Multiagent Deep Reinforcement Learning for Vehicular Computation Offloading in IoT

Xiaoyu Zhu, *Member, IEEE*, Yueyi Luo, Anfeng Liu, Md Zakirul Alam Bhuiyan, *Senior Member, IEEE*, and Shaobo Zhang

***Abstract*—The development of the Internet of Things (IoT) and intelligent vehicles brings a comfortable environment for users. Various emerging vehicular applications using artificial intelligence (AI) technologies are expected to enrich users' daily life. However, how to execute computation-intensive applications on resource-constrained vehicles based on AI still faces great challenges. In this article, we consider the vehicular computation offloading problem in mobile-edge computing (MEC), in which multiple mobile vehicles select nearby MEC servers to offload their computing tasks. We propose a multiagent deep reinforcement learning (DRL)-based computation offloading scheme, in which the uncertainty of a multivehicle environment is considered so that the vehicles can make offloading decisions to achieve an optimal long-term reward. First, we formalize a formula for the computation offloading problem. The goal of this article is to determine the optimal offloading decision to the MEC server under each observed system state, so as to minimize the total task processing delay in a long-term period. Then, we use a multiagent DRL algorithm to learn an effective solution to the vehicular task offloading problem. To evaluate the performance of the proposed offloading scheme, a large number of simulations are carried out. The simulation results verify the effectiveness and superiority of the proposed scheme.**

***Index Terms*—Computation offloading, deep reinforcement learning (DRL), mobile-edge computing (MEC), vehicular edge network.**

I. Introduction

With the quick development of the Internet of Things (IoT) and artificial intelligence (AI), vehicles are generally becoming intelligent terminals [1], [2]. The intelligent vehicles are basically equipped with an onboard device whose function is similar to a small computer with a network interface [3], [4]. Many researchers are devoted to developing novel vehicular applications to create a more comfortable and safe driving environment [5], [6]. However, more and more computation-intensive applications have emerged, such as real-time driving monitoring, video entertainment, vehicle-road coordination, and environment monitoring. Thus, it faces huge challenges to execute these computation-intensive applications on vehicles in a limited time. The vehicles with limited computing resources can hardly complete the computation-intensive tasks, which result in that the vehicles cannot ensure the needed quality of service.

Initially, the researchers proposed mobile cloud computing (MCC) [7] to solve the computation-intensive problem, which can make full use of the cloud center with powerful computing resources and efficient storage resources. However, MCC cannot satisfy the real-time response requirement for vehicle application scenarios. For example, MCC cannot achieve real-time feedback between the cloud center and vehicles. Furthermore, researchers proposed mobile-edge computing (MEC) [8] to solve the latency problem in MCC. Compared with MCC, MEC can extend the computation capacity of the vehicular edge network. MEC is a promising and alternative scheme because the MEC servers are closer to the users, which can largely decrease the communication delay. In vehicular edge network, the MEC servers are deployed along the roads, the vehicles can offload the computation tasks to the MEC server using wireless communication. Through computation offloading, the vehicles can largely decrease the task delay and improve the quality of service. Thus, the computation offloading problem in vehicular edge networks has attracted great interest from researchers.

Recently, AI [9] has been applied to many areas, such as image processing, language processing, and pattern recognition. Meanwhile, AI also involves vehicular applications, such as autonomous driving and real-time navigation. Many researchers proposed vehicular computation offloading schemes [10], [11]. However, multiple vehicles' locations are quickly moving as time changes, the computation offloading faces great challenges in the vehicular edge network. The vehicular computation offloading problem has great uncertainty, thus it is challenging to solve this problem using AI solutions. The AI methods that can support vehicular computation offloading are still in exploring stages.

Manuscript received September 8, 2020; revised October 31, 2020; accepted November 23, 2020. Date of publication November 26, 2020; date of current version June 7, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 62072475 and Grant 61772554; in part by the Joint Funds of the National Natural Science Foundation of China under Grant U2003208; and in part by the Hunan Provincial Natural Science Foundation of China under Grant 2020JJ4317. *(Corresponding author: Yueyi Luo.)*

Xiaoyu Zhu and Anfeng Liu are with the School of Computer Science and Engineering, Central South University, Changsha 410083, China (e-mail: zhuxiaoyu@csu.edu.cn; afengliu@mail.csu.edu.cn).

Yueyi Luo is with the School of Mathematics and Statics, and Network Resources Management and Trust Evaluation Key Laboratory of Hunan Province, Central South University, Changsha 410083, China (e-mail: luoyueyi@csu.edu.cn).

Md Zakirul Alam Bhuiyan is with the Department of Computer and Information Sciences, Fordham University, New York, NY 10458 USA (e-mail: zakirulalam@gmail.com).

Shaobo Zhang is with the School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China (e-mail: shaobozhang@hnust.edu.cn).

Digital Object Identifier 10.1109/JIOT.2020.3040768

Fortunately, as a promising AI method, deep reinforcement learning (DRL) [12] has huge advantages in the uncertain decision problem. DRL is a technique that may replace traditional learning methods. To construct the intelligent vehicular computation offloading system and make it work normally, there are mainly two challenges.

1) The vehicular network is highly dynamic [13], [14], the constraints are more implicit, flexible, and dynamic.
2) The DRL model only contains one agent usually, but multiple vehicles participate in the intelligent computation offloading system, thus it is difficult to construct an appropriate DRL model to adapt to the vehicular edge network.

The traditional DRL algorithms are not suitable for the multivehicle environment. As each vehicle is constantly learning to improve its strategy, from the perspective of each vehicle, the multivehicle environment is dynamic and unstable, which does not meet the convergence conditions. To some extent, it is impossible to adapt to the dynamic and unstable environment by merely changing the vehicle's strategy. Due to the unstable environment, the key techniques such as experience replay cannot be used directly. For example, the actor–critic (AC) algorithm [15] cannot handle complex multivehicle scenarios. Recently, researchers have proposed the multiagent DRL algorithm to handle complex multiagent scenarios [16]. The multiagent DRL algorithm can be adopted to reduce the instability of the multivehicle environment while ensuring that each vehicle's strategy is updated to increase each vehicle's reward.

We combine multiagent DRL with MEC to solve the computation offloading problem of multiple vehicles in the vehicular edge network. The vehicles generate tasks in a period, each vehicle can select one appropriate MEC server to offload tasks. As vehicle mobility has an important impact on the task execution delay, thus we include the vehicle mobility into the problem statement. Our motivation is to learn the task offloading policy and make decisions based on the observed real-time state under the help of DRL. DRL can make decisions learning from historical data, and it can use future rewards as feedback from the environment. The vehicles can adjust the offloading policy to achieve the optimal long-term reward. It is important in the time-varying system, especially in the MEC system with multiple mobile vehicles.

In this article, we propose a multiagent DRL scheme for computation offloading in vehicular edge networks (MDRCO). We consider a vehicular edge network, which contains one data center, multiple MEC servers, and multiple mobile vehicles. The multiagent DRL model mainly has four key actors: 1) agent; 2) system state; 3) action; and 4) reward. The system state includes the task triggering time, the real-time vehicle location, and task information. MDRCO can decrease uncertainty in a multiagent environment. We do not make decisions using historical information, but we collect historical information to train the model. The model is trained at the data center to save a lot of computation cost, and the vehicles do not need to exchange with the data center when making the task offloading decision. The computation offloading decisions are made based on real-time vehicle information. The main contributions of this article are summarized as follows.

1) The computation offloading problem is modeled as a multiagent DRL problem to select the appropriate MEC server to execute tasks for multiple vehicles. Our model can describe the vehicle state in an accurate way. We observe and summarize the vehicles' state, which includes the vehicles' real-time locations and task information. Each vehicle can select an appropriate MEC server to offload tasks based on our multiagent DRL model.
2) The MDRCO is proposed in this article. Our objective is to minimize the total task execution delay of the whole system in a period. Through analyzing the communication and computation model, we take the task execution delay as the main performance factor. The data center shares all the vehicles' historical state, action, and policy, we minimize the instability of multiple vehicles' environment.
3) The centralized training algorithm and distributed execution algorithms are proposed based on multiagent DRL. The data center first trains the actor network and critic network in a centralized way. The vehicles can execute task offloading decisions in a distributed way. We evaluate our proposed scheme, the evaluation results show that MDRCO can achieve better performances. Compared to the nearest neighbor (NN) algorithm and AC algorithm, the delay can be decreased by 21.7% and 10.2%, respectively.

The remainder of the article is summarized as follows. In Section II, we propose the system model, which includes the network model, task model, and computation model. In Section III, we describe our optimization problem. In Section IV, we propose the MDRCO scheme. In Section V, we show the evaluation results. In Section VI, we give the final conclusion.

II. Related Work

The vehicular computation offloading problem involves researches on MEC, computation offloading, and AI.

A. Mobile-Edge Computing

MEC is a promising paradigm that attracts many researchers' attention, which can be applied to many applications [17], [18]. Ko *et al.* [19] proposed a MEC model for large-scale networks, and stochastic geometry was used to analyze network latency. Wang *et al.* [20] proposed a multilayer MEC model, which contains an edge device layer, multilayer MEC server layer, and cloud center layer. The MEC model was used to process data generated by edge devices heterogeneously. Huang *et al.* [21] used the deep reinforcement method to offload tasks in a wireless-powered MEC network. Jošilo and Dán [22] studied the allocation problem of wireless and computing resources. In order to minimize the computation time, the wireless devices decide to use shared resources or not. Dinh *et al.* [23] considered the fixed and elastic CPU frequencies for mobile devices, the semidefinite

relaxation algorithms were proposed to offload tasks in an optimal way. However, these works do not consider the characteristics of mobile vehicles. The delay of the computation offloading should be considered carefully as the vehicles are moving fast.

B. Computing Offloading

In recent years, researchers have proposed many computation offloading solutions in MEC. Tran and Pompili [24] solved the computation offloading problem for the multicell and multiserver network. Sorkhoh *et al.* [25] proposed a task offloading problem and adopted the Lagrangian relaxation technique to solve this problem. Du *et al.* [26] proposed to solve the computation offloading problem in cooperated fog and cloud system, the proposed schemes aimed at maximizing the total cost of the delay and energy consumption. Eshraghi and Liang [27] studied the network with a computing access point and a cloud server. The offloading problem is more complex under uncertain computing constraints. Bi and Zhang [28] considered the computation offloading problem for the wireless devices, a bisection search algorithm was used to allocate time in the wireless-powered MEC network.

Zhang *et al.* [29] proposed a task offloading scheme focused on smart mobile devices, which considered both energy consumption and latency. Guo and Liu [30] focused on the fiber-wireless network and proposed two computation offloading schemes, the approximation collaboration, and game theory techniques were adopted to solve the computation offloading problems. Zhao *et al.* [31] considered the vehicular networks and proposed a cloud–MEC collaborative solution to offload computation tasks. However, in the real environment with multiple vehicles and MEC servers, each vehicle's computation offloading policy is dynamically changing due to the time-varying characteristic of the vehicular network. Thus, most of these researchers cannot make real-time decisions to achieve a long-time optimal reward.

C. Artificial Intelligence

AI plays an important role in computation offloading problems in MEC. He *et al.* [32] proposed a framework to orchestrate resources in a software-defined network, the DRL technique was used to allocate resources. Huang *et al.* [33] proposed a distributed deep learning algorithm for multiple wireless devices, the proposed algorithm can make near-optimal offloading decisions. Min *et al.* [34] enabled IoT devices with energy harvesting to make an offloading decision based on reinforcement learning, the offloading decision was made according to the long-term award. Chen *et al.* [35] modeled the computation offloading problem as a Markov decision process, the offloading decision was made according to the task queue rate using the deep Q-network technique.

Liu *et al.* [36] proposed a computation offloading solution considering the vehicles as a vehicular edge server, the Q-learning and deep reinforcement algorithms were adopted in generating offloading policy. Li *et al.* [37] proposed a MEC–cloud collaboration solution, the computation task was decided to offload to the MEC server or cloud server using a DRL algorithm. He *et al.* [38] provided online services by collaborating local resources of vehicles, roadside units, and cloud computing.

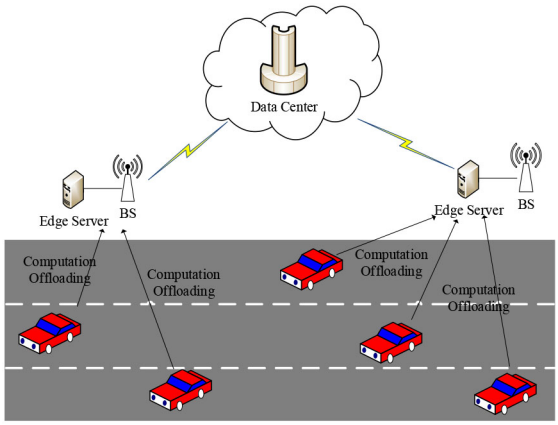

Fig. 1. Vehicular edge network model.

However, none of these works considered the computation offloading problem of multiple vehicles in a period of time. As the multivehicle environment is dynamic, it is difficult to make the optimal decision to change each vehicle's policy to adapt to the dynamic and unstable environment. Compared with the previous works, our research solves the computation offloading problem of multiple vehicles using a multiagent DRL algorithm.

III. System Model

A. Vehicle Edge Network Model

In this article, we consider a vehicle edge network composed of V vehicles, M MEC servers, and one data center. Fig. 1 shows the vehicle edge network model.

1) *Mobile Vehicles:* Many mobile vehicles are moving along the roads in the city, the vehicles are denoted as $\mathcal{V} = \{v_1, \ldots, v_i, \ldots, v_V\}$. We assume that each vehicle contains independent tasks where each task can be offloaded to a MEC server.
2) *MEC Servers:* The MEC servers are distributed along roads in the city, the MEC servers are denoted as $\mathcal{M} = \{m_1, \ldots, m_j, \ldots, m_M\}$.
3) *Data Center:* The data center is responsible for training the deep reinforcement model.

B. Problem Statement

Those important notations used throughout this article are listed in Table I.

We consider that vehicle v_i has a computation task $R_i = \{x_i, y_i\}$, where x_i is the input data of task, y_i is the needed computing resource for executing the task. The computation task can be offloaded to the MEC server. In the computation task, $a_{i,j}$ is the task offloading policy, where $a_{i,j} = 1$ denotes that the vehicle v_i offloads the computation task to the jth MEC server m_j. Thus, the following constraint should be satisfied:

$$\sum_{j \in M} a_{i,j} = 1. \tag{1}$$

TABLE I
NOTATIONS

Notation	Description
$\mathcal{V}$	Vehicles $\mathcal{V} = \{v_1, \cdots, v_i, \cdots, v_V\}$
$\mathcal{M}$	MEC servers $\mathcal{M} = \{m_1, \cdots, m_j, \cdots, m_M\}$
$a_{i,j}$	Task offloading indicator
x_i	Input data of computation task R_i
y_i	Task complexity of computation task R_i
$DT_{i,j}$	Task transmission time of R_i from v_i to m_j
$CT_{i,j}$	Task computation time of R_i from v_i to m_j
P	Vehicle transmission power
$r_{i,j}$	Data rate between vehicle v_i and MEC server m_j
$B_{i,j}$	The uplink transmission channel bandwidth
$f_{i,j}$	The computing resource that server m_j allocates to vehicle v_i
F_M	The computing capacity of MEC server
N_j	The number of tasks allocated to the MEC server m_j
$\boldsymbol{s}$	System state
$o_i(t)$	The observed state of vehicle v_i at time slot t
$\boldsymbol{a}$	The joint action of system
$\mathcal{A}$	The space of all eligible actions
$\mathcal{T}$	Time slots $\mathcal{T} = \{1, \cdots, t, \cdots, T\}$
$a_i(t)$	The offloading action of vehicle v_i at time slot t
π_i, π'_i	The main actor network and target actor networkactor-network
θ_i, θ'_i	The parameters of the main actor network and the target actor network
Q_i, Q'_i	The main critic network and the target critic network
ω_i, ω'_i	The parameters of the main critic network and the target critic network

The execution delay is very important. In the offloading process, we mainly consider two main steps: 1) transfer the input data and 2) compute tasks. The objective of our approach is to minimize the total time. Thereafter, we will evaluate with close formulas the above parameters.

C. Communication Model

Here, we study transmission latency due to communications between vehicles and MEC servers. As the downlink transmission delay is small, thus we omit the downlink transmission delay in this article. The data uplink transmission rate $r_{i,j}$ can be denoted as

$$r_{i,j} = B_{i,j}\log_2\left(1 + \frac{P\left(d_{i,j}\right)^{-\alpha}}{N_j\sigma^2}\right). \tag{2}$$

We assume that the MEC server divides its bandwidth W to vehicles in an average way. N_j is the number of tasks allocated to the MEC server m_j, $B_{i,j} = W/N_j$ is the uplink transmission channel bandwidth, and P is the vehicle transmission power. The variance of white noise in a complex Gaussian channel is denoted as σ^2. $d_{i,j}$ is the distance between the vehicle v_i and MEC server m_j, which can be calculated by vehicle v_i and MEC server m_j using their GPS information. $(d_{i,j})^{-\alpha}$ denotes the channel power gain.

As the vehicle v_i needs to transfer data x_i to MEC server to calculate the task, thus the uplink transmission latency between vehicle v_i and MEC server m_j is calculated as

$$\mathrm{DT}_{i,j} = \frac{x_i}{r_{i,j}}. \tag{3}$$

D. Computation Model

We assume that each server's computational resources are equally shared among all tasks when two or more tasks are offloaded to the same server. Then, the total number of computational resources allocated to each vehicle can be expressed as

$$f_{i,j} = \frac{F_M}{N_j} \tag{4}$$

where F_M is the computation capacity of the MEC server m_j, and we assume that all servers have the same computation capacities for simplicity. Thus, the computational delay can be given as

$$\mathrm{CT}_{i,j} = \frac{y_i}{f_{i,j}}. \tag{5}$$

When the task is calculated by the MEC server, then the data should be transmitted back to the vehicle. The total delay is the sum of data transmission delay and computation delay. The total delay $T_{i,j}$ can be calculated as

$$T_{i,j} = \mathrm{DT}_{i,j} + \mathrm{CT}_{i,j}. \tag{6}$$

E. Optimization Objective

Our optimization objective is to minimize the long-term total delay of all tasks. $a_{i,j}$ is the offloading decision that the task of vehicle v_i offloaded to the MEC server m_j. $a_{i,j}$ is a binary variable. We can write the following optimization goals and constraints:

$$\begin{aligned} \min \ & \sum_{i=1}^{V}\sum_{j=1}^{M} a_{i,j}T_{i,j} \\ \text{s.t.} \ & C_1\text{: } a_{i,j} \in \{0, 1\} \quad \forall v_i \in \mathcal{V} \quad \forall m_j \in \mathcal{M} \\ & C_2\text{: } \sum_{i=1}^{V} a_{i,j} \le 1. \end{aligned} \tag{7}$$

IV. DEEP REINFORCEMENT LEARNING IN VEHICULAR NETWORKS

The optimization problem has high complexity, (7) is an NP-hard problem, which is difficult to solve using traditional solutions. However, DRL is an effective way to solve this problem. In this section, we first formulate the computation offloading problem, we aim to maximize the long-term reward by determining the offloading decisions of vehicles. Each agent corresponds to a vehicle and interacts with the environment according to a policy that specifies how the agent selects actions at each state. Thus, we take advantage of the recent multiagent DRL to solve the computation offloading problem, which can decrease the instability of the multiple vehicle environment. We will introduce the system state, action, and reward in the multiagent model.

A. Modeling of Multivehicle Environment

System State $o_i(t) \in \mathcal{S}$: Let $\mathcal{T}$ be the duration time, which can be divided into T time slots. The state $o_i(t)$ is defined to describe the environment state for the task of vehicle v_i at time slot t, which includes the vehicle information. Suppose vehicle v_i generates a computation task $R_i(t) = \{x_i(t), y_i(t)\}$ at time slot t, it records the task status as well as the available computational resource information. Hence, we use $o_i(t) = \{\mu_i(t), g_i(t), x_i(t), y_i(t)\}$ to represent the vehicular edge computing state. The status contains the following.

1) *Task Trigger Time $\mu_i(t)$:* As time is continuous, we simply map the time to discrete data by 1 s per day.
2) *Vehicle Position $g_i(t)$:* Vehicle v_i's position is obtained through its GPS information.
3) *Task Information $x_i(t)$, $y_i(t)$:* $x_i(t)$ is the input data of the task and $y_i(t)$ is the needed computing resource for executing the task.

Action Space $a_i(t) \in \mathcal{A}$: At time slot t, the vehicle v_i takes an action $a_i(t)$, which indicates that the vehicle v_i will select the jth MEC server, according to the current state $o_i(t) \in \mathcal{S}$, base on the decision policy π_i. The space of all eligible actions is denoted by $\mathcal{A}$. The dimension of $\mathcal{A}$ is M.

Reward Function $r_i(t)$: According to state $o_i(t)$, the vehicle v_i takes an action $a_i(t)$ and obtains the immediate reward $r_i(t)$. The reward function of vehicle v_i can be expressed as

$$r_i(t) = \begin{cases} \frac{C_1}{C_2 + a_i(t) \cdot T_i(t)}, & \text{success} \\ 0, & \text{fail} \end{cases} \tag{8}$$

where C_1 and C_2 are constants, $T_i(t)$ is a vector constituted by $T_{i,j}$, and $a_i(t)$ is an M-dimensional vector.

The delay time $T_{i,j}$ is the task completion time when vehicle v_i offloads its task to a MEC server m_j, which can be computed through (6). $T_i(t) = (T_{i,1}, \ldots, T_{i,j}, \ldots, T_{i,M})$, the dimension of $T_i(t)$ is M. $a_i(t)$ is composed by the probability $p_{i,j}$ that vehicle v_i offloads its task to a MEC server m_j. Thus, $a_i(t) = (p_{i,1}, \ldots, p_{i,j}, \ldots, p_{i,M})$. Then, compute the vehicle's delay time $a_i(t) \cdot T_i(t) = p_{i,1}T_{i,1} + \cdots + p_{i,j}T_{i,j} + \cdots + p_{i,M}T_{i,M}$. Finally, we can obtain the reward $r_i(t)$ for vehicle v_i. Our optimization goal is to minimize the total tasks delay, so our reward function should be negatively correlated with time.

In the multivehicle environment, we use bold $\boldsymbol{s} = \{o_i(t)\}$, $\boldsymbol{a} = \{a_i(t)\}$, and $\boldsymbol{r} = \{r_i(t)\}$ to represent a joint state, action, and reward for multiple vehicles. In order to simplify the representation, we use o_i, a_i, and r_i to represent the state, action, and reward of the vehicle v_i in the current time slot. o'_i, a'_i, and r'_i represent the state, action and reward of the vehicle v_i in the next time slot.

Each vehicle attempts to maximize its own expected discounted reward with the interaction of the environment. To simplify the presentation, we use $Q_i(o_i, a_i)$ as the reward of vehicle v_i takes an action a_i at state o_i at time slot t. We use the Bellman equation to describe the action-value function as

$$Q_i(o_i, a_i) = \mathbb{E}\left[r_i + \gamma \mathbb{E}\left[Q_i\left(o'_i, a'_i\right)\right]\right] \tag{9}$$

where discount factor $\gamma \in [0, 1)$ controls the degree of how far the MDP looks into the future.

Algorithm 1 Execution Algorithm Using ε-Greedy Policy

Input: The weights θ' of target actor network π'
1: Initialize ε as 0.1
2: Load θ' to each vehicle's actor network
3: Each vehicle observes and obtains the state o_i
4: **For** $i = 1, 2, \ldots, V$ **do**
5: Choose a random probability $p \in [0, 1]$
6: **If** $p \leq \varepsilon$ **then**
7: Select a random action $a_i \in \mathcal{A}$
8: **Else**
9: Compute action value a_i where $a_i = \pi'_i(o_i)$
10: **End if**
11: **End for**
Output: Joint action $\boldsymbol{a} = (a_1, \ldots, a_V)$

B. MDRCO Network Architecture

Assume there are V vehicles, each vehicle v_i has an intelligent agent module i. Each agent has two parts: 1) actor network and 2) critic network.

1) *Actor Network:* The actor network contains the main actor network π_i and the target actor network π'_i, and their parameters are θ_i and θ'_i, respectively. The actor network takes the observed state as output and outputs the action through multiple full connection layers. The main actor network is used for actor network training, and the target actor network executes actions for the agents.
2) *Critic Network:* The critic network contains the main critic network Q_i and the target critic network Q'_i, and their parameters are ω_i and ω'_i, respectively. The critic network can evaluate the actor network's actions, it takes the state and action as inputs and outputs the Q value.

The actor network in MDRCO is illustrated in Fig. 2(a). The main actor network and the target actor network have the same network architecture, but they have different parameters. For each vehicle, the actor network's input is vehicle v_i's current observed state $o_i(t)$. The full connection layers and softmax layer output an action vector $a_i(t) = (p_{i,1}, \ldots, p_{i,j}, \ldots, p_{i,M})$. We use $\pi_i(o_i)$ to denote the action vector generation policy.

Algorithm 1 shows the execution algorithm using the ε-greedy policy. In the distributed execution process, the vehicle i first downloads the actor's training weights from the data center and loads them to its own target actor network. Then, the vehicle observes the environment and obtains a state s_i. We explore with the ε-greedy policy in our algorithm. Because the traditional greedy policy always selects the optimal action, which results in that many actions are not explored. The vehicle should explore unknown actions to get experience for some states. Once it gets the experiences, it can exploit what it already knows for the states but keep exploring at the same time. Thus, we select the ε-greedy policy, which can ensure that each action has some probability to be selected. We use the parameter ε to set the tradeoff between exploration and exploitation. We initialize ε as 0.1, the vehicle randomly chooses an action with the probability of 0.1 and computes the action value with a probability of 0.9.

The critic network in MDRCO is illustrated in Fig. 2(b). For each agent i, the critic network takes the system state $\boldsymbol{s}$

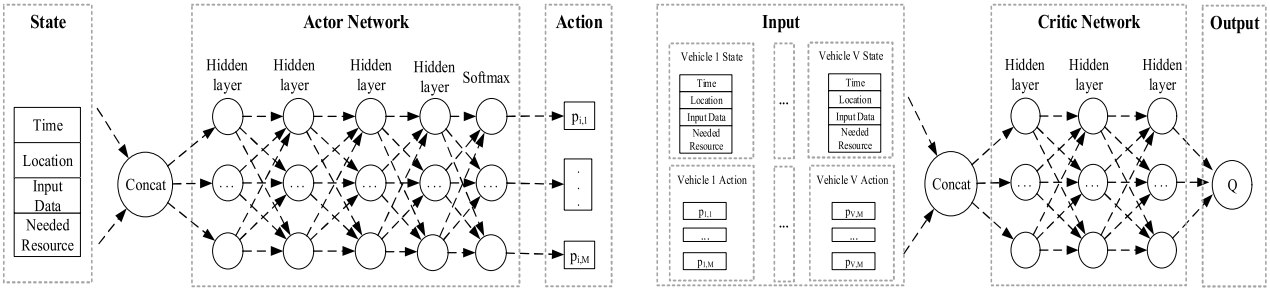

Fig. 2. Structure of the (a) actor network and (b) critic network.

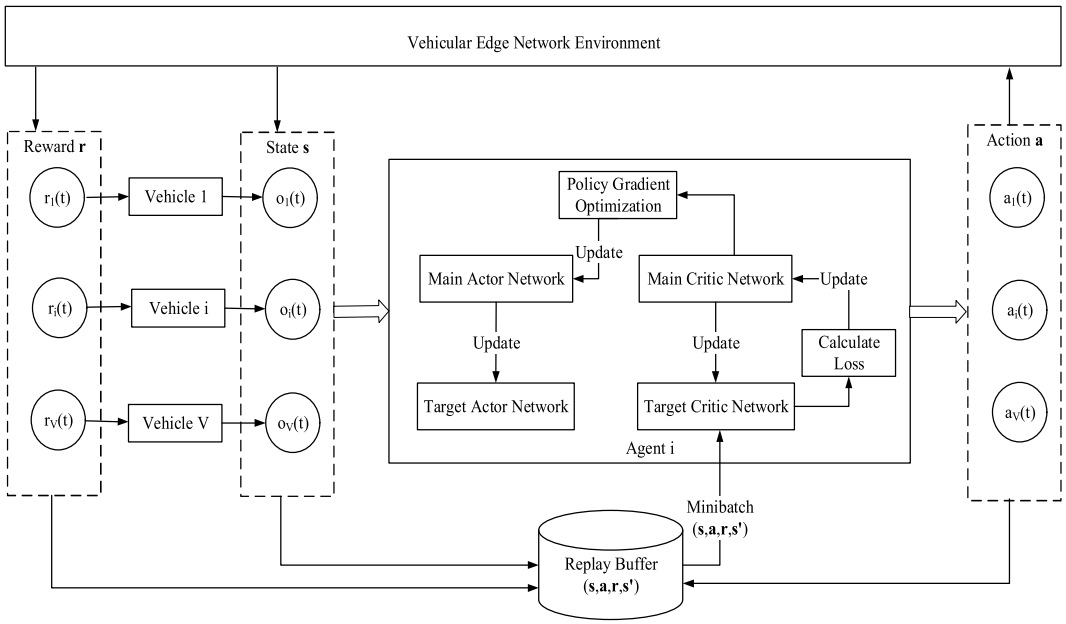

Fig. 3. Architecture for multiagent DRL-based computation offloading in the vehicular network.

and joint action $\boldsymbol{a} = (a_1, \ldots, a_V)$ as inputs and outputs the Q value of vehicle v_i's action. The output Q value is denoted as $Q_i(\boldsymbol{s}, a_1, \ldots, a_V)$.

C. Network Training and Execution

The architecture for multiagent DRL-based computation offloading in the vehicular network is shown in Fig. 3. In the MDRCO training process, the main actor network and main critic network are trained in a centralized way, the target actor network is executed in a decentralized way. Each agent i adopts the main actor network π_i to receive vehicle's state o_i. The vehicle takes the corresponding action a_i and receives the reward r_i. Then, the vehicle gets into the next state o_i'. The data center puts all the vehicles' joint state $\boldsymbol{s}$, joint action $\boldsymbol{a}$, joint reward $\boldsymbol{r}$, and next state into the replay buffer D.

The network training contains three steps: first, update the main critic network. Second, update the main actor network. Finally, update the target critic network and target actor network.

1) Update the Parameter of the Main Critic Network: For each agent i, the data center trains the main critic network. The objective of the training process is to maximize the Q value of vehicle, the Q value is defined as

$$Q = \mathbb{E}\big[r_i + \gamma \times \mathbb{E}\big[Q_i\big(\boldsymbol{s}', a'_1, \ldots, a'_V\big)\big]\big] \tag{10}$$

where γ is the discount factor, $a'_i = \pi'_i(o_i)$.

The data center samples a random batch of transitions from D, computes the Q value of main critic network, and updates the parameters ω_i of main critic network through minimizing the loss function. The loss function is defined as

$$L(\omega_i) = \mathbb{E}\Big[\big(y_i - Q_i(\boldsymbol{s}, a_1, \ldots, a_V)\big)^2\Big] \tag{11}$$

where

$$y_i = r_i + \gamma \times Q'_i\big(\boldsymbol{s}', \pi'_i\big(o'_1\big), \ldots, \pi'_i\big(o'_V\big)\big). \tag{12}$$

Thus, the parameter ω_i of the main critic network is updated through the gradient descent method.

2) Update the Parameter of the Main Actor Network: The parameter of the main actor network can be updated through the deterministic policy gradient descent method. The gradient of the loss function can be obtained by differentiation as

$$\nabla_{\theta_i} L \approx \mathbb{E}\big[\nabla_{\theta_i} \log(\pi_i(o_i)) \nabla_{a_i} Q_i(\boldsymbol{s}, \pi_i(o_1), \ldots, \pi_i(o_V))\big]. \tag{13}$$

3) Update the Parameters of Target Networks: The main target actor network's parameter θ'_i and main target critic network's parameter ω'_i are updated for each vehicle

$$\theta'_i = \tau\theta_i + (1-\tau)\theta'_i \tag{14}$$

$$\omega'_i = \tau\omega_i + (1-\tau)\omega'_i \tag{15}$$

where $\tau \in [0, 1]$ is the learning rate of the parameters.

In the process of training the network, the actor network should obtain the vehicle's local state, and the critic network needs all the vehicles' actions to output the Q value. After finishing the training process, the execution process only needs the target actor network, each agent can only make the optimal action according to its state. Because the training process needs strong computation power and long training time, thus the data center needs to transfer the training process.

The training algorithm of computation offloading in vehicular edge network is shown in Algorithm 2. Each vehicle's agent initializes its parameters of the actor network and critic network (lines 1–5). At each time slot t, each vehicle obtains its state $o(t)$ (lines 8–12). Then, each vehicle puts its state $o(t)$ as the input of its main actor network, and computes the joint action through Algorithm 1 (line 13). After each vehicle execute their actions, each vehicle obtains the reward and next state (line 14). Finally, all the tasks' offloading records $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s}')$ are stored in the replay buffer D (line 15). For every C time slots, each vehicle's agent updates its actor network and critic network. Randomly choose m records from historical replay buffer D (line 18), first updates the main critic network (line 21), then adopts the policy gradient method to update the main actor network (line 22). Finally, update the parameters of target networks for each vehicle (line 24).

Algorithm 2 Training Algorithm of Computation Offloading in the Vehicular Edge Network

Input: Historical vehicle trajectories
1: Initialize the learning rate τ as 0.05
2: Initialize a random integer variable C
3: Initialize replay buffer D to capacity N
4: Initialize the main actor network π with weights θ, the main critic network Q with weights ω
5: Initialize the target actor network π' with weights θ', the target critic network Q' with weights ω'
6: **For** episode $k = 1, 2, \ldots, K$ **do**
7: Initialize the state **s**
8: **For** each time slot $t = 1, 2, \ldots, T$ **do**
9: **For** each vehicle $i = 1, 2, \ldots, V$ **do**
10: Compute $\mu_i(t), g_i(t)$ based on the current time and v_i's GPS
11: Set $s_i(t) = (\mu_i(t), g_i(t), x_i(t), y_i(t))$
12: **End for**
13: Sample joint action $\boldsymbol{a}$ using Algorithm 1 for state $\boldsymbol{s}$
14: Execute action $\boldsymbol{a}$, compute reward $\boldsymbol{r}$, obtain next state $\boldsymbol{s'}$
15: Store all the transitions $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s'})$ in the replay buffer
16: **End for**
17: **If** $k\%C == 0$ **then**
18: Sample a batch of m transitions $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s'})$ from D
19: **For** $i = 1, 2, \ldots, V$ **do**
20: Set $y_i = r_i + \gamma \times Q'_i(s', \pi_i(o'_1), \ldots, \pi_i(o'_V))$
21: Update critic network by minimizing the loss: $L(\omega_i) = \frac{1}{m}\sum_i (y_i - Q_i(\boldsymbol{s}, a_1, \cdots, a_V))^2$
22: Update actor network θ_i by policy gradient method
23: **End for**
24: Update the parameters of target networks for vehicles: $\theta'_i = \tau\theta_i + (1-\tau)\theta'_i$, $\omega'_i = \tau\omega_i + (1-\tau)\omega'_i$
25: **End if**
26: **End for**
Output: The parameters of target actor θ'

V. Experimental Results

We use a real vehicle data set to verify our experiment. The data set includes the GPS of Rome city collected in 30 days, the time ranges from 2014/02/01 to 2014/03/02. The period is from 7 A.M. to 8 A.M. Fig. 4 shows the real vehicle trajectory of Rome city.

Each trajectory consists of a set of GPS points with timestamps. The GPS point includes taxi ID, timestamp, and taxi location. The first 15 days are training data, the next 15 days are test data. After we preprocess the raw data, 25 vehicles are moving on the road every day. We select 10–20 vehicles' trajectories as the learning data, there are 50 MEC servers randomly deployed on the road. The actor network has four fully connected layers and two hidden layers, the hidden layers have 512 and 128 neurons, respectively. The critic network has three fully connected layers and two hidden layers, the hidden layers have 512 and 256 neurons, respectively. The actor's learning rate is 0.001, the critic's learning rate is 0.01. The discount factor is 0.95.

The experiment contains three processes: 1) the preprocessing process; 2) offline training process; and 3) online computation offloading process. The experiments are all implemented by Python. In the offline training process, we train the actor network and critic network. In the online computation offloading process, each vehicle chooses to offload its computation task to one MEC server. Table II shows the experimental parameters.

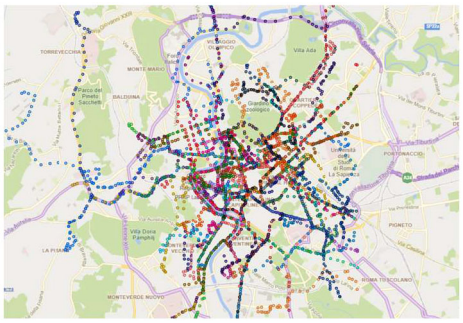

Fig. 4. Real vehicle trajectory.

A. Network Performance Evaluation

We compare our proposed MDRCO scheme with the other two schemes.

1) *NN Algorithm:* NN algorithm is the most direct method, which does not have the training process. In the task offloading process, NN directly selects the nearest MEC server to offload tasks.
2) *AC Algorithm:* The AC algorithm is applied in [15], which can be used to process continuous state and action

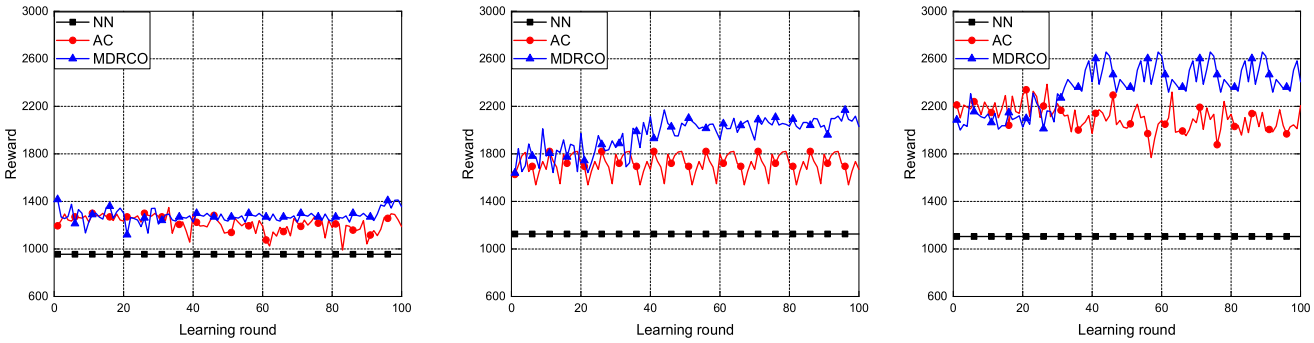

Fig. 5. Performance comparison of reward in different learning rounds. (a) Number of vehicles = 10. (b) Number of vehicles = 15. (c) Number of vehicles = 20.

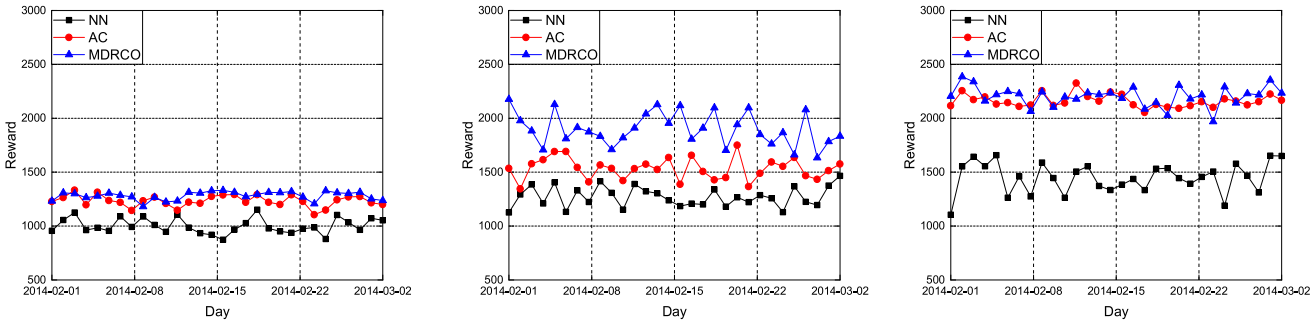

Fig. 6. Performance comparison of reward in different days. (a) Number of vehicles = 10. (b) Number of vehicles = 15. (c) Number of vehicles = 20.

TABLE II
EXPERIMENTAL PARAMETERS

Parameter	Value
Number of vehicles	[10, 15, 20]
The GPS range	(41.8,12.4)-(41.95,12.52)
Number of MEC servers	50
The maximum bandwidth of the MEC server	2MHZ
The capability of MEC server	6.3GHZ
The time slot	1s
Noise power	1Mbytes
The channel gain	0.25

space. We consider the AC algorithm uses a data center to train model and make offloading decisions in a centralized way.

Fig. 5 shows the performance comparison of reward in different learning rounds. The reward denotes the total reward of all the vehicles. The proposed MDRCO algorithm achieves better reward performance than the NN algorithm and AC algorithm, it converges at a fast speed and obtains maximum reward only after 55 learning rounds. Compared with the AC algorithm, the MDRCO algorithm can obtain better reward performance, and the convergence is more stable and less fluctuation. Because MDRCO considers the impact of the multiple mobile vehicle environment on the training stability and performance. The NN algorithm does not change with the increase of learning rounds and it has the minimum reward compared with other algorithms. Because the NN algorithm does not have the offline training process, and it directly selects a MEC server to offload tasks. Our MDRCO algorithm trains vehicles in the data center and does not keep the global information. The training stability is largely improved, the MDRCO algorithm can obtain maximum reward and converge quickly.

Fig. 6 shows the performance comparison of reward on different days. Our MDRCO algorithm can obtain maximum reward performance compared with AC and NN algorithms. The data center in the AC algorithm makes offloading decisions and transfers decisions to vehicles, but the vehicles' locations change with time, which results in a decrease of reward. The NN algorithm obtains the minimum reward in three algorithms, as the NN algorithm selects the nearest MEC server. Some MEC servers are prone to be overloaded and unable to process tasks efficiently. MDRCO algorithm makes task offloading decisions in a distributed way, thus the vehicles can make real-time task offloading and obtain a maximum reward. Compared to the NN algorithm, the MDRCO algorithm can improve the reward by 29.4%, 92.7%, and 54.3% approximately when the number of vehicles is 10, 15, and 20, respectively. Compared to the AC algorithm, the MDRCO scheme can improve the reward by 4.3%, 24.8%, and 1.7% approximately when the number of vehicles is 10, 15, and 20, respectively. The performance of reward can be optimized by 58.8% and 10.3% using our MDRCO algorithm compared with the NN and AC algorithms.

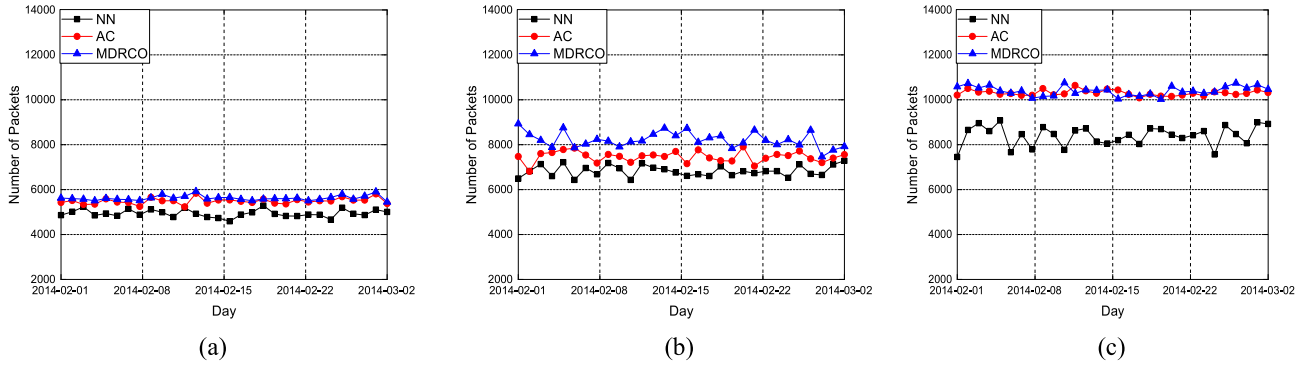

Fig. 7. Performance comparison of packets in different days. (a) Number of vehicles = 10. (b) Number of vehicles = 15. (c) Number of vehicles = 20.

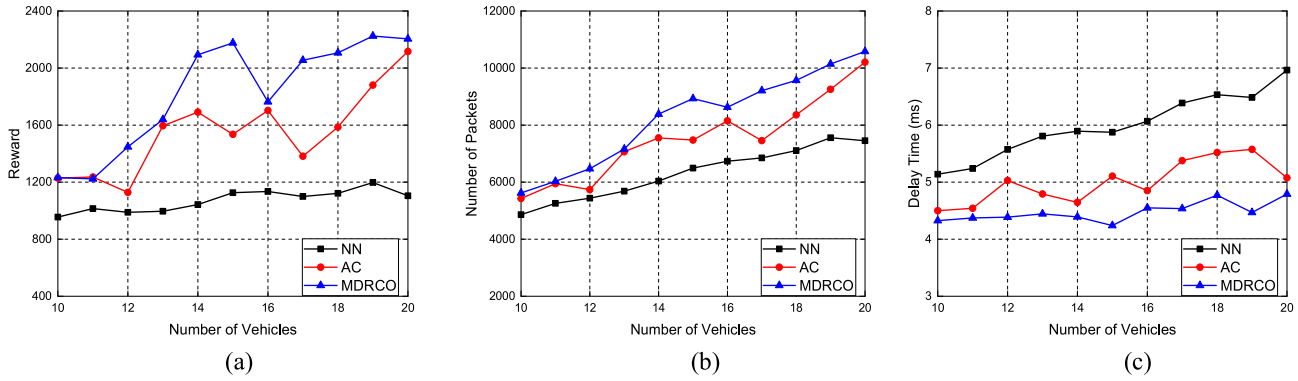

Fig. 8. Performance comparison in different number of vehicles. (a) Reward. (b) Packets. (c) Delay time.

The packets denote the number of data processed by the system, which represents the processing capacity of the system. Fig. 7 compares the performance of the packet of three algorithms on different days. The MDRCO algorithm can process maximum packets compared with NN and AC algorithms because the offloading policies in multiple vehicles coordinate with each other and prevent choosing the same MEC server simultaneously, which can accelerate the processing rate of the system. Compared to the NN algorithm, the MDRCO algorithm can improve the packets by 14.1%, 21.4%, and 24.3% approximately when the number of vehicles is 10, 15, and 20, respectively. Compared to the AC algorithm, the MDRCO scheme can improve the packets by 2.6%, 10.7%, and 0.7% approximately when the number of vehicles is 10, 15, and 20, respectively. The performance of packets can be optimized by 19.3% and 4.7% using our MDRCO algorithm compared with the NN and AC algorithms.

Fig. 8(a) compares the reward performance of three algorithms in a different number of vehicles. Our proposed MDRCO algorithm is significantly better than the other two algorithms. The vehicles in MDRCO can make full use of cooperation and obtain a better reward. Fig. 8(b) compares the packets of three algorithms in a different number of vehicles. The MDRCO algorithm's packets are maximum in three algorithms. NN and AC algorithms do not consider other vehicles' information, which may lead to the scenario that multiple tasks are sent to the same MEC server and decrease the processed packets. Fig. 8(c) compares the delay performance of three algorithms in different numbers of vehicles. The delay time denotes the execution time of all the tasks. MDRCO algorithm's delay time is minimum in three algorithms, MDRCO decreases delay 21.7% and 10.2% compared with the AC algorithm and NN algorithm separately. Our goal is to minimize the delay time of the system. MDRCO uses vehicles' information to train the model, and vehicles learn to cooperate to obtain a better reward. Thus, our MDRCO algorithm has a good performance on the delay time.

In conclusion, our vehicular computation offloading scheme integrates with the AI method. The proposed MDRCO scheme can improve vehicles' reward, process more packets, and decrease delay time. The MDRCO scheme can be applied into many scenarios, such as autonomous driving, smart city and intelligent transportation.

VI. Conclusion

Intelligent vehicles are becoming an emerging trend, there are more and more intelligent vehicles moving along the roads. However, as the vehicles are computation intensive, the tasks need to be offloaded to MEC servers. In order to make intelligent task offloading decisions, we proposed a multiagent deep reinforcement computation offloading scheme in the vehicular edge network. The vehicular edge network contains a data center, MEC servers, and vehicles. The data center is responsible for the training model, each vehicle selects a MEC server to offload tasks in a distributed way. Our MDRCO scheme can decrease the unstability of a multivehicle environment and make real-time offloading decisions. Our objective is to minimize the task execution delay in a long-term time period. The

multiagent DRL algorithm trains the actor network and critic network in a centralized way. The vehicles make real-time task offloading decisions in a distributed way. We evaluated our MDRCO scheme in a real vehicle data set, the evaluation results showed that our scheme can achieve maximum reward compares with the NN algorithm and AC algorithm.

REFERENCES

[1] A. Yang, J. Weng, K. Yang, C. Huang, and X. Shen, "Delegating authentication to edge: A decentralized authentication architecture for vehicular networks," *IEEE Trans. Intell. Transp. Syst.*, early access, Sep. 24, 2020, doi: 10.1109/tits.2020.3024000.

[2] X. Liu, H. Song, and A. Liu, "Intelligent UAVs trajectory optimization from space-time for data collection in social networks," *IEEE Trans. Netw. Sci. Eng.*, early access, Aug. 19, 2020, doi: 10.1109/tnse.2020.3017556.

[3] S. Misra and S. Bera, "Soft-VAN: Mobility-aware task offloading in software-defined vehicular network," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2071–2078, Feb. 2020, doi: 10.1109/TVT.2019.2958740.

[4] V. H. Hoang, T. M. Ho, and L. B. Le, "Mobility-aware computation offloading in MEC-based vehicular wireless networks," *IEEE Commun. Lett.*, vol. 24, no. 2, pp. 466–469, Feb. 2020, doi: 10.1109/LCOMM.2019.2956514.

[5] A. Yang, J. Weng, N. Cheng, J. Ni, X. Lin, and X. Shen, "DeQoS attack: Degrading quality of service in VANETs and its mitigation," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4834–4845, May 2019, doi: 10.1109/TVT.2019.2905522.

[6] Y. Ren, T. Wang, S. Zhang, and J. Zhang, "An intelligent big data collection technology based on micro mobile data centers for crowdsensing vehicular sensor network," *Pers. Ubiquitous Comput.*, to be published, doi: 10.1007/s00779-020-01440-0.

[7] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 18, no. 2, pp. 319–333, Feb. 2019, doi: 10.1109/TMC.2018.2831230.

[8] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6774–6785, Aug. 2019, doi: 10.1109/JIOT.2019.2911455.

[9] X. Zhu, Y. Luo, A. Liu, W. Tang, and M. Z. A. Bhuiyan, "A deep learning-based mobile crowdsensing scheme by predicting vehicle mobility," *IEEE Trans. Intell. Transp. Syst.*, early access, Oct. 7, 2020, doi: 10.1109/tits.2020.3023446.

[10] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint offloading and resource allocation in vehicular edge computing and networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–7, doi: 10.1109/GLOCOM.2018.8648004.

[11] Z. Wang, Z. Zhong, D. Zhao, and M. Ni, "Vehicle-based cloudlet relaying for mobile computation offloading," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11181–11191, Nov. 2018, doi: 10.1109/TVT.2018.2870392.

[12] H. Zhang, W. Wu, C. Wang, M. Li, and R. Yang, "Deep reinforcement learning-based offloading decision optimization in mobile edge computing," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2019, pp. 1–7, doi: 10.1109/WCNC.2019.8886332.

[13] Y. Liu, Z. Zeng, X. Liu, X. Zhu, and M. Z. A. Bhuiyan, "A novel load balancing and low response delay framework for edge-cloud network based on SDN," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 5922–5933, Jul. 2020, doi: 10.1109/JIOT.2019.2951857.

[14] S. Huang, A. Liu, S. Zhang, T. Wang, and N. Xiong, "BD-VTE: A novel baseline data based verifiable trust evaluation scheme for smart network systems," *IEEE Trans. Netw. Sci. Eng.*, early access, Aug. 7, 2020, doi: 10.1109/tnse.2020.3014455.

[15] H. Yang, X. Xie, and M. Kadoch, "Intelligent resource management based on reinforcement learning for ultra-reliable and low-latency IoV communication networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4157–4169, May 2019, doi: 10.1109/TVT.2018.2890686.

[16] Z. Li and C. Guo, "Multi-agent deep reinforcement learning based spectrum allocation for D2D underlay communications," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 1828–1840, Feb. 2020, doi: 10.1109/TVT.2019.2961405.

[17] Y. Xu, J. Ren, G. Wang, C. Zhang, J. Yang, and Y. Zhang, "A blockchain-based nonrepudiation network computing service scheme for industrial IoT," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3632–3641, Jun. 2019, doi: 10.1109/TII.2019.2897133.

[18] S. Zhang, G. Wang, M. Z. A. Bhuiyan, and Q. Liu, "A dual privacy preserving scheme in continuous location-based services," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 4191–4200, Oct. 2018, doi: 10.1109/JIOT.2018.2842470.

[19] S.-W. Ko, K. Han, and K. Huang, "Wireless networks for mobile edge computing: Spatial modeling and latency analysis," *IEEE Trans. Wireless Commun.*, vol. 17, no. 8, pp. 5225–5240, Aug. 2018, doi: 10.1109/TWC.2018.2840120.

[20] P. Wang, Z. Zheng, B. Di, and L. Song, "HetMEC: Latency-optimal task assignment and resource allocation for heterogeneous multi-layer mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 18, no. 10, pp. 4942–4956, Aug. 2019, doi: 10.1109/twc.2019.2931315.

[21] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020, doi: 10.1109/TMC.2019.2928811.

[22] S. Jošilo and G. Dán, "Wireless and computing resource allocation for selfish computation offloading in edge computing," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, 2019, pp. 2467–2475, doi: 10.1109/INFOCOM.2019.8737480.

[23] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017, doi: 10.1109/TCOMM.2017.2699660.

[24] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019, doi: 10.1109/TVT.2018.2881191.

[25] I. Sorkhoh, D. Ebrahimi, R. Atallah, and C. Assi, "Workload scheduling in vehicular networks with edge cloud capabilities," *IEEE Trans. Veh. Technol.*, vol. 68, no. 9, pp. 8472–8486, Jul. 2019, doi: 10.1109/tvt.2019.2927634.

[26] J. Du, L. Zhao, J. Feng, and X. Chu, "Computation offloading and resource allocation in mixed fog/cloud computing systems with min–max fairness guarantee," *IEEE Trans. Commun.*, vol. 66, no. 4, pp. 1594–1608, Apr. 2018, doi: 10.1109/TCOMM.2017.2787700.

[27] N. Eshraghi and B. Liang, "Joint offloading decision and resource allocation with uncertain task computing requirement," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, 2019, pp. 1414–1422, doi: 10.1109/INFOCOM.2019.8737559.

[28] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018, doi: 10.1109/TWC.2018.2821664.

[29] J. Zhang *et al.*, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2633–2645, Aug. 2018, doi: 10.1109/JIOT.2017.2786343.

[30] H. Guo and J. Liu, "Collaborative computation offloading for multiaccess edge computing over fiber-wireless networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4514–4526, May 2018, doi: 10.1109/TVT.2018.2790421.

[31] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7944–7956, Jun. 2019, doi: 10.1109/tvt.2019.2917890.

[32] Y. He, F. R. Yu, N. Zhao, V. C. M. Leung, and H. Yin, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017, doi: 10.1109/MCOM.2017.1700246.

[33] L. Huang, X. Feng, A. Feng, Y. Huang, and L. P. Qian, "Distributed deep learning-based offloading for mobile edge computing networks," *Mobile Netw. Appl.*, to be published, doi: 10.1007/s11036-018-1177-x.

[34] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019, doi: 10.1109/TVT.2018.2890685.

[35] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019, doi: 10.1109/JIOT.2018.2876279.

[36] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11158–11168, Nov. 2019, doi: 10.1109/TVT.2019.2935450.

[37] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Jun. 2018, pp. 1–6, doi: 10.1109/WCNC.2018.8377343.

[38] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018, doi: 10.1109/TVT.2017.2760281.

Xiaoyu Zhu (Member, IEEE) received the B.Sc. degree in electronic information engineering, the M.Sc. degree in computer science, and the Ph.D. degree in computer science from Central South University, Changsha, China, in 2010, 2013, and 2019, respectively.

She is currently a Postdoctoral Fellow with the School of Automation, Central South University. She has been a visiting student with Temple University, Philadelphia, PA, USA. Her research interests include vehicular network, cloud/edge computing, and network security.

Yueyi Luo received the B.Sc. and M.Sc. degrees from Central South University, Changsha, China, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, School of Computer Science and Engineering.

He is currently a Lecturer with the School of Mathematics and Statics, Central South University. His research interests include edge network, vehicular network, and machine learning.

Anfeng Liu received the M.Sc. and Ph.D. degrees in computer science from Central South University, Changsha, China, in 2002 and 2005, respectively.

He is a Professor with the School of Information Science and Engineering, Central South University. His major research interest is wireless sensor networks.

Prof. Liu is a member (E200012141M) of the China Computer Federation.

Md Zakirul Alam Bhuiyan (Senior Member, IEEE) received the Ph.D. degree from Central South University, Changsha, China, in 2013.

He is currently an Assistant Professor with the Department of Computer and Information Sciences, Fordham University, New York, NY, USA, and the Founding Director of the Fordham Dependable and Secure System Lab (DependSys). Earlier, he worked as an Assistant Professor with Temple University, Philadelphia, PA, USA. His research focuses on dependability, cybersecurity, big data, and IoT/CPS applications. His work (including 40+ JCR Q1 papers) in these areas published in top-tier venues, such as IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, *IEEE Communications Magazine*, *IEEE Internet of Things*, *ACM Transactions on Sensor Networks*, *ACM Transactions on Autonomous and Adaptive Systems*, ACM CS, *Information Sciences* (Elsevier), *Information Systems* (Elsevier), *Journal of Systems Architecture* (Elsevier), *Journal of Network and Computer Applications* (Elsevier), and *Future Generation Computer Systems* (Elsevier).

Dr. Bhuiyan has received numerous awards, including the IEEE TCSC Early Career Researcher, the IEEE Outstanding Leadership Award, and the IEEE Service Award. Several research works of him have got recognitions of ESI Highly Cited Papers (since 2018). He has also served as an Organizer, the General Chair, the Program Chair, the Workshop Chair, and a TPC Member for various international conferences, including IEEE INFOCOM, IEEE ICCCN, and IEEE COMSAC. He is a member of ACM.

Shaobo Zhang received the B.Sc. and M.Sc. degrees in computer science from Hunan University of Science and Technology, Xiangtan, China, in 2003 and 2009, respectively, and the Ph.D. degree in computer science from Central South University, Changsha, China, in 2017.

He is currently a Lecturer with the School of Computer Science and Engineering, Hunan University of Science and Technology. His research interests include privacy and security issues in social networks and cloud computing.