electronics

MDPI

Article

A Delay-Optimal Task Scheduling Strategy for Vehicle Edge Computing Based on the Multi-Agent Deep Reinforcement Learning Approach

Xuefang Nie, Yunhui Yan *, Tianqing Zhou, Xingbang Chen and Dingding Zhang

School of Information Engineering, East China JiaoTong University, Nanchang 330013, China
* Correspondence: yunhuiyan@ecjtu.edu.cn

Abstract: Cloudlet-based vehicular networks are a promising paradigm to enhance computation services through a distributed computation method, where the vehicle edge computing (VEC) cloudlet are deployed in the vicinity of the vehicle. In order to further improve the computing efficiency and reduce the task processing delay, we present a parallel task scheduling strategy based on the multi-agent deep reinforcement learning (DRL) approach for delay-optimal VEC in vehicular networks, where multiple computation tasks select the target threads in a VEC server to execute the computing tasks. We model the target thread decision of computation tasks as a multi-agent reinforcement learning problem, which is further solved by using a task scheduling algorithm based on multi-agent DRL that is implemented in a distributed manner. The computation tasks, with each selection acting on the target thread acting as an agent, collectively interact with the VEC environment and receive observations with respect to a common reward and learn to reduce the task processing delay by updating the multi-agent deep Q network (MADQN) using the obtained experiences. The experimental results show that the proposed DRL-based scheduling algorithm can achieve significant performance improvement, reducing the task processing delay by 40% and increasing the processing probability of success for computation tasks by more than 30% compared with the traditional task scheduling algorithms.

Keywords: task scheduling; deep reinforcement learning; vehicle edge computing; Internet of Vehicles

Citation: Nie, X.; Yan, Y.; Zhou, T.; Chen, X.; Zhang, D. A Delay-Optimal Task Scheduling Strategy for Vehicle Edge Computing Based on the Multi-Agent Deep Reinforcement Learning Approach. *Electronics* **2023**, *12*, 1655. https://doi.org/10.3390/electronics12071655

Academic Editor: Rameez Asif

Received: 23 February 2023
Revised: 23 March 2023
Accepted: 29 March 2023
Published: 31 March 2023

1. Introduction

With the rapid improvement in the data rate in 5G communications, a significant number of computation-sensitive services, such as AR and VR, real-time navigation, and autonomous driving, are emerging [1–4]. Autonomous driving technology, including target detection, information perception, and intelligent decision making, requires computation resources to process various tasks within a limited latency [5]. However, the computation capabilities of the vehicular terminals are generally limited and cannot meet real-time task processing requirements [6,7]. Although the remote computation cloud has enough computation resources to provide efficient computing services, the long-distance data transmission of the task files between the remote cloud and the local vehicular terminals will lead to significant communication delays [8,9]. To address this problem, vehicle edge computing (VEC) is widely recognized as a promising solution to enhance computing service performance, where the computation resources are pushed to the radio access networks and provide computing service close to the vehicular terminals [10–12].

From the cost perspective of network operators, the computation resources of VEC clouds may be limited [13,14]. During peak traffic, offloading tasks from vehicular terminals increase sharply. As a result, the VEC cloud will be overloaded, and performance degradation in task processing is incurred [15,16]. In order to cope with this problem, an efficient task scheduling strategy based on VEC servers is indeed required [17]. To this end,

we apply efficient deep reinforcement learning (DRL) to design a task scheduling algorithm based on multi-agent deep Q networks (MADQN) for improving computing efficiency and thereby reducing the task processing delay.

Current works generally assume that the computation resources of the server are enough to process the offloading tasks from vehicular terminals [6,7]. However, as mentioned above, the computation resources configured in the VEC server are generally limited. Especially during peak periods, the VEC server is overloaded, which will prolong the task processing delay because of the increase in the queuing time. In order to enhance the computing efficiency of the VEC server, this paper solves the task scheduling problem to improve the execution efficiency of VEC servers in mobile edge computing networks. We assume that each VEC server has multiple threads and consider a parallel task scheduling scheme. We model the target thread decision of computation tasks as a multi-agent DRL problem, which is then solved by using a DRL-based approach to improve the computing efficiency and reduce the task processing delay. The main contributions of this paper are summarized as follows:

- We model the task decision of the target thread as a multi-agent DRL problem and employ recent progress in DRL to develop a distributed task decision algorithm based on the MADQN framework, which enhances the computing efficiency of the VEC server.
- We illustrate that via an appropriate training mechanism with a time-related reward, the task decisions can learn from interactions with the VEC environment and decide upon a clever strategy in a distributed manner that simultaneously ensures the rapid convergence of the proposed algorithm and reduces the task processing delay.
- We use the intensive and diversified computing task data to carry out simulation experiments, and the experiment results demonstrate the effectiveness of the proposed algorithm. Compared with the existing benchmark algorithms, the DRL-based algorithm proposed in this paper can obtain a lower system delay and has relatively stable performance gain in dense multiple-task computing scenarios.

The remainder of the paper is summarized as follows. We discuss the related work in Section 2. Then, we introduce the system model, which includes the task-computing model and problem statement. In Sections 4 and 5, we describe the MADQN algorithm framework of task scheduling. Finally, we show the simulation results in Section 6 and conclude our work in Section 7.

2. Related Work

Currently, research on task scheduling for vehicular networks has received extensive attention. However, most existing works highly depend on the static system model. The authors of [18] proposed a genetic algorithm-based approach to minimize the weighted sum of the service time and energy consumption by jointly optimizing the task caching and computation offloading in VEC networks. The authors of [19] applied Lyapunov optimization technology and the greedy heuristics approach to minimize the response time in caching-assisted vehicular edge computing. Meanwhile, energy consumption was taken into account. The authors of [20] proposed a heuristic algorithm based on the ant colony algorithm to solve the service arrangement problem among fog computing nodes. However, the accuracy of both intelligent algorithms and heuristic algorithms greatly depends on accurate system models. Practically, the offloading tasks from vehicular terminals belong to a dynamical process, and thus the traditional static models cannot capture the dynamic features of offloading tasks [21]. Deep reinforcement learning, integrating the perceptual ability of deep learning (DL), and the decision making ability of reinforcement learning (RL) can be used to solve the decision-making problem in complex environments [22,23].

At present, many scholars have used the DRL method to solve the problem of an optimal task scheduling strategy [24–27]. The authors of [24] used the Markov decision process (MDP) to establish a task scheduling model to solve the problems of task processing delay and system energy consumption minimization in VEC task scheduling scenarios, and then they

optimized the task scheduling scheme through the DRL method. The experiment results verified the effectiveness of the DRL method in solving complex decision-making problems. The authors of [25] used the DRL method to optimize the scheduling strategy of joint wireless transmission and computing resources for the vehicle-road cloud cooperation scenario to reduce the task processing delay. The authors of [26] proposed a dynamic framing offloading algorithm based on a double deep Q network to minimize the total delay and waiting time of computing offloading in vehicular edge computing networks. The authors of [27] used the DRL method to design an optimized content caching scheme under the high mobility of vehicles and dynamic wireless channels. However, with the development of vehicular networks, the computing dimension of task files is increasing sharply, and the traditional DRL methods cannot adapt to scenarios of highly complex task computing [28,29].

Fortunately, artificial intelligence algorithms based on multi-agent deep reinforcement learning (MADRL) can efficiently solve complex computation problems [30]. The authors of [23] applied the MADRL method to obtain a spectrum allocation scheme of vehicle-to-infrastructure (V2I) links for maximizing the overall network throughout. The authors in that paper assumed that each vehicle-to-vehicle (V2V) link worked as an agent, and multiple V2V agents jointly explored the environment. Then, they optimized the power control and spectral allocation strategies according to their own observations from the environment state. The authors of [31] designed the MADRL algorithm to solve the optimal decision problem of task offloading from the VEC vehicle terminals and for minimizing the task execution delay. The existing works assumed that the computing resources of edge servers are sufficient. However, during peak traffic, the edge server cannot effectively schedule the computing tasks with limited resources, which results in computation performance degradation and decreases the service quality of the VEC. To this end, we propose a DRL-based framework to solve this problem and design a MADQN-based task scheduling scheme to decrease the task processing latency for VEC networks.

3. System Model

In this section, we first describe the system model, and the task scheduling problem is then formulated. The characteristics of the computation task scheduling problem based on the DRL approach are further analyzed.

We consider the VEC scenario of a vehicular network shown in Figure 1. The VEC server is connected to the macro base station (BS), which acts as a data center for model training. Each roadside BS is connected to the VEC server by optical fiber. We assume that each VEC server has M threads. The thread set of the VEC server is denoted by $\mathcal{S} = \{1, 2, \ldots, M\}$. The corresponding computation resources of thread m ($1 \leq m \leq M$) is denoted by f_m in GHz/s. The computation tasks offloaded by vehicular terminals are modeled according to a uniform distribution [32]. The set $\mathcal{D} = \{1, 2, \ldots, K\}$ represents the computation tasks cached in the VEC server. Each computation task k ($1 \leq k \leq K$) has two parameters: task file size d_k in MB and delay constraint τ_k in ms. Both d_k and τ_k are modeled according to a uniform distribution [21]. Set $\mathcal{T} = \{t_1, t_2, \ldots, t_M\}$ represents the waiting time of the threads in the VEC server. This means that the scheduled task may not be processed immediately when the target thread is busing. $\mathcal{T}$ is modeled according to a Gaussian distribution, while $x_{k,m}$ denotes the target thread index of computation task k and $x_{k,m} = 1$ if the computation task k selects the thread m; otherwise, $x_{k,m} = 0$. We assume that each computation task can only occupy one thread, and thus $\sum_{m \in \mathcal{S}} x_{k,m} = 1$ for a given task k. We model each selecting action of the target thread as an agent and perform a multi-agent decision mechanism. The DRL-based framework of task scheduling in the VEC server is shown in Figure 2.

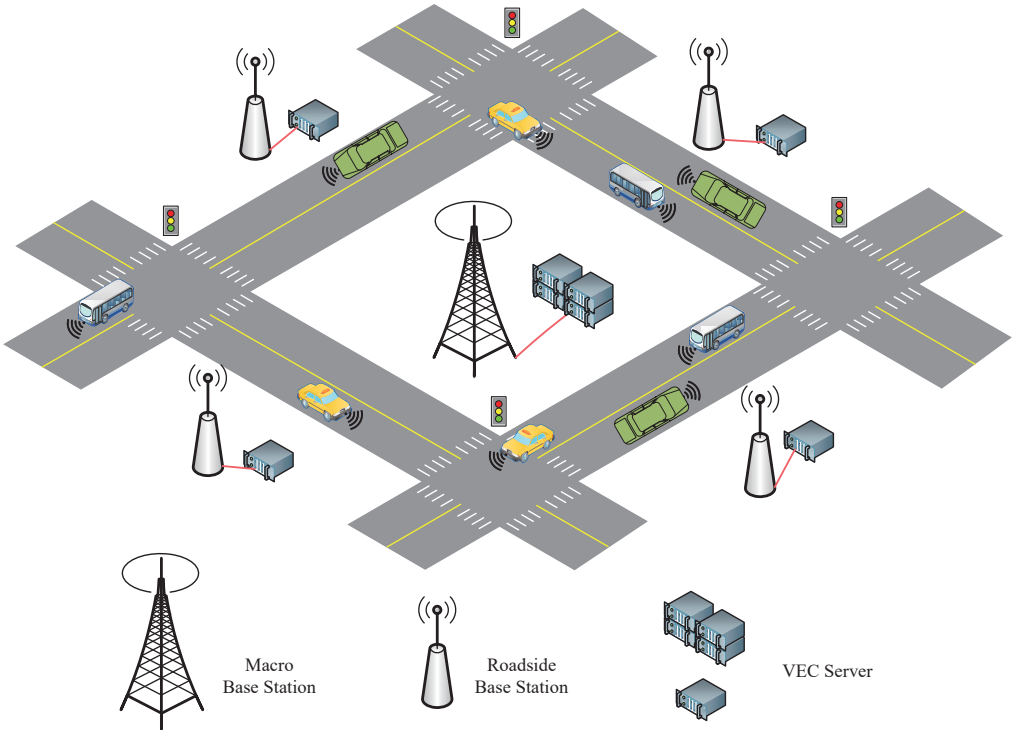

Figure 1. A VEC scenario of vehicular networks.

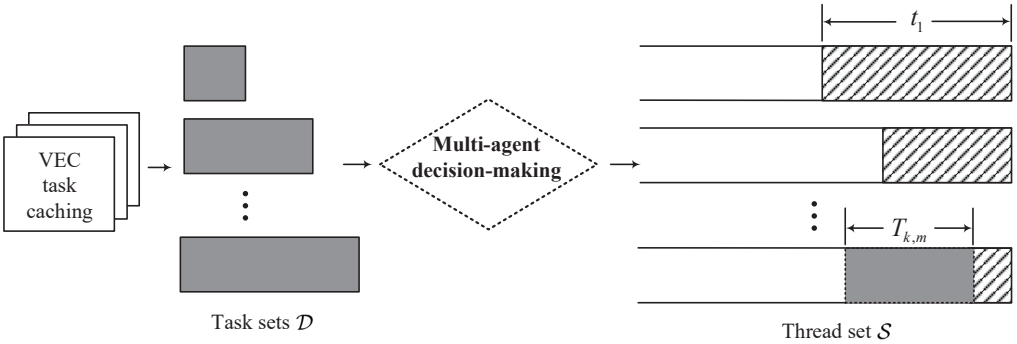

Figure 2. The DRL-based framework of task scheduling in the VEC server.

3.1. Task Computing Model

If the task k selects the thread m, then the task processing delay $T_{k,m}$ of task k in thread m is given by

$$T_{k,m} = x_{k,m} \times d_k \times e / f_m \,, \tag{1}$$

where e denotes the computational intensity (in CPU cycles per MB), representing how many CPU cycles are needed to process one MB of data. From the latency calculation in Equation (1), we can see that both the task file size d_k and computation resources f_m have a significant impact on $T_{m,k}$. We denote the processing delay of thread m as T_m, which can be calculated by

$$T_m = t_m + \sum_{k \in \mathcal{D}} T_{k,m} \,, \tag{2}$$

where t_m represents the task waiting time of thread m. From Equation (2), we can notice that T_m increases with the number of tasks scheduled to thread m. Parallel task scheduling

is adopted in this model, and then the total task processing delay T_s of the VEC server is calculated as follows:

$$T_s = \max\{T_1, \ldots, T_m, \ldots, T_M\} . \tag{3}$$

From Equation (3), we can find that T_m has an important impact on T_s. In order to reduce T_s, avoiding scheduling a large task to a thread with fewer computation resources is necessary.

3.2. Problem Statement

The overall goal of the task scheduling strategy is to minimize the total task processing delay under delay constraints. The optimization problem can be expressed as

$$\mathbf{P1}. \quad \min_{\{k,m\}} \left\{ \max_{m \in \mathcal{S}} T_m - \min_{m \in \mathcal{S}} T_m \right\} \tag{4}$$

$$\text{s.t.} \quad C_1 : \; T_{k,m} \leq \tau_k \tag{5}$$

$$C_2 : \; x_{k,m} \in \{0, 1\} \tag{6}$$

$$C_3 : \; \sum_{m \in \mathcal{S}} x_{k,m} = 1 , \tag{7}$$

where C_1 represents the delay constraints of the computation task, C_2 represents the target thread index, and C_3 means that each task can only be scheduled by one thread. Because of the integer constraint (C_2), the optimization problem in Equation (4) is NP-hard [21].

4. Multi-Agent DQN-Based Task Scheduling

The task scheduling scenario for the VEC server in a vehicular network is shown in Figure 2. Multiple computation tasks attempt to occupy thread resources, which is formulated as a multi-agent DRL problem. Each computation task acts as an agent and interacts with the VEC thread environment to obtain experiences by receiving observation results, which are then applied to modify their policy and enhance the computing efficiency of the VEC server. While computation task scheduling may result in a competitive game problem, we transform it into a cooperative mode by employing the same reward based on the task processing delay to improve the computing efficiency of the VEC server.

The proposed MADQN-based approach is divided into two stages: the centralized learning (training) phase and the distributed implementation phase. In the centralized learning phase, each task agent can easily obtain a computing efficiency-oriented reward and then refine its actions to an optimal strategy by updating its deep Q network (DQN). In the distributed implementation phase, each task agent observes the local environment state and then picks an action on the basis of its learned DQN. The main elements of the MADQN-based task scheduling design are described as followed in detail.

4.1. State and Observation Space

In the multi-agent RL of the task scheduling problem, each task k acts as an agent, collectively interacting with the unknown VEC environment [21]. This problem can be formulated as a Markov decision process (MDP). As illustrated in Figure 3, given the current VEC environment state s_t at each time t, each task agent k gains an observation result $Z_t^{(k)}$, determines the function O with $Z_t^{(k)} = O(s_t, k)$, and then executes an action $A_t^{(k)}$. Thus, these agents form a joint action $\boldsymbol{A}_t$. Subsequently, a reward R_{t+1} is achieved by the agent k, and the VEC environment turns into the next state s_{t+1} with a probability $p(s', R_t|s, a)$. In the next step $t + 1$, each agent receives the new observations $Z_{t+1}^{(k)}$. It is worth noting that all task agents employ the same reward to encourage cooperative behavior among task agents.

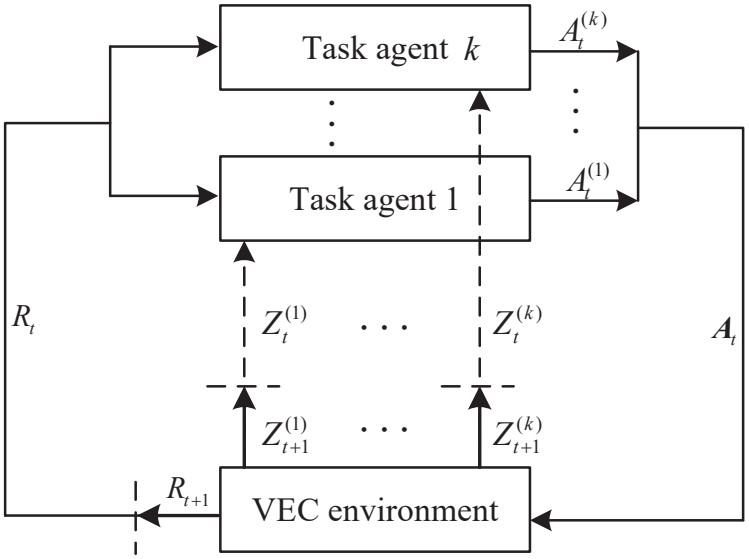

Figure 3. The multi-agent environment interaction with the DRL-based framework.

The true VEC environment state s_t includes all computation resources of all threads and all agents' behaviors. However, each task agent can only observe a part of the VEC environment. The observation space of task agent k contains the computation resources of the VEC thread f_m, the processing delay $T_{m,k}$ of the VEC thread m, the file size d_k of the input task, and the delay constraint τ_k. Therefore, the observation function for task agent k can be expressed as

$$S_t^{(k)} = \{f_m, T_{m,k}, d_k, \tau_k\} . \tag{8}$$

4.2. Action Space

The task scheduling strategy based on the MADQN approach is to select the optimal thread to minimize the task processing delay. While the VEC server breaks into M threads, at each time t, each agent k decides the target thread m to use to process the task from the M threads. As a result, the action of each task agent is to select the target thread and decide the value of the target thread index $x_{k,m}$.

4.3. Reward Design

The computing efficiency of the VEC server can be improved when the designed reward correlates with the system objective. Our objectives are twofold: minimizing the task processing latency of thread m and the total latency of the VEC server while increasing the success process probability of computation tasks under a certain time constraint τ_k. Therefore, the system reward must be consistent with the objective of improving the system's performance. In response to the objective of reducing the task processing latency, the designed reward at each time slot t is formulated as follows:

$$R_t = \min_{m \in \mathcal{S}} T_m - \max_{m \in \mathcal{S}} T_m . \tag{9}$$

From the reward function in Equation (9), we can observe that the designed reward R_t is a negative value, which means that it is a better choice to keep the load balance among the threads in the VEC server. The objective of learning based on the MADQN approach is to find the optimal strategy π_* that maximizes the expected return from any state s,

denoted by G_t, which is defined as the cumulative rewards with a discount fraction λ; in other words, we have

$$G_t = \mathbb{E}_\pi \left\{ \sum_{k=0}^{\infty} \lambda^k R_{t+k+1} \right\}, \ 0 \le \lambda \le 1 \, . \tag{10}$$

5. MADQN-Based Task Scheduling Algorithm

The structure of MADQN-based task scheduling is shown in Figure 4. In the learning process, an evaluation network is constructed to generate scheduling policies. The input parameter is the server state space s, and the output result is the scheduling function of the corresponding action-value function $Q(s, a; \omega)$, where ω is the weight parameter of the evaluation network. The function $Q(s', a'; \omega')$ represents the next state, and ω' is the weight parameters of the target network, which are duplicated from the training Q network parameter set ω.

The evaluation network and target network of each task agent in this paper are composed of an input layer, three hidden layers, and an output layer. Each layer is composed of multiple neurons and connected with the neurons in the next layer. There are relevant weight parameters between the interconnected neurons. Different weight parameters indicate the importance of variables. Larger weight parameters contribute more to the output. In this paper, the number of neurons of the input layer is the length of the state space s. The number of neurons in the three hidden layers is 256, 128, and 64. The number of neurons in the output layer is the number of server threads M. When the agent selects the maximum scheduling value function Q from the output layer, it also indicates the selection of server threads.

In the training process, the loss function *loss* is used to evaluate the accuracy of the network. The goal of training is to minimize the loss function by adjusting the weight parameters. In the process of weight parameter updating, only the weight parameters of the evaluation network ω are updated. The weight parameters of the target network ω' are updated by copying from the evaluation network $\omega' \leftarrow \omega$ after a certain number of rounds. In this way, the target of the evaluation network weight update is a fixed target for a period of time, increasing learning stability. Each task agent has an independent evaluation network and target network, which is an independent DQN, while the experienced playback is public to encourage task agents to explore the environment cooperatively.

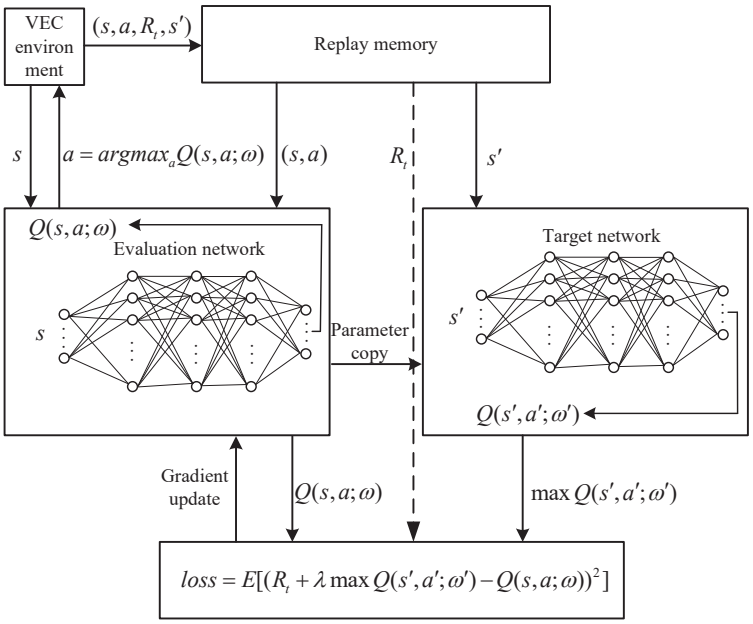

Figure 4. Structure of the MADQN-based task scheduling algorithm.

5.1. Training Process

We utilized the DRL experiential replay to effectively train the multi-tasking agent for learning the selection strategy of the target thread. The MADQN-based training process is shown in Figure 5.

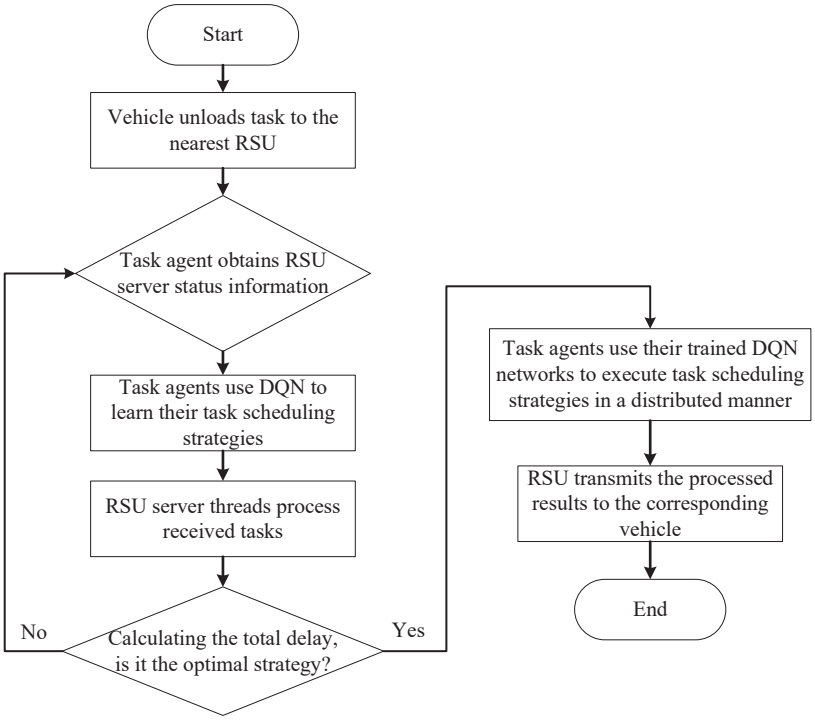

Figure 5. MADQN-based training process.

In the training process, each episode begins with a randomly initialized VEC environment state, including ω, which is the parameter of the evaluation network. Then, the evaluation network receives the environment state s_t and the output results of all action value functions $Q(s_t, a_t; \omega)$, which can be calculated as

$$Q(s_t, a_t; \omega) = R_t + \lambda E[Q(s_{t+1}, a_{t+1}; \omega)] \,, \tag{11}$$

where R_t is defined in Equation (9). Then, the largest Q value from the action value function $Q(s_t, a_t; \omega)$ is obtained, and the action corresponding to the largest Q value $Q(s_t, a_t; \omega)$ is executed. The environment turns into the next state s_{t+1} and saves the transition tuple (s_t, a_t, R_t, s_{t+1}) in a replay memory.

A batch of experiences in each episode $\mathcal{K}$ is uniformly sampled from the replay memory, which will be used as the input of the target network. Then, the target network outputs all the action values $Q(s_{t+1}, a_{t+1}; \omega')$ of the next state s_{t+1} and calculates the target value U_t of the evaluation network. U_t can be computed as follows:

$$U_t = R_t + \lambda \underset{a_{t+1}}{\max} Q(s_{t+1}, a_{t+1}; \omega') \,. \tag{12}$$

The goal of training is to minimize the loss function *loss* to ensure the correctness of the fitting of any given observation. The *loss* function is defined as follows:

$$loss \triangleq \frac{1}{B} \sum_{\mathcal{K}} [U_t - Q(s_t, a_t; \omega)]^2 \,, \tag{13}$$

where B is the sample size. The weight parameters are adjusted through gradient descent so that the model can determine the direction of error reduction. The gradient descent of the loss function can be calculated as follows:

$$\nabla_w loss = \frac{1}{B}\sum_{\mathcal{K}} 2[U_t - Q(s_t, a_t; \omega)]\nabla Q(s_t, a_t; \omega)\ . \tag{14}$$

Then, the evaluation network weight parameter is further updated to ω_{up}, which can be calculated as follows:

$$\omega_{up} = \omega + \eta\frac{1}{B}\sum_{\mathcal{K}} 2[U_t - Q(s_t, a_t; \omega)]\nabla Q(s_t, a_t; \omega)\ , \tag{15}$$

where η is the learning rate of the evaluation network. After the weight parameter ω has been updated, the environment turns into the next state $s \leftarrow s_{t+1}$. The weight parameter ω' of the target network is duplicated from set ω periodically. The algorithm procedure is illustrated in Algorithm 1. In Algorithm 1, we use an ε-greedy strategy to explore the state and action space. This means that the random action is checked with the probability of ε, and the action with the maximum Q value is checked with the probability of $1-\varepsilon$. Here, ε is a global variable which decreases with the number of training sessions and finally tends toward a fixed value.

Algorithm 1 Task scheduling algorithm based on MADQN

for each episode **do**
  for each step t **do**
    for each agent **do**
      Evaluating network reception s_t, choose action a_t at random with probability ε, with probability $1-\varepsilon$ choose action $a_t = argmax_a Q(s_t, a_t; \omega)$, ε decreases with training sessions;
    end for
    To perform an action $\boldsymbol{a}_t$,get environmental rewards R_t, the environment updates to the next state s_{t+1};
    for each agents **do**
      Saves experience (s_t, a_t, R_t, s_{t+1}) to experience playback
    end for
  end for
  for each agents **do**
    Select a batch of experiences from experience playback $(s_t^i, a_t^i, R_t^i, s_{t+1}^i)(i \in B)$
    Calculate the target value U_t according to Equation (12)
    Update ω according to Equation (15) to reduce loss
    The weight $\omega' \leftarrow \omega$ of the target network is updated every a certain episode
  end for
end for

5.2. Distributed Implementation

In the distributed implementation stage, at each time slot t, the task agent k receives the VEC environment state and chooses an optimal action value based on its trained Q network. Then, each thread of the VEC server starts to process the received tasks. It is worth noting that the intensive training procedure of VEC computation in Algorithm 1 can be executed offline. When the environment features have been changed significantly, the trained Q network needs to be updated according to the environmental dynamics and system performance requirements.

6. Simulation Results

In this section, the experiment results are provided to verify the effectiveness of the MADQN-based task scheduling strategy for vehicular networks. The software environment

for the simulation experiments was Python 3.6, and the convolutional neural network was created based on TensorFlow. The hardware platform for the simulation experiments was based on a desktop computer. Unless otherwise stated, the major simulation parameter settings of the VEC environment and neural network are shown in Table 1.

Table 1. Simulation parameters.

Simulation Parameters	Value
Number of tasks (k)	10 [33]
Task data size (d_k)	U(5,50) MB, U(50,200) MB [21]
Task delay constraint (τ_k)	10∼50 ms
Number of server threads (m)	4
Total computation resources	8 GHz
Server thread computing resources (f)	0.5, 1.5, 2.5, 3.5 GHz
Thread wait time (κ)	1∼3 ms
Computational intensity (e)	0.001 GHz/MB [33]
Number of neuron and hidden layers	5,3
Number of neurons	256, 128, 64
Neural network activation function	ReLU
Gradient descent algorithm	RMSProp
Learning rate of neural network (η)	0.001
Discount fraction (λ)	0.99

We investigated the convergence behaviors of the proposed MADQN-based algorithm, with the training rewards versus the number of training episodes shown in Figure 6. We can see that the training rewards improved with the number of training episodes. It has been demonstrated that the proposed MADQN-based algorithm is effective. The training rewards gradually converged when the episode count reached approximately 2500. In order to provide a security guarantee of algorithm convergence, the number of training episodes was set to 3000 for each agent in the Q networks.

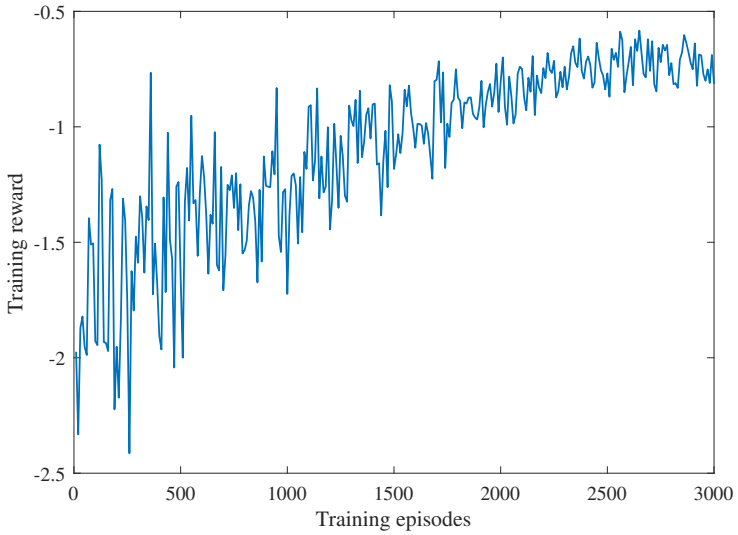

Figure 6. Training rewards versus the number of training episodes.

Figure 7 shows the total processing delay versus the number of training episodes. We can see that with the increase in the number of training episodes, the total processing delay decreased. This further demonstrates the effectiveness of the proposed MADQN-based algorithm.

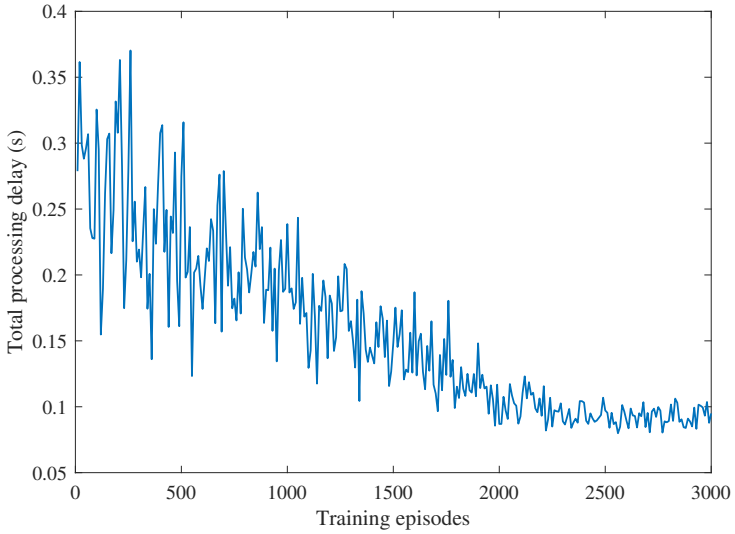

Figure 7. Total processing delay versus the number of training episodes.

In order to verify the performance of the proposed algorithm, the following two baseline algorithms were executed:

- Single-agent deep Q networks (SADQN) is a traditional deep reinforcement learning algorithm which was proposed in [34], where at each time slot t, only one task agent replaces its action while the other agents' actions remain unchanged.
- The shortest time (ST) scheduling algorithm, proposed in [35], is an algorithm where the task agent always selects the thread with the lowest processing time.

Figure 8 shows the impact of the data size of the task file on the task processing delay of different scheduling algorithms, including the SADQN, ST, and our proposed MADQN-based algorithms. We can see from Figure 8 that our proposed MADQN-based algorithm obtained the minimum task processing delay compared with the scheduling algorithms based on SADQN and the ST. With the increase in the data size of the task files, the task processing delay gradually increased. Intuitively, the larger the data size of the task file, the longer the execution time for a given computing resource. We can further observe that the performance gains obtained by the proposed MADQN-based algorithm were improved compared with scheduling algorithms based on SADQN and the ST. It is worth noting that our proposed MADQN-based algorithm reduced the task processing delay by more than 28% and 40% compared with the SADQN algorithm and ST algorithm, respectively, when the data size of the input tasks was greater than 50 MB.

The effect of the data size of the task file on the successful processing probability of computation tasks is shown in Figure 9, where the data size of the computation task file is uniformly distributed with U(5,50) MB. We can see that the success probability of task processing decreased with the increasing data size of the task file. This is due to the fact that a larger file size requires more computation resources. When the VEC server was overloaded, the degradation in computing efficiency resulted in an increase in the task processing delay, which led to a reduction in the successful processing probability. It is worth noting that compared with the SADQN and ST algorithms, the proposed MADQN-based algorithm could achieve significant performance gains in the successful processing probability of more than 30% when the data size of the task file was greater than 50 MB.

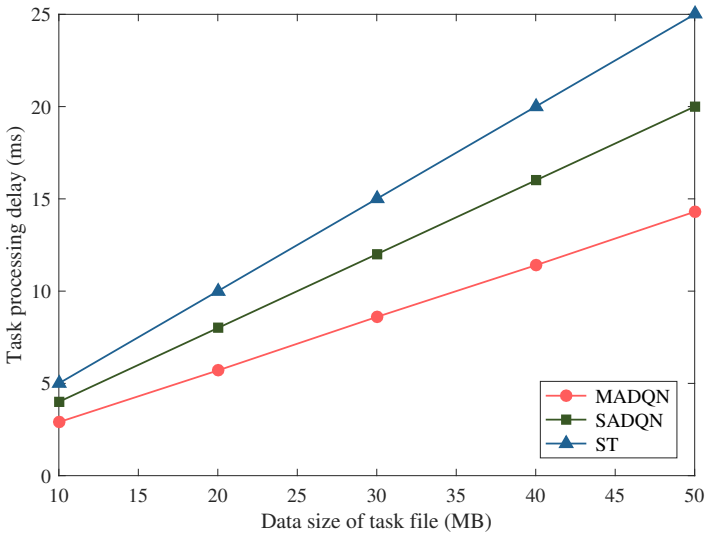

Figure 8. The effect of the data size of the task file on the task processing delay.

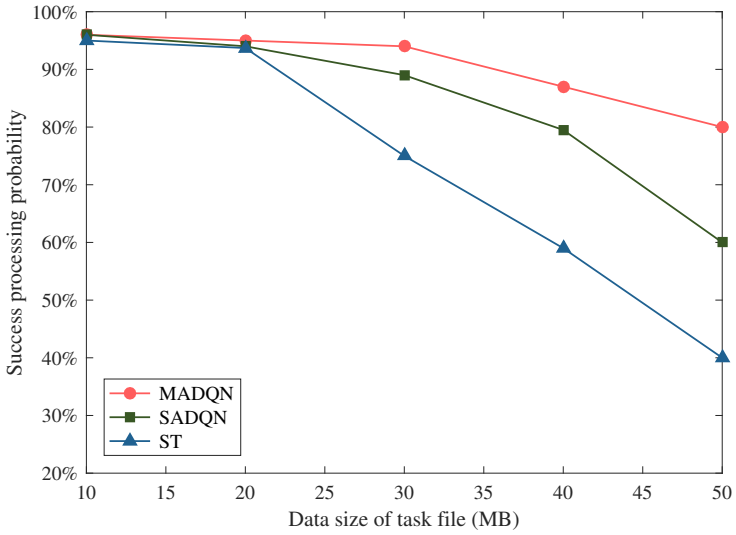

Figure 9. The effect of the data size of the task file on the successful processing probability.

We further investigated the effect of the number of computation tasks on the total processing delay, where the data size of the computation tasks was uniformly distributed with U(5,50) MB. From Figure 10, we can note that the total processing delay of different scheduling algorithms increased with the increasing number of computation tasks. We can further observe that our proposed MADQN-based algorithm obtained the minimum task processing delay. With the ever-increasing number of computation tasks, the performance gain obtained by the proposed algorithm improved continuously. Compared with the SADQN algorithm and the ST algorithm, the proposed MADQN based algorithm was superior because a distributed implemented approach was introduced and task processing was conducted in a parallel manner by the proposed algorithm, which is more suitable for solving complex problems. The total processing delay of the MADQN-based algorithm decreased by more than 2% when the number of computation tasks was more than four.

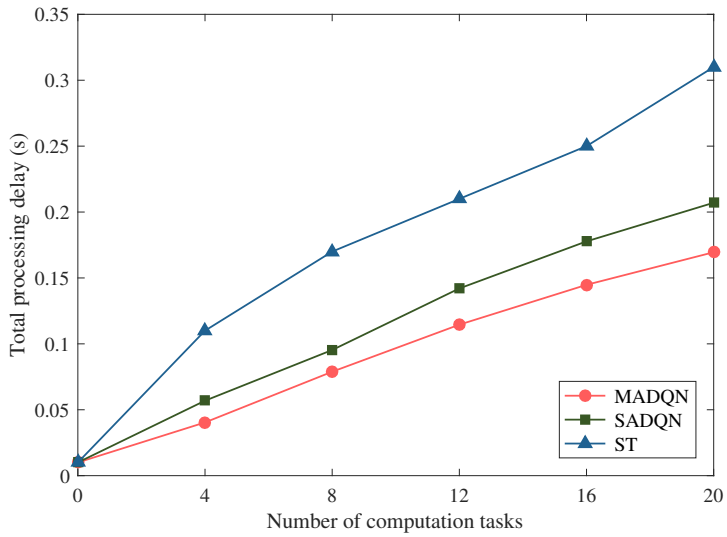

Figure 10. The effect of the number of computation tasks on the total processing delay.

The impact of the number of computation tasks on the successful processing probability with different scheduling algorithms was investigated, with the results shown in Figure 11. We can see from Figure 11 that with the increasing number of computation tasks, the successful processing probabilities of different scheduling algorithms decreased. This is due to the fact that a large number of computation tasks require more computing resources. As the number of computation tasks increases, the VEC servers are overloaded, the computing efficiency decreases, and thus the successful processing probability decreases. Fortunately, we can observe that the proposed algorithm outperformed the traditional SADQN algorithm and ST algorithm. Even when the workload of the VEC servers was deeply saturated, the significant performance improvement obtained by the proposed algorithm was greater than 5% compared with the traditional algorithms. This indicates that the proposed algorithm based on MADQN in this paper is effective for VEC servers in peak traffic periods.

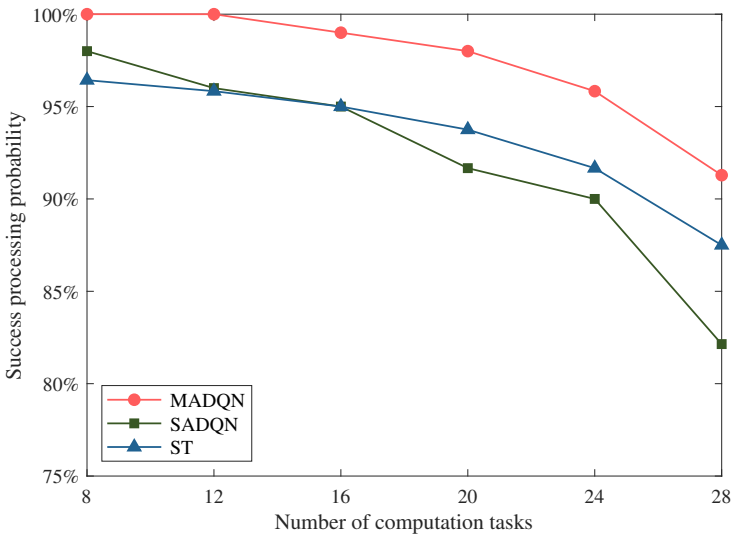

Figure 11. The effect of the number of computation tasks on the successful processing probability.

In order to demonstrate the superiority of the proposed algorithm based on MADQN, we investigate the calculation performance of different algorithms with diversified computation tasks (task types shown in Table 2), where task types 1, 2, and 3 represent the scenarios of a light load, medium load and heavy load for the VEC servers, respectively.

From Figure 12, we can clearly observe that the proposed MADQN-based algorithm can significantly reduce the task processing delay for the different computing scenarios of VEC servers with a light load, medium load or heavy load. Especially under the condition of a heavy load, the total processing delay can be reduced by more than 20% compared with the traditional algorithms. The reason behind the performance growth is the fact that the proposed MADQN-based algorithm introduces a distributed implementation method and parallel task processing. The experiment demonstrated that the proposed algorithm is effective in enhancing the computing efficiency of VEC servers.

Table 2. Three different types of computing tasks.

Task Types	The Number of Computing Tasks	Data Size of Computing Tasks (Uniform Distribution)
Task type 1	10	U(5,6) MB
Task type 2	10	U(50,55) MB
Task type 3	5 and 5	U(5,6) MB and U(50,55) MB

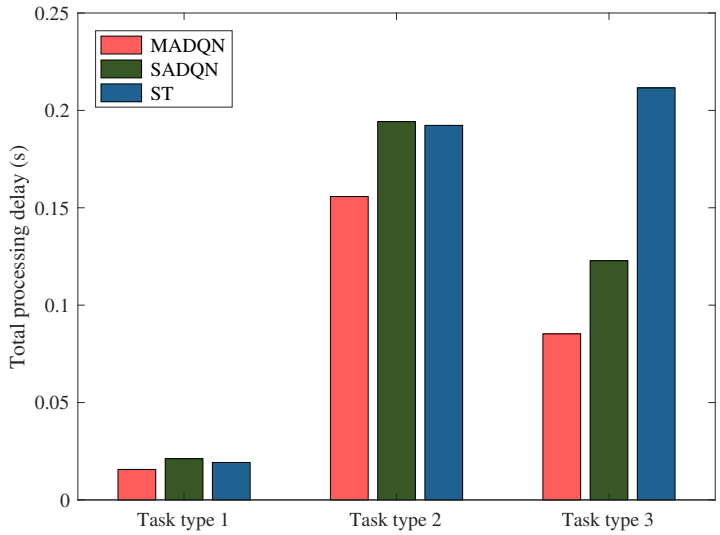

Figure 12. The effect of different data sizes of computing tasks on the total processing delay.

In order to provide a design perspective for VEC server deployment, we investigated the effect of different computing resource configurations of threads on the total processing delay, where the configured computing resources are shown in Table 3. Thread type A represents the most unbalanced computing resource configuration, and thread type D represents that all thread resources are equal. The total processing delays for different thread configurations are shown in Figure 13, where the total computing resources were constant. We can note that thread type A, with the most unbalanced resource configuration, required the minimum latency. This means that distinguished resource configurations for different threads in the VEC servers is a better choice.

Table 3. The computing resource configuration for the threads.

Thread Types	Computing Resources of Threads			
	Thread 1	Thread 2	Thread 3	Thread 4
Thread type A	0.5 GHz/s	1.5 GHz/s	2.5 GHz/s	3.5 GHz/s
Thread type B	1 GHz/s	1.5 GHz/s	2.5 GHz/s	3 GHz/s
Thread type C	1.5 GHz/s	2 GHz/s	2 GHz/s	2.5 GHz/s
Thread type D	2 GHz/s	2 GHz/s	2 GHz/s	2 GHz/s

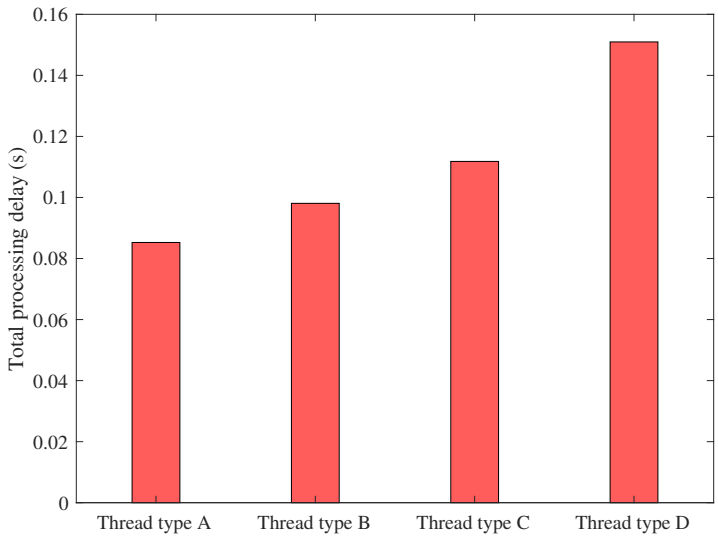

Figure 13. The effect of different computing resources on the total processing delay.

7. Conclusions

In order to reduce the task processing delay of the VEC server and improve the service quality of vehicular networks, the proposed MADQN-based algorithm was divided into two stages: centralized training in a data training center and distributed implementation on the VEC servers adjacent to the vehicular terminals. Our objective was to minimize the task processing delay, especially during the peak period of traffic in vehicular networks. The experiment results demonstrated that the proposed MADQN-based algorithm could obtain significant performance gains compared with the traditional SADQN algorithm and ST algorithm. We have illustrated the effectiveness of the proposed algorithm. Compared with traditional algorithms, the total processing delay can be reduced by more than 5% when the VEC servers are overloaded.

In the future, we will consider the scenarios of joint wireless channels and vehicle mobility to design a more realistic VEC environment. At the same time, we will further investigate the application of other DRL algorithms, such as Duel DQN and D3QN, for VEC networks.

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

Funding: This work was supported in part by the National Natural Science Foundation of China (61961020, 62261020, and 61861017), the Natural Science Foundation of Jiangxi Province (20212BAB202004, 20212BAB202005, and 20224BAB202001), the Key Research and Research and Development Program of Jiangxi Province (20202BBEL53014), and the Jiangxi Provincial Department of Education Science and Technology Program Funding Project (GJJ180312).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Zhao, J.; Sun, X.; Ma, X.; Zhang, H.; Yu, F.R.; Hu, Y. Online distributed optimization for energy-efficient computation offloading in air-ground integrated networks. *IEEE Trans. Veh. Technol.* **2022**. [CrossRef]
2. Aslam, N.; Wang, K.; Pan, C.; Aslam, N. Joint trajectory and passive beamforming design for intelligent reflecting surface-aided UAV communications: A deep reinforcement learning approach. *IEEE Trans. Mob. Comput.* **2022**. [CrossRef]
3. Liu, Y.; Peng, M.; Shou, G.; Chen, Y.; Chen, S. Toward edge intelligence: Multiaccess edge computing for 5G and Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 6722–6747. [CrossRef]
4. Zhao, J.; Ni, S.; Gong, Y.; Zhang, Q. Pilot contamination reduction in TDD-based massive MIMO systems. *IET Commun.* **2019**, *13*, 1425–1432. [CrossRef]
5. Yang, M. Research on vehicle automatic driving target perception technology based on improved MSRPN algorithm. *J. Comput. Cogn. Eng.* **2022**, *1*, 147–151.
6. Ji, L.; Guo, S. Energy-efficient cooperative resource allocation in wireless powered mobile edge computing. *IEEE Internet Things J.* **2018**, *6*, 4744–4754. [CrossRef]
7. Yang, L.; Zhang, H.; Li, M.; Guo, J.; Ji, H. Mobile edge computing empowered energy efficient task offloading in 5G. *IEEE Trans. Veh. Technol.* **2018**, *67*, 6398–6409. [CrossRef]
8. Mustafa, E.; Shuja, J.; Bilal, K.; Mustafa, S.; Maqsood, T.; Rehman, F.; ur Khan, A.R. Reinforcement learning for intelligent online computation offloading in wireless powered edge networks. *Clust. Comput.* **2022**, *26*, 1053–1062. [CrossRef]
9. 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]
10. Mustafa, E.; Shuja, J.; uz Zaman, S.K.; Jehangiri, A.I.; Din, S.; Rehman, F.; Mustafa, S.; Maqsood, T.; Khan, A.N. Joint wireless power transfer and task offloading in mobile edge computing: A survey. *Clust. Comput.* **2022**, *25*, 2429-2448. [CrossRef]
11. Liu, Y.; Wang, S.; Zhao, Q.; Du, S.; Zhou, A.; Ma, X.; Yang, F. Dependency-aware task scheduling in vehicular edge computing. *IEEE Internet Things J.* **2020**, *7*, 4961–4971. [CrossRef]
12. Zhao, J.; Sun, X.; Li, Q.; Ma, X. Edge caching and computation management for real-time internet of vehicles: An online and distributed approach. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 2183–2197. [CrossRef]
13. Xu, X.; Xue, Y.; Li, X.; Qi, L.; Wan, S. A computation offloading method for edge computing with vehicle-to-everything. *IEEE Access* **2019**, *7*, 131068–131077. [CrossRef]
14. Li, H.; Xu, H.; Zhou, C.; Lü, X.; Han, Z. Joint optimization strategy of computation offloading and resource allocation in multi-access edge computing environment. *IEEE Trans. Veh. Technol.* **2020**, *69*, 10214–10226. [CrossRef]
15. Zheng, Z.; Song, L.; Han, Z.; Li, G.Y.; Poor, H.V. A stackelberg game approach to proactive caching in large-scale mobile edge networks. *IEEE Trans. Wirel. Commun.* **2018**, *17*, 5198–5211. [CrossRef]
16. Zhao, J.; Nie, Y.; Zhang, H.; Yu, F.R. A UAV-aided vehicular integrated platooning network for heterogeneous resource management. *IEEE Trans. Green Commun. Netw.* **2023**, *7*, 512–521. [CrossRef]
17. Qi, F.; Zhuo, L.; Xin, C. Deep reinforcement learning based task scheduling in edge computing networks. In Proceedings of the IEEE International Conference on Communications in China (ICCC), Chongqing, China, 9–11 August 2020; pp. 835–840.
18. Tang, C.; Wu, H. Joint optimization of task caching and computation offloading in vehicular edge computing. *Peer-to-Peer Netw. Appl.* **2022**, *15*, 854-869. [CrossRef]
19. Tang, C.; Zhu, C.; Wu, H.; Li, Q.; Rodrigues, J.J.P.C. Toward response time minimization considering energy consumption in caching-assisted vehicular edge computing. *IEEE Internet Things J.* **2022**, *9*, 5051–5064. [CrossRef]
20. Huang, T.; Lin, W.; Xiong, C.; Pan, R.; Huang, J. An ant colony optimization-based multiobjective service replicas placement strategy for fog computing. *IEEE Trans. Cybern.* **2020**, *51*, 5595–5608. [CrossRef]
21. Dai, Y.; Zhang, K.; Maharjan, S.; Zhang, Y. Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond. *IEEE Trans. Veh. Technol.* **2020**, *69*, 12175–12186. [CrossRef]
22. Ke, H.; Wang, J.; Deng, L.; Ge, Y.; Wang, H. Deep reinforcement learning-based adaptive computation offloading for MEC in heterogeneous vehicular networks. *IEEE Trans. Veh. Technol.* **2020**, *69*, 7916–7929. [CrossRef]
23. Liang, L.; Ye, H.; Li, G.Y. Spectrum sharing in vehicular networks based on multi-agent reinforcement learning. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 2282–2292. [CrossRef]
24. Zhan, W.; Luo, C.; Wang, J.; Wang, C.; Min, G.; Duan, H.; Zhu, Q. Deep-reinforcement-learning-based offloading scheduling for vehicular edge computing. *IEEE Internet Things J.* **2020**, *7*, 5449–5465. [CrossRef]
25. Luo, Q.; Li, C.; Luan, T.H.; Shi, W. Collaborative data scheduling for vehicular edge computing via deep reinforcement learning. *IEEE Internet Things J.* **2020**, *7*, 9637–9650. [CrossRef]
26. Tang, H.; Wu, H.; Qu, G.; Li, R. Double deep Q-network based dynamic framing offloading in vehicular edge computing. *IEEE Trans. Netw. Sci. Eng.* **2022**. [CrossRef]
27. Dai, Y.; Xu, D.; Zhang, K.; Maharjan, S.; Zhang, Y. Deep reinforcement learning and permissioned blockchain for content caching in vehicular edge computing and networks. *IEEE Trans. Veh. Technol.* **2020**, *69*, 4312–4324. [CrossRef]
28. Yang, Y.; Luo, R.; Li, M.; Zhou, M.; Zhang, W.; Wang, J. Mean field multi-agent reinforcement learning. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 5571–5580.

29. Tang, F.; Mao, B.; Kato, N.; Gui, G. Comprehensive survey on machine learning in vehicular network: Technology, applications and challenges. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 2024–2057. [CrossRef]
30. Li, T.; Zhu, K.; Luong, N.C.; Niyato, D.; Wu, Q.; Zhang, Y.; Chen, B. Applications of multi-agent reinforcement learning in future internet: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2022**, *24*, 1240–1279. [CrossRef]
31. Zhu, X.; Luo, Y.; Liu, A.; Bhuiyan, M.Z.A.; Zhang, S. Multiagent deep reinforcement learning for vehicular computation offloading in IoT. *IEEE Internet Things J.* **2020**, *8*, 9763–9773. [CrossRef]
32. 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]
33. Qi, Q.; Zhang, L.; Wang, J.; Sun, H.; Zhuang, Z.; Liao, J.; Yu, F.R. Scalable parallel task scheduling for autonomous driving using multi-task deep reinforcement learning. *IEEE Trans. Veh. Technol.* **2020**, *69*, 13861–13874. [CrossRef]
34. Ye, H.; Li, G.Y.; Juang, B.H.F. Deep reinforcement learning based resource allocation for V2V communications. *IEEE Trans. Veh. Technol.* **2019**, *68*, 3163–3173. [CrossRef]
35. Singh, K.; Alam, M.; Sharma, S.K. A survey of static scheduling algorithm for distributed computing system. *Int. J. Comput. Appl.* **2015**, *129*, 25–30. [CrossRef]