mathematics

MDPI

Article

Intelligent Vehicle Computation Offloading in Vehicular Ad Hoc Networks: A Multi-Agent LSTM Approach with Deep Reinforcement Learning

Dingmi Sun, Yimin Chen and Hao Li *

The School of Information Science and Engineering, Yunnan University, Kunming 650504, China; dmsun@mail.ynu.edu.cn (D.S.); chenyimin@ynu.edu.cn (Y.C.)
* Correspondence: lihao707@ynu.edu.cn

Abstract: As distributed computing evolves, edge computing has become increasingly important. It decentralizes resources like computation, storage, and bandwidth, making them more accessible to users, particularly in dynamic Telematics environments. However, these environments are marked by high levels of dynamic uncertainty due to frequent changes in vehicle location, network status, and edge server workload. This complexity poses substantial challenges in rapidly and accurately handling computation offloading, resource allocation, and delivering low-latency services in such a variable environment. To address these challenges, this paper introduces a "Cloud–Edge–End" collaborative model for Telematics edge computing. Building upon this model, we develop a novel distributed service offloading method, LSTM Muti-Agent Deep Reinforcement Learning (L-MADRL), which integrates deep learning with deep reinforcement learning. This method includes a predictive model capable of forecasting the future demands on intelligent vehicles and edge servers. Furthermore, we conceptualize the computational offloading problem as a Markov decision process and employ the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) approach for autonomous, distributed offloading decision-making. Our empirical results demonstrate that the L-MADRL algorithm substantially reduces service latency and energy consumption by 5–20%, compared to existing algorithms, while also maintaining a balanced load across edge servers in diverse Telematics edge computing scenarios.

Keywords: edge computing; computation offloading; resource allocation; deep reinforcement learning

MSC: 68T07; 68T20

check for updates

Citation: Sun, D.; Chen, Y.; Li, H. Intelligent Vehicle Computation Offloading in Vehicular Ad Hoc Networks: A Multi-Agent LSTM Approach with Deep Reinforcement Learning. *Mathematics* **2024**, *12*, 424. https://doi.org/10.3390/math12030424

Academic Editor: Faheim Sufi

Received: 20 December 2023
Revised: 17 January 2024
Accepted: 25 January 2024
Published: 28 January 2024

1. Introduction

With the rapid development of telematics technology, the computing needs for delay-sensitive tasks such as pattern recognition, image and video processing, target detection and route planning continue to increase. In this type of task, results must be achieved in the shortest possible time so that the vehicle can quickly make the next decisions. However, due to the limited computing resources of vehicles, task computations for computationally intensive and delay-sensitive tasks cannot be performed locally in a short time. To address this challenge, Mobile Edge Computing (MEC) provides us with a solution that offloads tasks to edge servers to fully utilize their powerful computing resources to perform task computations.

In MEC, the Roadside Unit (RSU) plays the role of providing richer computing and storage resources. With the continuous development of 5G technology, communication latency has been significantly reduced, making task offloading more practical and feasible.When studying task offloading solutions, traditional solutions usually only consider the situation where the user does not leave the base station's communication range. However, in the context of the Internet-of-Vehicles (IoV), intelligent vehicles move very quickly on the highway and reach speeds of up to 80 km/h, which brings new challenges for communication between vehicle and base station. Due to the short communication range of 5G communication, the communication time between vehicle and base station may be very limited, which directly limits the data transmission speed. In addition, if the vehicle has left the communication range of the current base station before the RSU completes the task calculation, the calculation results will not be returned to the vehicle in a timely manner. Therefore, in the Internet-of-Vehicles, traditional task offloading solutions may fail, and there is an urgent need to propose a new task shifting solution that combines vehicle movements.

The incorporation of Multi-Agent Deep Reinforcement Learning (MADRL) enhances the stability of decision-making processes within Telematics networks, ensuring a continuous refinement of policies for each vehicle to maximize overall rewards. Through the integration of MADRL with Mobile Edge Computing (MEC), our research addresses the challenges associated with computational offloading in vehicular edge networks. Our methodology carefully considers the mobility patterns of vehicles, a crucial factor significantly influencing task execution latency. Our primary emphasis is on formulating effective offloading strategies based on real-time observations. The adaptability of Deep Reinforcement Learning (DRL) to dynamic environments remains pivotal, particularly in the context of sophisticated multi-vehicle MEC systems.

To address these challenges, this paper introduces a reinforcement learning offloading scheme based on vehicle trajectory prediction. The main contributions are as follows:

(1) We model the computational offloading issue in multi-vehicle environments as a Multi-Agent DRL problem, concentrating on the selection of the most appropriate MEC servers for various vehicle tasks. Our model adeptly captures the real-time state of vehicles, encompassing location and task-specific information, thus enabling each vehicle to make well-informed offloading decisions.
(2) We introduce a novel distributed task collaboration framework using a multi-agent deep deterministic policy gradient (L-MADRL) network. This framework effectively tackles the combined optimization challenge of offloading and resource allocation, exhibiting notable performance improvements in task completion delays, vehicle energy costs, and load balancing.
(3) We takes into account the urgency of specific tasks. This article makes priority judgments on different types of tasks during the offloading decision generation process to ensure that high-delay-sensitive tasks are completed locally first and prevent potential catastrophic harm.

The remainder of the paper is structured as follows: Section 2 provides a review of related research; Section 3 outlines the system model, including network, task, and computation models; Section 4 introduces the L-MADRL scheme; Section 5 presents evaluation results; and Section 6 offers concluding remarks.

2. Related Work

Tian et al. [1] introduce a dynamic task offloading algorithm based on greedy matching, focusing on real-time responsiveness in vehicle networks. Alqarni et al. [2] employ GPU-based particle swarm optimization for high-performance vehicular edge computing, demonstrating efficient task offloading. Shu and Li [3] propose a joint offloading strategy using quantum particle swarm optimization in MEC-enabled vehicular networks, expanding the optimization horizon. Bozorgchenani et al. [4] address heterogeneous vehicular edge networks with online and off-policy bandit solutions for computation offloading. Materwala et al. [5] contribute a QoS-SLA-aware adaptive genetic algorithm for multi-request offloading in integrated edge-cloud computing for the Internet-of-Vehicles. Wang et al. [6] present a sustainable Internet-of-Vehicles system, employing an improved genetic algorithm for task offloading with a focus on sustainability.

Wang et al. [7] explore computation offloading in MEC-assisted V2X networks, introducing game theory to foster cooperative environments. Xu et al. [8] propose a distributed approach using game theory and fuzzy neural networks for task offloading in the Internet-of-Vehicles, emphasizing collective decision-making. Zhang et al. [9] extend game theory principles to symmetric MEC-enabled vehicular networks, showcasing its adaptability in different vehicular scenarios.Ashraf et al. [10] conduct an analysis of link selection challenges in underwater routing protocols, shedding light on the complexities of communication in submerged environments. Sundararajan et al. [11] enhance sensor linearity by implementing translinear circuits with piecewise and neural network models, contributing to improved sensor performance. Khan et al. [12] focus on location-based reverse data delivery between infrastructure and vehicles, presenting a novel approach to data transmission in vehicular networks.Nguyen et al. [13] explore cooperative task offloading and block mining in blockchain-based edge computing using multi-agent deep reinforcement learning, offering insights into collaborative computing paradigms. Lang et al. [14] contribute to the field with cooperative computation offloading in blockchain-based vehicular edge computing networks, emphasizing collaborative strategies. In a subsequent work, Lang et al. [15] extend their contributions to blockchain by introducing secure handover mechanisms, ensuring a comprehensive approach to secure computation offloading in vehicular edge computing networks.

However, traditional methods for vehicular network computation offloading often require manual adjustment of parameters and rules, making it challenging to dynamically adapt to constantly changing network and vehicle conditions. In contrast, reinforcement learning, as an automated learning approach based on intelligent agents and reward mechanisms, brings new possibilities to address the challenges of vehicular network computation offloading.

The survey landscape on vehicular task offloading, particularly focusing on reinforcement learning (RL) and deep reinforcement learning (DRL), is comprehensively explored in the literature. Liu et al. [16] present a thorough survey on vehicular edge computing and networking, providing an overarching view of the field. Ahmed et al. [17] delve into the classification, issues, and challenges of vehicular task offloading, offering a roadmap for future research. In the realm of RL and DRL, Liu et al. [18] provide an extensive survey, tracing the evolutionary path of reinforcement learning in vehicular task offloading. Moving to specific RL-based methods, Hazarika et al. [19] contribute with a DRL-based resource allocation approach, highlighting dynamic and learning-centric task offloading. Mirza et al. [20] focus on DRL-assisted delay-optimized task offloading in Automotive-Industry 5.0 based Vehicular Edge Computing Networks (VECNs), addressing the nuances of delay-sensitive applications. Jia et al. [21] introduce a learning-based queuing delay-aware task offloading approach, showcasing the adaptability of RL in addressing temporal considerations. Luo et al. [22] contribute by minimizing the delay and cost of computation offloading in vehicular edge computing, emphasizing efficiency. Binh et al. [23] present a reinforcement learning framework specifically tailored for optimizing delay-sensitive task offloading in Vehicular Edge-Cloud Computing environments. Shang et al. [24] and Vemireddy and Rout [25] explore deep learning and fuzzy reinforcement learning for energy-efficient task offloading, bridging machine learning paradigms with sustainability in vehicular networks. These works collectively represent the diverse and evolving landscape of RL and DRL applications in vehicular task offloading, addressing challenges and contributing to the intelligent, adaptive, and efficient resource allocation in vehicular networks.

In the above-mentioned studies, the task offloading models did not consider the nodes' movement speed, making them only suitable for static nodes in task offloading scenarios. Intelligent vehicles, traveling rapidly on highways, may leave the communication range of the current base station before completing a computing task, especially in the context of 5G communication. Therefore, these research methods become ineffective in the new scenario.

To address the aforementioned offloading challenges, Huang et al. [26] propose a vehicle speed-aware computing task offloading and resource allocation scheme based on multi-agent reinforcement learning, emphasizing the importance of real-time vehicular dynamics in edge computing networks. Zhao et al. [27] introduce MESON, a mobility-aware dependent task offloading scheme tailored for urban vehicular edge computing, demonstrating the significance of considering vehicle movement patterns for efficient offloading decisions. Additionally, trajectory prediction plays a pivotal role in task offloading strategies. Chen et al. [28] leverage multiagent deep reinforcement learning for dynamic avatar migration in AIoT-enabled vehicular metaverses, incorporating trajectory prediction as a crucial element. Zeng et al. [29] contribute a task offloading scheme that combines deep reinforcement learning and convolutional neural networks, specifically designed for vehicle trajectory prediction in smart cities. Yan et al. [30] propose a vehicle trajectory prediction method to enhance task offloading in vehicular edge computing, focusing on predicting the future positions of vehicles to optimize offloading decisions. These approaches collectively underscore the significance of integrating real-time vehicle dynamics and trajectory prediction into location-aware task offloading strategies, paving the way for more adaptive and context-aware vehicular edge computing solutions.

Table 1 summarizes the advantages and disadvantages of different algorithms.

Table 1. Comparison of different algorithms.

Method	Advantages	Shortcomings
Greedy Matching-Based Methods	Real-time responsiveness in vehicle networks.	May lack adaptability to diverse vehicular scenarios.
Heuristic	Can find relatively good solutions in large-scale problems.	Results may not be globally optimal.
Particle Swarm Optimization	1. High-performance vehicular edge computing. 2. Efficient task offloading.	Quantum computing may have limited practical applications.
Genetic Algorithm-Based Methods	Focus on sustainability.	The complexity of genetic algorithms may affect real-time responsiveness.
Blockchain-Based Methods	1. Exploration of collaborative computing paradigms. 2. Introduction of secure handover mechanisms.	Requires a lot of computing resources.
Game Theory-Based Methods	1. Foster cooperative environments and Showcase adaptability in different vehicular scenarios	1. Effectiveness may depend on specific network conditions. 2. Fuzzy neural network effectiveness not guaranteed.
Specific RL and DRL-Based Methods	1. Adapt to complex network relationships. 2. Able to effectively process large amounts of data in vehicle networks.	1. Long training time: Model training can be time-consuming. 2. High dependence on large amounts of labeled data.
Location-Aware Task Offloading	1. Task offloading scheme combining DRL and convolutional neural networks for vehicle trajectory prediction. 2. Vehicle trajectory prediction method to enhance task offloading.	Ineffective for rapidly moving vehicles in certain scenarios.

3. System Model

3.1. Model Architecture

This section focuses on presenting the system architecture of the vehicular edge computing task offloading solution based on distributed reinforcement learning.

In this research, we examine a tri-layered edge-cloud collaborative system architecture for Telematics, as illustrated in Figure 1. Within the Vehicular Edge Computing (VEC) framework, vehicles function as dynamic nodes of distributed computing, the vehicle side generates a large number of computing tasks in real time, including perception tasks, decision-making and planning tasks, vehicle control tasks, and data transmission tasks. Some of these computing tasks require a large amount of computing resources, such as target detection and lane detection, traffic signal recognition, path planning, etc.Road-Side Units (RSUs), distributed throughout the urban infrastructure, play a pivotal role in processing vehicular data, orchestrating computational offloading and caching strategies, and delivering essential services, including video transmission and traffic management. In this framework, each RSU is equipped with an edge computing server to handle computational tasks. It's important to note that each edge computing server associated with the RSUs has a service radius denoted by $'L'$ defining the geographical scope within which it provides its computational and communication services. The cloud layer, equipped with expansive computing and storage resources, oversees data from mobile nodes and edge servers, providing centralized governance and strategic decision-making. This "Cloud–Edge–End" architectural model adeptly caters to a variety of edge computing and communication requirements, offering efficient and dependable services.

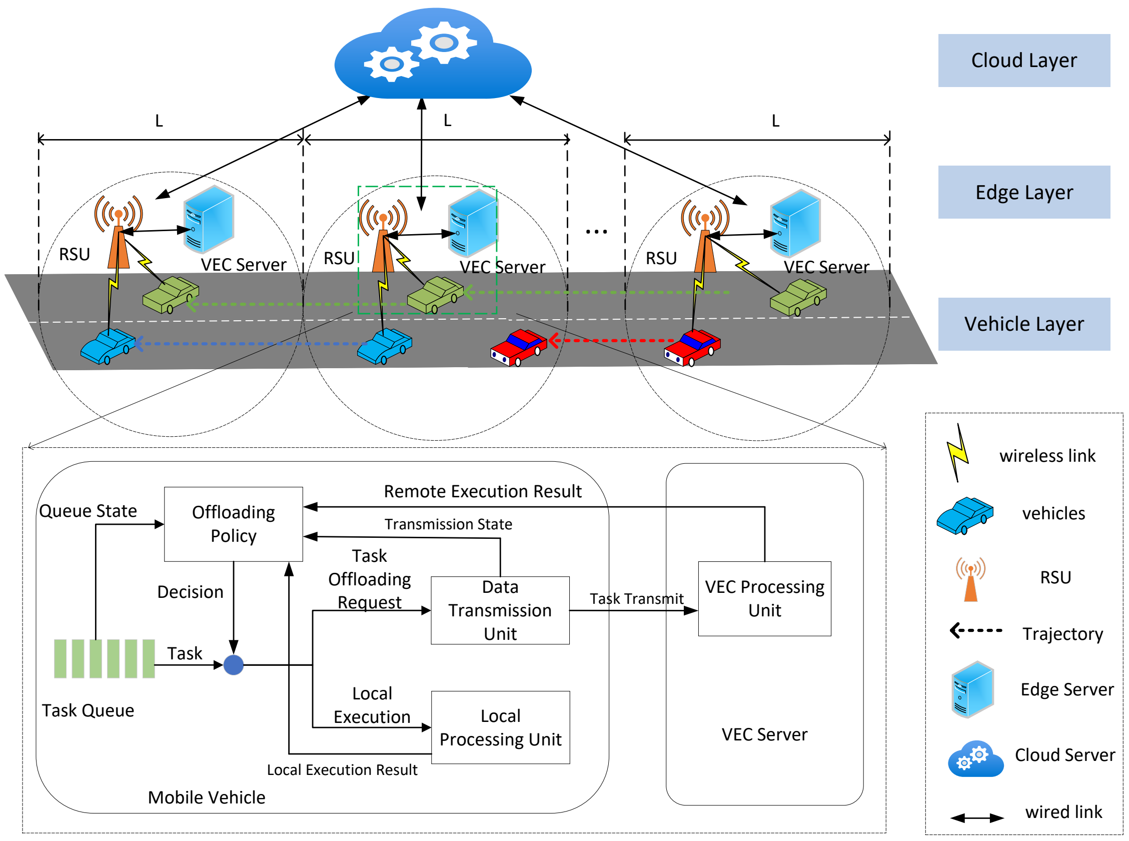


Figure 1. Architecture of computing task offloading in VEC network.

Consider a region with M edge servers, represented as $\{E_1, E_2 \dots E_M\}$. Each server, characterized by a coverage radius L_i, possesses computational capabilities defined by CPU processing frequency f_i^{mec}. Owing to the edge servers' finite computational resources, tasks awaiting immediate execution are queued. The queue length at each server is expressed as $q_i^{mec} = \sum_{t_i \in Q_i} c_k$, where c_k signifies the CPU cycles required for task t_k in the queue. Consequently, the computational state of each edge server is represented by $o_i^{mec} = \{f_i^{mec}, q_i^{mec}\}$.

At any given moment, the set of vehicles within the coverage of edge server E_i is $\{v_{i,1}, v_{i,2} \ldots v_{i,K}\}$. The current task of each vehicle $v_{i,K}$ is denoted as $t_{i,k} = \{d_{i,k}, c_{i,k}, t_{i,k}^{max}\}$, where $d_{i,k}$ denotes the task data size generated by the vehicle, $c_{i,k}$ represents the required CPU cycles for task completion, and $t_{i,k}^{max}$ is the task's maximum tolerable latency. Given the limited computing capacity of intelligent vehicles, unoffloaded tasks can be executed on-board. The computational capability of vehicle $v_{i,K}$ is indicated by its CPU processing frequency f_i^v, and its task queue length by q_i^v. Hence, the state of vehicle $v_{i,K}$ is captured by its computational state $\{f_i^v, q_i^v\}$ and task information $t_{i,k} = \{d_{i,k}, c_{i,k}, t_{i,k}^{max}\}$, collectively denoted as $o_i^{mec} = \{f_i^v, q_i^v, d_{i,k}, c_{i,k}, t_{i,k}^{max}\}$.

In this setting, the vehicle task offloading challenge is defined thus: at each time instance T, vehicles $\{v_{i,1}, v_{i,2} \ldots v_{i,K}\}$ within each edge server E_i's service range decide on offloading actions $\{a_{i,1}^T, a_{i,2}^T \ldots a_{i,k}^T\}$, where $a_{i,k}^T \in \{0, 1\}$ denotes a binary offloading decision (0 for local computation, 1 for offloading). The objective is to learn each edge server's offloading strategy, ensuring efficient computational resource distribution and minimizing overall costs in the task offloading system.

3.2. Network Communications Model

Firstly, we compute the communication delay induced by wireless transmission between vehicles and RSUs. In the wireless transmission process, we assume the RSU's position is fixed and denote it as $P_M^i = (X_M^i, Y_M^i)$. The position of vehicle v_i changes with time, and at time t, the position of vehicle v_i is denoted as $P_v^t = (x_v^t, y_v^t)$. Therefore, the distance between the vehicle and RSU at time t can be expressed as $D_{v,m}^t = \sqrt{|X_M^i - x_v^t|^2 + |Y_M^i - y_v^t|^2}$. In wireless communication, the delay incurred when intelligent vehicle v offloads tasks to RSU M in time slot t depends on the transmission rate of the uplink. The communication mode between the vehicle and RSU utilizes wireless communication, with a transmission rate R_i given by Equation (1):

$$R_i = B_i log_2(1 + \frac{p_i h_{v,m}^t}{\sigma^2}) \tag{1}$$

Here, p_i represents the transmission power of vehicle v, B_i denotes the channel bandwidth between the vehicle and RSU, $h_{v,m}^t$ represents the channel gain between the vehicle and RSU, and this study considers Rayleigh channel fading, where $h_{v,m}^t = A(\frac{l}{4\pi f D_{v,m}^t})^2$. A is the channel gain coefficient, l is the speed of light, f is the carrier frequency, and $D_{v,m}^t$ is the Euclidean distance between intelligent vehicle v and RSU m at time slot t. σ^2 represents the noise of the transmission. According to Shannon's theorem, the communication transmission rate R_i can be derived.

3.3. Computational Model

At time slot T, for the current computational task $t_{i,k}$ of vehicle $v_{i,K}$, two different computational modes are generated depending on its offloading decision $a_{i,K}^T$: executing the computation locally or offloading the computation.

3.3.1. Local Computing

(1) Service Latency for Local Execution
When the task requested by a vehicle is executed locally, the vehicle does not need to upload the task to the edge server and can simply process it using local computational resources. In the local computation mode, task $t_{i,k}$ enters the task queue $P_{i,k}$ of vehicle $v_{i,K}$ and executes the tasks in the queue step by step according to the First-In-First-Out (FIFO) queuing rule.

Therefore, for vehicle $v_{i,K}$ and its generated service request $t_{i,k} = \{d_{i,k}, c_{i,k}, t_{i,k}^{max}\}$, we can calculate the latency for the vehicle to fulfill its service request locally by Equation (2):

$$T_i^{loc} = T_{i,k}^{wait} + \frac{C_i(t)}{f_i^v} \tag{2}$$

In this framework, $T_{i,k}^{wait}$ signifies the waiting delay within the task queue. $C_i(t)$ denotes the CPU cycles required for the execution of the task, and f_i^v represents the inherent computational capacity of the vehicle.

(2) Energy Consumption for Local Execution
The local computational energy consumption can be modeled as Equation (3)

$$E_i^{loc} = P_i^{loc} * T_i^{loc} \tag{3}$$

Here, P_i^{loc} is the power consumption of the device when executing the task locally, and T_i^{loc} is the duration of executing the task. Therefore, the energy consumption of local computation is denoted as E_i^{loc}.

3.3.2. Offloading Calculations

Latency

(1) Service Latency for Offloading to Edge Server
When offloading a vehicle's service requirements to an edge server for processing, it is crucial to consider multiple latency factors. These include: (1) Waiting Delay: Caused by waiting for a free wireless channel. (2) Transmission Delay: Incurred in uploading the service request from the local system to the edge server. (3) Queue Waiting Delay: At the edge server while awaiting task execution. (4) Execution Delay: For processing the service request on the edge server.The time required for a vehicle to upload a computing request to an edge server can be calculated using Formula (4):

$$T_i^{tra} = \frac{d_i}{R_i} \tag{4}$$

where d_i represents the size of the computational task generated by the *i-th* vehicle and R_i signifies the transmission rate of the task uploaded to the RSU.
The execution delay for processing vehicle tasks at the edge server is computed by Equation (5)

$$T_i^{mec} = \frac{C_i}{f_i^{mec}} \tag{5}$$

Here C_i denotes the number of CPU cycles required for the vehicle task and f_i^{mec} denotes the current computational power of the edge server.
Consequently, if the vehicle's computational service is offloaded to the edge server for execution, the resulting service offloading delay is $T_i^{off-mec}$, which can be calculated using Equation (6).

$$T_i^{off-mec} = T_i^{wait-v} + T_i^{tra} + T_i^{mec} + T_i^{wait-q} \tag{6}$$

Here T_i^{wait-v} represents the waiting delay of the vehicle while waiting an idle channel, T_i^{tra} denotes the transmission delay of the vehicle task, T_i^{mec} signifies the processing delay of the vehicle task at the edge server, and T_i^{wait-q} is the queuing delay of the vehicle task at the edge server.

(2) Service Latency for Offloading to Cloud Server
In accordance with the 5G distributed vehicular network edge computing model outlined in Section 3.1, if a user's service requirement necessitates execution in the cloud, the service input parameters are initially transmitted from the user's vehicle to the edge layer via a wireless channel. Subsequently, these parameters are for-

warded to the cloud for processing through a wired channel by the respective edge computing node. Given the substantial computational capabilities of cloud-based servers, the processing delay is relatively negligible compared to the transmission delay. Consequently, when a user v_i offloads service $t_{i,k}$ for execution in the cloud, the resulting service delay primarily comprises three components: the waiting delay for a free wireless channel, the transmission delay for uploading input parameters to the edge computing node, and the round-trip time (RTT) involved in transmitting data between the edge computing node and the cloud server. The calculation formula is as Equation (7):

$$T_i^{off-cloud} = T_i^{wait-v} + T_i^{tra} + RTT \tag{7}$$

Due to the significant geographical separation between cloud servers and edge computing nodes, the incurred time delay in both forwarding input parameter data from the edge computing node to the cloud and returning the processed service results from the cloud tends to be similar. Importantly, this delay remains unaffected by the data volume of the input parameters. As a result, the Round-Trip Time (RTT) can be expressed using Formula (8):

$$RTT = 2t_{off}^{cloud} \tag{8}$$

Here t_{off}^{cloud} represents the delay incurred in transferring data from the edge server to the cloud.

Energy Consumption

(1) Service Energy Consumption for Offloading to Edge Server
The energy consumption associated with offloading services to the edge server is modeled using Equation (9):

$$E_i^{off-mec} = E_i^{tra} + E_i^{mec} \tag{9}$$

The energy consumption for the service offloaded to the edge server is calculated as detailed in Equation (9). In this equation, E_i^{tra} signifies the energy consumption required to transfer the computation task to the edge server, while E_i^{mec} denotes the energy consumption generated by the task computed at the edge server, Thus, the total energy consumption resulting from offloading the computation is expressed as $E_i^{off-mec}$.

(2) Service Energy Consumption for Offloading to Cloud Server
The energy consumption associated with offloading the service to the cloud server is modeled using Equation (10):

$$E_i^{off-cloud} = E_i^{tra} + E_i^{cloud} + E_i^{RTT} \tag{10}$$

The energy consumption for the service offloaded to the cloud server is calculated as detailed in Equation (10). Here, E_i^{tra} represents the energy consumption required to transfer the computation task to the edge server, E_i^{cloud} denotes the energy consumption generated by the task computed at the cloud server, and E_i^{RTT} signifies the energy consumption arising from offloading the computation.

3.3.3. Load Calculation

In scenarios where achieving optimal latency and minimizing energy consumption are of utmost importance, the simultaneous assignment of a large number of computational offloading tasks to a single edge server can result in an excessive workload for that specific server. Hence, we introduce the concept of the load balancing rate, denoted as L_n, for the edge server at time t. This factor plays a pivotal role in collaborative offloading, ensuring that edge servers can efficiently and effectively process offloading tasks.

To quantitatively assess the disparity between the load of each individual server and the average load across all edge servers, this study utilizes the load balancing rate as a metric for evaluating the load distribution among edge servers. In this context, To quantify the degree of deviation between each server's load and the overall average load, this paper employs the load balancing rate as a metric to evaluate the load balance of edge servers. In this context, $\bar{x}$ represents the average load of all edge servers, x_i signifies the load of the *i-th* edge server, and the overall system load balancing rate L_n is computed using Formula (11).

$$L_n = \frac{1}{N}\sum_{i=1}^{N}\frac{|x_i - \bar{x}|}{\bar{x}} \tag{11}$$

This equation quantifies the degree of load balance across the edge servers, crucial for achieving optimal system performance.

3.4. Problem Definition

In a Mobile Edge Computing (MEC) system, the decisions related to distributed offloading and resource allocation play pivotal roles. Mobile users base their offloading decisions on local information, guiding MEC servers in optimizing resource allocation to minimize the overall cost. This study considers control variables, including execution delay, energy consumption, and load balancing for mobile users, and establishes a utility function. The utility function is carefully crafted to strike a balance between the execution time and energy consumption of mobile users while also considering server load. With the requirement of meeting maximum tolerable delay for each task, the description of Problem Q_1 is as Equation (12):

$$Q_1 : min \sum_{i=1}^{N}\sum_{j=1}^{M}(Cost_T + Cost_E + Cost_L) \tag{12}$$

Subject to the following constraints:

$$X_{i,j}^{i} \in \{0,1\}, \forall i \in V, \forall j \in M \tag{12a}$$

$$T_i^{off}, T_i^{loc}(t) \le T_i^{max} \tag{12b}$$

$$0 \le p_i \le P_{max}, \forall i \in N \tag{12c}$$

$$0 \le f_i^{v} \le F_{max}^{mec}, \forall i \in N \tag{12d}$$

$$\alpha_{i,j}(t) + \beta_{i,j}(t) + \gamma_{i,j}(t) = 1 \tag{12e}$$

$$0 \le \alpha_{i,j}(t), \beta_{i,j}(t), \gamma_{i,j}(t) \le 1 \tag{12f}$$

Constraint (12a): A task can only be executed either locally (0) or on an edge server (1), where 0 denotes local execution and 1 signifies offloading computation to the server.

Constraint (12b): Regardless of whether the computation task is executed locally or on an edge server, the task's completion time must not exceed the maximum tolerable delay.

Constraint (12c): The power consumption of the vehicle must remain within the limits defined by P_{max}.

Constraint (12d): The computational resources allocated to mobile vehicle i must not exceed the maximum computing resources available from the MEC, denoted as F_{max}^{mec}.

Constraint (12e): The sum of the weights for delay, energy consumption, and load balancing must equal 1.

Constraint (12f): The weights for delay, energy consumption, and load balancing should fall within the range [0, 1].

For computing tasks in vehicular networks, the total processing delay and energy consumption can be described by Equations (13) and (14) as follows:

$$T_n = \left(1 - X_{i,j}^{i}\right)T_i^{loc} + X_{i,j}^{i}T_i^{off}, \forall n \in N \tag{13}$$

$$E_n = \left(1 - X_{i,j}^{i}\right) E_i^{loc} + X_{i,j}^{i} E_i^{off},\ \forall n \in N \tag{14}$$

The overall system utility function is collectively determined by delay, energy consumption, and load. Its calculation formula is presented in Equation (15) as follows:

$$R_n = \alpha_{i,j}(t)\left(\frac{T_i^{loc} - T_n}{T_i^{loc}}\right) + \beta_{i,j}(t)\left(\frac{E_i^{loc} - E_n}{E_i^{loc}}\right) + \gamma_{i,j}(t) L_n \tag{15}$$

The primary challenge lies in enabling mobile users to make decentralized decisions to minimize their costs. The offloading decision and resource allocation among different mobile users pose NP problems. Furthermore, various variables related to mobile users' offloading decisions and resource allocation are interconnected, rendering Problem Q_1 an NP problem. Additionally, as the number of mobile users increases, the solution space for Problem Q_1 grows exponentially. This paper aims to optimize the selection of offloading strategies to maximize available computing resources while minimizing costs for mobile users.

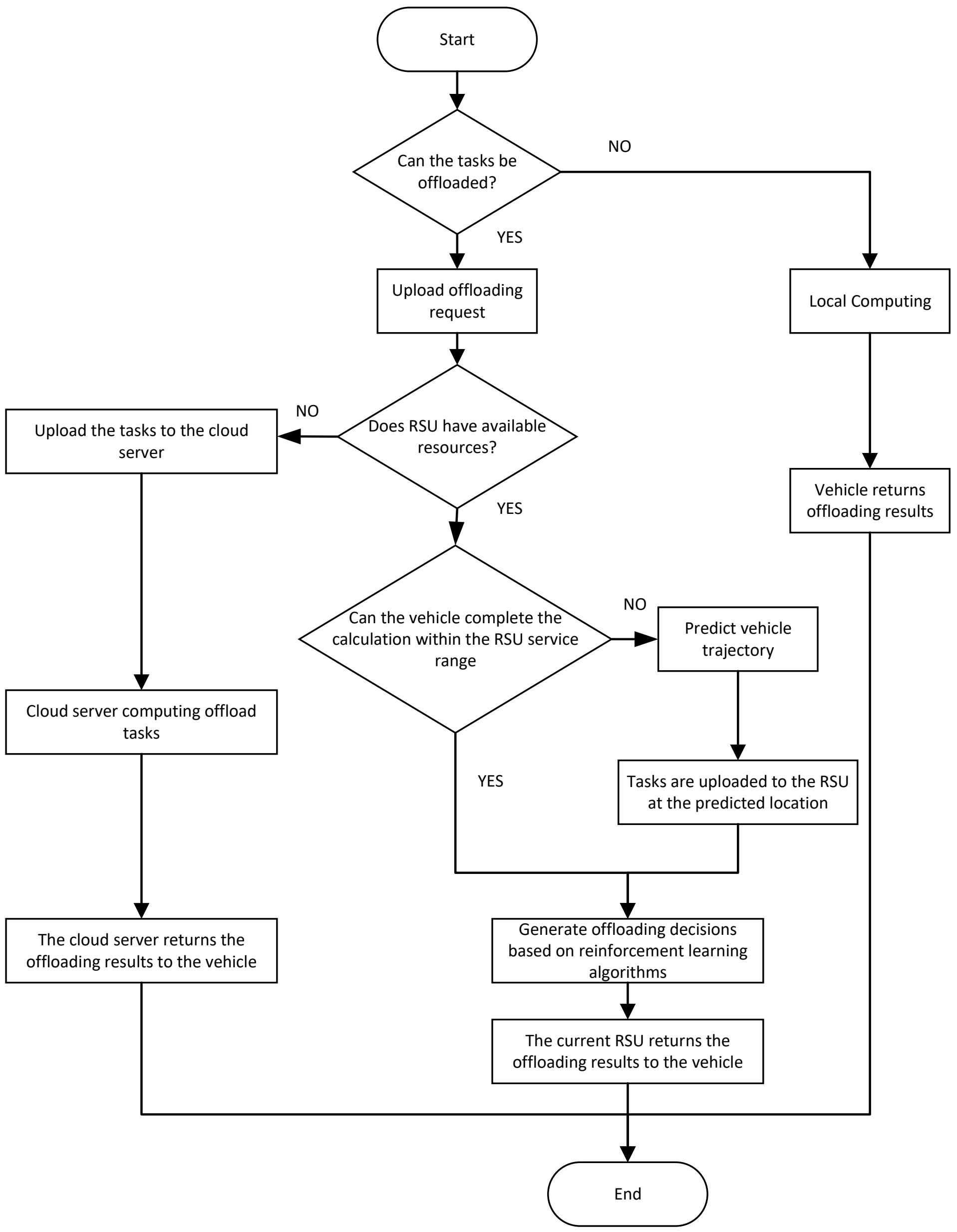
To address the challenges of computational offloading and resource allocation, this paper introduces an approach based on multi-agent strategies called L-MADRL. In contrast to value-based reinforcement learning methods, L-MADRL utilizes an Actor-Critic architecture, effectively handling continuous control problems. The following sections provide a detailed description of the L-MADRL algorithm and its specific structural components.

4. DRL-Based Vehicular Computation Offloading

In this section, We proposes a mobile offloading method for Internet-of-Vehicles based on trajectory prediction using reinforcement learning. Figure 2 illustrates the System flow chart.

First, when a computational task from a vehicle arrives, an assessment is made regarding the offloading feasibility of the task. Tasks sensitive to high latency, such as those involving control of vehicle direction or acceleration, cannot be offloaded and must be completed locally to avoid catastrophic consequences. If the vehicle's computational task is suitable for offloading, an offloading request is submitted. The current Roadside Unit (RSU) is checked for available computing resources. If the current RSU lacks sufficient computing resources, the task is sent to a cloud server for offloading. The cloud server performs the offloading task, and upon completion, the offloading results are returned to the vehicle. If the current RSU has sufficient computing resources, the dwell time of the vehicle within the current RSU's service range is calculated. If the vehicle can complete the offloading within this range, an offloading decision is generated using a reinforcement learning algorithm, and the current RSU returns the offloading results to the vehicle. If the vehicle cannot complete the offloading within the current RSU's service range, the Long Short-Term Memory (LSTM) algorithm is utilized to predict the vehicle's trajectory. Based on the predicted location of the vehicle at the next time step, the computational task is sent to the RSU server at that location. Within the service range of the predicted RSU, an offloading decision is generated using reinforcement learning, and the final decision is returned to the vehicle.

We introduce a distributed method called L-MADRL, specifically designed to address the service offloading problem within the Internet-of-Vehicles (IoV) edge computing environment. L-MADRL combines the LSTM and MADDPG algorithms to provide an effective solution. We begin by presenting an overview of the L-MADRL method, followed by a detailed explanation of its components. Figure 3 illustrates the framework of the L-MADRL algorithm.



Start
Can the tasks be offloaded?
NO
YES
Upload offloading request
Local Computing
Does RSU have available resources?
NO
YES
Upload the tasks to the cloud server
Vehicle returns offloading results
Can the vehicle complete the calculation within the RSU service range
NO
Predict vehicle trajectory
Cloud server computing offload tasks
YES
Tasks are uploaded to the RSU at the predicted location
The cloud server returns the offloading results to the vehicle
Generate offloading decisions based on reinforcement learning algorithms
The current RSU returns the offloading results to the vehicle
End

Figure 2. Task offloading process in Vehicular Edge Computing.

The input layer plays a crucial role in handling the state set of the moving user at a given time, denoted as $S(t)$. This layer captures essential information, including S1: the task queue of vehicles, S2: wireless availability for each communication link, S3: current resources of each Multi-Access Edge Computing (MEC) server, S4: the precise location and speed of vehicles. The position information of vehicles first undergoes a two-layer LSTM network to predict the future position at subsequent time intervals. Variables x_1, $x_2...x_{t-1}$ represent the historical trajectories of the vehicle, while x_t denotes the predicted trajectory for the vehicle at the next time step obtained through the LSTM network.The predicted location information is then incorporated as part of the state information for reinforcement learning, facilitating better decision-making and resource allocation. After collecting this state information data, it seamlessly transitions to the MADDPG layer. The MADDPG layer continuously trains the agents with the objective of optimizing latency, energy consumption, and load balancing, until the optimal computation offloading and resource allocation strategy is generated. If the decision is to offload to an edge server, an offloading request is submitted, waiting for the edge server to process the computation task and return the results. If local execution is chosen, the computation results are obtained directly.

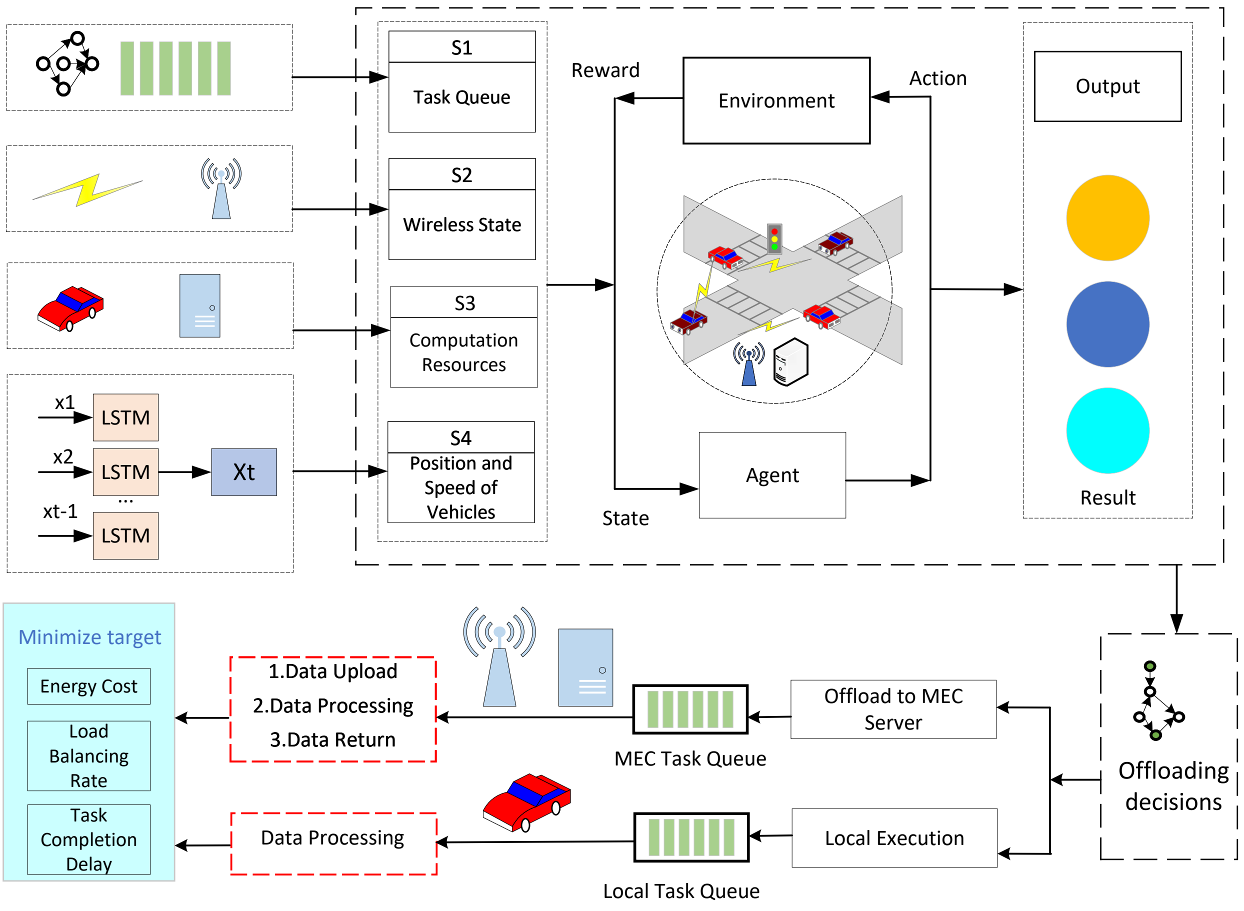


Figure 3. The Framework of the L-MADRL algorithm.

4.1. Vehicular Trajectory Prediction

Considering the uncertainty of vehicle movement and the complex dynamic road environment, a method based on trajectory prediction is proposed. Specifically, an LSTM-based encoder-decoder network architecture is employed to predict the future trajectory of the vehicle. We use historical trajectories and known information about the future to predict future trajectories, which is a sequence-to-sequence problem. To solve this problem, an encoder-decoder network structure that uses two LSTM layers as the feature extraction unit is proposed. As shown in Figure 4, we input two types of sequence data into the prediction network. When entering history, take into account the past sequence of vehicle position (x, y) and driving speed. An encoder-decoder prediction network based on LSTM is constructed to construct the feature representation of the historical input and future output. The encoder and decoder consist of two LSTM layers that capture long-term dependencies in the sequence. Historical features are encoded into context vectors in the encoder module and combined with future known features for decoding in the decoder module. All features are then passed to the linear module to predict the position of the target (x, y). In order to improve the nonlinear adaptability of the prediction network, we added the ReLU activation unit at the end. Figure 4 shows the LSTM algorithm framework for predicting vehicle trajectories.

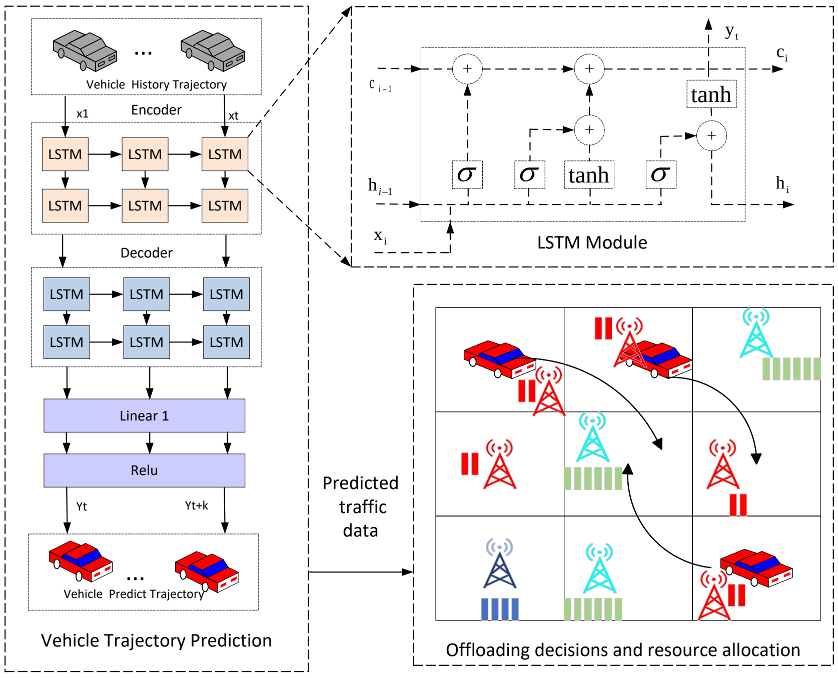


Figure 4. LSTM for vehicle trajectory prediction.

4.2. MADDPG-Based Optimal Offloading Decision

In tackling the computation offloading challenge, we incorporate the sophisticated Multi-Agent Deep Deterministic Policy Gradients (MADDPG) algorithm. This innovative approach is designed to craft decision-making policies for intelligent agents through the application of reinforcement learning. We frame the computation offloading dilemma as a Markov Decision Process (MDP), characterized by the four-tuple $\{S, A, P, R\}$. Here, S(state space) encapsulates the local observations of all agents, capturing dynamic environmental elements such as the workload on MEC servers and the computation resources. A(action space) includes the spectrum of possible actions for each agent, encompassing crucial decisions related to computation offloading and resource allocation, notably in the realms of communication and computational resource distribution. P(transition probability) signifies the likelihood of the environment transitioning between states in response to specific actions. R(reward function) gauges the benefits accrued from executing particular actions in given states. Through the MADDPG layer, agents are empowered to develop adaptive strategies within this fluid environment, basing decisions on real-time insights and experiential learning to optimize system efficiency. This framework represents a cutting-edge solution for the computation offloading conundrum.

Firstly, we transform the system utility maximization objective into a reward maximization challenge. This is accomplished by employing a multi-agent variation of the Markov Decision Process, termed the Markov Game, denoted by the tuple $< N,\ S,\ A,\ O >$. This approach treats each vehicle as an intelligent agent, which hones its optimal strategy by observing the L-MADRL environment and collaborating to maximize overall system utility. We then define the fleet of mobile vehicles as $N = \{1, 2 \ldots N\}$. Moreover, $S = \{s_1, s_2 \ldots s_N\}$ is designed as the set of states, $A = \{a_1, a_2 \ldots a_N\}$ represents the agents' action repertoire, and $O = \{o_1, o_2 \ldots o_N\}$ constitutes a sequence of observations available to the agents. Given that the collaborative L-MADRL scheme operates within discrete, equal, and non-overlapping time slots, and assuming consistent communication parameters within each time slot T, we establish the tuple for each time slot T accordingly. Figure 5 illustrates the framework for solving the computational offloading problem based on the MADDPG algorithm.

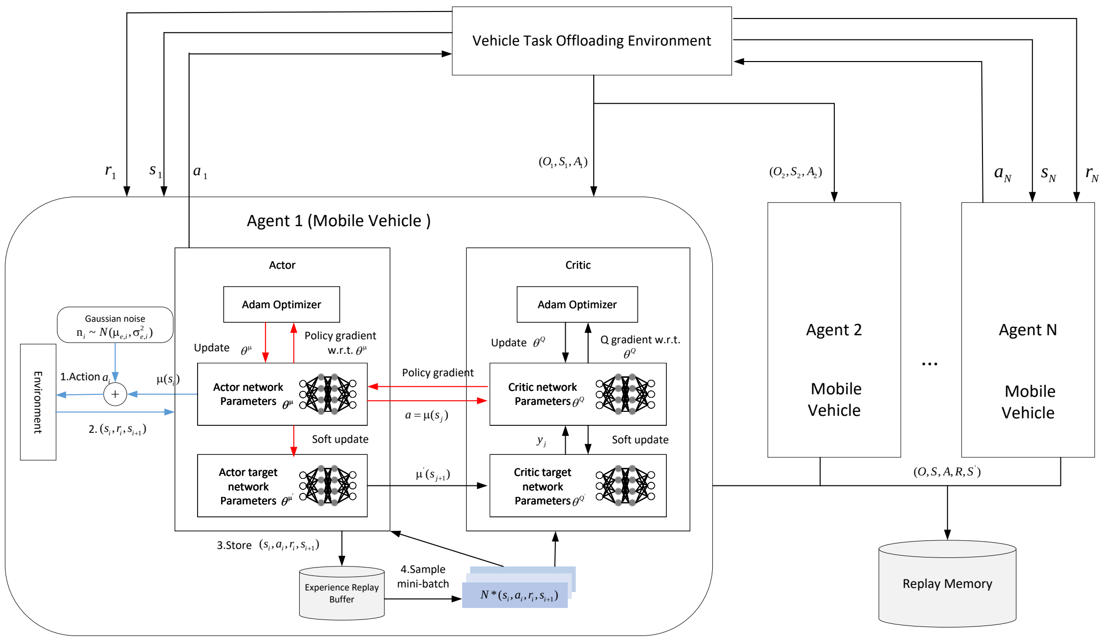


Figure 5. MADDPG Algorithm framework.

4.2.1. State S

The state environment within the L-MADRL network is a multifaceted construct, comprising five distinct states: task state, channel state, resource state and vehicle state. This composition is formally expressed as Equation (16):

$$S(t) = \{S_{task}(t), S_{channel}(t), S_{res}(t), S_{vehicle}(t)\} \tag{16}$$

The task state $S_{task}(t)$ is defined by the tuple $S_{task}(t) = [D(t), C(t), T^{max}(t)]$, $n \in N$. Here, $D(t)$ represents the volume of task data generated by a vehicle at t, $C(t)$ quantifies the CPU cycles needed for task completion, and $T^{max}(t)$ indicates the latest permissible completion time for the vehicle's task.

The channel state $S_{channel}(t)$ is articulated as Equation (17)

$$S_{channel}(t) = c_n^k(t) = \begin{bmatrix} c_{1,1} & \cdots & c_{1,K} \\ \vdots & \ddots & \vdots \\ c_{N,1} & \cdots & c_{N,K} \end{bmatrix} \tag{17}$$

In this matrix, $c_n^k(t)$ denotes whether vehicle n is utilizing sub-channel k at the moment t, with $c_n^k(t) = 1$, indicating usage and $c_n^k(t) = 0$ otherwise.

The resource state is depicted as $S_{res}(t)$

$$S_{res}(t) = \{f_1^{mec}(t), f_2^{mec}(t) \dots f_n^{mec}(t)\} \tag{18}$$

In this context, $f_i^{mec}(t)$ represents the computing resources available at the edge server.

$$S_{vehicle}(t) = \{v_1^{pos}(t), v_2^{pos}(t) \dots v_n^{pos}(t), v_1^{spe}(t), v_2^{spe}(t) \dots v_n^{spe}(t)\} \tag{19}$$

$S_{vehicle}(t)$ represents the status information of the vehicle, which consists of vehicle position and speed.

4.2.2. Action A

In the dynamic environment of the L-MADRL system, each vehicle, informed by the observed system state, must execute a sequence of critical actions at each discrete time step t. These actions are essential for the efficient management of task execution and encompass offloading decisions, channel selection, transmit power determination, and computational as well as CPU resource allocation. The action space is formally represented as Equation (20):

$$A(t) = \{x_n^k(t), k(t), f_{n,k}^{mec}(t)\} \tag{20}$$

The specific actions are defined as follows:

1. Offloading Decision $x_n^k(t)$: This binary decision, $x_n^k(t) \in \{0,1\}, (n \in N, k \in K)$, is made by each vehicle at time t, influenced by the current task state $S(t)$. n represents the number of vehicles within the service range of the edge server, k represents the number of subchannels used for transmission through the wireless channel, and x_n^k represents the offloading decision of a particular vehicle within the service range of the edge server for a specific wireless communication subchannel.The vehicle decides whether to process the task locally $x_n^k(t) = 0$ or to offload it to the MEC server via channel k $x_n^k(t) = 1$.

Upon deciding to offload the task, a series of resource allocations are required for the vehicle.

2. Channel Selection $k(t)$: The selection of $k(t)$ from the range $[1, 2, \ldots\ K]$ is based on the prevailing channel state $S_{channel}(t)$ Each vehicle selects an appropriate channel to transmit its task to the MEC server.
3. Computing Resource Allocation $S_{res}(t)$: The allocation of computing resources, denoted by $S_{res}(t) = [f_1^{mec}(t), f_2^{mec}(t) \ldots f_n^{mec}(t)]$, is determined based on the current computing resource state $S_{res}(t)$ and task state $S_{task}(t)$.This allocation ensures that each vehicle is equipped with the necessary computational resources to execute the computing task effectively.

4.2.3. Reward

In this study, we explore a system reward function that exhibits a positive correlation with the objective function of our optimization challenge. The system reward for a given time interval t is defined as the aggregate of the individual rewards $R(S_n(t), A_n(t))$ consequent to the execution of an action $A_n(t)$ in a distinct state $S_n(t)$. This collective reward mechanism is mathematically expressed as Equation (21):

$$r(S(t), A(t)) = \frac{1}{N} \sum_{n \in N} R(S_n(t), A_n(t)) = \frac{1}{N} \sum_{n \in N} R_n(t) \tag{21}$$

5. Algorithm Design

To address these complexities, we propose and implement a hybrid approach that combines centralized learning with decentralized execution in the L-MADRL algorithm. This strategy ensures comprehensive learning by incorporating global information and influences while enabling individual agents to execute based on their specific, locally-obtained insights. This dual approach aims to enhance the overall efficacy and adaptability of the L-MADRL algorithm in managing the sophisticated demands of Vehicular Computing Offloading and Resource Allocation.

Trajectory prediction and load prediction algorithms are important in self-driving vehicle management systems. The high maneuverability of intelligent vehicles is the main reason for the increase in the dynamic workload of RSUs, so it becomes crucial to capture the motion patterns of intelligent vehicles to calculate the potential future RSU workload. First, we use a trajectory prediction model to predict the future trajectories of intelligent vehicles.

In Algorithm 1, our initial step involves acquiring the set of intelligent vehicles. Subsequently, we gather both historical $\kappa_v^{his}(t)$ and future $\kappa_v^{fut}(t)$ trajectory data of these vehicles. During the training phase, we input the historical trajectories of each intelligent vehicle for each period into our trajectory prediction model. This model then outputs predicted trajectories. We assess the model's accuracy by comparing these predicted trajectories against the actual future trajectories of each vehicle, using the formula $Loss_v^{train} = (\kappa_v^{fut}(t) - \kappa_v^{pre}(t))^2$. The model is refined through a backpropagation-based method, leveraging $Loss_v$ for updates.

Once the model training is complete, we introduce the historical trajectory $\kappa_v^{his}(t')$ of an intelligent vehicle into this trained model during the evaluation phase. This process yields the predicted trajectory $\kappa_v^{pre}(t')$.The model's performance is evaluated using the mean square error (MSE) metric, and this error is utilized in the backpropagation of the trained model to compute the loss $Loss_v^{test} = (\kappa_v^{fut}(t') - \kappa_v^{pre}(t'))^2$.

Building upon the intelligent vehicles' trajectory prediction, we further devise a method for forecasting their future workload. Initially, we acquire the set $\{1, 2 \ldots, M\}$ of Road Side Unit (RSU) edge servers along with the respective positions P_m of each RSU. Next, we feed the predicted traffic data into the region selector of each RSU to estimate the future traffic $z_m^{t'}$ in RSU's region $z_m^{t'}$ is then input into an RSU workload predictor. Consequently, the anticipated RSU workload at time slot t' is computed as $L_m^{pre}(t') = L_m(t'-1) + \zeta z_m^{t'}$, where ζ is a coefficient that translates predicted traffic into the anticipated RSU workload.

Algorithm 1 Trajectory Prediction and Workload Prediction Algorithm

```
Input: Set of intelligent vehicles $V$, set of RSU edge servers$\{1, 2 \ldots\ M\}$, and each RSU's
    corresponding position $P_m$
Output: Predict workload of RSU m
    Training Process
 2: for each intelligent vehicle $v \in V$ do
        Obtain the historical trajectory $\kappa_v^{his}(t)$ and future trajectory $\kappa_v^{fut}(t)$ of the vehicle;
 4:     Predict trajectory $\kappa_v^{pre}(t)$ using $J_v$;
        Compute MSE loss for each vehicle $Loss_v^{train} = (\kappa_v^{fut}(t) - \kappa_v^{pre}(t))^2$;
 6:     Update the trajectory prediction model $J_v$ via backpropagation based on $Loss_v^{train}$;
    end for
 8: Evaluating Process
    for each intelligent vehicle $v \in V$ do
10:     Input the historical trajectory $\kappa_v^{his}(t')$into the trained trajectory prediction model to
        obtain the predicted trajectory $\kappa_v^{pre}(t')$;
        Compute the evaluation loss $Loss_v^{test} = (\kappa_v^{fut}(t') - \kappa_v^{pre}(t'))^2$;
12: end for
    Workload Prediction
14: for $m = 1, 2 \ldots M$ do
        Input predicted trajectories $\kappa_v^{pre}(t')$into RSU m region selector to obtain the predicted
        traffic volume $z_m^{t'}$ in the RSU m service region;
16:     Calculate the predicted workload of RSU m: $L_m(t'-1) + \zeta z_m^{t'}$;
    end for
```

The LSTM module in our model is tasked with learning from historical data to predict key network parameters like edge server load and bandwidth. Its ability to recognize temporal patterns allows for accurate future state predictions. These predictions are then utilized by the MADDPG module, an algorithm effective in multi-agent environments for optimizing decision-making. MADDPG adapts each agent's behavior—either an edge server or network node—for optimal resource allocation and task scheduling. By integrating LSTM and MADDPG, our model not only responds to current network conditions but also proactively adjusts to future changes, enhancing network efficiency and responsiveness, particularly in dynamic edge computing environments with fluctuating demands and limited resources.

In Algorithm 2, we introduce a Multi-Agent Deep Deterministic Policy Gradient (MA-DDPG) approach, tailored to address the complexities of distributed vehicular network computing offloading and resource allocation. This method, distinct from conventional reinforcement learning techniques, incorporates Deep Neural Networks (DNNs) as non-linear approximators. These networks sample the loss function, thereby reducing the computational burden associated with large-scale offloading challenges. The agents in this setup collaboratively offload tasks to Mobile Edge Computing (MEC) servers, establishing a mutual learning environment that includes all agents and MEC servers. During centralized training phases, MEC servers collate state-action data from all agents to train the DRL model. This arrangement allows each agent to acquire a comprehensive understanding of the learning environment, promoting collaborative learning, enhancing environmental stability, and bolstering convergence efficiency.

In our framework, we define the decision set of all agents as $\pi = \{\pi_1, \pi_2, \ldots, \pi_N\}$, where $\theta = \{\theta_1, \theta_2, \ldots \theta_N\}$ represents the parameter set for each respective policy. Each agent is tasked with updating its parameter set θ_n in pursuit of the optimal policy $\pi^*_{\theta_n} = argmax_{\theta_n} J(\theta_n)$. This involves maximizing its objective function $J(\theta_n)$, which is the agent's reward function, formulated as per Equation (21).

Algorithm 2 The MA-DDPG Training Procedure in MEC System

```
Input: Replay buffer D,time T,exploration probability$\epsilon$, discount factor $\gamma$, update step $\chi$
Output: The optimal policy $\pi^*_{\theta_n}$ and maximum reward $r^*(s,a)$
    for each episode do
        Initialize the state $s_0 \leftarrow \{S_{task}(t), S_{channel}(t), S_{res}(t), S_{vehicle}(t)|_{t=0}\}$;
3:      for $t = 1, 2 \ldots T$ do
            Select a random action $a_j(t)$ with probability $\epsilon$, Otherwise, choose action $a_j(t)$ by
            executing the policy: $a_j(t) = \pi_{\theta_j}\big(s_j(t)\big)$;
            Each vehicle intelligent agent executes actions a$(t) = a_1(t), a_2(t), \ldots, a_N(t)$ by
            performing offloading decision $x_n^k(t)$,channel selection k(t), computational resource
            allocation $f_n^{mec}(t)$;
6:          Get the system reward $r(t)$ and the state $s'$;
            Store $(s(t), a(t), r(t), s'(t))$ into the memory D;
            for agent $j = 1$ to $N$ do
9:              Sample random mini batch of transitions $(s_j, a_j, r_j, s'_j)$ from D;
                Calculate $y_j = r_j + \gamma Q_j^{\pi}\Big(s'_j, a'_1, a'_2, \ldots a'_N\Big)|_{a'_j = \pi'_j(o_j)}$;
                Update the parameter matrix of critic's online network by minimizing the loss
                    $$L(\theta_j) = \frac{1}{S}\sum_j [y_i - Q_j^{\pi}(s_j, a_1, \ldots, a_N)]^2$$
12:             Update actor by using the sampled policy gradient:
                    $$\nabla_{\theta_j} J(\pi_j) = \frac{1}{S}[\nabla_{\theta_j} Q_j^{\pi}(s'_j, a_1, \ldots, a_N)\nabla_{\theta_j}\pi_j(a_j|s_j)]$$
            end for
            Update the target network parameters for each agent
                    $$\theta'_j \leftarrow \epsilon\theta_j + (1-\epsilon)\theta'_j$$
15:     end for
    end for
```

Our approach leverages the Multi-Agent Deep Deterministic Policy Gradient (MA-DDPG), a deterministic policy gradient method particularly adept for continuous action spaces within multi-agent environments. During the training phase, actors within this framework execute deterministic actions via their behavioral networks, while critics appraise these actions using target networks. The actors subsequently refine their behavioral networks by computing the gradient of the objective function $J(\theta_n)$ as detailed in Equation (22):

$$\nabla_{\theta_n} J(\pi_n) = E_{o,a\sim D}[\nabla_{\theta_n} Q_n^{\pi}(o, a_1, \ldots, a_N)\nabla_{\theta_n}\pi_n(a_n|o_n)] \tag{22}$$

Here, $o = \{o_1, o_2, \ldots, o_N\}$ signifies the set of observations, $Q_n^{\pi}(o, a_1, a_2 \ldots a_n)$ is the centralized action-value function, $a_1, a_2 \ldots a_n$ are the actions learned by the agents, with D representing the experience replay memory buffer that contains multiple event samples (o, a, r, o'). Concurrently, the critic refines the Q function Q_n^{π} by minimizing the loss function, as expressed in Equation (23):

$$L(\theta_n) = E_{o,a,r,o'}[(y_n - Q_n^{\pi}(o, a_1, \ldots, a_N))^2] \tag{23}$$

where $y_n = r_n + \gamma Q_n^{\pi}(o', a_1', a_2', \ldots a_N')|_{a_n' = \pi_n'(o_n)}$ represents the Temporal Difference (TD) target, and $\pi_n'(o_n)$ defines the target policy with delayed parameters θ_n'. Algorithm 2 outlines the training process, encompassing two primary phases: planning and updating. In the planning phase, an ϵ-greedy strategy is employed to strike a balance between exploration and exploitation. During this phase, the Q function is updated. Each mobile device, at every time step T, executes an action, estimates the system reward, and records training data in an experience replay pool. Subsequent to each action, the mobile device advances to the next step, updating both the critic and actor networks, as well as their respective target networks. This training is iterative, continuing until the desired system reward performance is attained.

The update of the target network θ_n using the strategy gradient method ∇_{θ_n} guides the intelligent Edge Device (ED) in executing correct actions to achieve the optimal strategy. The values of the Deep Neural Networks (DNNs) in the goal-based actor network remain fixed over several iterations, while the weights of the DNNs in the actor-behavior network are updated. In the distributed vehicular network's MEC system, all intelligent EDs aim to maximize their expected function $J(\theta_n)$ (i.e., user utility) by interacting with the environment, thus attaining a stable strategy. This approach ensures stability in the learning environment, even amidst strategy shifts, which enhances the quality of strategy evaluation and, consequently, the overall utility of the system. Upon completing the training, the ED downloads the learned policy network parameters from the MEC server and updates the target network parameters of the actors and critics, following the update process outlined in Equation (24):

$$\theta_j' \leftarrow \epsilon\theta_j + (1-\epsilon)\theta_j' \tag{24}$$

6. Results

This section commences with an extensive overview of our experimental platform, offering intricate details regarding the configuration of the experimental parameters. We then proceed to demonstrate the efficacy of the L-MADRL method through a series of convergence experiments. Building upon this foundational analysis, we delve into comprehensive evaluations, particularly focusing on energy consumption and latency, under various system resource state scenarios for the L-MADRL approach. Moreover, a thorough comparative study is conducted to juxtapose the L-MADRL method with other existing service offloading techniques.

6.1. Simulation Settings and Dataset

The simulation experiments were executed on a system running the Windows 11 operating system. The central processing unit (CPU) employed in these experiments was the Intel(R) Core(TM) i7-14700KF, operating at 3400 Mhz. Our experimental framework was developed and implemented using Python 3.11 and Pytorch 2.1.0. For the purpose of the simulation, we utilized the Shenzhen City vehicle dataset. Each data entry in the vehicular dataset is structured as a quaternion, encapsulating various aspects of in-vehicle information: (ID, time, longitude, latitude, flag, speed) Here, 'ID' denotes the vehicle number, 'time' specifies when the data was received, 'longitude' and 'latitude' pinpoint the geographical coordinates of the user's vehicle, and 'flag' indicates the association of a vehicle user with a particular roadside unit.

In this experiment, the trajectory data of vehicles is real vehicle trajectory data from Shenzhen city, while the remaining parameters are set based on simulation parameters in real-world scenarios. We randomly generated coordinates for 10 base stations located at different positions. Each generated base station establishes communication links with vehicles within its coverage area, facilitating data transmission. All base stations collectively form a mesh network structure. Tasks are randomly assigned to vehicles, assuming the CPU frequency for local computation of vehicles is in the range [0.1, 1] GHz, the CPU frequency for RSU is 5 GHz, and the CPU frequency required for each computing task is in the range [0.05, 1.5] GHz. The number of vehicles within the service range of the edge server varies from 10 to 50, and the transmission power ranges from 0 to 24 dBm. The system bandwidth is 20 MHz. The specific simulation parameters are presented in Table 2.

Table 2. IoV parameters setting.

Definitions	Notations	Value
Number of EDs	N_v	[10, 50]
Size of Task	D_n	[0.1–5] MB
The transmit power	P_n^k	[0–24] dBm
The background noise variances	σ^2	-100 dBm
The system bandwidth	W	20 MHZ
The ED's and MEC server's computing capability	F_n^{vec}, F_n^{mec}	[0.1–1] GHz, 5 GHz
The ED's energy coefficient	k	5×10^{-27}
The weights of time and energy costs	λ_n^T, λ_n^E	0.6, 0.4
Number of base stations	N_b	10
Range of RSU	R_n	[100–500] m^2

The experimental simulation setup is visually depicted in Figure 6, where the vehicle's trajectory position information is marked in red, and the randomly generated base station information is highlighted in blue.

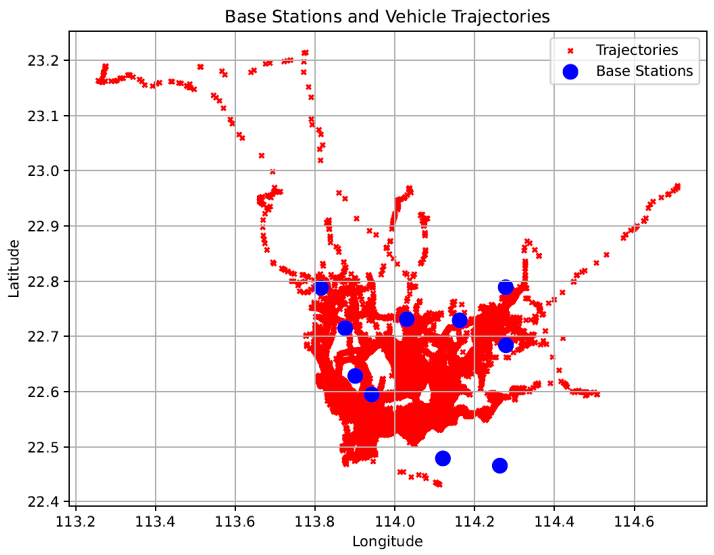


Figure 6. Base station and vehicle trajectory.

6.2. Simulation Results

6.2.1. Trajectory Prediction Evaluation

In the preliminary phase of the experiment, we conducted a comprehensive assessment of the LSTM prediction model. We carefully partitioned the dataset into training, testing, and validation sets to ensure thorough and adequate model training. Post-training, our prediction network exhibited outstanding performance on both the testing and validation sets, particularly excelling in vehicle trajectory prediction. Figure 7 clearly illustrates a gradual reduction in the loss function with an increasing number of training iterations, reflecting the model's gradual learning and adaptation to input data. Our experimental results further validate the reliability and robustness of the LSTM model in the task of vehicle trajectory prediction.

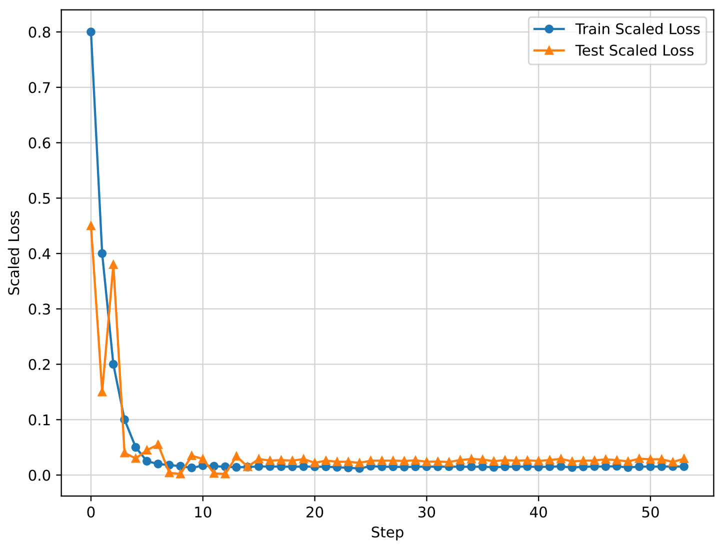


Figure 7. The train and valid scaled loss.

6.2.2. Comparison with Traditional Algorithms

We engage four traditional non-Deep Reinforcement Learning (non-DRL) algorithms to substantiate the efficiency of the L-MADRL algorithm in identifying optimal strategies for task offloading and resource allocation:

1. Local computing: In this approach, the computational workload in each region is managed directly by the vehicle side.
2. Edge computing: Here, vehicles in each region offload their computational workload to the nearest edge server (ES) for processing.
3. Cloud computing: This method involves processing computational tasks at the cloud server (CS), rather than at the local or edge level.
4. Random computing: In this strategy, decisions regarding task offloading and resource allocation are made randomly, selecting between vehicles, ES, or CS. Additionally, the allocation of resources and bandwidth for tasks is also randomized.

System Consumption and Success Rate

Figure 8 shows how system cost changes with varying vehicle numbers. Across all methods, there is a consistent trend of increased processing cost for vehicle computation tasks as the number of vehicles in the region increases. Notably, the random computing strategy is the least efficient, relying on random selection for task offloading and resource allocation, lacking the ability to make informed decisions in a dynamic network environment. In contrast, the L-MADRL algorithm consistently outperforms conventional task offloading and resource allocation methods across different vehicle densities.

The relationship between the number of vehicles and their average completion rate is illustrated in Figure 9. While the advantages of the L-MADRL algorithm may not be highly pronounced with a smaller vehicle count, its effectiveness becomes increasingly apparent as the number of vehicles rises. In contrast, the Local algorithm, when dealing with a large number of vehicles, faces substantial local computational strain, resulting in a diminished number of completed tasks within the specified timeframe. Simultaneously, both the Edge and Cloud algorithms encounter heightened transmission delays due to the increased vehicle count, leading to elevated network congestion and a subsequent decline in the timely completion of tasks. Conversely, the L-MADRL algorithm adeptly handles task offloading to RSUs even under heavy task loads, thus maintaining a high and stable task completion rate.

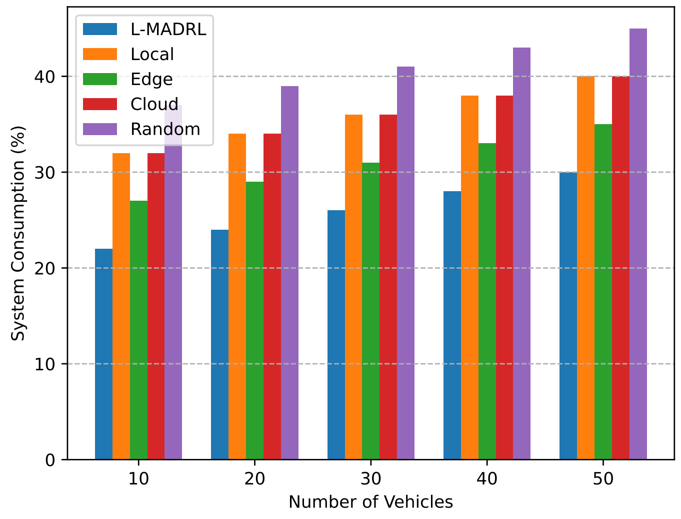


Figure 8. System Consumption.

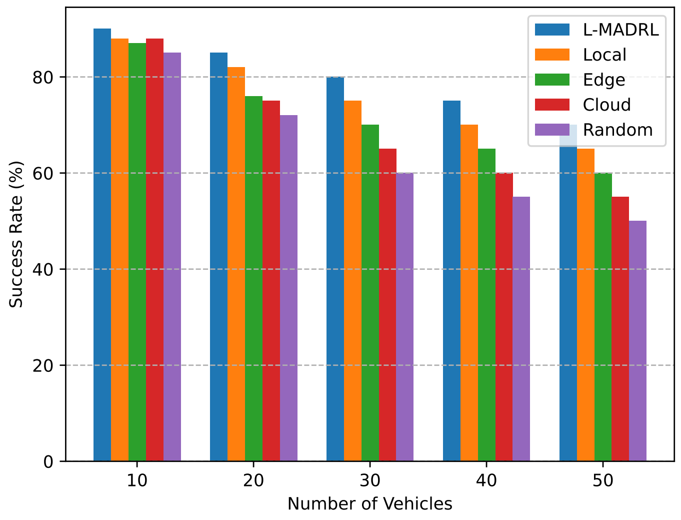


Figure 9. Success Rate.

Average delay time and energy consumption

Figure 10 illustrates the average processing latency across different methods for varying numbers of vehicles. A noticeable trend is the increase in processing latency corresponding to the growing workload in each region. In this context, the L-MADRL algorithm stands out by providing the lowest latency regardless of the number of vehicles. Conversely, the Cloud algorithm records the highest average processing latency, primarily due to significant transmission delays over wireless links when offloading computational tasks to the cloud server. Meanwhile, the stochastic computing algorithm's average processing delay exhibits a degree of randomness, depending on the prevailing network conditions. Lower latency is observed under favorable scenarios, while higher latency is noted in poorer conditions.

Figure 11 presents the average energy consumption of various processing algorithms across different numbers of vehicles. The Local algorithm maintains low energy consumption by processing tasks directly at the vehicle terminals, eliminating the need for task transfer. However, the system's average energy consumption increases with a higher number of vehicles. The L-MADRL algorithm excels in energy efficiency, significantly reducing average energy use by adapting to vehicle trajectories and edge server loads. This is a result of its optimized task offloading and resource allocation strategies, tailored to specific vehicle paths and server load dynamics.

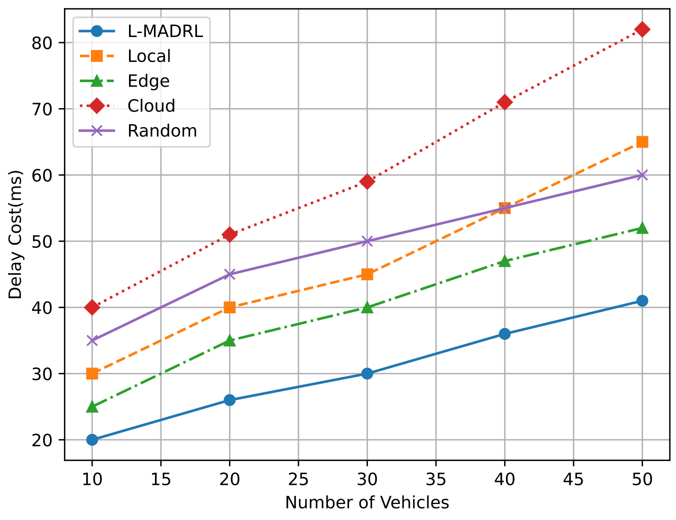


Figure 10. Delay Cost.

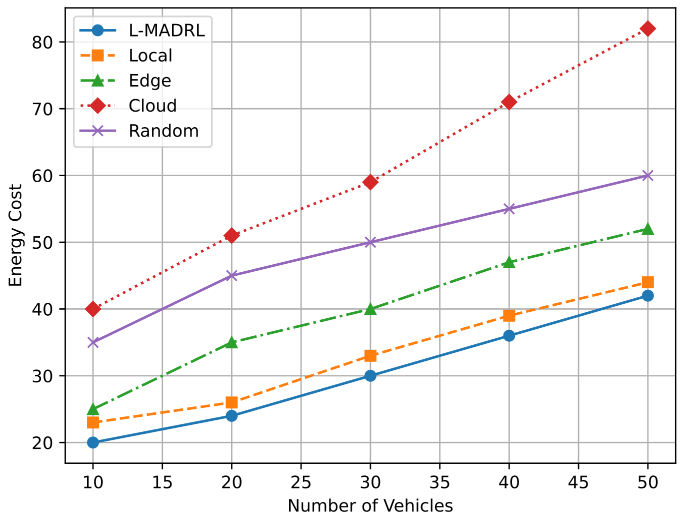


Figure 11. Energy Cost.

6.2.3. Comparison with Other Algorithms

DDPG: This algorithm employs the DDPG task offloading algorithm [31]. However, it does not account for the communication time between the vehicle and the base station. In scenarios where the vehicle departs before the computation concludes, the result cannot be returned, leading to task failure.

MADDPG:This algorithm employs the MADDPG task offloading algorithm [26]. The algorithm takes into account the perception of vehicle speed, but the actual relevance is limited as the vehicle speeds are randomly generated.

In the comparative experiments, the assessment of the system's overall cost involves diverse factors such as latency, energy consumption, and load balancing rate. Latency serves as a critical metric for gauging task execution speed, energy consumption directly influences the system's efficiency, and the load balancing rate signifies the equilibrium of resource utilization. We systematically consider these factors to thoroughly evaluate the system's performance across different task arrival rates ranging from 1.5 to 4 Mbps.

Concerning algorithmic comparison, we compare our proposed algorithm with DDPG and MADDPG algorithms. DDPG is a conventional algorithm for task offloading and resource allocation, while MADDPG is a multi-agent reinforcement learning algorithm. Through this comparative analysis, we acquire a comprehensive understanding of the superiority of our algorithm under varying load conditions, offering valuable insights for system design and optimization. Figures 12 and 13 respectively show the changes in system energy consumption and average reward as the vehicle task arrival ratio changes. The experimental results prove that our proposed algorithm significantly reduces system energy consumption and system average reward.

Impact of Range of RSU

In this experiment, we systematically investigate the impact of varying RSU coverage ranges on load balancing and task success offloading rates. The experiment is designed to compare the performance of our proposed algorithm with DDPG and MADDPG algorithms under different RSU coverage scenarios. Figures 14 and 15 respectively depict the variations in load balancing rate and task success completion rate of edge servers with the changing coverage range. Experimental results demonstrate that our proposed algorithm achieves a more balanced complexity and enhances the success completion rate of tasks compared to other algorithms.

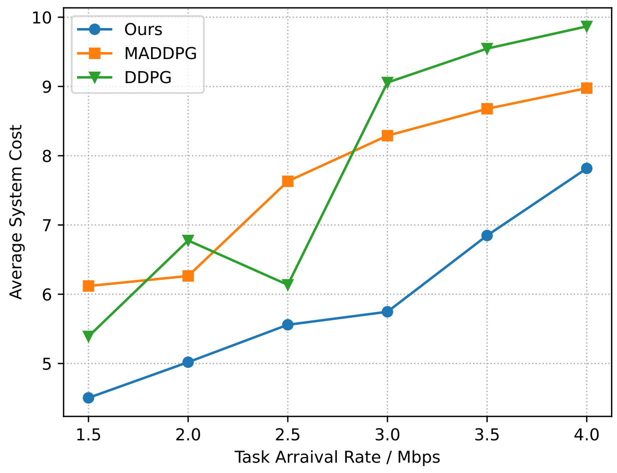


Figure 12. Task Arrival rate of System Cost.

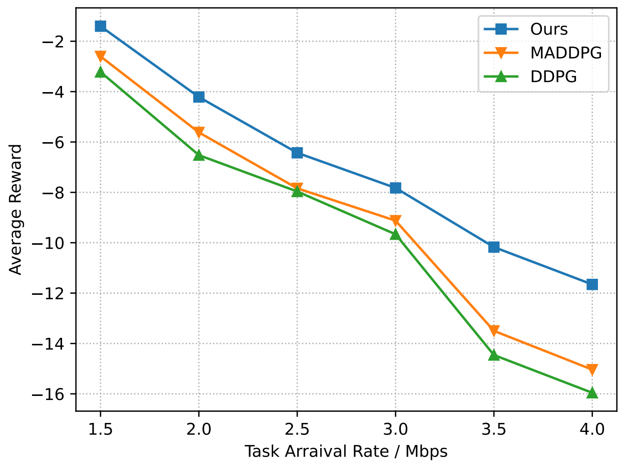


Figure 13. Task Arrival Rate of Success Rate.

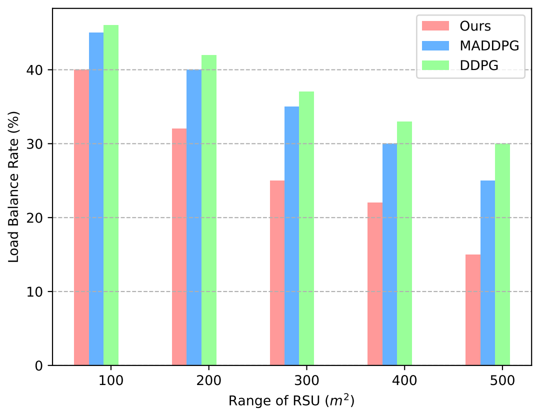


Figure 14. RSU of System Consumption.

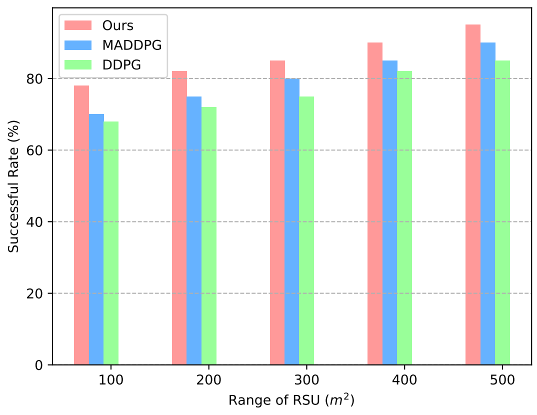


Figure 15. RSU of Success Rate.

6.2.4. Parameters Performance Evaluation

In Figure 16, the experiment delves into the impact of varying discount factors on the experimental reward values. We set three distinct discount factors—0.9, 0.95, and 0.99—and conducted multiple training and testing sessions on a consistent reinforcement learning task to evaluate their effects. The results clearly indicate that the discount factor significantly influences the reinforcement learning task's performance. With a discount factor of 0.9, the agent registers the highest cumulative reward values.When the discount factors are set to 0.95 and 0.99, there is a marginal decline in the agent's performance. Although these higher discount factors still account for future rewards, they may, in this specific task, cause the agent to overly prioritize long-term returns. This could potentially slow down the learning process and affect the stability of the learning outcomes.

The experimental results, as delineated in Figure 17, highlight the profound impact on system performance and learning efficacy that stems from varying the computational cutoff time and learning rate for different tasks.

Initially, the experiment focusing on the variation in task completion deadlines demonstrated a significant influence of the cutoff time on system performance. A trend was observed where a shorter task deadline led to a gradual decrease in the overall system reward, with the steepest decline noted at 0.6 s. This trend indicates that stringent time constraints for completing computational tasks necessitate heightened real-time processing capabilities, subsequently impacting the success rate of task execution and diminishing the overall system reward. Conversely, extending the task deadline allows for a progressive increase in the system reward, suggesting ample time for task processing and an enhanced success rate. The system reward stabilizes at a 1-s deadline, indicating an optimal balance between real-time performance and task completion success within this timeframe.

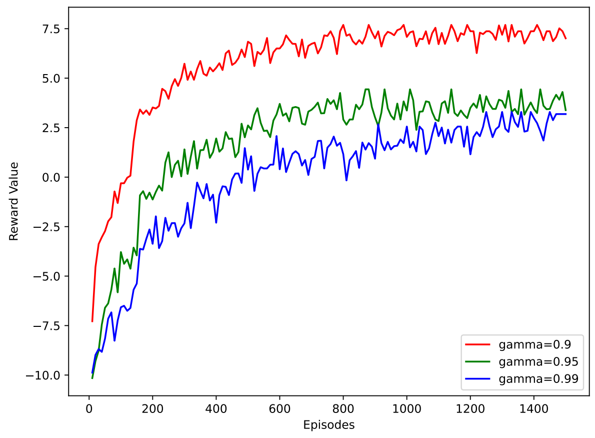


Figure 16. Different discount.

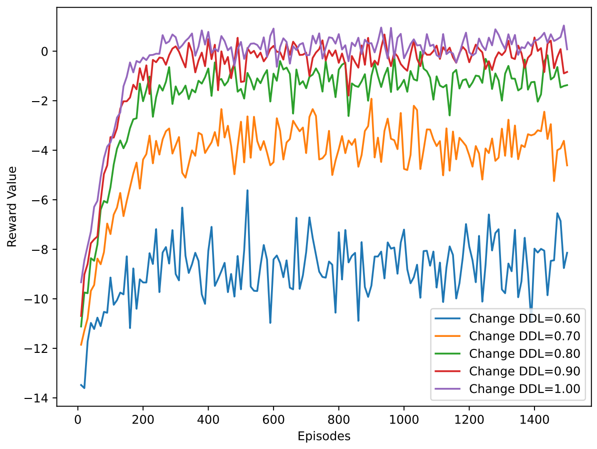


Figure 17. Different deadline.

7. Conclusions

In response to the growing need for low-latency services in Telematics, this study introduces L-MADRL, a distributed deep reinforcement learning-based approach for service offloading in 5G Telematics edge computing. The primary objective is to minimize the average service cost for Telematics users while ensuring a superior service experience. The approach utilizes Long Short-Term Memory (LSTM) for predicting future user service demands in Telematics, integrating this forecast into the L-MADRL algorithm to facilitate intelligent offloading decisions. Comparative experiments conducted with a real-world Telematics user service demand dataset demonstrate that the L-MADRL method significantly reduces average service costs, validating the effectiveness of the proposed approach. Future research will explore incorporating vehicles as computing service resources to further reduce service energy consumption and enhance resource utilization and load balancing at edge servers.

Author Contributions: Conceptualization, D.S., Y.C. and H.L.; methodology, D.S. and H.L.; software, D.S.; validation, Y.C., H.L. and D.S.; formal analysis, D.S.; investigation, Y.C.; resources, D.S.; data curation, Y.C.; writing—original draft preparation, D.S. and Y.C.; writing—review and editing, D.S. and H.L.; visualization, Y.C.; supervision, H.L.; project administration, H.L.; funding acquisition, H.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Yunnan Provincial Major Science and Technology Project: Research and Application of Key Technologies for Resource Sharing and Collaboration Toward Smart Tourism (Grant No. 202102AD080004) and supported in part by the Yunnan Provincial Major Science and Technology Project: Research and Application of Key Technologies for Scale Processing of Yunnan Characteristic Pre-Prepared Food (Grant No. 202202AE090019).

Data Availability Statement: The datasets used in this paper are public datasets. Datasets are available on the relevent website: https://people.cs.rutgers.edu/~dz220/data.html (accessed on 15 August 2023). The datasets used and analyzed during the current study are available from the corresponding website.

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

References

1. Tian, S.; Deng, X.; Chen, P.; Pei, T.; Oh, S.; Xue, W. A dynamic task offloading algorithm based on greedy matching in vehicle network. *Ad Hoc Netw.* **2021**, *123*, 102639. [CrossRef]
2. Alqarni, M.A.; Mousa, M.H.; Hussein, M.K. Task offloading using GPU-based particle swarm optimization for high-performance vehicular edge computing. *J. King Saud Univ. Comput. Inf. Sci.* **2022**, *34*, 10356–10364. [CrossRef]
3. Shu, W.; Li, Y. Joint offloading strategy based on quantum particle swarm optimization for MEC-enabled vehicular networks. *Digit. Commun. Netw.* **2023**, *9*, 56–66. [CrossRef]
4. Bozorgchenani, A.; Maghsudi, S.; Tarchi, D.; Hossain, E. Computation offloading in heterogeneous vehicular edge networks: On-line and off-policy bandit solutions. *IEEE Trans. Mob. Comput.* **2021**, *21*, 4233–4248. [CrossRef]
5. Materwala, H.; Ismail, L.; Hassanein, H.S. QoS-SLA-aware adaptive genetic algorithm for multi-request offloading in integrated edge-cloud computing in Internet of vehicles. *Veh. Commun.* **2023**, *43*, 100654. [CrossRef]
6. Wang, K.; Wang, X.; Liu, X. Sustainable Internet of Vehicles System: A Task Offloading Strategy Based on Improved Genetic Algorithm. *Sustainability* **2023**, *15*, 7506. [CrossRef]
7. Wang, H.; Lin, Z.; Guo, K.; Lv, T. Computation offloading based on game theory in MEC-assisted V2X networks. In Proceedings of the 2021 IEEE International Conference on Communications Workshops (ICC Workshops), Montreal, QC, Canada, 14–23 June 2021; pp. 1–6.
8. Xu, X.; Jiang, Q.; Zhang, P.; Cao, X.; Khosravi, M.R.; Alex, L.T.; Qi, L.; Dou, W. Game theory for distributed IoV task offloading with fuzzy neural network in edge computing. *IEEE Trans. Fuzzy Syst.* **2022**, *30*, 4593–4604. [CrossRef]
9. Zhang, K.; Yang, J.; Lin, Z. Computation Offloading and Resource Allocation Based on Game Theory in Symmetric MEC-Enabled Vehicular Networks. *Symmetry* **2023**, *15*, 1241. [CrossRef]
10. Ashraf, S.; Ahmad, A.; Yahya, A.; Ahmed, T. Underwater routing protocols: Analysis of link selection challenges. *AIMS Electron. Electr. Eng* **2020**, *4*, 234–248. [CrossRef]
11. Sundararajan, S.; Naduvil, M.K. Enhancing sensor linearity through the translinear circuit implementation of piecewise and neural network models. *AIMS Electron. Electr. Eng.* **2023**, *7*, 196–217.
12. Khan, F.; Nguang, S.K. Location-based reverse data delivery between infrastructure and vehicles. *AIMS Electron. Electr. Eng.* **2021**, *5*, 158–175. [CrossRef]
13. Nguyen, D.; Ding, M.; Pathirana, P.; Seneviratne, A.; Li, J.; Poor, V. Cooperative task offloading and block mining in blockchain-based edge computing with multi-agent deep reinforcement learning. *IEEE Trans. Mob. Comput.* **2021**. [CrossRef]
14. Lang, P.; Tian, D.; Duan, X.; Zhou, J.; Sheng, Z.; Leung, V.C. Cooperative computation offloading in blockchain-based vehicular edge computing networks. *IEEE Trans. Intell. Veh.* **2022**, *7*, 783–798. [CrossRef]
15. Lang, P.; Tian, D.; Duan, X.; Zhou, J.; Sheng, Z.; Leung, V.C. Blockchain-Based Cooperative Computation Offloading and Secure Handover in Vehicular Edge Computing Networks. *IEEE Trans. Intell. Veh.* **2023**, *8*, 3839–3853. [CrossRef]
16. Liu, L.; Chen, C.; Pei, Q.; Maharjan, S.; Zhang, Y. Vehicular edge computing and networking: A survey. *Mob. Netw. Appl.* **2021**, *26*, 1145–1168. [CrossRef]
17. Ahmed, M.; Raza, S.; Mirza, M.A.; Aziz, A.; Khan, M.A.; Khan, W.U.; Li, J.; Han, Z. A survey on vehicular task offloading: Classification, issues, and challenges. *J. King Saud Univ. Comput. Inf. Sci.* **2022**, *34*, 4135–4162. [CrossRef]
18. Liu, J.; Ahmed, M.; Mirza, M.A.; Khan, W.U.; Xu, D.; Li, J.; Aziz, A.; Han, Z. RL/DRL meets vehicular task offloading using edge and vehicular cloudlet: A survey. *IEEE Internet Things J.* **2022**, *9*, 8315–8338. [CrossRef]
19. Hazarika, B.; Singh, K.; Biswas, S.; Li, C.P. DRL-based resource allocation for computation offloading in IoV networks. *IEEE Trans. Ind. Inform.* **2022**, *18*, 8027–8038. [CrossRef]
20. Mirza, M.A.; Yu, J.; Raza, S.; Krichen, M.; Ahmed, M.; Khan, W.U.; Rabie, K.; Shongwe, T. DRL-assisted delay optimized task offloading in Automotive-Industry 5.0 based VECNs. *J. King Saud Univ. Comput. Inf. Sci.* **2023**, *35*, 101512. [CrossRef]
21. Jia, Z.; Zhou, Z.; Wang, X.; Mumtaz, S. Learning-based queuing delay-aware task offloading in collaborative vehicular networks. In Proceedings of the ICC 2021-IEEE International Conference on Communications, Montreal, QC, Canada, 14–23 June 2021; pp. 1–6.
22. Luo, Q.; Li, C.; Luan, T.H.; Shi, W. Minimizing the delay and cost of computation offloading for vehicular edge computing. *IEEE Trans. Serv. Comput.* **2021**, *15*, 2897–2909. [CrossRef]

23. Binh, T.H.; Vo, H.; Nguyen, B.M.; Binh, H.T.T. Reinforcement Learning for Optimizing Delay-Sensitive Task Offloading in Vehicular Edge-Cloud Computing. *IEEE Internet Things J.* **2023**, *11*, 2058–2069. [CrossRef]
24. Shang, B.; Liu, L.; Tian, Z. Deep learning-assisted energy-efficient task offloading in vehicular edge computing systems. *IEEE Trans. Veh. Technol.* **2021**, *70*, 9619–9624. [CrossRef]
25. Vemireddy, S.; Rout, R.R. Fuzzy Reinforcement Learning for energy efficient task offloading in Vehicular Fog Computing. *Comput. Netw.* **2021**, *199*, 108463. [CrossRef]
26. Huang, X.; He, L.; Zhang, W. Vehicle speed aware computing task offloading and resource allocation based on multi-agent reinforcement learning in a vehicular edge computing network. In Proceedings of the 2020 IEEE International Conference on Edge Computing (EDGE), Beijing, China, 19–23 October 2020; pp. 1–8.
27. Zhao, L.; Zhang, E.; Wan, S.; Hawbani, A.; Al-Dubai, A.Y.; Min, G.; Zomaya, A.Y. MESON: A Mobility-aware Dependent Task Offloading Scheme for Urban Vehicular Edge Computing. *IEEE Trans. Mob. Comput.* **2023**, 1–15. [CrossRef]
28. Chen, J.; Kang, J.; Xu, M.; Xiong, Z.; Niyato, D.; Chen, C.; Jamalipour, A.; Xie, S. Multiagent Deep Reinforcement Learning for Dynamic Avatar Migration in AIoT-Enabled Vehicular Metaverses with Trajectory Prediction. *IEEE Internet Things J.* **2024**, *11*, 70–83. [CrossRef]
29. Zeng, J.; Gou, F.; Wu, J. Task offloading scheme combining deep reinforcement learning and convolutional neural networks for vehicle trajectory prediction in smart cities. *Comput. Commun.* **2023**, *208*, 29–43. [CrossRef]
30. Yan, R.; Gu, Y.; Zhang, Z.; Jiao, S. Vehicle Trajectory Prediction Method for Task Offloading in Vehicular Edge Computing. *Sensors* **2023**, *23*, 7954. [CrossRef] [PubMed]
31. Chen, Z.; Wang, X. Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach. *EURASIP J. Wirel. Commun. Netw.* **2020**, *2020*, 1–21. [CrossRef]