mathematics

MDPI

Article

Vehicle Collaborative Partial Offloading Strategy in Vehicular Edge Computing

Ruoyu Chen [1,2], Yanfang Fan [2,*], Shuang Yuan [2] and Yanbo Hao [2]

[1] Institute of Intelligent Information Processing, Beijing Information Science and Technology University, Beijing 100192, China; chenruoyu@bistu.edu.cn
[2] Computer School, Beijing Information Science and Technology University, Beijing 100192, China; 2018020362@mail.bistu.edu.cn (S.Y.); 2022011016@bistu.edu.cn (Y.H.)
* Correspondence: fyfhappy@bistu.edu.cn

Abstract: Vehicular Edge Computing (VEC) is a crucial application of Mobile Edge Computing (MEC) in vehicular networks. In VEC networks, the computation tasks of vehicle terminals (VTs) can be offloaded to nearby MEC servers, overcoming the limitations of VTs' processing power and reducing latency caused by distant cloud communication. However, a mismatch between VTs' demanding tasks and MEC servers' limited resources can overload MEC servers, impacting Quality of Service (QoS) for computationally intensive tasks. Additionally, vehicle mobility can disrupt communication with static MEC servers, further affecting VTs' QoS. To address these challenges, this paper proposes a vehicle collaborative partial computation offloading model. This model allows VTs to offload tasks to two types of service nodes: collaborative vehicles and MEC servers. Factors like a vehicle's mobility, remaining battery power, and available computational power are also considered when evaluating its suitability for collaborative offloading. Furthermore, we design a deep reinforcement learning-based strategy for collaborative partial computation offloading that minimizes overall task delay while meeting individual latency constraints. Experimental results demonstrate that compared to traditional approaches without vehicle collaboration, this scheme significantly reduces latency and achieves a significant reduction (around 2%) in the failure rate under tighter latency constraints.

Keywords: mobile edge computing; vehicular edge computing; computation offloading; deep reinforcement learning; partial offloading

MSC: 68T07

Citation: Chen, R.; Fan, Y.; Yuan, S.; Hao, Y. Vehicle Collaborative Partial Offloading Strategy in Vehicular Edge Computing. *Mathematics* **2024**, *12*, 1466. https://doi.org/10.3390/math12101466

Academic Editor: Hari Mohan Srivastava

Received: 31 March 2024
Revised: 6 May 2024
Accepted: 7 May 2024
Published: 9 May 2024

1. Introduction

With the development of next-generation communication and information technologies such as 5G and artificial intelligence, the era of the Internet of Vehicles (IoV) [1] has arrived. Autonomous driving is a core application of vehicles as they evolve toward an intelligent and inter-connected state. Autonomous driving has strict real-time requirements for tasks such as environment sensing, decision making, and vehicle control, so the real-time completion of the computing tasks of vehicle terminals is directly related to the user's experience. Since computing tasks in VEC networks are latency-sensitive, computationally intensive, and highly mobile, ensuring the timely processing of vehicular tasks and improving the quality of service in VEC networks have become urgent problems to solve.

Several computation offloading approaches have been proposed to mitigate the aforementioned problems. Zhang [2] proposed a multilayered edge computing scheme in which backup computing servers were added between MEC servers and the problem of pricing for offloading computation to MEC and backup computing servers was transformed into a Stackelberg game. Guo et al. [3] constructed a cloud–MEC collaborative computation offloading scheme based on a fiber–wireless (FiWi) network. The offloading time to the

cloud was reduced via the FiWi connection, and task loads offloaded to the MEC servers and the cloud were balanced using a game theory-based method. In addition, several research studies [4] employed idle resources in Unmanned Aerial Vehicles (UAVs) as a supplement to edge computing servers for the efficient use of resources and to provide reliable service. However, the existing schemes face several challenges. Deploying additional MEC servers or UAVs incurs additional costs and leads to resource underutilization during periods of low network load. Simultaneously, intelligent vehicles in a VEC environment possess increasingly powerful computing capabilities, yet existing schemes largely neglect this valuable resource.

In this paper, vehicles with surplus computing and power resources and the ability to share their computing resources to serve other vehicles are referred to as collaborative vehicles. Vehicles that lack sufficient computing or power resources or those that cannot share their computing resources are referred to as non-collaborative vehicles. To address the aforementioned issues while avoiding the deployment of additional MEC servers, we propose a vehicle collaborative partial offloading strategy. This strategy, which takes into account dynamically changing vehicle and MEC computing resources, fully utilizes the computing and communication resources of each node in the network and offloads some computation tasks to collaborative vehicles, thus effectively reducing task delay in the computation-intensive and dynamic VEC network environment. Briefly, the contributions of this paper are as follows:

1. We design a collaborative partial computational offloading strategy based on divisible tasks. Each computational task can be divided into a maximum of three parts which can be processed locally by the task-generating vehicle, transferred to the MEC server, or offloaded to a nearby collaborative vehicle, allowing for better utilization of network resources.
2. We propose a multi-factor comprehensive evaluation method for collaborative vehicles. It takes into account factors such as the movement, remaining battery, and power of vehicles in the vicinity of the task-generating vehicle to find a more suitable candidate collaborative vehicle.
3. We propose a partial offloading approach based on deep reinforcement learning (DRL). On the basis of the collaborative vehicle evaluation method, the Double Deep Q Network (DDQN) algorithm is used to dynamically adjust the proportion of each offloading task part to achieve load balancing across service nodes and reduce delay.

The rest of this paper is organized as follows: related works are reviewed in Section 2. Section 3 briefly introduces the system model. The problem is formally formulated in Section 4, and our solution is proposed in Section 5. Furthermore, our experimental results are provided and analyzed in Section 6. Finally, the paper is concluded in Section 7.

2. Related Works

The industry and academia have been exploring ways of utilizing MECs to improve network performance and provide low-latency services to end users. However, emerging applications such as autonomous driving, augmented reality, speech-to-text translation, and image processing are generating huge numbers of computational tasks that are challenging the limited computing resources of MEC servers, especially in the Internet of Vehicles.

Approaches from different perspectives have been proposed to tackle the problem of a single server with insufficient resources, such as adding backup servers [2–6], balancing loads between MEC servers [7,8], and collaborative offloading using existing equipment [9–11]. Wang et al. [6] used a secondary MEC server to reduce the computational and communication load on a primary MEC server. Furthermore, they decomposed the mixed-integer nonlinear problem of minimizing the latency of system processing tasks into several subproblems and proposed a heuristic algorithm based on the priority of mobile devices and MEC servers to obtain a suboptimal device offloading policy. Experimental results show that this solution can effectively reduce a system's waiting time and improve system reliability. Huang et al. [9] considered an entire parking lot as a candidate server and modeled

the relationship between the user, the MEC server, and the parking lot as a Stackelberg game. They also proposed a subgradient-based iterative algorithm to solve the problem of workload distribution between parked vehicles and minimize overall cost. Similarly, Li et al. [5] designed an offloading incentive mechanism based on the Stackelberg game with the aim of maximizing the combined utility of vehicles, operators, and parking agents.

In terms of load balancing, Yang et al. [7] proposed a location-based offloading scheme that uploads tasks to the nearest MEC server based on the user's direction of travel. The convex optimization algorithm was utilized to generate an offloading strategy. Experiments show that the scheme can effectively reduce the system cost while satisfying delay constraints. Xiao et al. [8] proposed a deep learning-based prediction method to predict vehicle density and implemented a non-cooperative game-theoretic strategy based on regret matching to share the tasks of MEC servers in dense areas according to user task requirements. This approach takes into account the availability of nearby MEC server resources, but it has difficulty dealing with situations in which the nearby MEC server has no free resources. The above methods did not take into account the utilization of vehicle resources and the mobility of vehicles which comprise a constantly changing environment.

From the perspective of vehicle collaborative offloading, Huang et al. [9] proposed vehicular neighbor groups (VNGs) in which vehicles can ask for services for common interests, similar goals, and shared experiences. Similarly, Qiao et al. [10] proposed a collaborative task offloading scheme in which vehicles were divided into task-computation and task-offloading sub-clouds based on the similarity of tasks and the computational capability of the vehicles. With the help of these two sub-clouds, the number of similar tasks transferred to the MEC server can be effectively reduced. A dynamic pricing strategy is proposed in [11] to minimize the average cost of the MEC system under constraints on the quality of service by constantly adjusting the price based on the current system state. The above solutions utilize vehicle resources for collaborative offloading, but there is still a lack of vehicle collaborative partial offloading solutions.

A summary of recent studies on task offloading for VEC is given in Table 1.

Table 1. Summary of recent (2019–present) studies in task offloading for VEC.

Source	Feature	Offloading Type	Method	Result
Li (2019) [5]	Parking lot-assisted VEC	Full Offloading	Contract–Stackelberg Offloading Incentive; Backward Induction	Maximizes the utility of vehicles, operators, and parking lot agents
Yang (2019) [7]	MEC server-assisted mobility-aware task offloading	Full Offloading	Convex Optimization Algorithm	Reduces system cost while satisfying delay constraints
Xiao (2020) [8]	Heat-aware MEC cooperation	Full Offloading	Deep Learning; Non-Cooperative Game-Theoretic Strategy	Reduces system delay and enhances energy efficiency
Wang (2020) [6]	MEC network with secondary MEC servers	Full Offloading	Problem Decomposition; Heuristic Algorithm	Reduces system delay and improves system reliability
Dai (2022) [12]	VEC with edge-cloud Computing Cooperation	Full Offloading	Deep Q-Network	Reduces average delay
Liu (2023) [13]	Joint computation offloading and resource allocation	Full Offloading	Matching Theory-Based and Lagrangian-Based Algorithms	Improves system performance
Xu (2023) [14]	Joint task offloading and resource allocation	Full Offloading	Multi-Agent Distributed Distributional Deep Deterministic Policy Gradient (MAD4PG)	Improves system performance
This paper	Vehicle collaborative VEC	Partial Offloading	Double Deep Q-Network	Minimizes overall delay

While metaheuristic methods, like genetic algorithms, have been widely used for optimization problems, they can be computationally expensive. Recent research by Klar [15] demonstrates that well-performing reinforcement learning approaches can achieve similar or better solutions compared to top metaheuristic algorithms to cope with the dynamically changing network environment and selects collaborative vehicles that provide more stable computational offloading services by taking into account their mobility, remaining power, and computational capacity. At the same time, partial offloading allows the task to be divided into three sub-tasks to be processed locally by the MEC server or by collaborative vehicles, thus greatly reducing the burden on the MEC server and minimizing task latency.

3. System Model

3.1. Network Architecture of VEC

Figure 1 depicts a dynamic traffic scenario for a straight one-way road of a limited length. There are several roadside units (RSUs) at the roadside with a coverage radius of r and a MEC server with computational resources of f^{MEC} deployed on each RSU. In addition, a deep reinforcement learning (DRL)-based task offloading controller is also deployed on the MEC server to provide offloading decisions for vehicles. VTs report various information, such as position, speed, computational resources, etc., through vehicle-to-infrastructure (V2I) communication periodically. Depending on the demand for computing resources and the adequacy of its own computing capacity, each VT can be either a service requester or a service provider. We refer to these two different roles as offloading vehicles (OVs) and collaborative vehicles (CVs), respectively. We assume a time slot for each time and denote the time slot index by $t(t \in T = 0, 1, \cdots)$. It is assumed that the identity of each vehicle is stable and absolute in any time slot t and the speed and direction of each VT remain constant. Definitions of the notations used in this paper are given in Table 2.

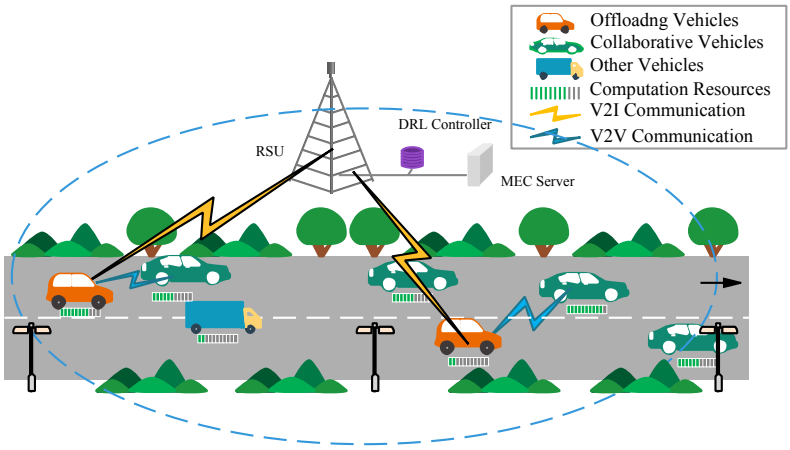

Figure 1. The vehicular edge computing network framework.

As illustrated in Figure 2, the collaborative task offloading process consists of eight steps:

1. The offloading vehicle generates a task and transmits a task request to the MEC server.
2. Upon receiving the request, the MEC server collects relevant information from both the offloading vehicle and potential collaborative vehicles.
3. Based on the system resource state, offloading targets, and available strategies, the MEC server calculates the optimal offloading strategy.
4. The MEC server informs the offloading vehicle and selected collaborative vehicles about the offloading decision.
5. Following the decision, the offloading vehicle divides the task and offloads portions to the MEC server and the designated collaborative vehicles.
6. The offloading vehicle, MEC server, and collaborative vehicles process their assigned task splits concurrently.

7. Once finished, the MEC server and collaborative vehicles return their processed results to the offloading vehicle.
8. Finally, the offloading vehicle aggregates the final results and provides feedback on the task execution.

Table 2. List of notations.

Notation	Definition	Notation	Definition
A, A^*	action space and optimal action set	$a_i, A_{i,t}$	offloading decision of task T_i; action set of vehicle i at time slot t
b_j, b^u, b^l	percentage of power remaining in vehicle j; upper and lower thresholds of remaining power	B^{V2I}, B^{V2V}	bandwidth values of V2I and V2V communication (Hz)
C_i	computational resources required for task i (cycles)	$V = V^O \cup V^C$	set of vehicles (union of offloading vehicles and collaborative vehicles)
D_i	size of task i (bit)	$\overline{d_i}$	average distance between vehicle i and RSU within RSU's coverage (meter)
$d_{i,j}, \overline{d_{i,j}}$	instantaneous and average distances between vehicles i and j (meter)	$d^{V2I}_{range}, d^{V2V}_{range}$	maximum communication distance between V2I and V2V
f^i, f^{MEC}	computational resources of vehicle i and MEC server (Hz)	G	transmission power of vehicle (Watt)
h	channel attenuation factor for uplinks	$l_i = (x_i, y_i)$	position of vehicle i (horizontal and vertical coordinates)
$H_i, H_{i,j}$	channel gain for communication from vehicle i to RSU and vehicle i to j	N_0	Gaussian white noise power (dB)
$p_L, p_M, p_{C,j}$	proportions of task T_i executed locally and offloaded to MEC servers and to collaborative vehicle j	$\overline{r_i^{V2I}}, \overline{r_{i,j}^{V2V}}$	average data transfer rate between $v_i \in V^O$ and RSU, v_i and $v_j \in V^C$ (bit/s)
t^u, t^l	upper and lower thresholds for time two vehicles remain within communication range	$tmax_i$	maximum tolerable delay of task T_i (second)
$t_i^{Loc}, t_i^{V2I}, t_{i,j}^{V2V}$	delay in local calculation; V2I and V2V offloading (second)	T_i	tasks for offloading vehicle $v_i \in V^O$
y_t, y'_t	predictions of main and target neural networks	$\gamma, \alpha, \varepsilon$	discount factor, learning rate, and exploration rate of greedy algorithms

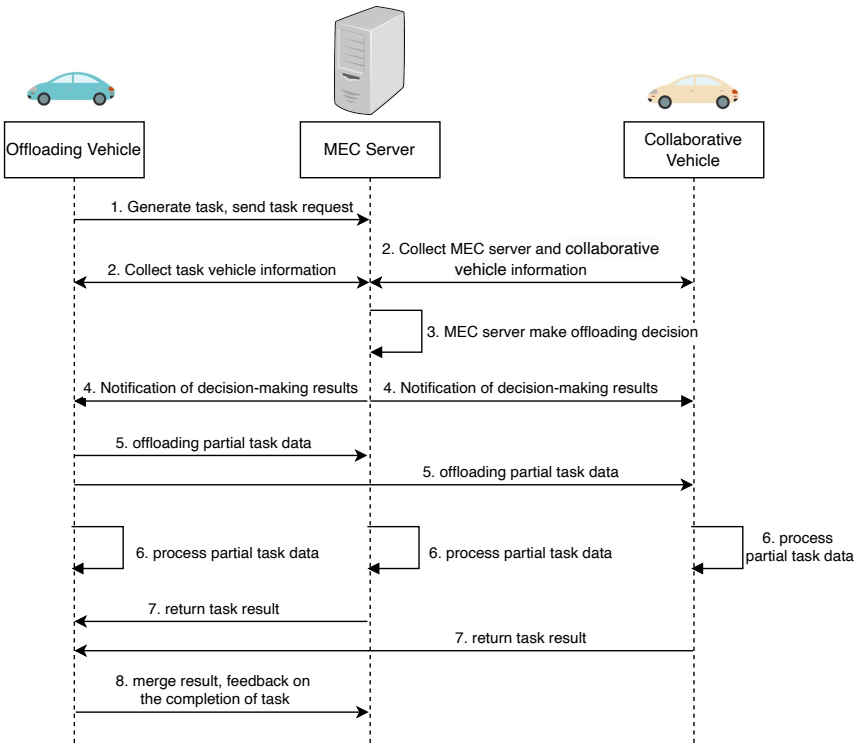

Figure 2. The task offloading process.

The set of all vehicles in the coverage area of an RSU is represented as $V = V^O \cup V^C = \{v_1, v_2, \cdots, v_n\}$, in which V^O and V^C are sets of offloading and collaborative vehicles, respectively. The available computational resources of any vehicle v_i are denoted by f^i. The

task generated by the offloading vehicle i is denoted by $T_i \triangleq (D_i, C_i, tmax_i)$, in which D_i, C_i, and $tmax_i$ denote the data size, required computational resources, and maximum tolerable delay of the task T_i, respectively. $V_i^C \subseteq V^C$ denotes the set of collaborative vehicles near vehicle i, and the available resources of collaborative vehicle j for offloading vehicle i are defined as f_i^j. Assuming that the RSU provides the origin coordinates, the center of the road is the x-axis, and the direction of travel of the vehicles is the positive direction; the position of vehicle i can be denoted as $l_i = (x_i, y_i)$. If vehicle i travels at a constant speed e_i in time slot t and the coverage radius of the RSU is r, after task generation, the time of the offloading vehicle i remaining within RSU coverage can be computed as follows:

$$T_i^{V2I} = \frac{r - x_i}{e_i}. \tag{1}$$

Let $T_{i,j}^{V2V}$ denotes the time elapsed from the generation of task T_i until the distance between offloading vehicle i and collaborative vehicle j reaches the maximum vehicle-to-vehicle (V2V) communication distance d_{range}^{V2V}. We obtain the following equation:

$$d_{range}^{V2V} = \sqrt{[(x_j + T_{i,j}^{V2V} \times v_j) - (x_i + T_{i,j}^{V2V} \times v_i)]^2 + (y_j - y_i)^2}. \tag{2}$$

Simplifying the above equation gives $T_{i,j}^{V2V} = {(\pm\sqrt{{d_{range}^{V2V}}^2 - (y_j - y_i)^2} - (x_j - x_i))}/{v_j - v_i}$. When $T_{i,j}^{V2V} \leq 0$, it means that the two vehicles cannot communicate with each other. Thus, the time the offloading vehicle i remains in the range of communication ($d_{i,j} < d_{range}^{V2V}$) with collaborative vehicle j can be expressed as follows:

$$T_{i,j}^{V2V} = \max\{\frac{\sqrt{{d_{range}^{V2V}}^2 - (y_j - y_i)^2} - (x_j - x_i)}{v_j - v_i}, \frac{-\sqrt{{d_{range}^{V2V}}^2 - (y_j - y_i)^2} - (x_j - x_i)}{v_j - v_i}, 0\}. \tag{3}$$

As each vehicle can only communicate with one vehicle at a time, the task T_i can be divided into three parts: local calculation, offloading to the MEC server, and offloading to a nearby collaborative vehicle. Let $a_i = [p_L, p_M, p_{C,j}]$ (in which $j \in V_i^C$ and $p_L + p_M + p_{C,j} = 1$) denote the offloading decision of vehicle i, in which p_L, p_M, and $p_{C,j}$ indicate the proportions of task T_i processed locally, offloaded to the MEC server, and offloaded to the collaborative vehicle j, respectively.

3.2. Collaborative Vehicle Discrimination Model

Any vehicle j, before it becomes a candidate collaborative vehicle, must satisfy three constraints in terms of mobility, available computational resources, and remaining power. To this end, three functions were designed to represent the impact of these three constraints on the likelihood of a vehicle being a collaborative candidate.

First, the distance between the candidate collaborative vehicle and the offloading vehicle should meet the V2V communication requirements during the task-offloading process. Function $P^T(i,j)$ represents the effect of mobility on the likelihood that vehicle j is a candidate collaborative vehicle for an offloading vehicle i:

$$P^T(i,j) = \begin{cases} 1 & T_{i,j}^{V2V} > t^u \\ \dfrac{T_{i,j}^{V2V} - t^l}{t^u - t^l} & t^l \leq T_{i,j}^{V2V} \leq t^u \\ 0 & T_{i,j}^{V2V} < t^l \end{cases}. \tag{4}$$

It can be seen that $P^T(i,j)$ is proportional to the time $T_{i,j}^{V2V}$ that the distance between the two vehicles remains within the maximum V2V communication distance. In this equation, t^l and t^u indicate the lower and upper thresholds for the time the two vehicles remain within the communication distance.

Secondly, the candidate collaborative vehicle should have sufficient computational resources to provide a service for the offloading vehicle. Function $P^F(i,j)$ refers to the

effect of the available computational resources of vehicle j on the likelihood that it will be a candidate collaborative vehicle for offloading vehicle i:

$$P^F(i,j) = \begin{cases} 1 & f_i^j > f^i \\ f_i^j / f^i & 0 < f_i^j < f^i \\ 0 & f_i^j \leq f_i^j \end{cases} . \quad (5)$$

From this equation, we can draw a conclusion that $P^F(i,j)$ is proportional to f_i^j, which represents the computational resources allocated to vehicle i by vehicle j. The more computational resources vehicle j can provide, the more likely it is to be a candidate collaborative vehicle for vehicle i.

Finally, a candidate collaborative vehicle should have sufficient power to provide a service. Function $P^B(i,j)$ describes the effect of the currently remaining power of vehicle j on its probability of being a candidate collaborative vehicle:

$$P^B(i,j) = \begin{cases} 1 & b_t^j > b^u \\ \dfrac{b_t^j - b^l}{b^u - b^l} & b^l \leq b_t^j \leq b^u \\ 0 & b_t^j < b^l \end{cases} . \quad (6)$$

We can see that $P^B(i,j)$ is proportional to vehicle j's remaining power. In this equation, b^u denotes the threshold at which vehicle j is willing to provide service when there is sufficient power, and $b^l (b^u > b^l)$ denotes the threshold at which vehicle j refuses to provide service when there is insufficient power.

In summary, $P^T(i,j)$, $P^F(i,j)$, and $P^B(i,j)$ denote the effects of mobility, available computational resources, and remaining power, respectively. Whether vehicle j can be a collaborative vehicle for vehicle i can be determined by the following matrix, mat:

$$mat(i,j) = \begin{cases} 1, & P^T(i,j), P^F(i,j), P^B(i,j) \geqslant \iota \\ 0, & P^T(i,j), P^F(i,j), P^B(i,j) < \iota \end{cases} . \quad (7)$$

Let ι be the threshold of the above functions P^T, P^F, and P^B. When $mat(i,j) = 1$, vehicle j is a candidate collaborative vehicle for the offloading vehicle i, and $j \in V_i^C$. Otherwise, $j \notin V_i^C$.

3.3. Computation Model

3.3.1. Local Calculation

If $p_L > 0$ in the offloading decision a_i, the delay of a task with a proportion p_i calculated locally can be computed as follows:

$$t_i^{Loc}(a_i) = \frac{p_L \times C_i}{f^i}. \quad (8)$$

3.3.2. Offloading to the MEC Server

If $p_M > 0$ in the offloading decision a_i, a proportion p_M of the task will be offloaded to the MEC server from vehicle $v_i \in V^O$ via vehicle-to-infrastructure (V2I) communication. The MEC server then processes the task and returns the task result to vehicle i. Usually, the size of the calculated result is much smaller than that of the input data; thus, the transmission time of computation results can be ignored. The delay in a task with a proportion of p_M offloaded to a MEC server is computed as follows:

$$t_i^{V2I}(a_i) = \frac{p_M \times D_i}{r_i^{V2I}} + \frac{p_M \times C_i}{f^{MEC}}. \quad (9)$$

The vehicle and RSU (MEC) communicate with each other via a direct C-V2X (Cellular-V2X) link. In this paper, the transmission link between the vehicle and RSU is set as a flat

Rayleigh fading channel without considering channel interference. According to Shannon's formula, the average uplink data transfer rate of vehicle i from the time of task generation to driving out of RSU coverage can be calculated as follows:

$$\overline{r_i^{V2I}} = B^{V2I} \log_2(1 + \frac{GH_i}{N_0}). \tag{10}$$

In which B^{V2I} denotes the uplink channel bandwidth, N_0 is the Gaussian white noise power, G is the transmitting power of the on-board device, $H_i = h^2 L^{V2I}$ is the channel gain parameter, h is the channel fading factor of the uplink, $L^{V2I} = 100.7 + 23.5 \log_{10}(\overline{d_i})$ refers to path loss between the vehicle and the RSU [16], and $\overline{d_i}$ represents the average distance between vehicle i and the RSU during the task-processing period (from task generation to the maximum tolerable delay), defined as $\overline{d_i} = \frac{\int_t^{t+tmax_i} d_i(t)dt}{T_i^{V2I}}$.

3.3.3. Offloading to Collaborative Vehicle

If $p_{C,j} > 0$ in the offloading decision a_i, a proportion $p_{C,j}$ of the task will be offloaded to a nearby vehicle $j \in V_i^C$ via V2V communication, and the delay of the task can be defined as follows:

$$t_{i,j}^{V2V}(a_i) = p_{C,j} \times \frac{D_i}{\overline{r_{i,j}^{V2V}}} + p_{C,j} \times \frac{C_i}{f_i^j}. \tag{11}$$

In short-range wireless communication scenarios, the IEEE 802.11p protocol is commonly used for V2V communication. The average data transfer rate between the offloading vehicle i and the collaborative vehicle j can be expressed as

$$\overline{r_{i,j}^{V2V}} = B^{V2V} \log_2(1 + \frac{GH_{i,j}}{N_0}). \tag{12}$$

In which $H_{i,j} = h^2 L^{V2V}$ is the channel gain parameter, the path loss [17] $L^{V2V} = 63.3 + 17.7 \log_{10}(\overline{d_{i,j}})$ dB, and $\overline{d_{i,j}}$ is the average distance between V_i^O and V_j^C when they remain within communication range.

4. Problem Formulation

This paper assumes all vehicle terminals use the collaborative partial offloading strategy. Under this assumption, the computational delay for each task is the maximum computational delay of each task portion, defined as follows:

$$t_i(a_i) = \max\{t_i^{Loc}(a_i), t_i^{V2I}(a_i), t_{i,j}^{V2V}(a_i)\}. \tag{13}$$

To improve the success rate of offloading, this paper aims to minimize the task delay. The vehicle collaborative partial offloading problem in a VEC scenario can be formulated as follows:

$$\begin{aligned} &\min \sum_{i \in V^O} t_i(a_i) \\ &= \min \sum_{i \in V^O} \max\{t_i^{Loc}(a_i), t_i^{V2I}(a_i), t_{i,j}^{V2V}(a_i)\}. \end{aligned} \tag{14}$$

$$s.t.\ a_i = [p_L, p_M, p_{C,j}]_{j \in V_i^C, p_L, p_M, p_{C,j} \geqslant 0}, \tag{15}$$

$$p_L + p_M + p_{C,j} = 1, \tag{16}$$

$$t_i(a_i) \leqslant tmax_i, \tag{17}$$

$$t_i^{V2I}(a_i) \leqslant T_i^{V2I}, \tag{18}$$

$$t_{i,j}^{V2V}(a_i) \leqslant T_{i,j}^{V2V}. \tag{19}$$

Problem (14) can be solved by determining the decision vector a_i. Five constraints govern this problem:

- Constraint (15) defines the offloading decision for task T_i.
- Constraint (16) ensures that task T_i is processed in its entirety.
- Constraint (17) guarantees that the processing time for each task meets the specified latency requirement.
- Constraint (18) ensures that the RSU transmits results back to vehicle i before it moves out of communication range.
- Constraint (19) dictates that vehicle $j \in V_i^C$ returns the results to vehicle i before they can no longer communicate.

5. Solution

Problem (14) is difficult to solve because it is a non-convex problem with multiple variables and many constraints. In this section, the offloading problem is converted into a Markov Decision Process (MDP), and then a Double Deep Q-Network (DDQN)-based vehicle collaborative partial computation offloading algorithm is proposed.

5.1. Markov Decision Process

A MDP can be formulated as $\{S, A, P(s_{t+1}|s_t, a), R(s, a)\}$, in which set S denotes the state space of the environment, set A represents the action space, $P(s_{t+1}|s_t, a)$ refers to the probability of state transfer from s_t to s_{t+1} after performing action a, and $R(s, a)$ indicates the immediate reward received for taking action a in state s. The goal of the MDP is to find a policy that maximizes the long-term expected reward. This can be expressed as the sum over the time period T, discounted by a factor γ between 0 and 1, of the rewards received: $\sum_{t=0}^{T} \gamma^t R(s, a)$. The discount factor γ balances immediate rewards with future rewards where higher values prioritize long-term gains. Then the states, actions, and rewards within the MDP can be defined as follows:

- State space: The state of vehicle V_i^O is described by its location, speed, on-board computing power, and the computing power available from the nearby MEC server and candidate vehicles:

$$s_i = \left\{ l_i, v_i, f^i, f^{MEC}, f_i^j \right\}. \tag{20}$$

  In which $l_i = \{x_i, y_i\}$ is the location of vehicle i. The state space S of the whole system is then composed of the location, speed, on-board computing power, MEC server computing power, and the determining matrix of all vehicles, mat.
- Action space: Within the proposed VEC network, a deep reinforcement learning (DRL)-based controller resides on the RSU, acting as the agent that interacts with the environment and generates decisions. In any given state, each OV chooses a specific offloading decision from a set of available options as action a_i. Collectively, the set of all possible offloading decisions for all OVs forms the system's action space:

$$A = \{a_1, a_2, \cdots, a_n\}. \tag{21}$$

- Reward function: Since minimizing the total time delay is our goal, the reward function is designed to be directly proportional to the negative of the delay:

$$r = -\frac{t_i(a_i)}{tmax_i}. \tag{22}$$

  To prevent the learning process from becoming stuck on suboptimal solutions, this paper proposes a reward normalization scheme for OVs. This scheme scales all OV action rewards to a range between -1 and 0. Additionally, any invalid action selection incurs a minimum reward of -1.

5.2. Computation Offloading Based on Reinforcement Learning

Since it is difficult to obtain the state transfer probability $P(s_{t+1}|s_t, a)$ in the MDP, the model-free Q-learning algorithm can be used to solve the decision problem in VEC. In Q-learning, the Q-value function $Q(s, a)$ indicates the expected reward of action a under a given state s, expressed as

$$Q^{\pi}(s, a) = \mathbb{E}(\sum_{t=0}^{T} \gamma^t R(s, a)|\pi). \tag{23}$$

A strategy π is considered optimal if it maximizes the expected reward for all possible states encountered by the agent. According to the Bellman equation, the optimal Q-value can be estimated as

$$Q^*(s, a) = \mathbb{E}'\left[R(s, a) + \gamma \max_{a'} Q^*(s', a')\right]. \tag{24}$$

Generally, the Q function is obtained iteratively through quintuple (s, a, r, s', a') in the next time state s', and the updated Q function can be expressed as

$$Q'(s, a) = (1 - \alpha)Q(s, a) + \alpha(R(s, a) + \gamma \max_{a' \in A} Q^*(s', a')). \tag{25}$$

where $\alpha \in [0, 1]$ denotes the learning rate. By setting an appropriate learning rate and repeatedly updating the Q-value, $Q(s, a)$ will converge to $Q^*(s, a)$. The best action set A^* can be obtained by repeatedly updating the Q value from each state a.

5.3. Computation Offloading Based on Deep Reinforcement Learning

In highly dynamic traffic scenarios, complexity arises from the vast state space (possible situations the agent encounters) and action space (possible maneuvers the agent can make). It becomes impractical to store each state–action pair in a traditional Q-table. This is where Deep Q-Networks (DQNs) come in. DQN leverages a Deep Neural Network (DNN) to approximate the Q-function, significantly reducing memory requirements compared to a Q-table. Additionally, DQNs utilize a memory buffer to store past experiences. During training, a subset of these experiences is randomly sampled to reduce correlations between training examples and improve learning efficiency. This approach makes DQNs well suited to highly dynamic environments.

However, a well-known challenge with DQNs is overestimation bias. This arises because the target Q-value used for training is the maximum of the estimated Q-values for the next state, all of which are calculated by the same network being trained. This can lead to overly optimistic estimates of future rewards, ultimately hindering the agent's ability to learn the optimal policy. The Double Deep Q-Network (DDQN) algorithm addresses this issue by decoupling the selection of actions for the target Q-value from its actual calculation. This means that in the DDQN architecture, two distinct networks are employed: main network for action selection and target network for Q-value evaluation. This decoupling helps reduce overestimation bias and improve the accuracy of Q-value estimates, ultimately leading to better policy learning. Thus, this paper proposes formulating the computational offloading problem as a parameter optimization problem suitable for solving with DDQN. In DDQN, the estimated value Q of the main network is as follows:

$$y = r' + \gamma \max_{a \in A} Q(s', \max_{a \in A} Q(s', a, \theta), \theta). \tag{26}$$

The actual value Q of the target network is

$$y' = r' + \gamma \max_{a \in A} Q(s', \max_{a \in A} Q(s', a, \theta), \theta'). \tag{27}$$

The loss function is defined as follows:

$$Loss(\theta) = (y' - y)^2. \tag{28}$$

The key steps involved in the DDQN-based collaborative partial computation offloading strategy for VEC is illustrated as follows (refer to Algorithm 1 for details).

- Initialization (Line 1): The algorithm begins by initializing the parameters for both the main and target networks. Additionally, an experience memory is created to store past interactions with the environment.
- Action Selection (Lines 5–9): At each time step, the algorithm employs the Epsilon-Greedy strategy to select an action a.
- Environment Interaction (Lines 10–11): The chosen action is then taken in the environment ($env(a)$). The environment responds with the next state (S_{t+1}) and a reward signal (r) indicating the outcome of the action (Line 10). The experience gained from this interaction (S_t, a, r, S_{t+1}) is stored in the experience memory D (Line 11).
- Network Update (Lines 12–17): At each time step, a mini-batch (size U) of experiences is randomly sampled from the experience memory and then used to update the parameters of the main network using the Bellman equation (Lines 12–14). The target network's parameters are periodically (every K time steps) updated with a copy of the main network's parameters (Lines 15–17).
- Offloading Decision (Line 19): At the end of each episode, the final optimal strategy is obtained.

Algorithm 1 Collaborative Partial Computation Offloading Algorithm based on DDQN

Require: U—sample size of experience replay. K—target network replacement frequency.

1: Initialize network parameters θ, θ' and experience memory D;
2: **for** each episode **do**
3: according to Equation (7), calculate determining matrix **mat** to obtain S_1;
4: **for** $t = 1$ to T **do**
5: **if** rand()>ε **then**
6: $a = \text{rand}(a)$;
7: **else**
8: $a = arg\max_{a_i \in A} Q(S_t, a \mid \theta)$;
9: **end if**
10: $S_{t+1}, r \leftarrow$ **env(***a***)**;
11: store (S_t, a, r, S_{t+1}) to experience memory D;
12: randomly select U samples (S_j, a_j, r_j, S_{j+1}) from D ;
13: according to Equation (26), calculate target Q value y';
14: according to Equation (27), calculate training loss to update θ;
15: **if** $t \bmod K \equiv 0$ **then**
16: update parameters θ' of target network: $\theta' \leftarrow \theta$;
17: **end if**
18: **end for**
19: select $A^* \sim \pi_\theta(S_t)$;
20: **end for**

6. Experiment and Analysis

6.1. Environment

Our experiments were implemented using TensorFlow. We simulated a one-way road scenario with 30 vehicles traveling at speeds between 10 and 20 m per second. The road was 400 m long and 10 m wide. We set the simulation to run for 300 episodes, with each episode consisting of 50 time steps ($T = 50$). Additional experiment parameters (hyperparameters) and their values/ranges are detailed in Table 3.

Table 3. Values and ranges of parameters and hyperparameters.

Parameter	Value/Range	Parameter	Value/Range
d_{range}^{V2I}	200 m	d_{range}^{V2V}	15 m
C_i	(1, 1000) Megacycles	D_i	$[0, 1]$ MB
f^{MEC}	10 GHz	f^i	$[0.5, 1]$ GHz
B^{V2I}	15 MHz	B^{V2V}	10 MHz
$tmax_i$	$(0, 1]$ s	N_0	-100 dB
t^u	$tmax_i$	t^l	$tmax_i/2$
b^u	80%	b^l	20%
h	1	γ	0.9
ε	0.9	α	0.01
U	3000	K	100

6.2. Analysis

To evaluate our proposed scheme, we compared it with four alternatives: Only Local, MEC Offloading, Collaborative Full Offloading, and Collaborative Partial Offloading (our scheme). In the Only Local scheme, all offloading vehicles process tasks locally. In the MEC Offloading scheme, tasks are divided into two parts. One part is processed locally on the vehicle, while the other part is offloaded to nearby MEC servers. Vehicles in the Collaborative Full Offloading scheme can choose to process tasks locally, offload them to MEC servers, or offload them to collaborative vehicles within the network.

Figure 3 depicts the convergence behavior of our algorithm with respect to the discount factor (γ). As shown, the algorithm converges relatively quickly when γ is set to a suitable value (around 75 episodes when $\gamma = 0.9$). However, the choice of γ is crucial:

- High discount factor (e.g., γ = 0.99): While it encourages considering future rewards, a very high discount factor can lead to convergence difficulties. This is because the agent has to consider the impact of actions over many future steps, making training more complex.
- Low discount factor (e.g., γ = 0.4): A very low discount factor can lead to faster convergence but also smaller long-term rewards. This happens because the agent prioritizes immediate rewards, neglecting the potential benefits of actions with delayed payoff. In simpler terms, the agent becomes less focused on the long-term consequences of its decisions.

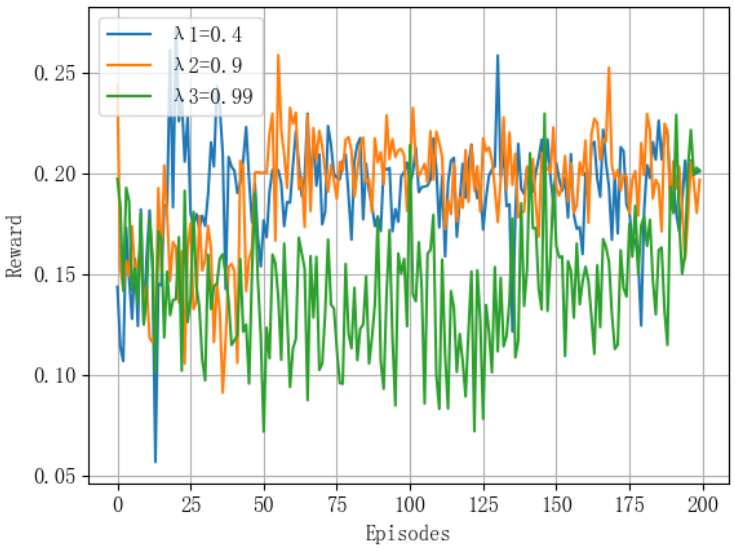

Figure 3. Convergence results.

Figure 4 illustrates how the task failure rate for different offloading schemes varies with the vehicle task generation probability. As expected, the task failure rate increases for all schemes as the task generation probability grows. This is a natural consequence of the system's finite computational resources becoming overwhelmed with an increasing workload. Notably, our proposed scheme consistently exhibits a lower failure rate compared to the alternatives. This advantage stems from our scheme's ability to leverage collaborative vehicles for partial offloading. By distributing the task splits across three processing units (a local vehicle, MEC servers, and collaborative vehicles), our approach significantly reduces the burden on any single unit, leading to a lower overall failure rate compared to other schemes.

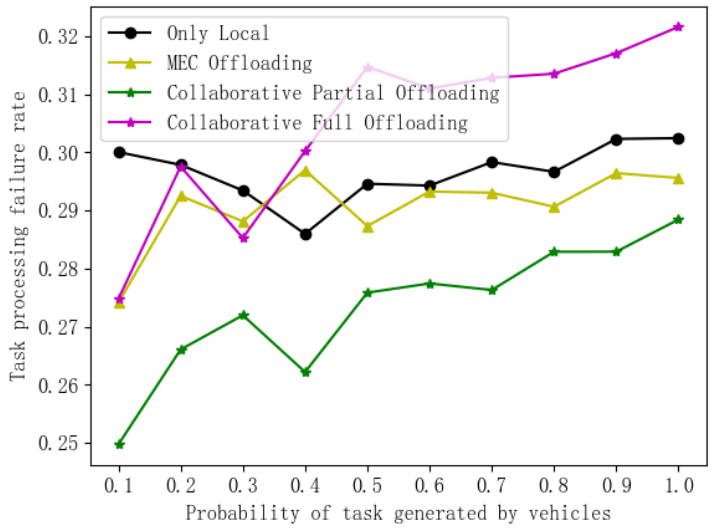

Figure 4. Comparison of failure rates under different task generation probabilities.

Figure 5 illustrates the average delay in tasks for different offloading schemes under varying vehicle task generation probabilities. Our proposed scheme consistently exhibits a lower average task delay compared to the alternatives. This advantage stems from the efficient use of computational resources through task division and partial offloading.

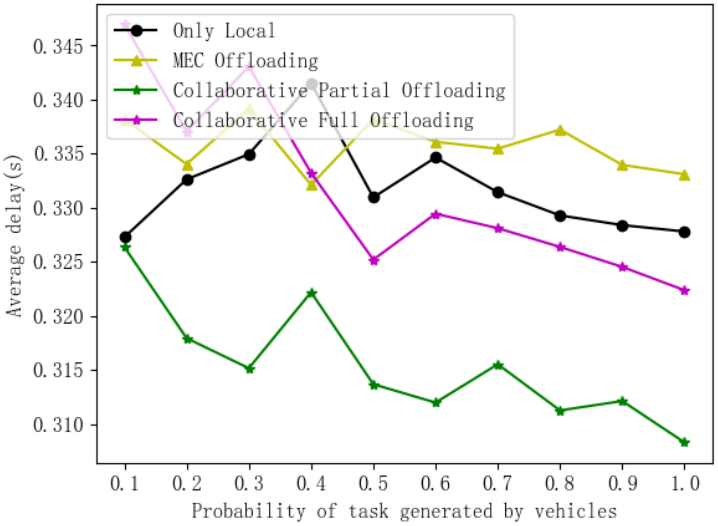

Figure 5. Comparison of average delays under different task generation probabilities.

Figure 6 reveals how the probability of a vehicle generating a task influences task allocation in our scheme. As this probability increases, the proportion of tasks offloaded to collaborative vehicles decreases. At the same time, the proportions of tasks executed locally

and offloaded to MEC servers are higher than those of CV offloading and failures. The reasons are twofold: First, in high-speed scenarios, OVs might move out of the communication range before receiving the results of tasks offloaded to MEC servers or CVs. This limits the feasibility of offloading at higher speeds. Second, with an increasing task generation probability, the total number of tasks in the system grows but the MEC server's resources and the number of suitable CV candidates remain constant. This increased workload strains the MEC server and the limited pool of CVs, making offloading less successful.

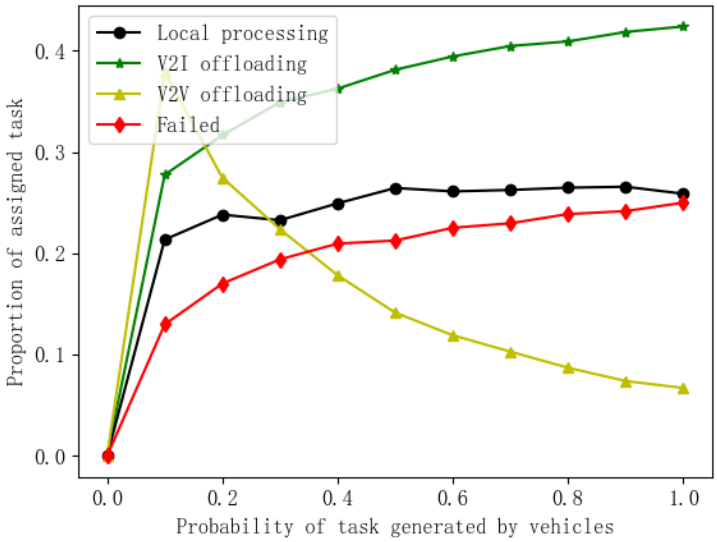

Figure 6. Comparison of task allocation proportions under different task generation probabilities.

From Figure 7, we can observe that the rate of task failures drops as the maximum tolerable time increases because the relaxed delay requirement alleviates pressure on MEC servers and CVs. Furthermore, the proportion of local processing and V2V offloading increases. This is because even with limited computational resources, CVs can potentially meet relaxed deadlines for tasks with higher tolerable delays, making local processing or offloading to OVs more viable.

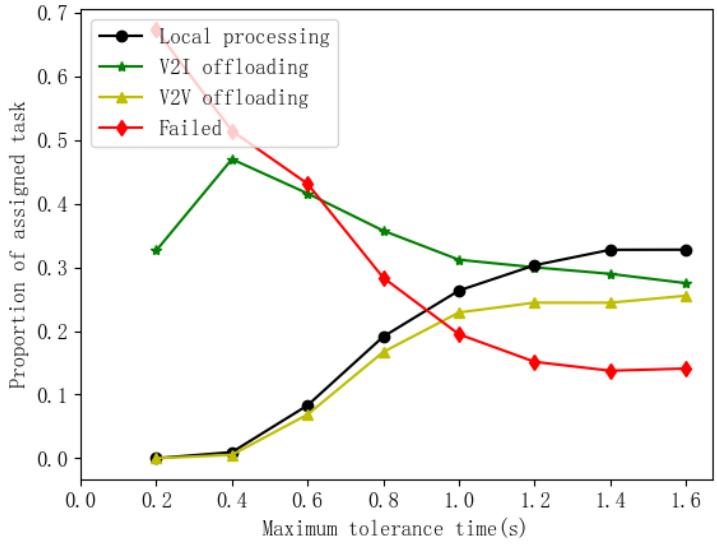

Figure 7. Comparison of task allocation proportions under different maximum tolerance times.

In Figure 8, it can be seen that as the maximum tolerance time for tasks increases, the task failure rate decreases for all offloading schemes. Meanwhile, our scheme demonstrates

a clear advantage in failure rate reduction, particularly when the tolerance time is low. Compared to other approaches, it achieves a significant reduction (around 2%) in the failure rate under tighter latency constraints. However, with looser latency constraints, the effect of reducing the failure rate is less pronounced as the main objective of this paper is to reduce latency rather than the task failure rate.

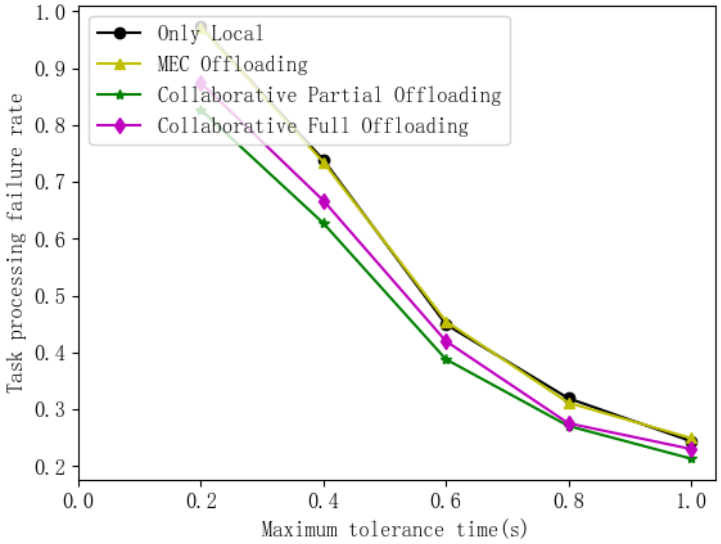

Figure 8. Comparison of failure rates under different maximum tolerance times.

The experimental results demonstrate that the proposed vehicle collaborative partial offloading scheme based on DDQN significantly outperforms the schemes of local execution only, offloading to the MEC server, and full collaborative vehicle offloading in terms of the task processing failure rate and average task execution delay. When the maximum tolerable delay of a task increases, the success allocation ratio of the task also increases steadily. Therefore, the proposed scheme is a valid and effective solution.

7. Conclusions

This paper tackles the challenge of limited processing power in a single, statically deployed MEC server handling demanding in-vehicle computation tasks. First, we propose a vehicle collaborative partial offloading strategy. In this approach, the computational task is divided into three parts processed by the vehicle terminal itself, by MEC servers, and by collaborative vehicles, respectively. To identify suitable collaborative vehicles, we investigate the impact of factors like a vehicle's location, available resources, and other states on the likelihood of the vehicle becoming a collaborative vehicle candidate. Second, we develop a model for the partial offloading process and formulate the delay minimization problem with a delay tolerance constraint as a Markov Decision Process (MDP). Finally, to adapt to dynamic traffic scenarios, we design and implement a partial computational offloading scheme using a Double Deep Q-Network (DDQN) algorithm. Simulation results show that this scheme significantly outperforms traditional approaches without vehicle collaboration. It achieves this by reducing latency and lowering the failure rate by around 2%, especially under stricter latency constraints. While this paper proposes a novel strategy for collaborative task offloading, there are limitations that can be addressed in future work:

- Currently, the model only considers the offloading vehicle's local MEC server and nearby collaborative vehicles. This can lead to increased task failure rates when dealing with a high task volume due to latency constraints. In future research, we can explore incorporating surrounding MEC servers to enable task relay and load balancing, potentially reducing task failure rates.
- The current approach prioritizes minimizing latency. However, other crucial factors like the task failure rate also exist. We can investigate combining these metrics into

a unified objective function to achieve a more balanced and optimized task offloading strategy.

- Incentive mechanisms for collaborative vehicles are not considered in this paper. By introducing well-designed incentive mechanisms, we can achieve a win–win situation for both collaborative vehicles and MEC servers. These incentives could be designed to maximize collaborative vehicles' benefits. This will help attract more vehicles to become collaborative vehicles and improve the success rate of task offloading. The incentive mechanism could also consider maximizing the benefits of service providers (MEC operators). This will help alleviate the burden on MEC servers.

Overall, the deep reinforcement learning method employed in this paper demonstrates promise in addressing the aforementioned limitations. We will further explore alternative approaches for enhanced performance.

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

Funding: This research was jointly supported by the Qin Xin Talents Cultivation Program, Beijing Information Science & Technology University (grant number QXTCP C202111), and Promoting the Diversified Development of Universities—College Students' Innovation and Entrepreneurship Training Program, Beijing Information Science & Technology University, Computer School (grant number 5112410852).

Data Availability Statement: The data used to support the findings of this study were simulated by the algorithm proposed in this article, and the parameters used in the simulation are included within the article.

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

References

1. Qi, Q.; Ma, Z. Vehicular edge computing via deep reinforcement learning. *arXiv* **2018**, arXiv:1901.04290.
2. Zhang, K.; Mao, Y.; Leng, S.; Maharjan, S.; Zhang, Y. Optimal delay constrained offloading for vehicular edge computing networks. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–6.
3. Guo, H.; Liu, J. Collaborative Computation Offloading for Multiaccess Edge Computing Over Fiber–Wireless Networks. *IEEE Trans. Veh. Technol.* **2018**, *67*, 4514–4526. [CrossRef]
4. Zhou, Z.; Feng, J.; Tan, L.; He, Y.; Gong, J. An Air-Ground Integration Approach for Mobile Edge Computing in IoT. *IEEE Commun. Mag.* **2018**, *56*, 40–47. [CrossRef]
5. Li, Y.; Yang, B.; Chen, Z.; Chen, C.; Guan, X. A Contract-Stackelberg Offloading Incentive Mechanism for Vehicular Parked-Edge Computing Networks. In Proceedings of the 2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring), Kuala Lumpur, Malaysia, 28 April–1 May 2019; pp. 1–5.
6. Wang, H.; Peng, Z.; Pei, Y. Offloading Schemes in Mobile Edge Computing With an Assisted Mechanism. *IEEE Access* **2020**, *8*, 50721–50732. [CrossRef]
7. Yang, C.; Liu, Y.; Chen, X.; Zhong, W.; Xie, S. Efficient Mobility-Aware Task Offloading for Vehicular Edge Computing Networks. *IEEE Access* **2019**, *7*, 26652–26664. [CrossRef]
8. Xiao, Z.; Dai, X.; Jiang, H.; Wang, D.; Chen, H.; Yang, L.; Zeng, F. Vehicular Task Offloading via Heat-Aware MEC Cooperation Using Game-Theoretic Method. *IEEE Internet Things J.* **2020**, *7*, 2038–2052. [CrossRef]
9. Huang, X.; Yu, R.; Kang, J.; He, Y.; Zhang, Y. Exploring mobile edge computing for 5G-enabled software defined vehicular networks. *IEEE Wirel. Commun.* **2017**, *24*, 55–63. [CrossRef]
10. Qiao, G.; Leng, S.; Zhang, K.; He, Y. Collaborative Task Offloading in Vehicular Edge Multi-Access Networks. *IEEE Commun. Mag.* **2018**, *56*, 48–54. [CrossRef]
11. Han, D.; Chen, W.; Fang, Y. A Dynamic Pricing Strategy for Vehicle Assisted Mobile Edge Computing Systems. *IEEE Wirel. Commun. Lett.* **2018**, *8*, 420–423. [CrossRef]
12. Dai, F.; Liu, G.; Mo, Q.; Xu, W.; Huang, B. Task offloading for vehicular edge computing with edge-cloud cooperation. *World Wide Web* **2022**, *25*, 1999–2017. [CrossRef]
13. Liu, S.; Tian, J.; Zhai, C.; Li, T. Joint computation offloading and resource allocation in vehicular edge computing networks. *Digit. Commun. Netw.* **2023**, *9*, 1399–1410. [CrossRef]

14. Xu, X.; Liu, K.; Dai, P.; Jin, F.; Ren, H.; Zhan, C.; Guo, S. Joint task offloading and resource optimization in NOMA-based vehicular edge computing: A game-theoretic DRL approach. *J. Syst. Archit.* **2023**, *134*, 102780. [CrossRef]
15. Klar, M.; Glatt, M.; Aurich, J.C. Performance comparison of reinforcement learning and metaheuristics for factory layout planning. *CIRP J. Manuf. Sci. Technol.* **2023**, *45*, 10–25. [CrossRef]
16. Luoto, P.; Bennis, M.; Pirinen, P.; Samarakoon, S.; Horneman, K.; Latva-aho, M. Vehicle clustering for improving enhanced LTE-V2X network performance. In Proceedings of the 2017 European Conference on Networks and Communications (EuCNC), Oulu, Finland, 12–15 June 2017; pp. 1–5.
17. Karedal, J.; Czink, N.; Paier, A.; Tufvesson, F.; Molisch, A.F. Path Loss Modeling for Vehicle-to-Vehicle Communications. *IEEE Trans. Veh. Technol.* **2011**, *60*, 323–328. [CrossRef]