an improved particle swarm genetic algorithm (IPSGA) to find the optimal offloading strategy, thereby minimizing the delay in task offloading. Wang et al. [15] proposed a task offloading scheme based on the alternating direction method of multipliers (ADMM) to obtain an approximate optimal solution by transforming the problem of minimizing system cost into a convex problem. Hong et al. [16] studied the problem of multihop cooperative computation offloading and routing, proposing a constraint-free mechanism integrated with potential game theory to enhance service quality. Liu et al. [17] utilized UAVs to assist ground vehicles with task offloading and proposed a second-order convex approximation-based continuous convex programming method to address the joint scheduling optimization problem for communication and computational resources. However, these traditional methods, such as dynamic programming, heuristic approaches, and convex optimization, often involve high computational complexity and require numerous iterations to achieve a local optimum [18]. These methods are not well suited for making real-time offloading decisions in dynamic and complex IoV systems.

B. Task Offloading Based on DRL

With the advancement of artificial intelligence, deep reinforcement learning methods have been employed to address task offloading and resource allocation problems in vehicular networks due to their powerful learning and decision-making capabilities [19]. Zhao et al. [20] developed a mobile-aware task offloading scheme to minimize the average response time and energy consumption in urban VEC systems and utilized the deep deterministic policy gradient (DDPG) algorithm to train the offloading strategy. He et al. [21] combined DDPG with prioritized experience replay (PER) and stochastic weight averaging (SWA) mechanisms to propose the PS-DDPG algorithm, aiming to improve the Quality of Experience

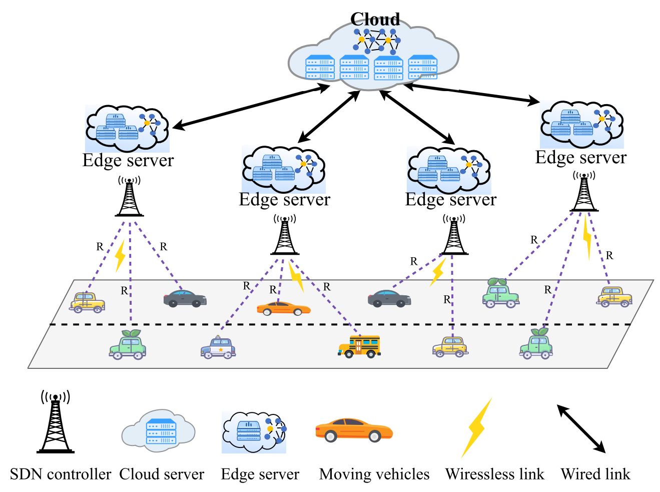


Fig. 1. Cloud–edge–vehicle offloading architecture.

(QoE) for vehicle task offloading. Tang et al. [22] decomposed vehicle tasks into multiple consecutive subtasks and proposed a novel dynamic offloading decision algorithm based on DDQN to optimize the total delay of subtasks with sequential dependencies. The above work is based on single agent learning algorithms. However, due to the high dynamism and decentralization in IoV, task offloading and resource allocation require multiple agents to interact and make informed decisions collectively, single-agent reinforcement learning algorithms struggle to learn effective strategies in such complex environments [18].

In addition, the enormous computational and storage resources of the cloud server have not been fully utilized in the aforementioned studies. In contrast to these works, we introduce cloud server for assisted computation in the IoV system and propose a novel MADRL algorithm to optimize the overall system's delay and energy consumption.

III. System Model

This section introduces the cloud–edge–vehicle system, mainly including its network model, communication model, computation model, and mobility perception model.

A. Network Model

We consider a cloud–edge–vehicle three-layer offloading system. Moreover, to enhance collaboration efficiency between vehicles and improve network management flexibility in the IoV system, we integrate software-defined networking (SDN) into the system, as shown in Fig. 1.

In a bidirectional road, multiple SDN controllers are deployed along one side. Each SDN controller is responsible for gathering service request information from vehicles, intelligently allocating computing resources, and providing services [23]. Each controller is equipped with an edge server that has a maximum data storage capacity of G and manages communication within a specific region with a maximum coverage range of R. In each region, there are K vehicles traveling on the road, represented by the set $\mathcal{V} = \{V_1, V_2, \ldots, V_i, \ldots, V_K\}$. We assume that vehicle V_i can only communicate with the SDN controller in its region via a wireless connection. The SDN controller is connected to the cloud server through optical fiber links. In this article, we focus on studying one region, and the situation in other regions is similar.

TABLE I
Notation and Definition

Notation	Definition
g_i	The channel power gain from vehicle V_i to SDN controller
d_i	The distance from vehicle V_i to SDN controller
p_i	The transmission power of vehicle V_i
R	Maximum communication range
B	The available bandwidth
D_i	The size of the task generated by vehicle V_i
C_i	The required CPU cycles to accomplish task M_i
H	Tower height of the SDN controller
T_i^{cover}	The communication coverage duration
a_i	Offloading decision of vehicle V_i
ω	Delay weight coefficient
η	Energy consumption weight coefficient
G	Maximum data storage capacity of edge server
ε_l	Local energy consumption parameter
ε_e	Edge server energy consumption parameter
ε_c	Cloud server energy consumption parameter
$T_{\max}$	Maximum tolerance time for each task
f_i^l, f_i^e, f_i^c	The computing resources that vehicle V_i/edge/cloud server allocats to task M_i
T_i^l, T_i^e, T_i^c	The computing delay of task M_i in local/edge/cloud processing
E_i^l, E_i^e, E_i^c	The computational energy consumption of task M_i in local/edge/cloud processing

We divide the system time into T equal time slots, denoted as $\mathcal{T} = \{1, 2, \ldots, t, \ldots, T\}$. Each vehicle is assumed to generate one compute-intensive task in each time slot, and the task arrivals follow a periodic deterministic arrival model, with tasks generated at regular intervals [24]. The task set is denoted as $\mathcal{M} = \{M_1, M_2, \ldots, M_i, \ldots, M_K\}$, where each task M_i is represented by a tuple $M_i = \{D_i, C_i\}$, with D_i representing the data size and C_i representing the number of CPU cycles required to process the task. The maximum tolerance time of a task is denoted as $T_{\max}$. The offloading decision for each task is represented by $a_i = \{0, 1, 2\}$, where $a_i = 0$ means the task M_i is processed locally, $a_i = 1$ means it is processed at the edge server, and $a_i = 2$ denotes offloading to the cloud server for processing. The important symbols used in this article are shown in Table I.

B. Communication Model

We assume that the upload link between the vehicles and the SDN controller is modeled as a flat Rayleigh fading channel, thereby ignoring the effects of channel interference. According to Shannon's formula, the communication rate r_i between vehicle V_i and the SDN controller can be expressed as follows:

$$r_i = B \log_2\left(1 + \frac{g_i d_i^{-\alpha} p_i}{N_0 B}\right) \tag{1}$$

where $g_i d_i^{-\alpha}$ represents the channel power gain between vehicle V_i and the SDN controller, d_i denotes the distance between vehicle V_i and the SDN controller, α is the path-loss exponent, B is the channel bandwidth, p_i represents the uplink transmission power of vehicle V_i, and N_0 denotes the noise power spectral density.

C. Computation Model

We assume each task is indivisible, meaning it can only be executed on either the local vehicle, the edge server, or the cloud server. In the following, we will provide a detailed description of the delay and energy consumption calculations for tasks under different execution methods.

1) Local Processing: If task M_i is executed locally, the local computation delay T_i^l can be expressed as

$$T_i^l = \frac{C_i}{f_i^l} \tag{2}$$

where f_i^l represents the computing resources that vehicle V_i allocats to task M_i.

The local computation energy consumption E_i^l for task M_i can be expressed as

$$E_i^l = \varepsilon_l \left(f_i^l\right)^2 C_i \tag{3}$$

where ε_l denotes the local energy consumption parameter, which depends on the CPU architecture [25].

2) Edge Server Processing: If task M_i is offloaded to the edge server for processing, the computation delay T_i^e for this process can be expressed as

$$T_i^e = \frac{D_i}{r_i} + \frac{C_i}{f_i^e} \tag{4}$$

where D_i/r_i represents the transmission delay of data D_i from the vehicle V_i to the edge server, and C_i/f_i^e is the processing delay of task M_i on the edge server, f_i^e denotes the computing resources that edge server allocats to task M_i. Since the amount of data for the result is usually small, the delay for receiving the result is negligible.

The computational energy consumption E_i^e for offloading task M_i to the edge server is expressed as

$$E_i^e = p_i \frac{D_i}{r_i} + \varepsilon_e \left(f_i^e\right)^2 C_i \tag{5}$$

where $p_i \cdot D_i/r_i$ represents the transmission energy consumption of data D_i from vehicle V_i to the edge server, p_i is the transmission power of V_i, $\varepsilon_e (f_i^e)^2 C_i$ denotes the processing energy consumption of task M_i on the edge server, and ε_e is the energy consumption parameter of the edge server.

3) Cloud Server Processing: If task M_i is offloaded to the cloud server for processing, the computation delay T_i^c for this process can be expressed as

$$T_i^c = \frac{D_i}{r_i} + \frac{D_i}{r_c} + \frac{C_i}{f_i^c} \tag{6}$$

where D_i/r_c is transmission delay of data D_i from the SDN controller to the cloud server, with r_c denoting the transmission rate, C_i/f_i^c is the processing delay of task M_i on the cloud server, and f_i^c represents the computing resources that cloud server allocats to task M_i.

The computational energy consumption E_i^c for this process is expressed as

$$E_i^c = p_i \frac{D_i}{r_i} + \varepsilon_c \left(f_i^c\right)^2 C_i \tag{7}$$

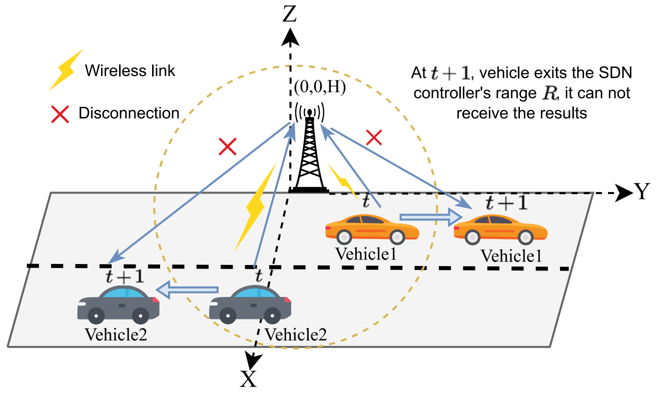

Z
Wireless link
Disconnection
(0,0,H)
At $t+1$, vehicle exits the SDN controller's range R, it can not receive the results
Y
t
$t+1$
Vehicle1
Vehicle1
$t+1$
t
Vehicle2
Vehicle2
X

Fig. 2. Vehicles out of range of SDN controller communication.

where $\varepsilon_c (f_i^c)^2 C_i$ is the processing energy consumption of task M_i on the cloud server, and ε_c denotes the energy consumption parameter of the cloud server. Compared to edge servers and vehicles, cloud servers possess greater computing power and a higher level of equipment integration, thus $f_i^c > f_i^e > f_i^l$ and $\varepsilon_c < \varepsilon_e < \varepsilon_l$ [26].

Therefore, the computation delay T_i^{comp} and the computation energy consumption E_i^{comp} of task M_i can be expressed as

$$T_i^{\text{comp}} = T_i^l \cdot \mathbf{1}_{\{a_i=0\}} + T_i^e \cdot \mathbf{1}_{\{a_i=1\}} + T_i^c \cdot \mathbf{1}_{\{a_i=2\}} \tag{8}$$
$$E_i^{\text{comp}} = E_i^l \cdot \mathbf{1}_{\{a_i=0\}} + E_i^e \cdot \mathbf{1}_{\{a_i=1\}} + E_i^c \cdot \mathbf{1}_{\{a_i=2\}} \tag{9}$$

where $\mathbf{1}_{\{\Omega\}}$ is an indicator function that equals 1 if condition Ω is satisfied, and 0 otherwise [27].

D. Mobility-Aware Model

Considering the mobility of vehicles in the system, a vehicle may move out of the maximum communication range of the SDN controller during the task offloading process. This would result in the final outcome not being returned to the vehicle via the SDN controller, as shown in Fig. 2. In this case, tasks cannot be processed effectively, leading to significant waste of time and energy. Therefore, for vehicles that are about to move out of the SDN controller's range, local processing is preferable. Although local computing capabilities are limited, for tasks with less stringent time requirements, such as predownloading multimedia files or batch processing traffic data, local processing can ensure the tasks are completed and avoid potential risks from communication interruptions. To prevent unreasonable offloading decisions, we propose a mobility detection method to estimate the communication coverage duration between the vehicle and the SDN controller.

In Fig. 2, we adopt a 3-D Euclidean coordinate system, assuming the SDN controller is located at $(0, 0, H)$, where H represents the height of the SDN controller's tower. Each vehicle on the road travels along the y-axis at a constant speed v. The position coordinate of vehicle V_i is $L_i(t) = (x_i(t), y_i(t), 0)$, where $y_i(t+1) = y_i(t) + v \times t$. The distance $d_i(t)$ between vehicle V_i and the SDN controller is calculated using the Euclidean formula

$$d_i(t) = \sqrt{x_i(t)^2 + y_i(t)^2 + H^2}. \tag{10}$$

The communication coverage duration $T_i^{\text{cover}}(t)$ between vehicle V_i and the SDN controller is defined as

$$T_i^{\text{cover}}(t) = \frac{R - d_i(t)}{v} \tag{11}$$

where the task offloading request of vehicle V_i has the opportunity to be accepted by the edge server only if $T_i^{\text{cover}}(t) \geq T_i^{\text{e}}(t)$. If not, the task will be processed locally on the vehicle.

E. Problem Description

Considering the limited computational resources of the vehicles and the data storage capacity constraints of the edge server, we jointly optimize the offloading decisions and resource allocation in IoV. Our goal is to minimize the computational delay of the vehicle terminal while ensuring low energy consumption of the entire system. The problem is formulated as follows:

$$\begin{aligned}
\text{P}_1: \quad & \text{minimize} \quad \sum_{t=1}^{T}\sum_{i=1}^{K}\left(\omega T_i^{\text{comp}}(t) + \eta E_i^{\text{comp}}(t)\right) \\
\text{s.t.} \quad & \text{C1:} \quad a_i \in \{0, 1, 2\} \quad \forall i \in K \\
& \text{C2:} \quad \omega \in (0, 1) \\
& \text{C3:} \quad \eta \in (0, 1) \\
& \text{C4:} \quad p^{\min} \leq p_i \leq p^{\max} \quad \forall i \in K \\
& \text{C5:} \quad f_l^{\min} \leq f_i^l \leq f_l^{\max} \quad \forall i \in K \\
& \text{C6:} \quad f_e^{\min} \leq f_i^e \leq f_e^{\max} \quad \forall i \in K \\
& \text{C7:} \quad f_c^{\min} \leq f_i^c \leq f_c^{\max} \quad \forall i \in K \\
& \text{C8:} \quad T_i(t) \leq T_{\max} \quad \forall i \in K \\
& \text{C9:} \quad \sum_{i \in K} D_i \cdot \mathbf{1}_{\{a_i=1\}} \leq G \quad \forall i \in K
\end{aligned} \tag{12}$$

where C1 indicates that each task can only be executed on one of the local, edge server, or cloud server layers. Constraints C2 and C3 denote the ranges for the delay and energy consumption weights, respectively. Constraints C4–C7 represent the constraints on transmission power and CPU frequency during task processing. Constraint C8 represents the computation delay for each task does not exceed the maximum tolerance time, and C9 ensures that the total data accepted by the edge server does not exceed its maximum storage capacity. In addition, since the cloud server has sufficiently abundant storage resources, we do not consider the limitation of the data storage capacity of this layer.

IV. Multiagent Deep Reinforcement Learning in IoV

Problem P_1 is a mixed-integer nonlinear programming problem, traditional machine learning methods exhibit high time complexity and low efficiency in solving such problems, making them unsuitable for application in the IoV environment [28]. However, the MADRL method, with its powerful learning and real-time decision-making capabilities, has become an effective tool for solving this complex optimization problem. In the multiagent environment, agents make decisions based on the joint actions of all agents, aiming to achieve a globally optimal decision rather than a locally optimal one. This approach enables vehicle users to autonomously explore dynamic environment and learn from it, thereby formulating better decision strategies [29]. Therefore, this article adopts MADRL to address task offloading and resource allocation in IoV. We design an MVPOA, which uses the actor–critic network architecture, where each vehicle in the system is considered as an agent. Next, we will first provide a detailed introduction to the state space, action space, and reward function of MVPOA. The network architecture and algorithm design of MVPOA will be detailed in the next section.

A. State Space

In MVPOA, the state $S_i(t)$ at time slot t for vehicle V_i is composed of the task $M_i(t)$, the vehicle's location $L_i(t)$, the communication coverage duration $T_i^{\text{cover}}(t)$, and the distance between the vehicle V_i and the SDN controller $d_i(t)$.

Therefore, the state $S_i(t)$ can be represented as

$$S_i(t) = \left\{M_i(t), L_i(t), T_i^{\text{cover}}(t), d_i(t)\right\} \tag{13}$$

where $M_i(t) = \{D_i, C_i\}$ and $L_i(t) = \{x_i(t), y_i(t), 0\}$.

B. Action Space

We classify actions into two categories: 1) vehicle-proposed action and 2) environment interaction action.

Vehicle-Proposed Action: This action is directly generated by the actor network. At time slot t, vehicle V_i inputs its state $S_i(t)$ into its actor network to obtain the action $\hat{A}_i(t)$, which is expressed as

$$\hat{A}_i(t) = \left\{\hat{a}_i(t), \hat{p}_i(t), \hat{f}_i^l(t), \hat{f}_i^e(t), \hat{f}_i^c(t)\right\} \tag{14}$$

where $\hat{a}_i(t)$ denotes the proposal offloading decision of vehicle V_i at time slot t, and $\hat{p}_i(t), \hat{f}_i^l(t), \hat{f}_i^e(t), \hat{f}_i^c(t)$ represent the transmission power, local computation resources, edge server computation resources, and cloud server computation resources that allocate to task $M_i(t)$, respectively, these values are continuous and fall within the range of $[-1, 1]$.

Environment Interaction Action: The environment interaction action is used by the agent to interact with the actual environment. This action is obtained by the above-mentioned vehicle-proposed action according to certain rules. The environment interaction action of vehicle V_i can be denoted as

$$A_i(t) = \left\{a_i(t), p_i(t), f_i^l(t), f_i^e(t), f_i^c(t)\right\}. \tag{15}$$

When $T_i^{\text{cover}}(t) < T_i^{\text{e}}(t)$, it means the communication coverage duration of vehicle V_i is insufficient to support task offloading, accordingly, we set $a_i(t) = 0$, indicating that task $M_i(t)$ is processed locally by vehicle V_i.

When $T_i^{\text{cover}}(t) \geq T_i^{\text{e}}(t)$, this indicates that vehicle V_i has sufficient communication coverage duration to support task offloading. If $\hat{a}_i(t) \leq 0$, it means vehicle V_i chooses to execute the task locally, accordingly, we set $a_i(t) = 0$. If $\hat{a}_i(t) > 0$, it indicates that the vehicle V_i proposes to offload the task to the edge server for processing. At this point, we need to assess the edge server's load status. If the total data volume of all the tasks proposed for offloading is less than the edge server's maximum storage capacity G, the SDN controller accepts the offloading proposals from all vehicles, we then set $a_i(t) = 1$, indicating that task M_i can be offloaded to the edge server for processing. If the total data volume of all the tasks proposed for offloading exceeds G, the SDN controller will decide which offloading proposals to accept based on the priority of the tasks (the priority of tasks will be discussed in detail in the next section) until the edge server's storage capacity G is reached. The remaining tasks not selected by the edge server will be transmitted by the controller to the cloud server for processing, accordingly, we set $a_i(t) = 2$, indicating that task M_i will be offloaded to the cloud server.

Since the values of $\hat{p}_i(t), \hat{f}_i^l(t), \hat{f}_i^e(t), \hat{f}_i^c(t)$ are in the range $[-1, 1]$, the values of $p_i(t)$, $f_i^l(t)$, $f_i^e(t)$, $f_i^c(t)$ need to be obtained using the following mapping formulas:

$$F = \left(\frac{x-b}{c-b}\right) \times (f_{\max} - f_{\min}) + f_{\min} \tag{16}$$

where $b = -1$, $c = 1$, and F represents the mapped results, which corresponds to $p_i(t), f_i^l(t), f_i^e(t), f_i^c(t)$, while x corresponds to $\hat{p}_i(t), \hat{f}_i^l(t), \hat{f}_i^e(t)$, and $\hat{f}_i^c(t)$. The values of $f_{\max}$ correspond to $p_i^{\max}$, $f_l^{\max}$, $f_e^{\max}$, and $f_c^{\max}$, and the values of $f_{\min}$ correspond to $p^{\min}, f_l^{\min}, f_e^{\min}$, and $f_c^{\min}$.

C. Reward Function

The core objective of reinforcement learning is to maximize the cumulative reward, reward function is usually related to the objective function [30]. We define a penalty for task M_i at time slot t as follows:

$$\text{penalty}_i(t) = \begin{cases} \left(T_i^{\text{comp}}(t) - T_{\max}\right) \times 10 & \text{if } T_i^{\text{comp}}(t) > T_{\max} \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

Therefore, the total reward of the system is expressed as

$$r(t) = -\sum_{i=1}^{K}\left(\omega T_i^{\text{comp}}(t) + \eta E_i^{\text{comp}}(t) + \text{penalty}_i(t)\right). \tag{18}$$

V. Network Architecture and Algorithm Design of MVPOA

Nonstationarity is a challenge faced in MADRL, however, existing work rarely considers this issue. Most approaches use decentralized critic networks, where each agent has its own independently trained critic network. This method ignores the strategies of other agents, which exacerbates the nonstationarity of the learning process and negatively impacts overall performance [31]. An effective method for dealing with nonstationarity is to use a centralized critic architecture, where all agents share critic network parameter and critic can collect observations and actions from all agents. This approach promotes collaboration among agents and reduces the nonstationarity of the learning process [32]. Therefore, considering the impact of nonstationarity, MVPOA adopts a centralized critic approach. The algorithm is built on the actor–critic framework, where the critic uses a value function to compute Q-value for each actor to evaluate the actions it selects, and the actor updates and optimizes its actions based on the critic's feedback.

A. Network Structure

The network structure of MVPOA consists of K distributed actor networks and a logically centralized critic network. Each vehicle deploys an actor network, and a centralized critic network is deployed on the SDN controller. The parameters of the K actor networks can be represented as: $\theta^\pi = \{\theta_1^\pi, \theta_2^\pi, \ldots, \theta_i^\pi, \ldots, \theta_K^\pi\}$, while the parameter of the centralized critic network is represented as θ^w.

For the ith actor network, its deterministic policy parameters can be represented as

$$\nabla_{\theta_i^\pi} J\left(\theta_i^\pi\right) \approx \mathbb{E}\left[\nabla_{\theta_i^\pi} \max Q\left(\mathcal{S}, \hat{\mathcal{A}}, S_i, \hat{A}_i; \theta^w\right)\right] \tag{19}$$

where $Q(\mathcal{S}, \hat{\mathcal{A}}, S_i, \hat{A}_i; \theta^w)$ is the Q-function used to calculate the Q-value, $\mathcal{S} = \{S_1, S_2, \ldots, S_i, \ldots, S_k\}$ is the set of states of K agents in the current time slot, and $\hat{\mathcal{A}} = \{\hat{A}_1, \hat{A}_2, \ldots, \hat{A}_i, \ldots, \hat{A}_k\}$ is the set of vehicle-proposed actions of K agents in the current time slot. S_i and $\hat{A}_i$ denote the state and vehicle-proposed action of agent V_i in the current time slot, respectively. The parameter θ_i^π of the ith actor network is updated using deterministic policy gradient descent, as described by (19).

For centralized critic network, its loss function is defined as

$$\mathcal{L}(\theta^w) = \mathbb{E}\left[\left(y - Q\left(\mathcal{S}, \hat{\mathcal{A}}, S_i, \hat{A}_i; \theta^w\right)\right)^2\right] \tag{20}$$

$$y = r + \gamma \times \max Q'\left(\mathcal{S}', \hat{\mathcal{A}}', S_i', \hat{A}_i'; \theta^{w'}\right) \tag{21}$$

where γ represents the discount factor, $\theta^{w'}$ is the parameter of the target critic network, $\mathcal{S}' = \{S_1', S_2', \ldots, S_i', \ldots, S_k'\}$ and $\hat{\mathcal{A}}' = \{\hat{A}_1', \hat{A}_2', \ldots, \hat{A}_i', \ldots, \hat{A}_k'\}$ represent the set of states and vehicle-proposed actions for K agents in the next time slot, respectively. S_i' and $\hat{A}_i{}'$ denote the state and vehicle-proposed action of agent V_i in the next time slot, respectively. The parameter θ^w of centralized critic network is updated using gradient descent, as described by (20).

The update of the target networks is represented as

$$\theta_i^{\pi\prime} = \tau\theta_i^\pi + (1-\tau)\theta_i^{\pi\prime} \tag{22}$$

$$\theta^{w\prime} = \tau\theta^w + (1-\tau)\theta^{w\prime} \tag{23}$$

where $\theta_i^{\pi\prime}$ is the target network parameter of the ith actor, and $\tau \in (0, 1)$ is the soft update parameter.

B. Algorithm Design

In MVPOA, the magnitude of the Q-value determines the priority for action selection, meaning that task offloading proposals with higher Q-values are prioritized by the edge server. Our algorithm primarily consists of two processes: 1) the vehicle decision-making process and 2) the training process. The MVPOA algorithm architecture is shown in Fig. 3, where Fig. 3(a) represents actor i and centralized critic networks, and Fig. 3(b) represents MVPOA algorithm training process.

1) Vehicle Decision-Making Process: The details of the vehicle decision-making process are described in Algorithm 1. First, initialize the parameters of each actor network, the parameters of the centralized critic network, and the vehicular network environment (lines 3 and 4). During each time step

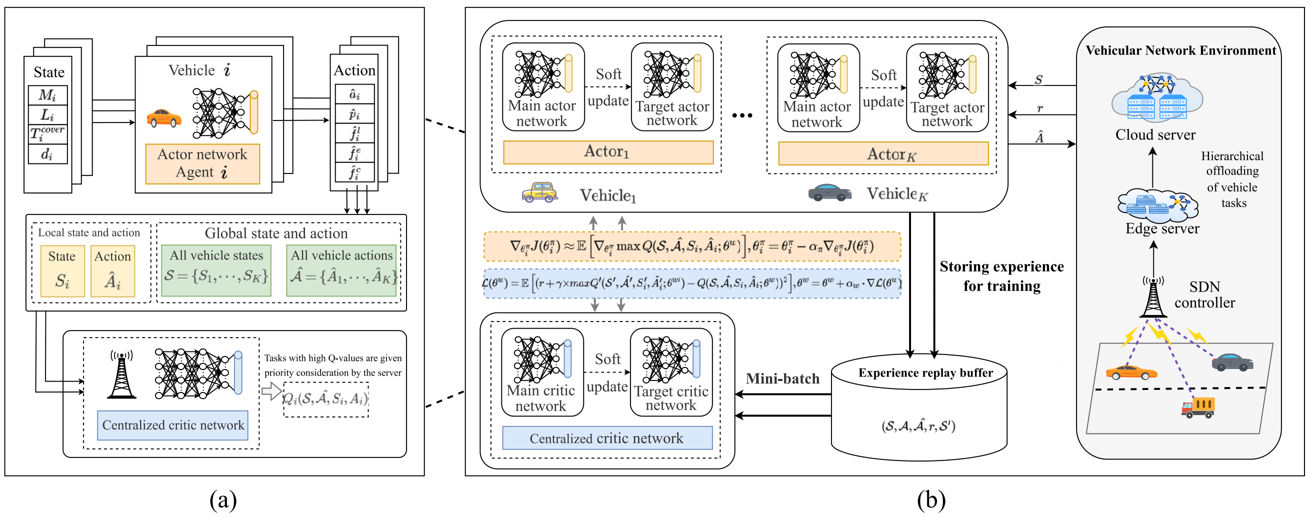

State
M_i
L_i
T_i^{cover}
d_i
Vehicle i
Actor network Agent i
Action
$\hat{a}_i$
$\hat{p}_i$
$\hat{f}_i^l$
$\hat{f}_i^e$
$\hat{f}_i^c$
Local state and action
State S_i
Action $\hat{A}_i$
Global state and action
All vehicle states $\mathcal{S}=\{S_1,\cdots,S_K\}$
All vehicle actions $\hat{\mathcal{A}}=\{\hat{A}_1,\cdots,\hat{A}_K\}$
Tasks with high Q-values are given priority consideration by the server
$Q_i(\mathcal{S},\mathcal{A},S_i,A_i)$
Centralized critic network
Main actor network
Soft update
Target actor network
Actor$_1$
Actor$_K$
Vehicle$_1$
Vehicle$_K$
$\nabla_{\theta_i^\pi}J(\theta_i^\pi)\approx\mathbb{E}\left[\nabla_{\theta_i^\pi}\max Q(\mathcal{S},\hat{\mathcal{A}},S_i,\hat{A}_i;\theta^w)\right],\theta_i^\pi=\theta_i^\pi-\alpha_\pi\nabla_{\theta_i^\pi}J(\theta_i^\pi)$
$\mathcal{L}(\theta^w)=\mathbb{E}\left[(r+\gamma\times maxQ'(\mathcal{S}',\hat{\mathcal{A}}',S_i',\hat{A}_i';\theta^{w\prime})-Q(\mathcal{S},\hat{\mathcal{A}},S_i,\hat{A}_i;\theta^w))^2\right],\theta^w=\theta^w+\alpha_w\cdot\nabla\mathcal{L}(\theta^w)$
Main critic network
Target critic network
Mini-batch
Storing experience for training
Experience replay buffer
$(\mathcal{S},\mathcal{A},\hat{\mathcal{A}},r,\mathcal{S}')$
$\mathcal{S}$
r
$\hat{\mathcal{A}}$
Vehicular Network Environment
Cloud server
Hierarchical offloading of vehicle tasks
Edge server
SDN controller
(a)
(b)

Fig. 3. MVPOA algorithm framework. (a) Actor i and centralized critic networks. (b) MVPOA algorithm training process.

t, vehicle V_i inputs its current state S_i into its actor network to obtain vehicle-proposed action $\hat{A}_i$. In order to ensure that the agent thoroughly explores potential optimal strategies, noise Ψ is added to each action (lines 8). If $\hat{a}_i \leq 0$, it indicates that vehicle V_i chooses to process the task locally, and the corresponding a_i is set to 0. If $\hat{a}_i > 0$, it means that vehicle V_i proposes to offload the task to the edge server. Considering vehicle mobility, some vehicles proposing task offloading may soon move out of the SDN controller's maximum communication range. This would result in the vehicles being unable to receive the processed results after offloading. Therefore, the communication coverage duration for vehicle V_i is predicted using (11). If vehicle V_i do not meet the communication time requirements, a_i is set to 0 (lines 11–14).

In the algorithm, we use the ϵ-greedy strategy to explore actions. We sample a value u uniformly from the interval $(0, 1)$. When the total data volume of tasks proposed for offloading by vehicles within the communication range exceeds the edge server's data capacity G, and if $u < \epsilon$, the SDN controller will randomly select tasks from these vehicles until the total accepted data volume reaches the edge server's capacity G, then the decision variable a for these selected vehicles will be set to 1. The remaining task offloading proposals will be rejected by the edge server and forwarded to the cloud server for processing, with the decision variable a for these vehicles set to 2 (lines 16–25). If $u \geq \epsilon$, we calculate the Q-values for all tasks proposed for offloading to the edge server. The Q-values will be used to determine the priority of task offloading to the edge server. For offloading requests from vehicles within communication range, the SDN controller will accept them in descending order based on their generated task Q-values, until the edge server's data storage capacity G is reached, then the decision variable a for these vehicles will be set to 1. The remaining tasks proposed for offloading will be transmitted to the cloud server for processing, and the decision variable a for these vehicles will be set to 2 (lines 26–29). If the total amount of task data that the vehicle proposes to offload is less than the capacity G of the edge server, then the edge server accepts all these tasks and sets their decision variable a to 1 accordingly (lines 32–34). At the end of each time step, the agent executes environment interaction action A_i, and stores the experience in a replay buffer for subsequent training process (lines 35–37).

2) *MVPOA Training Process:* The training process is shown in Algorithm 2. We randomly sample a mini-batch of size M from the experience replay buffer. Each sample consists of the state set $\mathcal{S}$, the vehicle-proposed action set $\hat{\mathcal{A}}$, the environment-interacted action set $\mathcal{A}$, the total reward r and the set of state $\mathcal{S}'$ for the next time slot of the K vehicles. First, we compute the vehicle-proposed action $\hat{A}'_{j,i}$ for the next time slot for each agent in every batch. Then, for each batch, we use $\hat{A}'_{j,i}$ to calculate the next time slot Q-values $Q'_{j,i}$ for each agent proposing offloading and select the maximum $Q'_{j,i}$ to calculate the target Q-values. If no vehicles propose offloading, a zero placeholder is used to calculate the target Q-value, ensuring normal feedback (lines 2–11). In lines 15–21, we calculate the Q-values for the current time slot, in lines 22–24, we compute the temporal-difference (TD) error for each sample in the entire batch and then take the average of these errors to obtain the batch's average loss, this loss is used to update the parameter of the centralized critic network via gradient descent. For the actor network, we update its parameter using the deterministic policy gradient method (lines 25–37).

VI. Experiment Results

In this section, we conduct simulation experiments to evaluate the effectiveness of MVPOA. First, we describe the experimental setup. Next, we compare and analyze MVPOA with three other approaches. Finally, we examine the impact of various parameters on MVPOA's performance.

A. Experiment Setup

We conduct a simulation experiment in the Python 3.9 environment with a cloud–edge–vehicular network system consisting of one edge server, one cloud server, and 30 vehicles. We set the maximum number $\boldsymbol{E}$ of episodes to 1000, with each episode consisting of five time steps, and each time step representing 5 s. In this system, each vehicle

Algorithm 1 MVPOA: Vehicle Decision-Making Process

```
1: Input: $\epsilon_0, \epsilon_{\min}$, actor network learning rate $\alpha_\pi$, critic network learning rate $\alpha_w$, reward discount factor $\gamma$
2: Output: Vehicle-proposed action $\hat{A}_i$ and environment interaction action $A_i$ for each agent
3: Initialize $\theta_i^\pi$ $(\forall i \in K)$, $\theta^w$; $\theta_i^{\pi\prime} \leftarrow \theta_i^\pi$ $(\forall i \in K)$, $\theta^{w\prime} \leftarrow \theta^w$
4: Initialize the environment for the IoV system
5: for episode = 1: E do
6:   Reset $S_i(t)(\forall i \in K)$, $\epsilon = \epsilon_{\min} + (\epsilon_0 - \epsilon_{\min}) \cdot e^{-\frac{\text{episode}}{E}}$
7:   for t = 1: T do
8:     Get $\hat{A}_i = \pi_i(S_i, \theta_i^\pi) + \Psi_i, \forall i \in K$
9:     The values of $p_i, f_i^l, f_i^e$, and $f_i^c$ are obtained from $\hat{A}_i$ using (16)
10:    Initialize task choice list $\mathcal{C}^L$, $Q$-value sorting list $\mathcal{Q}^S$ and $Q$-value index list $\mathcal{Q}^I$
11:    if $T_i^{\text{cover}} \geq T_i^e$ and $\hat{a}_i > 0, \forall i \in K$ then
12:      Append $i$ to $\mathcal{C}^L$
13:    else
14:      Set $a_i = 0$
15:    end if
16:    Sample $u \sim \text{U}(0, 1)$
17:    if len($\mathcal{C}^L$) > 0 and Sum($D_i, \forall i \in \mathcal{C}^L$) > $G$ then
18:      if $u < \epsilon$ then
19:        Initialize Datasize = 0
20:        while Datasize ≤ G do
21:          Randomly choose i from $\mathcal{C}^L$
22:          Set $a_i = 1$, Datasize = Datasize+ $D_i$
23:        end while
24:        Set $a_i = 2$
25:      else
26:        Calculate the $Q$-values according to $Q$-funtion $Q(\mathcal{S}, \mathcal{A}, S_i, \hat{A}_i; \theta^w), \forall i \in \mathcal{C}^L$ and append to $\mathcal{Q}^S$
27:        Sort the values in list $\mathcal{Q}^S$ in reverse order, and add their corresponding indices to list $\mathcal{Q}^I$
28:        Run similar lines 19 to 25 but choose i from $\mathcal{Q}^I$
29:      end if
30:    else
31:      if len($\mathcal{C}^L$) > 0 then
32:        Set $a_i = 1, \forall i \in \mathcal{C}^L$
33:      end if
34:    end if
35:    Execute environment interaction action $A_i$, observe total reward $r(t)$ and next state $S_i(t+1), \forall i \in K$
36:    Store $(S_i(t), \hat{A}_i(t), A_i, r(t), S_i(t+1)), \forall i \in K$ into buffer
37:    Update the state $S_i(t) = S_i(t+1), \forall i \in K$
38:  end for
39: end for
```

Algorithm 2 MVPOA: Training Process

```
1: Sample a random mini-batch of transitions $(\mathcal{S}, \mathcal{A}, \hat{\mathcal{A}}, r, \mathcal{S}')$ of size $M$ from replay buffer
2: for j = 1: M do
3:   Calculate $\hat{A}'_{j,i} = \pi_i(S'_{j,i}, \theta_i^{\pi\prime}), \forall i \in K$
4: end for
5: Initialize target $Q$-value list $\mathcal{Q}^T$, current $Q$-value list $\mathcal{Q}^C$
6: for j = 1: M do
7:   Initialize list $\mathcal{Q}'$
8:   if $\hat{a}'_{j,i} \geq 0, \forall i \in K$ then
9:     Calculate $Q'_{j,i} = Q(\mathcal{S}'_j, \hat{\mathcal{A}}'_j, S'_{j,i}, \hat{A}'_{j,i}; \theta^{w\prime})$ and append to $\mathcal{Q}'$
10:  end if
11:  $q' = Q(\mathcal{S}'_j, \hat{\mathcal{A}}'_j$, all zeros, all zeros; $\theta^{w\prime})$ if len($\mathcal{Q}'$) = 0 else max($\mathcal{Q}'$)
12:  $q^t = r_j + \gamma \cdot q'$
13:  if $a_{j,i} == 1, \forall i \in K$ then
14:    Append $q^t$ to $\mathcal{Q}^T$
15:    Calculate $q^c = Q(\mathcal{S}_j, \hat{\mathcal{A}}_j, S_{j,i}, \hat{A}_{j,i}; \theta^w)$ and append to $\mathcal{Q}^C$
16:  end if
17:  if len($\mathcal{Q}^C$)==0 then
18:    Append $q^t$ to $\mathcal{Q}^T$
19:    Calculate $q^c = Q(\mathcal{S}_j, \hat{\mathcal{A}}_j$, all zeros, all zeros; $\theta^w)$ and append to $\mathcal{Q}^C$
20:  end if
21: end for
22: Compute loss: $\mathcal{L}(\theta^w) = \frac{1}{\text{len}(\mathcal{Q}^T)} \sum_{j=1}^{\text{len}(\mathcal{Q}^T))} (\mathcal{Q}_j^T - \mathcal{Q}_j^C)^2$
23: Update critic parameters $\theta^w$:$\theta^w = \theta^w + \alpha_w \cdot \nabla \mathcal{L}(\theta^w)$
24: Update target critic $\theta^{w\prime} = \tau\theta^w + (1-\tau)\theta^{w\prime}$
25: for j = 1: M do
26:   Calculate $\hat{A}_{j,i} = \pi_i(S_{j,i}, \theta_i^\pi), \forall i \in K$
27: end for
28: for i = 1: K do
29:   Initialize actor loss list $\mathcal{Q}^{\text{loss}}$
30:   for j = 1: M do
31:     Initialize list $\mathcal{Q}^\pi$
32:     $\forall i \in K$, if $a_{j,i} = 1$ calculate $q = Q(\mathcal{S}_j, \hat{\mathcal{A}}_j, S_{j,i}, \hat{A}_{j,i}; \theta^w)$ and append to $\mathcal{Q}^\pi$
33:     Append $Q(\mathcal{S}_j, \hat{\mathcal{A}}_j$, all zeros, all zeros; $\theta^w)$ to $\mathcal{Q}^{\text{loss}}$ if len($\mathcal{Q}^\pi$) = 0 else append max($\mathcal{Q}^\pi$) to $\mathcal{Q}^{\text{loss}}$
34:   end for
35:   Compute gradient: $\nabla_{\theta_i^\pi} J(\theta_i^\pi) = -\frac{1}{\text{len}(\mathcal{Q}^{\text{loss}})} \sum_{j=1}^{\text{len}(\mathcal{Q}^{\text{loss}})} \nabla_{\theta_i^\pi} \mathcal{Q}_j^{\text{loss}}$
36:   Update actor parameters $\theta_i^\pi = \theta_i^\pi - \alpha_\pi \nabla_{\theta_i^\pi} J(\theta_i^\pi)$
37:   Update target actor networks $\theta_i^{\pi\prime} = \tau\theta_i^\pi + (1-\tau)\theta_i^{\pi\prime}$
38: end for
```

generates a task with a size in the range of [500, 1000] kb at each time step, the computational resources required for each task are randomly generated within the range of [500, 1500] Megacycles. When task M_i is processed locally, the allocated computing resources f_i^l are in the range of [0.05, 0.5] GHz, when processed on the edge server, f_i^e are in the range of [4, 8] GHz, and when processed on the cloud server, f_i^c are in the range of [10, 20] GHz.

Each actor network consists of three fully connected layers, including two hidden layers with 64 and 32 neurons, respectively, and uses ReLU activation functions, its output layer consists of five neurons with a tanh activation function. The centralized critic network also has three fully connected layers, with two hidden layers containing 512 and 128 neurons,

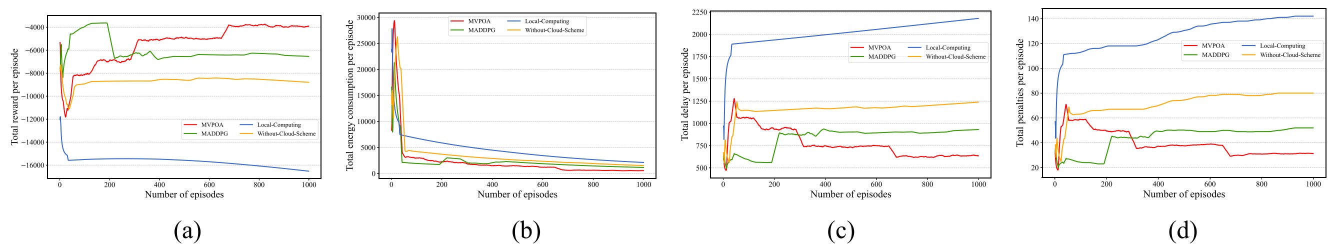


Fig. 4. Comparison of convergence performance. (a) Total reward of the system. (b) Total energy consumption. (c) Total delay of the system. (d) Total number of penalties.

TABLE II
SIMULATION PARAMETERS

Parameters	Value
The number of vehicles K	30
The channel power gain g_i	-30dB
The noise power spectral density N_0	4×10^{-20}W/Hz
The transmission power p_i	[0.5,2]W
Data storage capacity of the edge server G	10000Kbits
The bandwidth B	1.5MHz
The size of the task D_i	[500,1000]Kbits
The required CPU cycles of task C_i	[500,1500]megacycles
Each vehicle constant speed v	5m/s
Tower height of the SDN-controller H	10m
The transmission rate r_c	0.5Mbps
The path loss exponent α	4
Maximum tolerance time $T_{\max}$ for task	5s
Delay weight coefficient ω	0.7
Energy consumption weight coefficient η	0.3
ϵ_0 in ϵ-greedy strategy	1.0
$\epsilon_{\min}$ in ϵ-greedy strategy	0.01
Local energy consumption parameter ε_l	1.5×10^{-24}
Edge server energy consumption parameter ε_e	1.5×10^{-29}
Cloud server energy consumption parameter ε_c	1.5×10^{-31}

respectively, and employs ReLU activation functions. The learning rates for the actor and critic networks are set to 0.0001 and 0.001, respectively. We use the batch size M is 64 and set the discount factor γ to 0.99. The Adam optimizer is used to train both the actor and critic networks, other important parameters for this experiment are detailed in Table II.

To evaluate the performance of the proposed MVPOA, we compare it with three other different offloading strategies.

1) MADDPG: MADDPG uses the actor–critic architecture. In this scheme, task priority is determined by the time required for tasks to be offloaded to the edge server for processing, rather than by their Q-values.

2) Local-Computing: In this scheme, we use the MADDPG approach to allocate local resources, but all tasks can only be processed on the local vehicles.

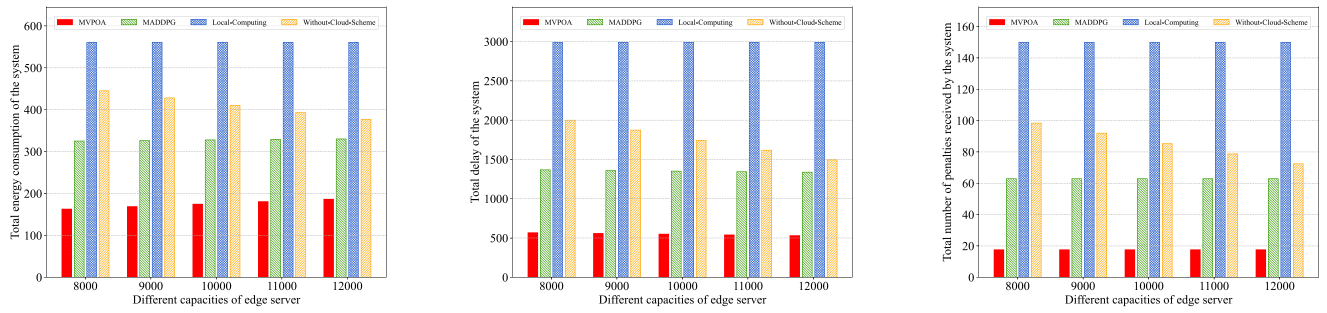
3) Without-Cloud-Scheme: In this scheme, we also use the above MADDPG to allocate local and edge resources, but there is no cloud server for assistance computation, tasks can only be offloaded to the edge server for processing. When the edge server data storage capacity is full, other tasks can only be processed locally.

B. Comparison With Other Algorithm Schemes

We set the rolling window size to 30 to perform rolling smoothing on the experimental data.

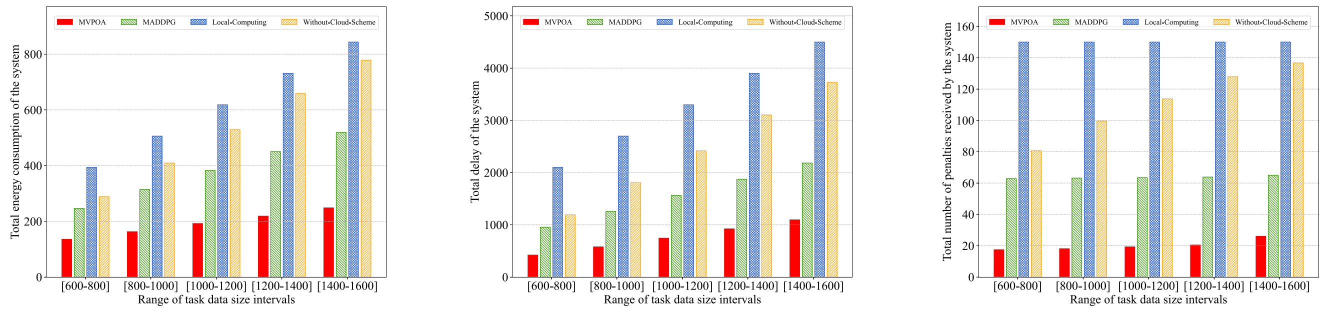
1) Comparison of Converged Values: Fig. 4 shows the comparison of the values of the different metrics for the four schemes. Fig. 4(a) illustrates the total reward during the training process for four different schemes. The results show that MVPOA achieves the highest converged reward, approximately 40.24% higher than MADDPG, 55.54% higher than the Without-Cloud-Scheme, and 76.33% higher than Local-Computing. Fig. 4(b) illustrates the total energy consumption over the training process for four different schemes. From the figure, it is evident that MVPOA has the lowest converged energy consumption, approximately 52.50% lower than MADDPG, 74.13% lower than Local-Computing, and 63.96% lower than Without-Cloud-Scheme. Fig. 4(c) shows the total delay during the training process for four different schemes. The results indicate that MVPOA achieves the lowest convergence delay, approximately 31.75% lower than MADDPG, 70.84% lower than Local-Computing, and 48.69% lower than Without-Cloud-Scheme. Fig. 4(d) shows the number of times penalties are incurred during the training process when tasks exceed the maximum tolerance time for four different schemes. From the figure, it can be concluded that MVPOA achieves the lowest convergence penalty value, which is approximately 39.81% lower than MADDPG, 77.96% lower than Local-Computing, and 60.88% lower than Without-Cloud-Scheme. Based on the above comparison, we observe that the Local-Computing scheme performs the worst among the four metrics. This is because all tasks in this scheme are executed locally, and local computational resources are relatively limited, followed by Without-Cloud-Scheme, as it lacks cloud server assistance for computation, when the edge server's storage is full, any remaining tasks must be processed locally, resulting in similarly poor performance. Additionally, in baseline MADDPG, its centralized critic network only calculates a global Q value for all agents to provide feedback, and the server determines the priority of the task according to the required offloading time, which limits its optimization ability. In contrast, in our MVPOA algorithm, the centralized critic network calculates an independent Q-value for each proposed offloading task, and the server preferentially offloads those tasks that have a greater impact on the system benefits based on the Q value of the task. Therefore, MVPOA is able to better balance the load, leading to superior performance.

2) Impact of Different Storage Capacities for Edge Server: We investigate the impact of different edge server storage capacities on the system's total delay, total energy consumption, and task penalties. The storage capacity of the edge server ranges from 8000 to 120 000 kb. The results are shown in Fig. 5. From Fig. 5(a)–(c), it can be observed that as the storage capacity of the edge server increases from 8000 to 120 000 kb, both the MVPOA and MADDPG schemes exhibit lower energy consumption, delay, and task



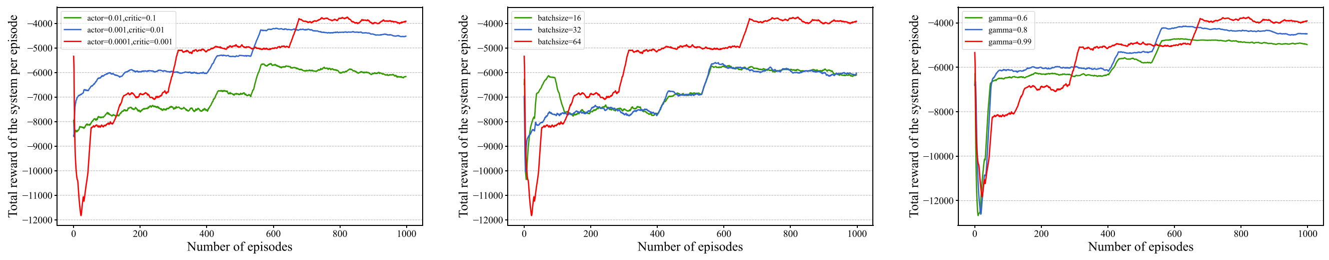
(a) (b) (c)

Fig. 5. Impact of different storage capacities on edge server. (a) Total energy consumption. (b) Total delay. (c) Total number of penalties.



(a) (b) (c)

Fig. 6. Impact of different tasks size. (a) Total energy consumption. (b) Total delay. (c) Total number of penalties.



(a) (b) (c)

Fig. 7. Different parameter analysis. (a) Different learning rate. (b) Different batch size. (c) Different discount factor.

penalties compared to the other two schemes, and they are largely unaffected by changes in storage capacity. This is because both MVPOA and MADDPG utilize cloud server for assisted computation. The Local-Computing scheme performs the worst, with high levels of energy consumption, delay, and task penalties, as all tasks must be processed locally with limited computational capabilities. It is also largely unaffected by changes in server storage capacity. On the other hand, the Without-Cloud-Scheme is significantly influenced by changes in server storage capacity because the edge server becomes its sole provider of advantageous resources, when the edge server's storage capacity increases, the energy consumption, delay, and task penalties decrease considerably. Overall, our proposed MVPOA performs the best among the four schemes.

3) Impact of Different Task Data Size: We set the edge server's storage capacity at 10 000 kb and investigate the impact of different task data sizes on the system's total energy consumption, delay, and the number of task penalties. The data size for each task increases from [600–800] to [1400–1600] kb, with the required CPU cycles correspondingly increasing from [600–800] to [1400–1600] megacycles, as shown in Fig. 6. From Fig. 6(a)–(c), it can be observed that as task data size and required CPU cycles increase, all four schemes exhibit higher energy consumption, delay, and task penalties. Among them, the Local-Computing scheme performs the worst due to the very limited local computing capacity. This leads to high energy consumption and delay during local processing, and most tasks cannot be completed within their maximum tolerable time. The Without-Cloud-Scheme also performs poorly because it lacks cloud server support for offloading. In contrast, our proposed MVPOA scheme consistently demonstrates superior performance in terms of energy consumption, delay, and task penalties compared to the other three schemes, reaffirming MVPOA's strong adaptability and superior effectiveness across various scenarios.

C. Parameter Analysis

To study the convergence of MVPOA, we analyze the impact of the following parameters on algorithm convergence: the learning rates of the actor and critic networks, the batch size, and the reward discount factor.

1) Impact of the Learning Rate on Performance: In Fig. 7(a), we investigate the impact of different learning rates for the actor and critic networks on the convergence of the reward values. The learning rates tested are: (actor = 0.01, critic = 0.1), (actor = 0.001, critic = 0.01), and (actor = 0.0001, critic = 0.001). From the figure, we can observe that setting the learning rate either too high will cause the algorithm to get trapped in local optima, resulting in suboptimal performance and poor results. Based on the results shown in the figure, it is evident that the algorithm achieves the highest reward convergence when the learning rates for the actor and critic are set to 0.0001 and 0.001, respectively. Therefore, we use the group of learning rates (actor = 0.0001, critic = 0.001) for simulation experiments.

2) Impact of the Batch Size on Performance: In Fig. 7(b), we investigate the impact of different batch sizes on the reward convergence of the algorithm. We conduct experiments with batch sizes of 16, 32, and 64. The figure clearly shows that when the batch size is set to 64, the reward convergence value is significantly higher than when the batch sizes are 16 and 32. This is because with smaller batch sizes, such as 16 and 32, the algorithm receives insufficient sample information during each training iteration, which negatively impacts the model's training effectiveness. Therefore, we choose a batch size of 64 for the simulation experiments.

3) Impact of Reward Discount Factor on Performance: We study the impact of different discount factors on the convergence of reward values. As shown in Fig. 7(c), when the discount factor is set to 0.99, the final converged reward value is higher than those achieved with discount factors of 0.6 and 0.8. This is because with a discount factor of 0.99, the agent tends to make decisions that favor long-term gains. Therefore, in order to obtain higher reward value, we set the discount factor to 0.99 in the simulation experiments.

VII. Conclusion

This article investigates the problem of task offloading and resource allocation in a cloud-assisted IoV system. Our objective is to minimize energy consumption and delay under various resource constraints. This optimization problem is formulated as a mixed-integer nonlinear programming problem, which is challenging to solve using traditional machine learning methods. Therefore, we propose a multiagent-reinforcement-learning-based vehicle proposal offloading algorithm (MVPOA). Each vehicle in the IoV system, after training, can learn an optimal offloading strategy and make real-time offloading decisions. We compare the performance of our proposed solution with other schemes in different scenarios. Simulation results demonstrate that our scheme outperforms other baseline algorithms in all evaluated metrics. This study focuses on a general offloading scenario with a single edge server and a single cloud server within a specific region. Future work will explore more complex situations involving multiple edge servers and cloud servers. We will investigate how to efficiently offload tasks to the edge server or cloud server in the next nearby region when a vehicle is about to drive away the communication range of the current region.

References

[1] W. Xiao, Y. Hao, J. Liang, L. Hu, S. A. Alqahtani, and M. Chen, "Adaptive compression offloading and resource allocation for edge vision computing," *IEEE Trans. Cogn. Commun. Netw.*, vol. 10, no. 6, pp. 2357–2369, Dec. 2024.

[2] R. Jin, J. Hu, G. Min, and J. Mills, "Lightweight blockchain-empowered secure and efficient federated edge learning," *IEEE Trans. Comput.*, vol. 72, no. 11, pp. 3314–3325, Nov. 2023.

[3] J. Lin, S. Huang, H. Zhang, X. Yang, and P. Zhao, "A deep-reinforcement-learning-based computation offloading with mobile vehicles in vehicular edge computing," *IEEE Internet Things J.*, vol. 10, no. 17, pp. 15501–15514, Sep. 2023.

[4] Z. Yu, J. Hu, G. Min, Z. Zhao, W. Miao, and M. S. Hossain, "Mobility-aware proactive edge caching for connected vehicles using federated learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 8, pp. 5341–5351, Aug. 2021.

[5] Y. Hao, J. Wang, D. Huo, N. Guizani, L. Hu, and M. Chen, "Digital twin-assisted URLLC-enabled task offloading in mobile edge network via robust combinatorial optimization," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 10, pp. 3022–3033, Oct. 2023.

[6] H. She, L. Yan, and Y. Guo, "Efficient end–edge–cloud task offloading in 6G networks based on multiagent deep reinforcement learning," *IEEE Internet Things J.*, vol. 11, no. 11, pp. 20260–20270, Jun. 2024.

[7] Y. Hao, L. Hu, and M. Chen, "Joint sensing adaptation and model placement in 6G fabric computing," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 7, pp. 2013–2024, Jul. 2023.

[8] L. Wu, P. Sun, Z. Wang, Y. Li, and Y. Yang, "Computation offloading in multi-cell networks with collaborative edge-cloud computing: A game theoretic approach," *IEEE Trans. Mobile Comput.*, vol. 23, no. 3, pp. 2093–2106, Mar. 2024.

[9] Y. Ding, K. Li, C. Liu, and K. Li, "A potential game theoretic approach to computation offloading strategy optimization in end-edge-cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 6, pp. 1503–1519, Jun. 2022.

[10] S. Pang, L. Hou, H. Gui, X. He, T. Wang, and Y. Zhao, "Multi-mobile vehicles task offloading for vehicle-edge-cloud collaboration: A dependency-aware and deep reinforcement learning approach," *Comput. Commun.*, vol. 213, pp. 359–371, Jan. 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366423004048

[11] L. Chen, J. Wu, J. Zhang, H.-N. Dai, X. Long, and M. Yao, "Dependency-aware computation offloading for mobile edge computing with edge-cloud cooperation," *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2451–2468, Oct.–Dec. 2022.

[12] D. Sun, Y. Chen, and H. Li, "Intelligent vehicle computation offloading in vehicular ad hoc networks: A multi-agent LSTM approach with deep reinforcement learning," *Mathematics*, vol. 12, no. 3, p. 424, 2024.

[13] J. Cai, W. Liu, Z. Huang, and F. R. Yu, "Task decomposition and hierarchical scheduling for collaborative cloud-edge-end computing," *IEEE Trans. Services Comput.*, vol. 17, no. 6, pp. 4368–4382, Nov./Dec. 2024.

[14] C. Li, L. Chai, K. Jiang, Y. Zhang, J. Liu, and S. Wan, "DNN partition and offloading strategy with improved particle swarm genetic algorithm in VEC," *IEEE Trans. Intell. Veh.*, early access, Dec. 25, 2023, doi: 10.1109/TIV.2023.3346506.

[15] X. Wang, S. Wang, X. Gao, Z. Qian, and Z. Han, "AMTOS: An ADMM-based multi-layer computation offloading and resource allocation optimization scheme in IoV-MEC system," *IEEE Internet Things J.*, vol. 11, no. 19, pp. 30953–30964, Oct. 2024.

[16] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial IoT–edge–cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2759–2774, Dec. 2019.

[17] Y. Liu et al., "Joint communication and computation resource scheduling of a UAV-assisted mobile edge computing system for platooning vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 8435–8450, Jul. 2022.

[18] S. Zhang, A. Liu, C. Han, X. Liang, X. Xu, and G. Wang, "Multiagent reinforcement learning-based orbital edge offloading in SAGIN supporting Internet of Remote Things," *IEEE Internet Things J.*, vol. 10, no. 23, pp. 20472–20483, Dec. 2023.

[19] J. Wang, J. Hu, J. Mills, G. Min, M. Xia, and N. Georgalas, "Federated ensemble model-based reinforcement learning in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 6, pp. 1848–1859, Jun. 2023.
[20] L. Zhao et al., "MESON: A mobility-aware dependent task offloading scheme for urban vehicular edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 4259–4272, May 2024.
[21] X. He, H. Lu, M. Du, Y. Mao, and K. Wang, "QoE-based task offloading with deep reinforcement learning in edge-enabled Internet of Vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 4, pp. 2252–2261, Apr. 2021.
[22] H. Tang, H. Wu, G. Qu, and R. Li, "Double deep Q-network based dynamic framing offloading in vehicular edge computing," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 3, pp. 1297–1310, May/Jun. 2023.
[23] Z. Zhang and F. Zeng, "Efficient task allocation for computation offloading in vehicular edge computing," *IEEE Internet Things J.*, vol. 10, no. 6, pp. 5595–5606, Mar. 2023.
[24] L. Li, W. Chen, and K. B. Letaief, "Wireless communications with hard delay constraints: Cross-layer scheduling with its performance analysis," *IEEE Internet Things J.*, vol. 11, no. 20, pp. 32540–32556, Oct. 2024.
[25] G. Wu et al., "Combining Lyapunov optimization with actor–critic networks for privacy-aware IIoT computation offloading," *IEEE Internet Things J.*, vol. 11, no. 10, pp. 17437–17452, May 2024.
[26] Z. Zhang, N. Wang, H. Wu, C. Tang, and R. Li, "MR-DRO: A fast and efficient task offloading algorithm in heterogeneous edge/cloud computing environments," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 3165–3178, Feb. 2023.
[27] G. Zhang, S. Ni, and P. Zhao, "Learning-based joint optimization of energy delay and privacy in multiple-user edge-cloud collaboration MEC systems," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1491–1502, Jan. 2022.
[28] J. Mills, J. Hu, and G. Min, "Multi-task federated learning for personalised deep neural networks in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 630–641, Mar. 2022.
[29] L. Wang, H. Liang, and D. Zhao, "Deep-reinforcement-learning-based computation offloading and power allocation within dynamic platoon network," *IEEE Internet Things J.*, vol. 11, no. 6, pp. 10500–10512, Mar. 2024.
[30] J. Eschmann, "Reward function design in reinforcement learning," in *Reinforcement Learning Algorithms: Analysis and Applications*. Cham, Switzerland: Springer, 2021, pp. 25–33.
[31] A. Shojaeighadikolaei, Z. Talata, and M. Hashemi, "Centralized vs. decentralized multi-agent reinforcement learning for enhanced control of electric vehicle charging networks," 2024, *arXiv:2404.12520*.
[32] G. Papoudakis, F. Christianos, A. Rahman, and S. V. Albrecht, "Dealing with non-stationarity in multi-agent deep reinforcement learning," 2019, *arXiv:1906.04737*.

Wenjing Xiao received the Ph.D. degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China, in 2023.

She is a Research Assistant Professor with the School of Computer, Electronic and Information, Guangxi University, Nanning, China. Her Google Scholar Citations reached 450+ with an H-index of 13. Her research interests include efficient artificial intelligence, edge intelligence, Internet of Things, and wireless network.

Xin Ling received the bachelor's degree in engineering from Guangxi University, Nanning, China, in 2022, where he is currently pursuing the M.E. degree with the School of Computer, Electronics, and Information.

His research interests include edge computing, Internet of Vehicles, and task offloading.

Miaojiang Chen received the Ph.D. degree in computer science from Central South University, Changsha, China, in 2023.

He is currently an Assistant Professor with the School of Computer and Electronic Information, Guangxi University, Nanning, China. His major research interests include deep reinforcement learning, Internet of Things, edge computing, and optimization.

Junbin Liang received the B.E. and M.S. degrees from Guangxi University, Nanning, China, in 2000 and 2005, respectively, and the Ph.D. degree from Central South University, Changsha, China, in 2010.

He was a Visiting Professor with the University of British Columbia, Vancouver, BC, Canada, from 2019 to 2020. He is currently a Professor with Guangxi University. His research interests include sensor–cloud systems, fog computing, and distributed computing.

Salman A. Alqahtani is currently a Full Professor with the Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia. He also serves as a Senior Consultant in computer communications, integrated solutions, and digital forensics for few development companies, and government sectors in Saudi Arabia. His main research interests include radio resource management for wireless and cellular networks (4G, 5G, IoT, Industry 4.0, and digital forensics).

Min Chen (Fellow, IEEE) is a Full Professor with School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. He is also the Director of Embedded and Pervasive Computing (EPIC) Lab at Huazhong University of Science and Technology.

He is the founding Chair of IEEE Computer Society Special Technical Communities on Big Data. He is a Fellow of IEEE and IET.