Deep reinforcement learning based distributed computation offloading in vehicular edge computing networks

Liwei Geng, Hongbo Zhao, Jiayue Wang, Aryan Kaushik, Shuai Yuan, Wenquan Feng

Publication date
10-06-2023

Document Version
Accepted version

Citation for this work (American Psychological Association 7th edition)
Geng, L., Zhao, H., Wang, J., Kaushik, A., Yuan, S., & Feng, W. (2023). *Deep reinforcement learning based distributed computation offloading in vehicular edge computing networks* (Version 1). University of Sussex. https://hdl.handle.net/10779/uos.23494646.v1

Published in
IEEE Internet of Things Journal

Link to external publisher version
https://doi.org/10.1109/JIOT.2023.3247013

Deep Reinforcement Learning Based Distributed Computation Offloading in Vehicular Edge Computing Networks

Liwei Geng, Hongbo Zhao*, Jiayue Wang, Aryan Kaushik, Shuai Yuan, Wenquan Feng

***Abstract*—Vehicular edge computing has emerged as a promising paradigm by offloading computation-intensive latency-sensitive tasks to mobile-edge computing (MEC) servers. However, it is difficult to provide users with excellent quality-of-service (QoS) by relying only on these server resources. Therefore, in this paper, we propose to formulate the computation offloading policy based on deep reinforcement learning (DRL) in a vehicle-assisted vehicular edge computing network (VAEN) where idle resources of vehicles are deemed as edge resources. Specifically, each task is represented by a directed acyclic graph (DAG) and offloaded to edge nodes according to our proposed subtask scheduling priority algorithm. Further, we formalize the computation offloading problem under the constraints of candidate service vehicles models, which aims to minimize the long-term system cost including delay and energy consumption. To this end, we propose a distributed computation offloading algorithm based on multi-agent DRL (DCOM), where an improved actor-critic network (IACN) is devised to extract features, and a joint mechanism of prioritized experience replay and adaptive n-step learning (JMPA) is proposed to enhance learning efficiency. The numerical simulations demonstrate that, in VAEN scenario, DCOM achieves significant decrements in the latency and energy consumption compared with other advanced benchmark algorithms.**

***Index Terms*—Computation offloading, deep reinforcement learning, mobile-edge computing, vehicular edge computing networks.**

I. Introduction

THE Internet of Vehicles (IoV), a typical scenario of the Internet of Things (IoT) focused on the transportation industry, has recently attracted tremendous attention from academia and industry [1], [2]. Constructing a network topology among communication entities such as vehicles, pedestrians, and roadside units (RSU), IoT can provide efficient and secure information services to meet the increasing demand for a transportation environment [3]. With the rapid development of IoV and artificial intelligence (AI), more and more vehicles prompt a large number of smart applications, such as smart driving and augmented reality (AR) [4], [5], which are largely safety-related and require ultra-low latency constraints. Some of these applications are also computation-intensive, resulting in a large amount of energy consumption for their execution. However, relying solely on resource-constrained vehicle devices (VDs) is not sufficient to meet the requirements of these applications, in terms of latency or energy consumption.

As a promising solution, mobile-edge computing (MEC) has attracted tremendous attention by extending the computation capacity of the vehicular edge layer, which consists of fixed edge servers and RSUs near VDs [6]. Specifically, computation offloading technology, which offloads computation-intensive and latency-sensitive tasks to the edge servers, not only breaks the resource limits of VDs, but also reduces the high transmission latency due to mobile cloud computing (MCC) [7]. Nevertheless, as the vehicle density increases, the resources required for the MEC server are not sufficient to guarantee the quality-of-service (QoS) of all vehicles [8]. Moreover, if we only consider MEC servers as edge nodes, to reduce the latency and energy consumption, most vehicles will offload tasks, which causes a great burden on MEC servers. In addition, the resources of MEC servers are expensive, and extensive use increases the economic cost of the system. To this end, it is necessary to fully exploit the underutilized resources in vehicular edge computing networks (VECNs). Recently, the technical enhancements in device-to-device (D2D) communication technologies and vehicle hardware have made it possible to leverage the idle resources of some vehicles [9].

Therefore, to make the most of such resources, in this paper, we consider a vehicle-assisted vehicular edge computing network (VAEN) scenario, where mobile edge nodes are composed of idle vehicles with resources and fixed edge servers. Recently, several studies have considered the private properties of vehicles and thus utilized federal learning to protect user privacy or designed incentives to encourage vehicle participation in edge services. However, in this paper, we focus on the resources of idle vehicles and the formulation of offloading decisions. By utilizing the idle resources of vehicles as edge resources and providing computation offloading services near the request vehicles (RVs), tasks execution latencies are reduced, the resource utilization of RVs is optimized, and the load pressure on MEC servers is relieved with the addition of mobile edge nodes. In addition, vehicle-to-vehicle (V2V) communication can ensure the success of tasks, improves the stability of the system, and enhances user experience. Particularly, the idle resources of RVs are fully utilized in the VAEN scenario, which is critical for VECNs with valuable resources.

Recently, two models of computation offloading tasks have

This work was supported by the National Natural Science Foundation of China under Grant 61901015.

Liwei Geng, Hongbo zhao, Jiayue Wang, Shuai Yuan and Wenquan Feng are with the School of Electronic and Information Engineering, Beihang University, Beijing 100191, China (e-mail: liviageng@buaa.edu.cn; bhzhb@buaa.edu.cn; wangjiayue@buaa.edu.cn; ysleon@buaa.edu.cn; buaafwq@buaa.edu.cn).

Aryan. Kaushik is with the School of Engineering and Informatics, University of Sussex, Brigthon BNI 9RH, United Kingdom (e-mail: aryan.kaushik@sussex.ac.uk)

been considered in VECNs, binary offloading and partial offloading [10]. Specifically, in binary offloading, the task is treated as a whole and either executed locally or offloaded [11], [12]. In partial offloading, the task is arbitrarily divided into two parts, one for local execution and the other for offloaded execution [13]. However, due to the rapid progress of AI, numerous computation-intensive and latency-sensitive applications are generated that can be divided into several subtasks with considerable and complex dependencies between them [14]. Therefore, computation offloading faces mainly two challenges in the VAEN scenario. 1) Designing offloading strategies for tasks with complex subtask dependencies is a challenge. Technically, on the one hand, the scheduling order of subtasks is difficult to determine due to the existence of serial and parallel dependencies of subtasks. On the other hand, the dependencies of subtasks increase the complexity of the system state space.

2) Considering V2V communication in the high-dynamic VAEN scenario is another challenge. Due to the relative movement of the vehicles, considering V2V communication will make the vehicle mobility models and constraints more flexible and dynamic. Specifically, if the communication between the RV and the SV is interrupted, the execution of the task will be disrupted.

Furthermore, the essence of computation offloading is to develop an optimal offloading strategy, which means that optimization methods should be paid utmost attention. However, applying conventional methods, such as convex optimization and game theory, in large-scale scenarios leads to an exponential increase in search space and heavy computational burdens [13]. Recently, deep reinforcement learning (DRL) has attracted the attention of academia and industry in searching for the optimal solutions for computation offloading problems [5]. The actor-critic algorithm shows promising performance in formulating offloading decisions [15], [16].

In the conventional actor-critic, both actor and critic networks are composed of fully connected networks (FCNs) [17], which makes model training inefficient due to a large number of parameters. Moreover, one-step Temporal-Difference Learning (TD) is commonly utilized to train the critic network, causing the critic network to converge slowly [18]. Additionally, in the training phase, transitions are sampled uniformly from the experience replay buffer, ignoring the differences in the importance of the samples.

To address these challenges and dilemma of VAEN, in this paper, we divide the task into multiple subtasks and depict the dependencies with a directed acyclic graph (DAG). Moreover, based on the DAG, we propose a subtask scheduling priority algorithm to further reduce the task execution latency. As for vehicle mobility, we conduct a candidate service vehicle (CSV) model to undertake communication constraints. Further, we propose a distributed computation offloading algorithm based on multiagent DRL (DCOM). Specifically, to better extract the features of system state spaces, we devise an improved actor-critic network (IACN) composed of 2-D convolution (Conv2D) layers and a long short-term memory (LSTM) layer. In addition, to improve the network learning efficiency, we propose a joint mechanism of prioritized experience replay and adaptive n-step learning (JMPA).

In summary, in this paper, we focus on making real-time offloading decisions for fine-grained tasks in the VAEN scenario, with the aim of minimizing the long-term system cost, including delay and energy consumption. Specifically, the contributions of the paper are as follows.

1) Modeling: We consider a VAEN scenario, wherein idle vehicles and MEC servers are jointly utilized as edge nodes to provide offloading services for fine-grained tasks. Specifically, a CSV model is constructed and a subtask scheduling priority algorithm is devised to reduce the delay of task executions.

2) Algorithm: We formulate an optimization problem with integrated considerations of delay and energy consumption, represent the problem as a Markovian decision process (MDP), and design the state, action and reward specifically based on VAEN. Furthermore, a multiagent DRL-based method, namely DCOM, is proposed to solve the optimization problem. Especially, IACN and JMPA are devised to extract features and improve the learning efficiency in DCOM.

3) Simulation: We evaluate the performance of DCOM with numerical simulations. The simulation results demonstrate that VAEN, IACN, and JMPA are effective in increasing the rewards of RVs. Furthermore, DCOM significantly outperforms other benchmarks in reducing latency and energy consumption under different parameter settings.

The remainder of this paper is organized as follows. Section II reviews the related work. Section III presents the system model. The problem formulation and offloading strategy are elaborated in Section IV and Section V, respectively. Section VI analyzes and discusses the experimental results, and Section VII concludes this paper.

II. Related Work

In this section, we review the related work concerning computation offloading in cellular networks and vehicular networks.

A. Computation Offloading in MEC

In recent years, mobile edge computing offload technologies have received considerable attention in both academia and industry. Initially, most work focused on single-user scenarios. For example, Mao et al. [19] proposed a dynamic binary computing offloading algorithm based on Lyapunov optimization, which aimed to minimize the delay. Mohamed et al. [20] presented an offloading decision algorithm that minimized energy consumption. Subsequently, to apply to the actual application scenario, computation offloading in multi-user scenarios was studied. In [21], a computation offloading method based on the game theory was proposed to trade off the energy consumption and delay in a multi-user and multi-channel environment. Wu et al. [22] studied a multi-user task offloading strategy based on the game theory to reduce computational complexity. Despite the research progress, most of the above work focused on the binary offloading method. However, the binary offloading is a rough 0-1 offloading method that fails to make full use of system resources, i. e. bandwidth, computing resources. Moreover, in practice, many large-scale applications, i.e., AR applications [23], are composed of many components. Therefore, if tasks are simply considered as a whole, the binary offload model is struggling to satisfy the delay constraint when large-scale applications are generated [24].

To fully utilize system resources and reduce task execution latency, many studies considered to split these tasks and performed partial offloading [25], [26]. Zhang et al. [27] formalized the partial offloading decision problem as a highly complex nonlinear constraint in the multi-mobile device scenario. Mao et al. [28] used buffer stability constraints to formulate energy minimization problems, and then proposed an online algorithm based on Lyapunov optimization to determine the optimal CPU frequency. Meanwhile, with the development of IOV, many studies have extended edge networks to vehicular networks and further proposed VECNs. Specifically, Zhao et.al [29] constructed an intelligent offloading system for VECNs, where the task scheduling and resource allocation strategy were formulated as a joint optimization problem to maximize the user's quality-of-experience (QoE). Zhao et al. [30] studied a computation offloading strategy to consider the cloud-edge cooperation. In addition, the private attributes of vehicles have attracted significant attention. Feng et al. [31] studied the optimization design of federated learning in MEC systems and achieved the privacy protection of private users. Lu et al. [32] combined federated learning and blockchain in the context of MEC to address the problem of user privacy and security. Zhou et al. [33] proposed an auction-based incentive mechanism to incentivize private users to utilize and participate in the offloading service.

By simply dividing the task into two parts, the task execution latency relative to binary offloading are reduced. However, with the increasing complexity of IoV applications, internal dependencies of tasks become essential. Therefore, we divide the computation-intensive and delay-sensitive tasks into multiple subtasks and consider the dependencies between them to ensure the original task execution order. Further, we propose a subtask scheduling priority algorithm to further reduce the task execution latency. Additionally, we consider a VAEN scenario in which idle vehicles are seen as edge nodes, which can make fully utilize the resources of the VECNs and improve the QoS.

B. Computation Offloading Optimization Technology

The computation offloading problem mentioned above can be formulated optimally as a mixed-integer nonlinear program, and thus it is critical to use an objective optimization method applicable to VECNs to make real-time offloading decisions.

In recent years, researchers conducted extensive research on such objective optimization problems, especially based on the game theory [34]-[36], convex/non-convex optimization [37], [38], and other optimization techniques for formulating offloading decisions. However, the highly dynamic nature of VECNs poses a challenge for these conventional optimization methods. Furthermore, these conventional optimization algorithms have difficulty in obtaining optimization results when the optimization variables or constraints become more complex [7].

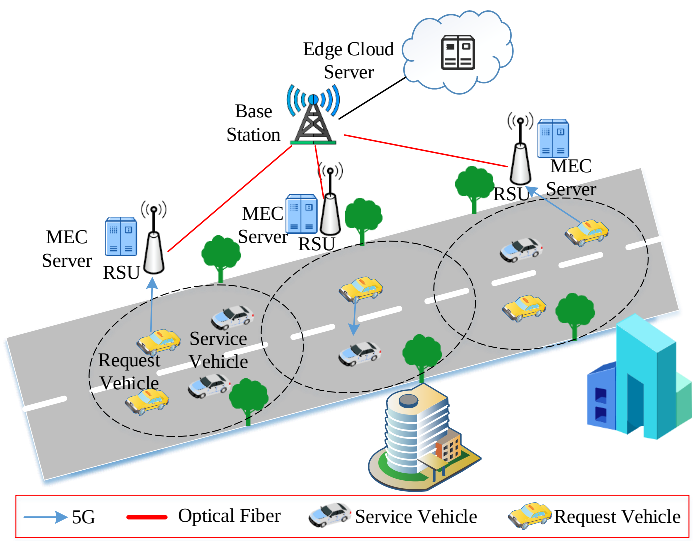


Fig. 1. Vehicle-assisted computation offloading network model.

Fortunately, the application of DRL in regards to dynamic control problems has inspired researchers, and recent attention has shifted to finding DRL-based solutions to computation offloading problems [39]-[41]. DRL is a combination of reinforcement learning (RL) and deep learning (DL). Specifically, the problem definition and objective optimization are achieved with RL, the strategy and value function are solved by DL, and the objective function is optimized by error backpropagation algorithms [42]-[44].

As for the application of DRL in computation offloading, Zhang et.al [45] considered the high dynamic of vehicular networks, constructing a synchronized random walk model and using a deep-Q-network (DQN) to reduce system delay. Wu et.al [46] adopted the DQN algorithm, combining Q-Learning and deep learning to achieve joint optimization of delay and energy consumption. Moreover, Song et al. [47] investigated a task offloading problem, proposing a semi-online computation offloading model using Dueling Deep-Q networks, which jointly considered user behavior predictions and server load balancing. With the development of DRL, an actor-critic algorithm was proposed by combining policy gradient and temporal difference, consisting of an actor network and a critic network [48]. Liu et.al [49] proposed the actor-critic algorithm to jointly optimize the computation offloading policies and resource allocation. Zhan et.al [14] studied the task offloading problem and formulated it as a decentralized computation offloading game in each time slot, by taking into account both communication and computation costs. Additionally, actor-critic reinforcement learning was used to obtain the optimal offloading strategy. Geng et.al [50] used the DDPG approach to optimize the task scheduling policy for VECNs scenarios considering channel time variations. Lu et al. [51] considered the QoE when constructing the computation offloading model, and proposed an improved DDPG algorithm named Double Duel Deterministic Policy Gradient (D(3)PG).

Although the adoption of DRL enables the agents to make real-time offloading decisions in highly dynamic large-scale

TABLE I
NOTATION

Notation	Description
$\mathcal{RV}/\mathcal{SV}$	Set of Request vehicles/Service vehicles
$\mathcal{R}$	Set of all RSUs
$\mathcal{S}$	Road Segment
$\mathcal{M}$	Set of all MEC servers
N	The number of RVs
G	The number of SVs
$\mathcal{F}_{\mathcal{RV}}/\mathcal{F}_{\mathcal{SV}}$	The computing capacity of RVs/SVs
$\mathcal{F}_{\mathcal{MEC}}$	The computing capacity of MEC servers
$\mathcal{T}_i$	The task generated by RV i
G_i	Describe the task of RV i with a DAG
E_i	All edges of a DAG
T_{ij}	The $j-th$ subtask of RV i
d_{ij}^{in}	The size of the subtask input data
c_{ij}	Required CPU cycles to accomplish the subtask
$t_i^{\max}$	Maximum permissible processing delay of the task
$t_{ij}^{\max}$	Maximum tolerated delay of the $j-th$ subtask
τ	Time slot
$\mathbf{lp}, \mathbf{mp}, \mathbf{vp}$	Task offloading symbolic factors
$\lvert T_i \rvert$	Number of subtasks for RV i
$a_{i,j}$	Task offloading decision
p_i	The upload power of RV i
$r_{i,m}^{V2I}/r_{i,g}^{V2V}$	Data rate between RV i and MEC server m/SV g
B^{V2I}/B^{V2V}	The channel bandwidth of MEC server/SV
$h_{i,m}^{V2I}$	Channel gain between RV i and MEC server m
$h_{i,g}^{V2V}$	Channel gain between RV i and SV g
(x_i, y_i)	The coordinates of RV i
(x_g, y_g)	The coordinates of SV g
v_i/v_g	The velocity of RV i/SV g
R_C	The communication range of vehicles
$P_{cand}^{i,g}$	CSV binary symbolic factor
ρ	The threshold value to determine CSVs

VECNs, the use of fully-connected networks and the uniform sampling method in the experience replay are not conducive to feature extraction and model training.

Consequently, we propose DCOM with IACN and JMPA to improve the learning efficiency of the network.

III. System Model

In this section, we elaborate on the system model of VAEN, which includes network, communication, computation, and CSV models.

A. Network Model

As shown in Fig. 1, we consider an urban environment scenario in which a one-way road has two parallel lanes with multiple vehicles moving in the lane. In the scenario, two types of vehicles are included, i.e., the request vehicles (RVs) and service vehicles (SVs). We denote RVs as $\mathcal{RV} = \{1, 2, ..., N\}$ with generation task, and SVs as $\mathcal{SV} = \{1, 2, ..., G\}$ with spare resources to provide computation offloading services for RVs. It is crucial to note that SVs can be both private and public. In other words, we only focus on the idle resources of SVs, without considering other attributes. In addition, we divide the time into many equal time slots, defined as $\tau = \{1, 2, \ldots, T\}$, and assume that each RV generates only one task in a time slot. Moreover, RSUs are deployed on one side of the road, denoted as $\mathcal{R} = \{R_1, R_2, \ldots, R_M\}$, and the neighboring RSUs are connected via fibers. According to the coverage of RSUs, the road is divided into different segments $\mathcal{S} = \{S_1, S_2, ..., S_M\}$. MEC servers $\mathcal{M} = \{1, 2, ..., M\}$ are connected to RSUs and possess a large number of computing resources to provide highly reliable computation offloading services for RVs. For simplification, an MEC server and the connected RSU are represented by a fixed edge node. Furthermore, a base station (BS) is responsible for managing all RSUs and connects to RSUs via fiber optics, and the edge cloud server serves as a backup for RVs. In this paper, we assume that tasks can be completed depending solely on the edge layer nodes. For a practical scenario, we define both vehicles and MEC servers are heterogeneous, i.e., the computing capacity set of RVs, SVs, and MEC servers are $\mathcal{F}_{\mathcal{RV}} = \{f_1^{RV}, f_2^{RV}, ..., f_N^{RV}\}$, $\mathcal{F}_{\mathcal{SV}} = \{f_1^{SV}, f_2^{SV}, ..., f_G^{SV}\}$, and $\mathcal{F}_{\mathcal{MEC}} = \{f_1^{MEC}, f_2^{MEC}, ..., f_M^{MEC}\}$, respectively. The main notations used in this Section are listed in Table I.

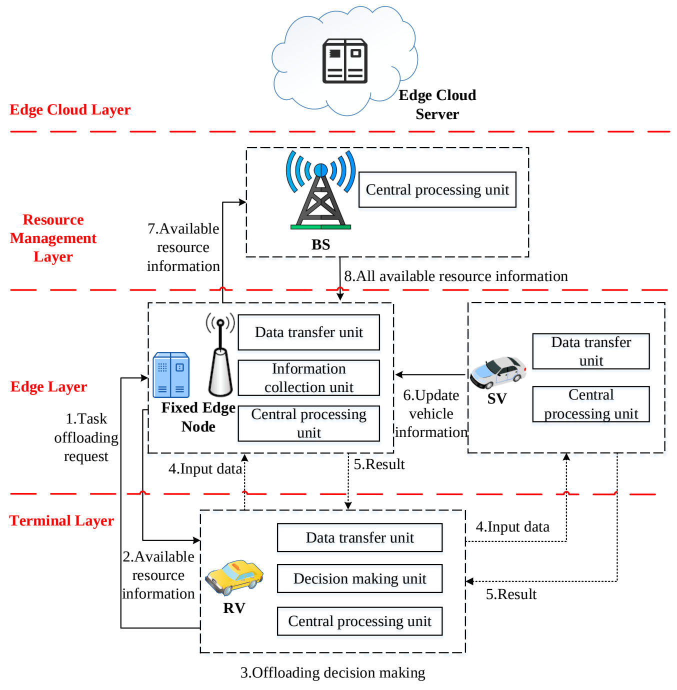


Fig. 2. Information interaction in vehicle-assisted vehicular edge computing network scenario.

As shown in Fig. 2, the steps for computation offloading are as follows.

1) Task offloading requests sending: When RV i generates a task, it sends a task execution request to its corresponding RSU to obtain the information of the connected MEC server and all SVs.

2) Offloading decision making and task execution: Each RV makes its offloading strategy based on the known information and then processes tasks based on the offloading decisions. If

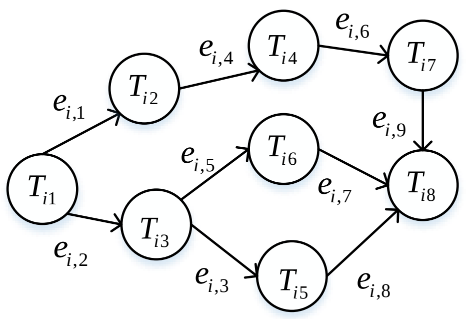


Fig. 3. DAG structure diagram of task $\mathcal{T}_i$.

RVs decide to offload tasks, the uploading of the original data and the downloading of the execution results are required.

3) SVs update information: After the task execution step is completed, all SVs broadcast their information, such as locations, speeds, remaining resources, etc., to the corresponding RSUs at regular intervals.

4) Vehicular network information summary: All RSUs transmit the information of SVs to the BS for aggregation and then the BS broadcasts to all RSUs, thus ensuring that all RSUs have all the relevant information about SVs.

B. Task Model

Considering that most applications are fine-grained tasks. A task generated by RV i at time slot t can be represented by a set of subtasks denoted as $\mathcal{T}_i = \left\{T_{i1}, T_{i2}, ..., T_{i|T_i|}\right\}$, $\forall i \in \mathcal{RV}$, where $|T_i|$ denotes the number of subtasks. As shown in Fig. 3, we apply DAG $G_i = (\mathcal{T}_i, E_i)$ to depict the dependencies between subtasks, where E_i is the set of edges. E_i is defined by $E_i = \left\{e_{i,1}, e_{i,2}, ..., e_{i,|E_i|} \mid e_{i,e} = (j, j^{'})\right\}$ and $e_{i,e} = (j, j^{'})$ indicates that the execution of subtask $T_{ij'}$ depends on the result of the execution of subtask T_{ij}. Therefore, we consider subtask T_{ij} as the predecessor subtask of subtask $T_{ij'}$. Correspondingly, subtask $T_{ij'}$ is the successor subtask of subtask T_{ij}. We assume these subtasks are atomic and can not be subdivided, and we define the $j-th$ subtask of RV i as

$$T_{ij} \triangleq \{d_{ij}^{in}, c_{ij}, t_{ij}^{\max}, t_i^{\max}\}, \tag{1}$$

where d_{ij}^{in} is the input data size of the subtask, c_{ij} is the number of CPU cycles to complete the subtask, $t_{ij}^{\max}$is the maximum allowable delay of subtask $T_{i,j}$ and $t_i^{\max}$ is the maximum allowable delay of task $\mathcal{T}_i$.

According to the aforementioned, in the VAEN scenario, the edge layer is composed of fixed edge servers and SVs. Specifically, after generating task $\mathcal{T}_i$, three ways can be utilized to process it by RV i, i.e., executing locally, offloading to a fixed service node, and offloading to a surrounding SV. Considering that all vehicles are equipped with one radio, a subtask can select at most one server for task processing, and an SV can serve at most one RV in a time slot. Therefore, for RV i, there are $G+2$ offloading decisions. We define task offloading symbolic factors for RV i are $lp_{ij} \in \{0,1\}$, $mp_{ij} \in \{0,1\}$ and $vp_{ij} \in \{0,1\}$. Specifically, $lp_{ij} = 1$ when subtask T_{ij} is executed locally, otherwise $lp_{ij} = 0$; $mp_{ij} = 1$ when subtask T_{ij} is offloaded to a fixed edge node, otherwise $mp_{ij} = 0$; $vp_{ij} = 1$ when subtask T_{ij} is offloaded to SV g, otherwise $vp_{ij} = 0$. The offloading decision of the T_{ij} is defined as $a_{i,j} \in \{-1, 0, 1, 2, \ldots, G\}$. Further, the task offloading symbolic factors and the offloading decision have the corresponding relationships as follows: If $lp_{ij} = 1$ then $a_{i,j} = 0$, if $mp_{ij} = 1$ then $a_{i,j} = -1$, and if $vp_{ij} = 1$ then $a_{i,j} = g (1 \le g \le G)$, which indicates RV i decides to offload subtask T_{ij} to SV g.

C. Communication Model

In the following, we discuss the vehicle-to-infrastructure (V2I) communication model and V2V communication model.

1) V2I communication model: RVs need to communicate with RSUs when offloading tasks to fixed edge nodes. In VAEN scenario, we set the upload link from the RVs to the fixed edge node as a flat Rayleigh fading channel, regardless of channel interference. Assuming that the bandwidth of RSU is B^{V2I} and there are n RVs offloading tasks to the MEC server m. The uplink data rate between RV i and fixed edge node m can be calculated as

$$r_{i,m}^{V2I} = \frac{B^{V2I}}{n} \log_2(1 + \frac{p_i h_{i,m}^{V2I}}{\sigma^2}), \tag{2}$$

where p_i is the upload power of RV i, $h_{i,m}^{V2I}$ is the channel gain between the $i-th$ RV and the fixed edge nodes m, and σ^2 denotes the background noise power.

2) V2V communication model: We consider RVs can communicate directly with SVs via D2D-based V2V communications. The achievable transmission rates between RV i and SV g can be written as

$$r_{i,g}^{V2V}(t) = B^{V2V} \log_2(1 + \frac{p_i h_{i,g}^{V2V}}{\sigma^2}), \tag{3}$$

where B^{V2V} denotes the communication bandwidth of SV g and $h_{i,g}^{V2V}$ is the channel gain.

D. Computation Model

Based on the VAEN scenario, we analyze the computation model of the system.

1) Processing locally: If subtask T_{ij} is selected to be executed locally, the execution delay is

$$T_{i,j,l} = \frac{c_{ij}}{f_i^{RV}}, \tag{4}$$

and the energy consumption of local execution is

$$E_{i,j,l} = \kappa ({f_i}^{RV})^2 c_{ij}, \tag{5}$$

here κ is the effective switching capacitance in the chip [43].

2) Processing at a MEC server: If subtask T_{ij} is offloaded to a fixed edge node for execution, it follows a first-come-first-serve queuing principle. Due to the high-speed mobility of vehicles, if RV i drives away from the road segment S_m during task execution, the result of T_{ij} is transmitted via inter-MEC server communication. Because the data volume of the returned calculation result is relatively small, the download delay can be ignored, and transmission delay of subtask T_{ij} can be calculated as

$$T^{V2I}_{i,up} = \frac{d^{in}_{ij}}{r^{V2I}_{i,m}}. \tag{6}$$

The queuing time can be expressed as

$$T^{V2I}_{i,queue} = \frac{\sum_{n\in k, n\neq i} c_{nj}}{f^{MEC}_{m}}, \tag{7}$$

where k is the set of RVs that offload tasks to RSU m before RV i.

The execution time of subtask T_{ij} can be given as

$$T^{V2I}_{i,exe} = \frac{c_{ij}}{f^{MEC}_{m}}. \tag{8}$$

If RV i drives away from the road segment S_m, the communication time between RSUs is

$$T^{V2I}_{i,backhaul} = \Delta x * t_{backhaul}, \tag{9}$$

where $\triangle x$ is the number of road segments traveled by RV i, $\triangle x = 0, 1, 2, \ldots$, and $t_{backhaul}$ is the time of adjacent RSUs communication, which we define as a constant.

The total time delay offloading to a MEC server for execution is

$$T^{V2I}_{i,j,m} = T^{V2I}_{i,up} + T^{V2I}_{i,exe} + T^{V2I}_{i,backhaul} + T^{V2I}_{i,queue}, \tag{10}$$

and the energy consumption of RV i is

$$E^{V2I}_{i,j,m} = p_i T^{V2I}_{i,up}. \tag{11}$$

3) Processing at a SV: If T_{ij} is offloaded to the SV g, the delay consists of two parts, i.e., transmission delay and execution delay. The transmission delay between the RV i and SV g can be given as

$$T^{V2V}_{i,g,up} = \frac{d^{in}_{i,j}}{r^{V2V}_{i,g}}, \tag{12}$$

and the execution delay of subtask T_{ij} can be expressed as

$$T^{V2V}_{i,g,exe} = \frac{c_{ij}}{f^{SV}_{g}}. \tag{13}$$

Therefore, the total delay of offloading to SV g can be calculated as

$$T^{V2V}_{i,j,g} = T^{V2V}_{i,g,up} + T^{V2V}_{i,g,exe}. \tag{14}$$

We consider that task transfer consumes energy, so the energy consumption of RV i is

$$E^{V2V}_{i,j,g} = p_i T^{V2V}_{i,g,up}. \tag{15}$$

By analyzing the computation model, we can conclude that the task delay and energy consumption of T_{ij} can be given as

$$T_{i,j} = lp_{ij} \cdot T_{i,j,l} + mp_{ij} \cdot T^{V2I}_{i,j,m} + vp_{ij} \cdot T^{V2V}_{i,j,g}, \tag{16}$$

$$E_{i,j} = lp_{ij} \cdot E_{i,j,l} + mp_{ij} \cdot E^{V2I}_{i,j,m} + vp_{ij} \cdot E^{V2V}_{i,j,g}. \tag{17}$$

E. CSV Model

Due to the high-speed mobility of vehicles, we consider the communication range of vehicles when RVs perform V2V communication. Assume that the communication range of vehicles is R_C. When the task $\mathcal{T}_i$ is generated, the coordinates of RV i and SV g are (x_i, y_i), (x_g, y_g), and the velocities of them are v_i, v_g respectively. Accordingly, the distance between RV i and SV g is

$$D_{i,g} = \sqrt{(x_i - x_g)^2 + (y_i - y_g)^2}. \tag{18}$$

If $D_{i,g} > R_C$, SV g cannot be a CSV for RV i. If $D_{i,g} < R_C$, the maximum communication duration between RV i and SV g can be calculated as

$$CT_{i,g} = \frac{R_C \cos\alpha - (D_{i,g}\cos\alpha)\beta}{\Delta v_{i,g}}, \tag{19}$$

where $\alpha = \arcsin(\frac{|y_i - y_g|}{R_C})$, $\beta = \frac{(x_i - x_g)(v_i - v_g)}{|x_i - x_g||v_i - v_g|}$ and $\Delta v_{i,g} = |v_i - v_g|$.

We define $P^{i,g}_{cand\_t}$ as the effect of the mobility on SV g becoming a CSV for RV i, and represent it as

$$P^{i,g}_{cand\_t} = \begin{cases} 1, & \text{if} \quad CT_{i,g} > t^{\max}_{i,j} + \varsigma \\ \frac{CT_{i,g} - T^{V2V}_{i,j,g}}{t^{\max}_{i,j} + \varsigma - T^{V2V}_{i,j,g}}, & \text{if} \quad T^{V2V}_{i,j,g} < CT_{i,g} < t^{\max}_{i,j} + \varsigma \\ 0, & \text{if} \quad CT_{i,g} < T^{V2V}_{i,j,g} \end{cases}, \tag{20}$$

where ς is the guaranteed factor for successfully executing a task.

In addition, the volume of computing resources in SVs is a critical factor when RV i determines the CSVs. We define $P^{i,g}_{cand\_r}$ as the effect of the computing resources on SV g being a CSV for RV i.

$$P^{i,g}_{cand\_r} = \begin{cases} 1, & \text{if} \quad E_g > E^g_{i,j} \\ 0, & \text{if} \quad E_g < E^g_{i,j} \end{cases}, \tag{21}$$

where E_g is the computing resource of SV g, $E^g_{i,j}$ is the computing resource required for $T_{i,j}$ and it is defined as $E^g_{i,j} = \kappa({f_g}^{SV})^2 c_{ij}$.

We define ρ as a threshold value to determine whether an SV can be a CSV and use a matrix $\mathbf{P}_{cand} = [P^{i,g}_{cand}]_{N\times G}$ to depict the candidate relationship between RVs and SVs. The SV g is able to become a CSV for RV i when $P^{i,g}_{cand} = 1$, and $P^{i,g}_{cand}$ can be expressed as follows

$$P^{i,g}_{cand} = \begin{cases} 0, & \text{if} \quad P^{i,g}_{cand\_t} \cdot P^{i,g}_{cand\_r} < \rho \\ 1, & \text{if} \quad P^{i,g}_{cand\_t} \cdot P^{i,g}_{cand\_r} > \rho \end{cases}. \tag{22}$$

In the next section, we discuss formulating an optimization problem.

IV. Problem Formulation

In this section, we formulate the optimization problem for computation offloading in the VAEN scenario. Specifically, we aim to seek out the optimal offloading to minimize the task completion delay and energy consumption, and define the system cost of RV i as

$$U_i = \sum_{j=1}^{|T_i|} (\lambda_t T_{i,j} + \lambda_e E_{i,j}), \tag{23}$$

where $\lambda_t, \lambda_e \in [0,1]$ and $\lambda_t + \lambda_e = 1$.

According to the task model introduced above, we define the task offloading symbolic factors as $\mathbf{lp} = [lp_{ij}]_{N\times G}$, $\mathbf{mp} = [mp_{ij}]_{N\times G}$, $\mathbf{vp} = [vp_{ij}]_{N\times G}$ $(0 < i \le N,\ 0 < j \le |T_i|)$, respectively. Furthermore, we define the offloading decision for all RVs as $\mathbf{A}^t = [a_{i,j}]_{N\times G}$ $(0 < i \le N,\ 0 < j \le |T_i|)$. The optimization problem can be formulated as the long-term total system cost minimization problem, described as follows:

$$\begin{aligned} P1:\ & \min_{\{\mathbf{lp},\mathbf{mp},\mathbf{vp},\mathbf{A}^t\}} \sum_{i\in N} U_i \\ \text{s. t.}\ & C1: \sum_{j=1}^{|T_i|} T_{i,j} \le {t_i}^{\max} \\ & C2: lp_{ij} + mp_{ij} + vp_{ij} = 1 \\ & C3: a_{i,j} \in \{-1,0,1,2,...,G\} \\ & C4: \sum_{j=1}^{|T_i|} E_{i,j} \le {E_i}^{\max} \\ & C5: 0 \le p_i \le P_{\max} \end{aligned} . \tag{24}$$

The main goal of the above optimization problem is to minimize total system cost by making reasonable offloading decision $\mathbf{A^t}$. Here, constraint C1 ensures that the execution delay does not exceed the maximum allowable delay. C2 and C3 are constraints on the decision variables. The constraint among lp_{ij}, mp_{ij} and vp_{ij} indicates that a subtask can only be executed on one server, and the a_{ij} describes further the specific offloading destination of a subtask. Constraint C4 states that the energy consumption of RV i does not exceed its own maximum number of resources, i.e., $E_i^{\max}$. C5 constrains that the transmission power of the RV i does not exceed the maximum transmission power, i.e., $P_{\max}$.

To solve problem P1, it is significant to obtain an optimal task offloading decision with minimum task delay and task energy consumption. However, it is clear that problem P1 is a nonlinear programming problem, which is generally NP-hard [32], and we prove it afterwards. Moreover, in a highly dynamic VAEN scenario, we need a time-vary task offloading decision that can be adaptively adjusted. Therefore, solving problem P1 with conventional methods is difficult to achieve. To this end, we design a task offloading algorithm based on RL to solve this problem.

Theorem 1. Problem P1 is NP-hard.

Proof: Refer to Appendix A.

V. Reinforcement Learning-Based Solution

In this section, we consider a distributed structure to approximate the optimization problem as an MDP to minimize the system cost including delay and energy consumption. Specifically, we introduce a subtask scheduling priority algorithm, and define state, action, and reward function. Finally, we propose a DCOM which integrates IACN and JMPA to obtain the optimal policy. The main notations used in this Section are listed in Table II.

A. Priority of Task Scheduling

The fine-grained task offloading decision process is divided into two steps, i.e., offloading order and offloading decision. To reduce the total processing latency of an application, it is necessary to obtain the optimal subtask offloading order. Two important concepts are proposed for subtask T_{ij}, i.e., priority and execution cost. The subtask priority is obtained with execution cost and internal dependencies, and the execution cost can be formulated as

$$\overline{\mathrm{cost}_{i,j}} = \frac{(T_{i,j,l} + T_{i,j,m}^{V2I} + \sum_{g=1}^{G_i'} T_{i,j,g}^{V2V})}{G_i' + 2}, \tag{25}$$

where G_i' is the number of CSVs about RV i.

In a task described by a DAG, we define the last subtask as the exit subtask, and each subtask has at least one path to the exit subtask, which we refer to as the exit critical path $\mathbf{ep}$. In this paper, we use a reverse recursive approach to compute the priority of subtasks. Specifically, the priority of the exit

TABLE II
NOTATION

Notation	Description
λ_t/λ_e	The weight of delay/energy consumption
$succ(T_{ij})$	Successor subtasks set of subtask T_{ij}
$pre(T_{ij})$	Precursor subtasks set of subtask T_{ij}
$\overline{\cos t_{i,j}}$	Execution cost of the $j-th$ subtask
G_i'	The number of CSVs abou RV i
$w_{i,j,j'}$	The communication overhead between subtasks
$\mathcal{T}_i'$	Subtask priority sequence of $\mathcal{T}_i$
$\mathbf{s}_{i,j}(t)$	The state space of RV i
$r_{i,j}$	Reward function
$\theta^\pi/\theta^{\pi'}$	The parameters of main/target actor network
$\omega^Q/\omega^{Q'}$	The parameters of main/target critic network
π/π'	The main/target actor network
Q, Q'	The main/target critic network
γ	Discount factor
n_{batch}	The mini-batch in DRL
$n_{\max}$	The predetermined value for the length of transitions
W	The size of replay buffer
ξ	Target networks updating rate

Algorithm 1 Subtask Scheduling Priority

```
Input: The set of RVs $\mathcal{RV}$, the set of SVs $\mathcal{SV}$, the set of MEC servers $\mathcal{M}$, computing capacity of RVs $\mathcal{F_{RV}}$, computing capacity of SVs $\mathcal{F_{SV}}$, computing capacity of MEC servers $\mathcal{F_{MEC}}$
1: for RV $i = 1,2,...,N$ do
2:     Generate task $G_i = (\mathcal{T}_i, E_i)$.
3:     for $j = 1,2,...,|T_i|$ do
4:         Get information about a subtask $T_{ij} = \{d_{ij}^{in}, c_{ij}, t_{ij}^{\max}, t_i^{\max}\}$ .
5:         Calculate the processing cost $\overline{\mathrm{cost}_{i,j}}$ by (25).
6:         Calculate $P(T_{ij})$ by (26).
7:     end for
8:     Sorting subtasks by priority $\mathcal{T}_i'$= $\left\{T_{i1}', T_{i2}', ..., T_{i|T_i|}'\right\}$, $\forall i \in \mathcal{RV}$.
9: end for
Output: Subtask priority sequence $\mathcal{T}'$= $\left\{\mathcal{T}_1', \mathcal{T}_2', ..., \mathcal{T}_N'\right\}$
```

subtask is calculated first, and then the priority of each non-export subtask is calculated in the reverse direction according to the structure of the DAG. It should be noted that when calculating the priority of non-export subtasks, the exit critical path with the highest priority is chosen. The priority of subtask T_{ij} can be defined as

$$P(T_{ij}) = \begin{cases} \overline{\cos t_{i,j}}, & \text{if } T_{ij} \text{ is exit subtask} \\ \max\limits_{\mathbf{ep}} P(T_{ij'}) + \overline{\cos t_{i,j}} + w_{i,j,j'}, & \text{otherwise} \end{cases}. \tag{26}$$

Here, $w_{i,j,j'}$ is the communication overhead between subtask T_{ij} and its successor subtask $T_{ij'}$.

Specifically, the pseudo-code of the subtask scheduling priority algorithm is provided in Algorithm 1.

B. State, Action and Reward Definition

RL can be described as an intelligence interacting with its environment to continuously learn for a specific goal, such as obtaining the maximum reward value. In this paper, we apply a distributed structure to obtain the optimal offloading policy, with the agents referring to all RVs and the environment referring to VAEN. In the following, the state space, action space, and reward function of the system are defined based on our proposed model.

1) State Space: We define the system state of RV i at time slot t as $\mathbf{s}_{i,j}(t) = \left\{\mathbf{S}_{i,j}^{sti}(t), \mathbf{S}_{i,j}^{vmi}(t), \mathbf{S}_{i,j}^{vod}(t)\right\}$, where $0 < i \leq N$ and $0 < j \leq |T_i|$. The state parameters are depicted in detail as follows.

a) $\mathbf{S}_{i,j}^{sti}(t)$ is the information of the current subtask, including the data size vector of all subtasks $\mathbf{D}_i^{in} = \left[d_{i1}^{in}, d_{i2}^{in}, ..., d_{i|T_i|}^{in}\right]$, the number of CPU cycles to complete all subtask $\mathbf{C}_i = \left[c_{i1}, c_{i2}, ..., c_{i|T_i|}\right]$, the set of predecessor subtasks $pred(T_{ij})$ of the current subtask and the set of successor subtasks $succ(T_{ij})$, which can be formulated as follows.

$$\mathbf{S}_{i,j}^{sti}(t) = \left\{\mathbf{D}_i^{in}, \mathbf{C}_i, pred(T_{ij}), succ(T_{ij})\right\}. \tag{27}$$

b) $\mathbf{S}_{i,j}^{vmi}(t)$ is the information related to the system and can be expressed as

$$\mathbf{S}_{i,j}^{vmi}(t) = \{RV_{pos}, RV_{sp}, P_{candi}(i), MEC_{pos}, SV_{pos}, F_{SV}\}. \tag{28}$$

Here, RV_{pos} and RV_{sp} denote the coordinates and velocity of RV i, respectively. $P_{cand}(i)$ denotes the CSV model and MEC_{pos} denotes the coordinates of MEC servers. Moreover, SV_{pos} and $\mathcal{F}_{SV}$ are the coordinates and CPU frequencies of SVs, respectively.

c) $\mathbf{S}_{i,j}^{vod}(t)$ denotes the offloading decisions of the previous L subtasks of the current subtask T_{ij} for all RVs. Note that if $j \leq L$, the tasks can be traced back to the previous time slot. For simplicity, here we do not delve into the range of j. Specifically, when $j \leq L$, we consider the offloading decision of the previous time slot. Therefore, $\mathbf{S}_i^{vod}(t)$ can be described as

$$\mathbf{S}_{i,j}^{vod}(t) = \begin{bmatrix} a_{1,(j-1)} & a_{1,(j-2)} & \cdots & a_{1,(j-L)} \\ a_{2,(j-1)} & a_{2,(j-2)} & \cdots & a_{2,(j-L)} \\ \cdots & \cdots & \cdots & \cdots \\ a_{N,(j-1)} & a_{N,(j-2)} & \cdots & a_{N,(j-L)} \end{bmatrix}. \tag{29}$$

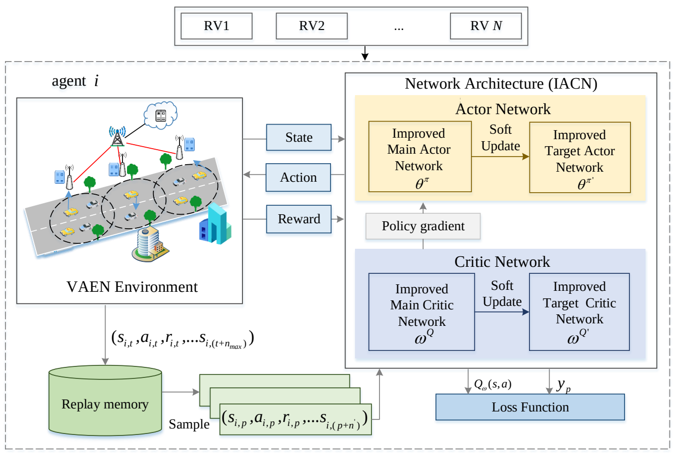

RV1
RV2
...
RV N
agent i
VAEN Environment
State
Action
Reward
Network Architecture (IACN)
Actor Network
Improved Main Actor Network θ^{π}
Soft Update
Improved Target Actor Network $\theta^{\pi'}$
Policy gradient
Critic Network
Improved Main Critic Network ω^{Q}
Soft Update
Improved Target Critic Network $\omega^{Q'}$
$(s_{i,t}, a_{i,t}, r_{i,t}, ...s_{i,(t+n_{max})})$
Replay memory
Sample
$(s_{i,p}, a_{i,p}, r_{i,p}, ...s_{i,(p+n')})$
$Q_{\omega}(s,a)$
y_p
Loss Function

Fig. 4. The structure of the distributed computation offloading algorithm based on multiagent DRL.

2) Action Space: In VAEN, an RV i needs to make an offloading decision based on the system state $\mathbf{s}_{i,j}(t)$ and decision policy, specifically, determining the target server for each subtask. Technically, the action space of the RV i at time slot t can be represented as

$$\mathbf{A}_i^t = [a_{i,1}, a_{i,2}, ..., a_{i,j}, ..., a_{i,|T_i|}], \tag{30}$$

where $a_{i,j}$ is the $j-th$ subtask execution destination. In particular, if $a_{i,j} = -1$, the subtask T_{ij} will be offloaded to the MEC server, if $a_{i,j} = 0$, then the subtask T_{ij} will be executed locally, and if $a_{i,j} > 0$, the subtask T_{ij} will be executed on the corresponding SV. According to the aforementioned, the legal action space of $a_{i,j}$ is $\{-1, 0, 1, 2, \ldots, G\}$ and the dimension is $G + 2$.

3) Reward Function: In the time slot t, the environment gives rewards according to the actions taken by the agent. In this work, we have $\mathbf{R}(t) = [r_{i,j}]_{N\times G}$ $(0 < i \leq N$, $0 < j \leq |T_i|$, where $r_{i,j}$ is the improvement of the performance brought by RV i, and can be expressed as

$$r_{i,j} = \lambda_t \frac{T_{i,j,l} - T_{i,j}}{T_{i,j,l}} + \lambda_e \frac{E_{i,j,l} - E_{i,j}}{E_{i,j,l}}. \tag{31}$$

C. Distributed Computation Offloading Algorithm based on multiagent DRL

Afterwards, the overview of the algorithm, IACN, JMPA and DCOM descriptions are introduced.

1) Overview: To make real-time offloading decisions, we apply an actor-critic algorithm which combines a policy-based approach with a value-based approach. The actor-critic technique can compensate for the drawback of policy round update with the idea of DQN, and learn the policy function and value function together to get an optimal behavior through their interactions. Specifically, the actor selects an appropriate action according to the state of the environment, which is a policy-based approach. The critic is a value-based network that evaluates the quality of the action, and in turn the evaluation results act on the actor network for training, which can accelerate the learning process and achieve satisfactory results.

Therefore, to obtain an optimized computation offloading policy, we design DCOM.

Fig. 4 illustrates the overall framework of DCOM with N agents. Each agent possesses actor networks and critic networks. The actor networks include a main actor network π, with parameters θ^{π} and a target actor network $\pi^{'}$ with parameters $\theta^{\pi^{'}}$. The main actor network and target actor network share the same structure, which reduces the oscillation of DNN and achieves stable convergence. In the process of generating action by the actor network, the conventional greedy algorithm only selects the optimal action and fails to achieve the purpose of full exploration. To balance the relationship between exploration and exploit, we adopt an ε greedy strategy, which can fully explore the actions in action space. After the main actor network outputs an action, the agent interacts with the environment to obtain the next state and immediate reward, and stores a transition $(s_{i,j}, a_{i,j}, r_{i,j}, s_{i,(j+1)})$ into the experience replay buffer for use in the subsequent training phase. With experience replay buffer, the dependency between sample data is broken.

As mentioned before, a critic network consists of a main critic network Q with parameters ω^{Q} and a target critic network $Q^{'}$ with parameters $\omega^{Q^{'}}$. In the training phase, a mini-batch transition is sampled from the experience replay buffer to train the critic network. Suppose the $p-th$ transition sampled from the experience replay buffer as $(s_{i,p}, a_{i,p}, r_{i,p}, s_{i,(p+1)})$, $(s_{i,p}, a_{i,p})$ is taken as the input of the main critic network, and the output of the value estimation network can be calculated as $Q(s_{i,p}, a_{i,p}\left|\omega^{Q}\right.)$. Further, taking $(s_{i,(p+1)}, \pi(s_{i,(p+1)}))$ as the input of the target critic network, the target Q value y_p can be expressed as

$$y_p = r_{i,p} + \gamma Q^{'}(s_{i,(p+1)}, \pi^{'}(s_{i,(p+1)}\left|\theta^{\pi^{'}}\right.)\left|\omega^{Q^{'}}\right.). \tag{32}$$

Therefore, the TD-error can be calculated by

$$TD_p = y_p - Q(s_{i,p}, a_{i,p}\left|\omega^{Q}\right.), \tag{33}$$

and loss function of the critic network is given by

$$loss = \frac{1}{n_{\text{batch}}}\sum_{p=1}^{n_{\text{batch}}} TD_p^2. \tag{34}$$

Here, n_{batch} is the size of the mini-batch. The parameters of the main critic network can be updated by minimizing (34).

The main actor network π updates its parameters θ^{π} in the direction of getting a larger cumulative discounted reward, that is

$$\nabla_{\theta} L(\pi) = E_{\pi}[\nabla_{\theta}\pi(s_{i,p})\nabla_{\theta}Q(s_{i,p}, a_{i,p}\left|\omega^{Q}\right.)\left|_{a_{i,p}=\pi(s_{i,p})}\right.]. \tag{35}$$

Then, the parameters of the target network are updated by soft update algorithm as follows

$$\begin{array}{l} \omega^{Q^{'}} \leftarrow \tau^{'}\omega^{Q} + (1-\tau^{'})\omega^{Q^{'}} \\ \theta^{\pi^{'}} \leftarrow \tau^{'}\theta^{\pi} + (1-\tau^{'})\theta^{\pi^{'}} \end{array}, \tag{36}$$

where $\tau^{'}$ is 0.001 in general.

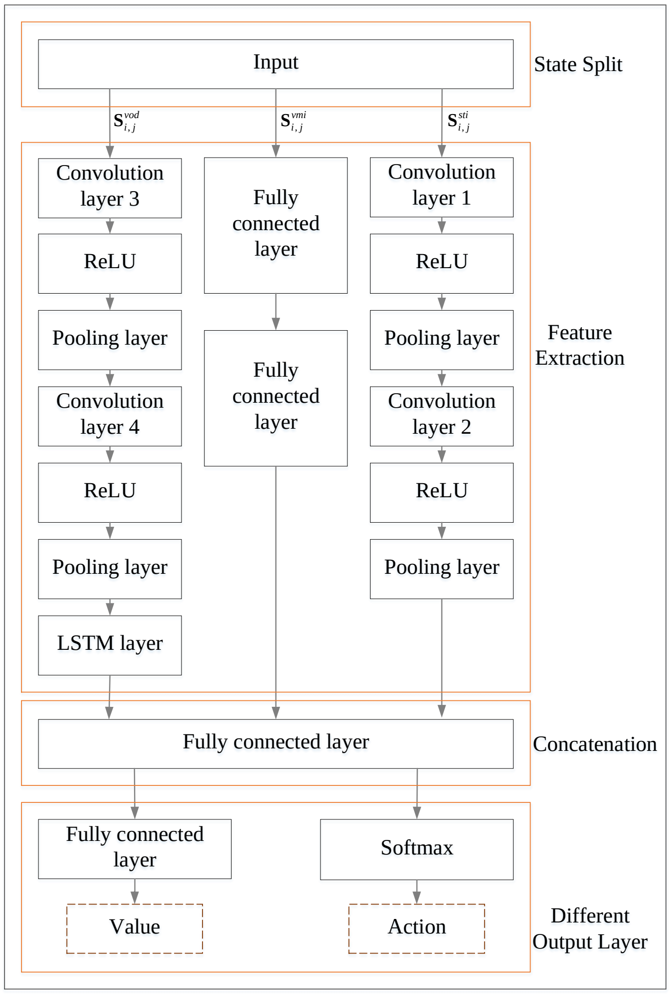


Fig. 5. Structure of improved actor-critic network.

TABLE III
SETTING OF THE NETWORK STRUCTURE CORRESPONDING TO $\mathbf{S}_{i,j}^{sti}$

Layer	Kernels	Pooling	Stride	Output
Input	/	/	/	10×20
Convolution layer1	$5 \times 5 \times 32$	/	1	$10 \times 20 \times 32$
Pooling layer	/	2×2	2	$5 \times 10 \times 32$
Convolution layer2	$5 \times 5 \times 32$	/	1	$5 \times 10 \times 32$
Pooling layer	/	2×2	1	$2 \times 5 \times 32$

2) IACN: The size of the state space is very large due to $S_{i,j}^{sti}(t)$ and $S_{i,j}^{vod}(t)$, and the parameters of $S_{i,j}^{vod}(t)$ are temporal variations. Specifically, we assume that the number of RVs is 20, and each task can be partitioned into up to 10 interdependent subtasks. The parameter $S_{i,j}^{sti}(t)$ represents the structure information of a DAG with dimension 10×20, and $S_{i,j}^{vod}(t)$ possesses temporal correlation. However, in a conventional actor-critic network, both its actor network and critic network are composed of FCNs. Although the FCNs can achieve feature extraction, it has inherent disadvantages. On the one hand, a large number of parameters are contained in

TABLE IV
SETTING OF THE NETWORK STRUCTURE CORRESPONDING TO $\mathbf{S}_{i,j}^{vod}$

Layer	Kernels	Pooling	Stride	Output
Input	/	/	/	20×5
Convolution layer1	$5 \times 5 \times 16$	/	1	$20 \times 5 \times 16$
Pooling layer	/	2×2	2	$10 \times 2 \times 16$
Convolution layer2	$5 \times 5 \times 16$	/	1	$10 \times 2 \times 16$
Pooling layer	/	2×2	2	$5 \times 1 \times 16$

FCN, making it difficult to train, especially in a large-scale and highly dynamic VAEN scenario. On the other hand, using FCNs for feature extraction makes it difficult to capture the time-varying features of task sequences. Therefore, the training efficiency and results of fitting offloading strategies using only FCNs can not apply for the VAEN scenario.

As shown in Fig. 5, to solve this problem, we first divide the state space into three parts, and then design the IACN using 2-D convolution layers and an LSTM network. Taking the actor network as an example, for $S_{i,j}^{sti}(t)$, we consider 5×5 2D convolution filters to extract features as the input of a rectified linear unit (ReLU). Further, the max-pooling with 2×2 filters and stride 2 in the pooling layer is applied, and a convolution layer 2, ReLU and a pooling layer are followed. The details of the network structure corresponding to the $S_{i,j}^{sti}(t)$ state are shown in Table III. Moreover, we use two 2-D convolution layers and an LSTM network to extract features of $S_{i,j}^{vod}(t)$, and the detailed settings of the network structure are shown in Table IV. Other information in the state space is fed directly into the FCNs. Finally, the state space is spliced and inputted to an FCN. Note that the actor network and the critic network are different, where the former obtains the offloading decision through a Softmax layer and the letter connects an FCN to output the state value information.

3) JMPA: In the conventional actor-critic algorithm, a transition is stored as $(s_{i,p}, a_{i,p}, r_{i,p}, s_{i,(p+1)})$, i.e., one-step TD learning is applied to estimate the long-term reward. However, $Q(s_{i,p+1}, \pi((s_{i,(p+1)} \mid \theta^{\pi^{'}}) \mid \omega^{Q^{'}})$ is an inaccurate estimate of the future reward, so it is easy to have high bias and low efficiency at the early stage of training. In other words, the critic network will converge more slowly, which makes it difficult to give an accurate evaluation of the actor network. In addition, we observe that the uniform sampling in conventional actor-critic algorithm impacts the convergence of model training. Specifically, the model presents different performance when learning with different transitions in its replay buffer, which reveals different significance of these transitions for model training. To improve the speed and stability of convergence, we propose a JMPA which integrates prioritized experience replay and adaptive n-step learning. On the one hand, the importance of transitions is fully considered. On the other hand, the network is trained utilizing n-step transitions which can estimate the long-term reward adaptively.

In terms of n-step learning, n-step transitions, i.e. $(s_{i,p}, a_{i,p}, r_{i,p}, s_{i,(p+1)}, ..., a_{i,(p+n-1)}, r_{i,(p+n-1)}, s_{i,(p+n)})$,

```
Algorithm 2 JMPA
Input: The size of mini-batch $n_{\text{batch}}$, the prioritization parameter $\alpha$, correction parameter $\varphi$
1: for RV $i = 1, 2, ..., N$ do
2:     Sample a prioritized mini-batch of $n_{\text{batch}}$ transitions with length $n_{\max}$ from replay buffer.
3:     for $p = 1, 2, ..., n_{\text{batch}}$ do
4:         Compute the adaptive n-step for the $p-th$ transition and return $n^{'}$ using (38).
5:         Replace the $p-th$ transition with $\left(s_{i,p}, a_{i,p}, r_{i,p}, ..., s_{i,(p+n')}\right)$.
6:     end for
7: end for
Output: Selected transition $\left(s_{i,p}, a_{i,p}, r_{i,p}, ..., s_{i,(p+n')}\right)$
```

are cached in the experience replay memory, and the update objective can be described as

$$\begin{aligned} Q(s_{i,p}, a_{i,p}|\omega^{Q}) &= r_{i,p} + \gamma r_{i,(p+1)} + ... + \gamma^{n-1} r_{i,(p+n-1)} \\ &+ \gamma^{n} E[Q(s_{i,(p+n)}, \pi(s_{i,(p+n)} \left| \theta^{\pi^{'}} \right.)|\omega^{Q^{'}})] \end{aligned}, \tag{37}$$

where n is a predetermined value, and discount factor $\gamma \in [0, 1]$ controls the degree of how far the MDP looks into the future.

Since the transition we select may be under a different strategy, to introduce an update with a fixed value of n directly may cause a large variance. Therefore, we calculate the update value $n^{'}$ for the selected transition, namely adaptive n-step learning, which can adjust the value of $n^{'}$ dynamically to correct the "policy gap". In the initialization process, the maximum value of n is set to $n_{\max}$. In each training iteration, a trajectory of length is chosen to update the parameters. Suppose $(s_{i,p}, a_{i,p}, r_{i,p}, ..., s_{i,(p+n_{\max})})$is a trajectory that exists in experience replay buffer, $n^{'}$ can be calculated as

$$n^{'} = \begin{cases} n_{\max}, \text{if} \{\underset{\text{k}}{\arg} \Gamma\{k > \sum\limits_{j=1}^{k} I^{j}\} = 1\} = \Phi \\ \min\{\{\underset{k}{\arg} \Gamma\{k > \sum\limits_{j=1}^{k} I^{j}\} = 1\}, \text{otherwise} \end{cases}. \tag{38}$$

where $k \in (1, 2, ..., n_{\max} - 1)$, $I^{j} = \Gamma\{a_{i,(p+j)} = \pi(s_{i,(p+j)} \mid \theta^{\pi})\}$ and $\Gamma\{*\}$ is an expressivity function. When $*$ is true, $\Gamma\{*\} = 1$, otherwise $\Gamma\{*\} = 0$.

According to JMPA, in the training phase, we can rank the importance of the sample based on the absolute TD-error, i.e., a larger absolute TD-error indicates higher importance. The absolute TD-error of sample p can be calculated by (33). Note that target Q value y_p in JMPA is

$$\begin{aligned} y_p &= r_{i,p} + ... + \gamma^{n^{'}-1} r_{i,(p+n_{\max}-1)} \\ &+ \gamma^{n_{\max}} Q^{'}(s_{i,(p+n_{\max})}, \pi^{'}(s_{i,(p+n_{\max})} \left| \theta^{\pi^{'}} \right.) \left| \omega^{Q^{'}} \right.) \end{aligned}. \tag{39}$$

Based on absolute TD-error, priority $rank(p)$ of the $p-th$ transition is can be obtain.

Algorithm 3 DCOMSDRL: Interaction process

```
Input: The set of RVs $\mathcal{RV}$, the set of SVs $\mathcal{SV}$, the set of MEC servers $\mathcal{M}$, road segment $\mathcal{R}$, computing capacity of RVs $\mathcal{F}_{\mathcal{RV}}$, computing capacity of SVs $\mathcal{F}_{\mathcal{SV}}$, computing capacity of MEC servers $\mathcal{F}_{\mathcal{MEC}}$, the size of replay buffer $W$, predetermined value $n_{\max}$
1: Initialize locations and velocities of all vehicles
2: Initialize main actor network with $\theta^{\pi}$ and main critic network with $\omega^{Q}$
3: Initialize target actor network with $\theta^{\pi^{'}} \leftarrow \theta^{\pi}$; target critic network with $\omega^{Q^{'}} \leftarrow \omega^{Q}$
4: for $episode = 1, 2, ..., E_{\max}$ do
5:   for RV $i = 1, 2, ..., N$ do
6:     get subtask scheduling priority (see Algorithm 1).
7:     obtain initial state $s_{i,1}$ of the VAEN.
8:   end for
9:   for $t = 1, 2, ..., |T_i|$ do
10:    for RV $i = 1, 2, ..., N$ do
11:      Select action $a_{i,t}$ according to $\varepsilon$-greedy policy.
12:      Execute action $a_{i,t}$ and interact with the environment.
13:      Obtain reward $r_{i,t}$ and the next state $s_{i,(t+1)}$.
14:      Store transition $\left(s_{i,t}, a_{i,t}, r_{i,t}, ..., s_{i,(t+n_{\max})}\right)$ in replay memory buffer and calculate initial priorities using (40).
15:    end for
16:  end for
17:  BS broadcasts the information to all RSU.
18: end for
Output: Offloading desicion $a_{i,t}$
```

Further, the probability of sampling transition p is

$$P(p) = \frac{(z_p)^{\alpha}}{\sum\limits_{g \in W} (z_g)^{\alpha}}, \tag{40}$$

where $z_j = 1/rank(p)$, and α is the prioritization parameter.

Finally, to correct the bias introduced by changing the state visitation frequency, [52] introduces the importance-sampling weights, which can be calculated by

$$\phi_p = \frac{1}{W^{\varphi} \cdot P(p)^{\varphi}}. \tag{41}$$

Here, W is the size of the replay buffer, $P(p)$ is the probability of the sampled experience p, and the parameter φ controls to the extent of used corrections.

The pseudocode of JMPA is summarized in Algorithm 2.

4) DCOM Descriptions: In DCOM, we divide it into two phases, i.e., the interaction phase and learning phase.

a) Interaction phase: The detail of the interaction phase is described in Algorithm 3. Each RV initializes the parameters of the actor network and critic network (Lines 1-3). For each episode, each RV generates fine-grained tasks, obtains the scheduling priority of subtasks according to Algorithm 1(Lines 4-6), and acquires the initial state of the environment (Line 7). At the beginning of a time slot, each RV selects an action based on the obtained state of the environment, and interacts with the environment to obtain a reward and the next state(Lines 9-13). Further, transition $\left(s_{i,t}, a_{i,t}, r_{i,t}, ..., s_{i,(t+n_{\max})}\right)$ is stored in the replay buffer and the priority is calculated by (40) (Line 14). Finally, after each task is processed completely, the BS broadcasts the information to all RSU (Lines 17).

Algorithm 4 DCOMSDRL: Learning process

```
Input: The size of mini-batch $n_{\text{batch}}$, target networks updating rate $\xi$, the prioritization parameter $\alpha$, correction parameter $\varphi$, soft update parameter $\tau^{'}$
1: for RV $i = 1, 2, ..., N$ do
2:   Sample $n_{\text{batch}}$ transitions from the replay buffer according to JMPA (see Algorithm 2).
3:   Calculate the $y_p$ in the target critic network with (39).
4:   Update main critic network according to (34).
5:   Update main actor network according to (35).
6:   Update the target policy network and target Q networks according to (36) with an updating rate $\xi$.
7: end for
Output: Updated parameters $\theta^{\pi}$ of main actor network $\pi$, updated parameters $\theta^{\pi^{'}}$ of target actor network, updated parameters $\omega^{Q}$ of main critic network $Q$, updated parameters $\omega^{Q^{'}}$ of main critic network $Q^{'}$
```

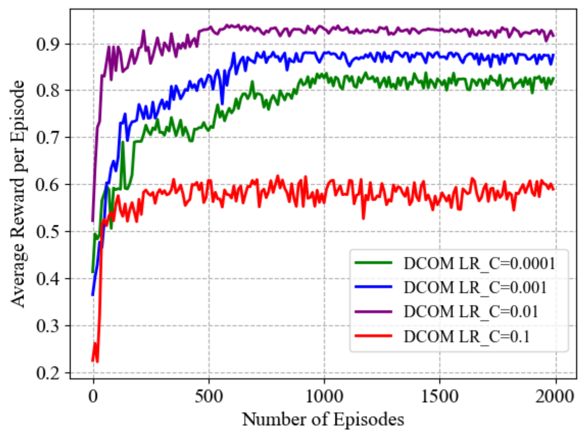
b) Learning phase: We provide the details of the learning phase in Algorithm 4. The learning phase is started based on known actor network and critic network parameters and other parameters. For each RV, we sample transitions in the replay buffer according to the priority, and the detailed procedure of the algorithm is described in Algorithm 2 (Line 2). Further, the value of the target critic network is calculated by (39) (Line 3). Then the mian critic network and main actor network are updated by (34) and (35), respectively (Lines 4-5). Finally, the target actor network and target critic network are updated based on (37) (Line 6).

VI. Numerical Result and Analysis

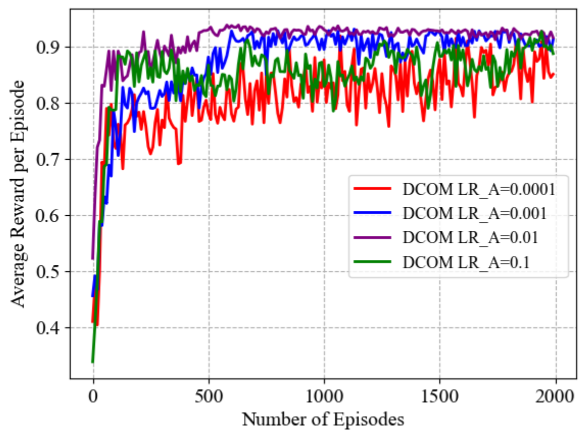
In this section, we first introduce detailed parameter settings for simulation. Moreover, we verify the convergence of DCOM as well as the effectiveness of IACN and JMPA. Finally, we compare DCOM with other benchmark algorithms.

A. Simulation Settings

We perform the distributed computing offload algorithm with tensorflow 1.5 in python 3.7. In our simulations, we set up 3 edge servers, 20 RVs, and 8 SVs distributed on a 1000-meter one-way road. Each RV generates a computation-intensive latency-sensitive task at each time slot, which can be divided into inter-dependent subtasks with an upper limit of 10 subtasks. We assume that the size of each subtask's input data is limited by [100, 500] KB, and its volume ratio ranges from [100, 500] cycles/byte. In addition, the maximum time delay allowed by task $\mathcal{T}_i$ is selected from [0.5, 2] s, and the bandwidth of an RSU is 30 MHz. The transmission power and the CPU cycle frequency of each RV range from [0.1, 0.5] W and [0.5, 2] GHz, respectively. Moreover, the CPU frequency of the MEC server is selected from 4 GHz to 8 GHz, the CPU



(a) Critic network learning.



(b) Actor network learning.

Fig. 6. Average reward with different learning rate.

frequency of each SV is selected from 2 GHz to 4 GHz and the maximum communication range of SVs is 200 m.

As for the design of four networks of DCOM, the specific network structure parameters are presented in Tables II and Tables III. The size of the experience replay buffer is set to 1000, the mini-batch is set to 128 and the discount factor is set to 0.9, respectively. During training, an Adam optimizer is adopted to optimize the loss function.

B. Convergence Performance

We first verify the convergence of the proposed DCOM at different learning rates of its actor network and critic network. Fig. 6 illustrates the relationship between episodes and the average reward at different learning rates. By fixing the actor network's learning rate to 0.01 and increasing the critic network's learning rate from 0.0001 to 0.1, it is observed that the best convergence and reward values are achieved when the actor network's learning rate is 0.01. Similarly, by fixing the learning rate of the critic network at 0.01 and increasing the learning rate of the actor network from 0.0001 to 0.01, the average reward curve converges accelerated and the reward value increases gradually. If the actor network's learning rate is increased by 0.1, the reward value is significantly decreased, which is caused by the local optimum of the algorithm due to a large learning rate. Therefore, in our subsequent experiments, we fix the learning rate of the actor network and critic network to 0.01.

C. Effectiveness simulations

To evaluate the effectiveness of the proposed VAEN, IACN, and JMPA, we design the following benchmark algorithms for comparison.

1) DCOM-without-V2V: DCOM is used to formulate offloading decisions without considering V2V communication, where only MEC servers are considered as edge nodes.

2) AC-JMPA: JMPA is adopted on the basis of a conventional AC. Note that the network structures are FCNs. As for the design of the four networks, an input layer, two hidden fully-connected layers, and an output layer are conducted. The two hidden layers contain 200 nodes and 100 nodes, respectively.

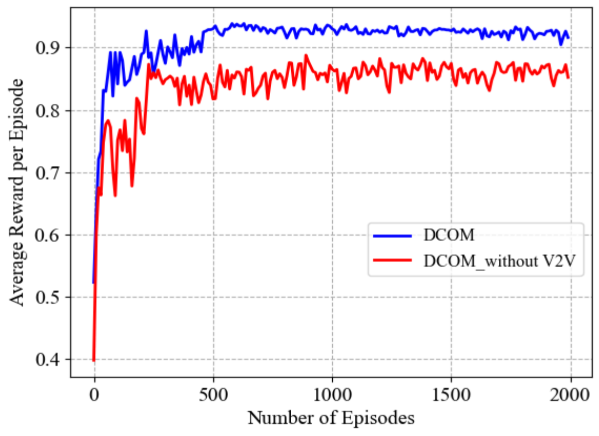


Fig. 7. Comparison of the reward of the two scenarios.

3) AC-IACN: IACN is used based on a conventional AC. It is noted that this algorithm applies uniform sampling for one-step transition.

4) AC: The conventional AC algorithm applies an FCN structure and uniform sampling.

Fig. 7 illustrates the curves of the average reward, and depict the average reward for the VAEN scenario and the computation offloading scenario without considering V2V communication. In Fig. 8, with the increase of episodes, both DCOM and DCOM-without-V2V achieve convergence. However, due to the action space dimension is large in VAEN scenario, DCOM converges faster than DCOM-without-V2V. Specifically, DCOM achieves convergence at around the 500-th episode, while the DCOM-without-V2V converges at about the 400-th episode. In comparison, the DCOM can achieve

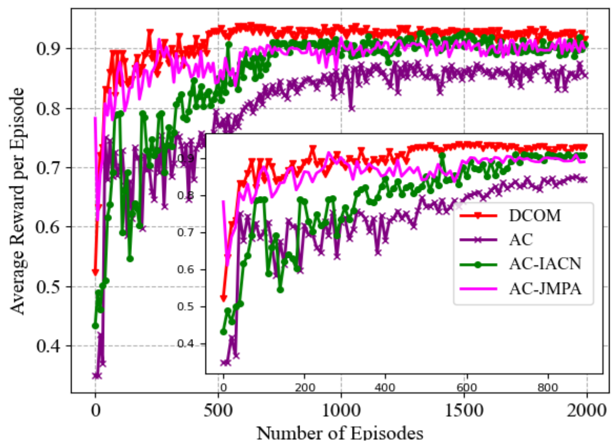


Fig. 8. Average reward for different algorithms.

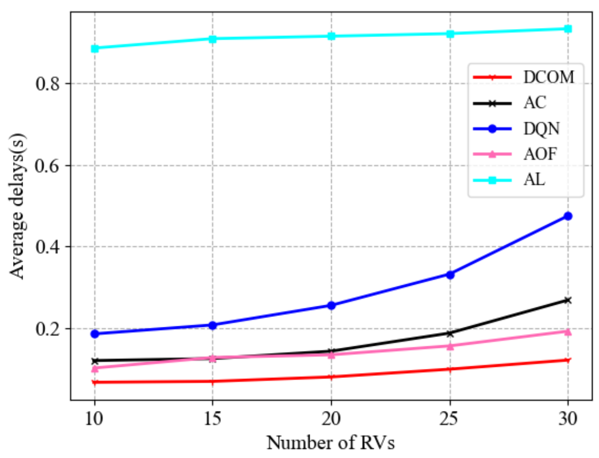


Fig. 9. Average delay with different numbers of RVs.

a higher reward value, which is increased by about 13%. That is because RVs reduce the queuing time at fixed MEC servers by offloading tasks to SVs, thus the task completion delay is shortened and the reward is increased. Therefore, it is necessary to consider the resources of SVs in multi-vehicle offloading scenarios to enhance the system utility.

Fig. 8 illustrates the reward curves for the four benchmark algorithms of DCOM, AC, AC-JMPA, and AC-IACN. It is observed that the agents of different algorithms achieve convergence by interacting with the environment, but the speeds of convergence and the average reward values after stabilization reveal different. Moreover, AC converges around 800 episodes and AC-JMPA converges around 600 episodes. The reason is that AC-JMPA adopts JMPA, where prioritized experience replay (PER) can make full use of valuable transitions and the adaptive n-step method accelerate the convergence of the critic network. Compared with AC, the average reward value of AC-JMPA is increased by about 4.5%, and AC-IACN achieves convergence at around 730 episodes and the average reward increases by about 5%. The reason behind that is the network structure that replaces FCN with Conv2D helps the agent to extract the features of its state space. In addition, LSTM captures the long-term changes of time-dependent data, which helps the agent to select an appropriate action and get the best reward. DCOM combines IACN and JMPA, not only has a substantial improvement in convergence speed compared with AC but also improves the average reward value by about 9%. In summary, IACN and JMPA are effective in speeding up convergence and improving the system utility.

D. Overall Performance Comparison

To evaluate the performance of DCOM, we design three benchmark algorithms to investigate the impact of the number of RVs, the weight of delay, the bandwidth and total computing capacity of fixed MEC servers, and the task data size on system performance. The settings of these benchmark algorithms are as follows.

1) AC: A conventional AC algorithm with an FCN structure and uniform sampling.

2) DQN: A value-based conventional DQN algorithm, which utilize value functions to guide the agent selection strategy in the VAEN scenario.

3) AllOff: The agent's action space is G+1, i.e., it only selects to offload to the nearest fixed MEC server or SV g for execution according to DCOM.

4) AL: All tasks generated by RVs are processed locally.

1) Impact of the number of RVs: Fig. 9 depicts the task average delay of DCOM, AC, DQN, AllOff, and AL with different numbers of RVs. The number of RVs is varied from 10 to 30 with an increment of 5. As illustrated in Fig. 9, the task average delay of DCOM, AC, DQN, and AllOff increases with the number of RVs, because as the number of RVs become larger, the RVs get less upload bandwidth from the fixed MEC server and the queuing time becomes longer. Therefore, the transmission and computation delay in the fixed MEC server become large which results in an increase in the average delay of those tasks. In addition, as the number of RVs increases, the delay is impacted more and more. When the increment of RVs is small, the system resources are still relatively sufficient, thus the delay is not highly impacted. However, when the number of RVs is large enough, the system's bandwidth and computation resources are highly competitive, and the delay is greatly impacted. Compared with AC, DQN and AllOff, the delay of DCOM is the smallest all the time. On average, the DCOM reduces the latency by 40% compared with AC. Compared with DQN and AllOff, the latency of DCOM is reduced by 61% and 38%, respectively. Unlike the trends of the four curves mentioned above, the average task latency of AL is not significantly impacted by the number of RVs, because AL processes all tasks locally and is only related to the computation capabilities of RVs instead of their number. We can conclude that the latency performance of DCOM consistently outperforms the other four benchmark algorithms as the number of RVs increases.

2) Impact of the weight of delay: In Fig. 10, we illustrate the trade-off between the average delay and average energy consumption of tasks with different delay weights, which

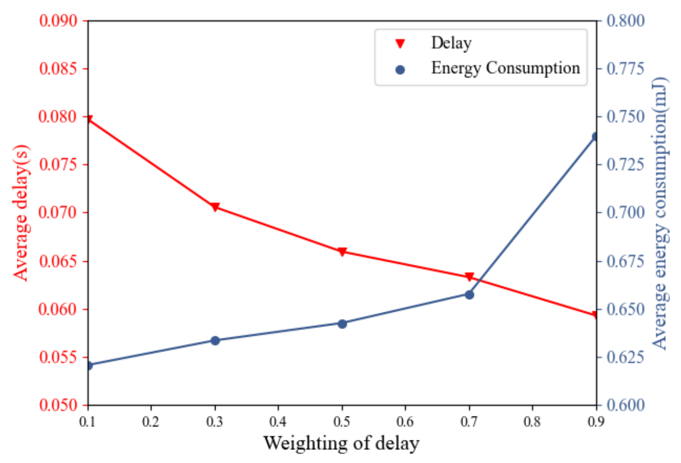


Fig. 10. The tradeoff between the average delay and average energy under different weights of delay.

TABLE V
COMPARISON OF AVERAGE SYSTEM COST
FOR DIFFERENT WEIGHTS OF DELAY

Algorithm/λ	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**
DCOM	0.141	0.133	0.130	**0.129**	0.133
AC	0.267	0.258	0.257	**0.255**	0.277
DQN	0.436	0.428	0.427	**0.416**	0.442
AL	1.989	1.989	1.989	**1.989**	1.989
AllOff	0.198	0.186	0.181	**0.169**	0.175

varies from 0.1 to 0.9 with an increment of 0.1. Based on the previous description in Section IV, it is known that $\lambda_t + \lambda_e = 1$. As illustrated in Fig. 10, with the increase of λ_t, RVs achieve a low delay but high energy consumption. This is because a large λ_t indicates that the agent is concerned with reducing the execution delay of delay-sensitive tasks, and the demand for low energy consumption has a lower priority. Therefore, in practical applications, the weights can be adjusted according to the state of RVs when tasks are highly latency-sensitive, such as driving safety-related tasks, λ_t can be increased. For example, when the battery energies of RVs are low, λ_t can be decreased to maintain the power level.

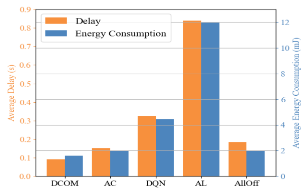
We further compare the system cost performance with different offloading algorithms under different delay weights. Table IV records the system cost of DCOM, AC, DQN, AllOff, and AL when λ_t takes different values. It is observed that the system cost values of DCOM, AC, DQN, and AllOff reveal undulatory as λ_t increases, and the minimum system cost is achieved when λ_t=0.7. In addition, the system cost values of DCOM are smaller than the other four benchmark algorithms regardless of the value of λ_t. In the subsequent simulations, we consider λ_t=0.5 because the scenario studied in this work is equally significant in terms of the time delay and energy consumption.

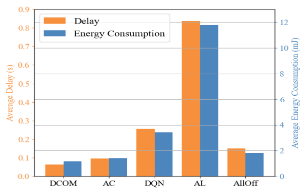
3) Impact of bandwidth of MEC server: We evaluate the impact of total bandwidth of MEC server on the average delay and average energy consumption. Fig. 11 illustrates the average delay and average energy consumption of tasks for DCOM, AC, DQN, AllOff, and AL with different total bandwidths of MEC server. The total bandwidth of MEC server is varied from 10 MHz to 40 MHz with an increment of 10 MHz. It is observed from Fig. 11(a)-(d) that as the total bandwidth increases, the average delay and average energy consumption of the task decreases. This is because the larger the total bandwidth, the higher the upload rate of the task is offloaded, resulting in lower transmission delay and lower transmission energy consumption. Therefore, increasing the total bandwidth results in small average latency and low average energy consumption of the task. In addition, when the bandwidth is increased from 30 MHz to 40 MHz, the average delay and average energy consumption of DCOM, AC, DQN, and AllOff almost no change. The reason for this is when the total bandwidth increases to a certain value, the effect of bandwidth on system utility is no longer obvious, that is, in the current environment, the competition for bandwidth resources is no longer intense and the magnitude of system utility is constrained by other factors in the environment. From Fig.11 (a)-(d), it is observed that DCOM has the lowest latency and energy consumption compared with the other four benchmark algorithms regardless of any value of the total bandwidth. In this experiment, we can infer that larger the total bandwidth leads to smaller the average latency and energy consumption of the task.

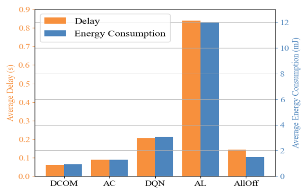
4) Impact of total computing capacity of MEC server : We consider the impact of different computing capacities of the MEC server. Fig. 12 illustrate the average delay of DCOM, AC, DQN, AllOff, and AL with the MEC server's computing capacity, which varied from 1 to 9 with an increment of 2. In Fig. 12, the average delay of tasks decreases with the computing capacity of MEC server increases. This is because larger the computing capacity of the MEC server leads to smaller the task execution delay at the MEC server. Since AL is independent of the MEC server, it is not impacted by its computing capacity. In addition, the latency of the DCOM is lower than that of the other four benchmark algorithms regardless of the variation of the computing capacity of the MEC server.

Fig. 13 shows the variation of the average energy consumption of the five algorithms with the computing capacity of the MEC server. For DCOM, AC, DQN, and AllOff, the energy consumption decreases significantly when the computing capacity of MEC server increases from 1GHz to 3GHz, but it reveals little changes when it continues to increase. Intrinsically, when the computing capacity of MEC server reaches 1GHz, most of RVs choose to offload to SVs or local execution due to the insufficient resources of MEC, which results in high energy consumption. When the computing capacity of MEC server is increased to 3GHz, the resources of MEC are sufficient, and the energy consumption of the system drops significantly. When the computing capacity of MEC server continues to be increased, the impact is on the processing time of the task on MEC, and the impact on both the transmission delay and energy consumption are small.

Similar to Fig. 12, the average energy consumption of


(a) Bandwidth=10MHz


(b) Bandwidth=20MHz


(c) Bandwidth=30MHz

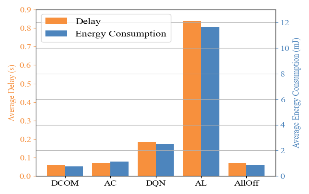

(d) Bandwidth=40MHz

Fig. 11. Impact of total bandwidth on average delay and average energy.

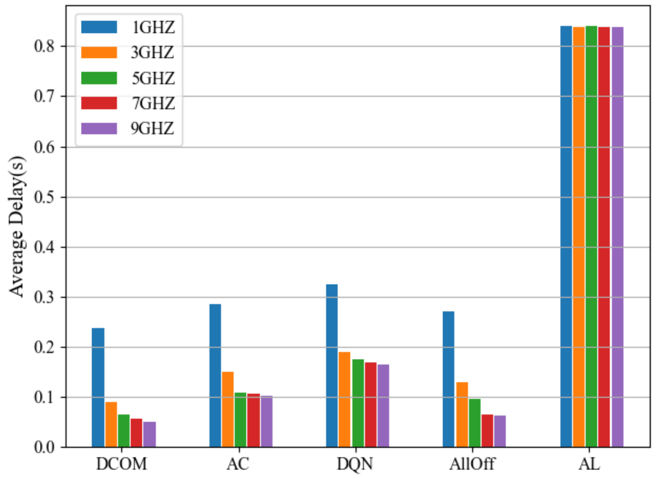


Fig. 12. Impact of total computing capacity of MEC server on average delay.

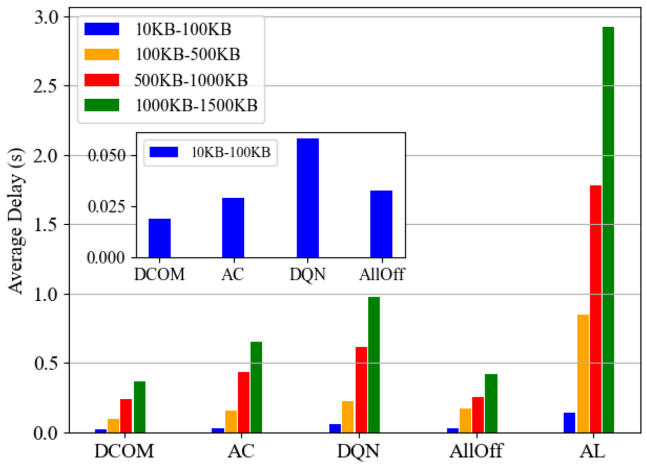


Fig. 14. Impact of subtask data size on average delay.

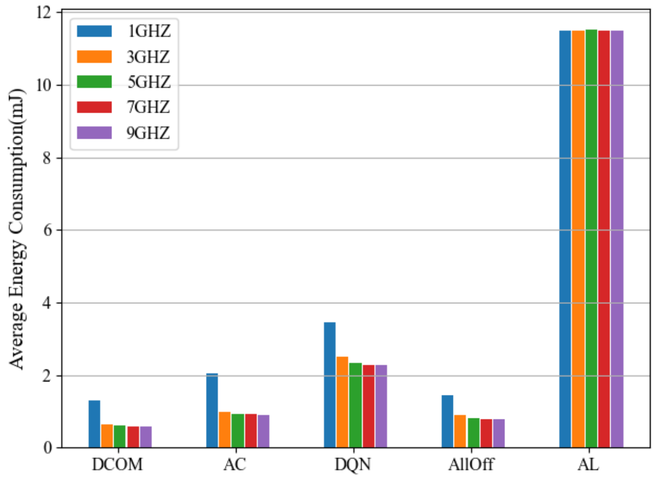


Fig. 13. Impact of total computing capacity of MEC server on average energy.

DCOM is lower than that of the other four benchmark algorithms regardless of the variation of MEC servers' computing capacities. As illustrated in Fig. 13, averagely, DCOM reduces 35.8% of the average energy consumption compared with AC. In comparison with DQN and AllOff, DCOM reduces 74.4% and 23% of the average energy consumption, respectively.

5) *Impact of task data size:* Figs. 14 and 15 depict the effect of subtask data size on the average delay and average energy consumption of the system. The data size is varied from [10-100] KB to [1000-1500] KB. It is observed that the average delay and average energy consumption increase as the task data size increases. The key reason is larger the amount of task data leads to greater processing and transmission latency when the task is offloaded. When the subtask data size is [500-1000] KB, the delay of AL exceeds the maximum time delay allowed by the task, which causes the task execution to fail. Similarly, it also causes large energy consumption during local execution and transmission. In addition, it is observed that the average delay and average energy consumption of DCOM are minimized compared with other four benchmark algorithms. DCOM, AC, DQN, and AllOff are optimal because they can adaptively learn to make decisions, which are different from AL. However, DCOM, AC and AllOff achieve better performance than DQN because the actor network and critic network are utilized to make decisions. Furthermore, DCOM achieves better performance than AC because DCOM combines IACN and JMPA, which reduces the average delay and average energy consumption and improves the learning efficiency. Compared with AllOff, DCOM utilizes more system resources, such as RV local resources. Moreover, the average delay and average energy consumption of AL increase rapidly with the increase of its task data size, which indicates that the impact of task data size on AL is significant. Therefore, it is necessary for RVs to make appropriate offloading decisions when generating tasks with large data volumes.

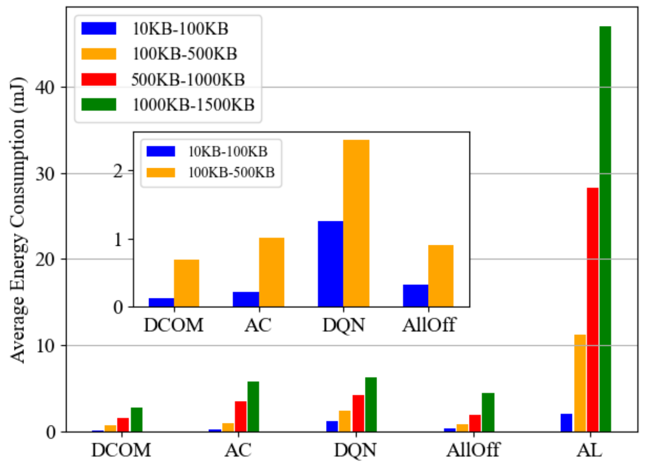


Fig. 15. Impact of subtask data size on average energy consumption.

VII. Conlusion

In this paper, we investigate the computation offloading problem in VAEN scenario with multi-vehicle, where both MEC servers and SVs are considered as edge nodes to provide computation resources. We consider each RV having a computation-intensive and latency-sensitive task that could be partitioned into interdependent subtasks and need to be completed within a time slot. Moreover, under the CSV model constraint, we have formulated the optimization problem by minimizing the delay and energy consumption. Further, to tackle this problem, we propose DCOM adopting IACN and JMPA to make an offloading decision in real-time. Simulation results demonstrate the effectiveness of the VAEN scenario, IACN, and JMPA, and validate the superiority of DCOM compared with other benchmark algorithms. Although satisfactory results are achieved by this work, it is still necessary to take care of the privacy issues during data transmission. In the future, we will further explore the computation offloading based on federated learning.

Appendix A
Proof of Theorem 1

First, we introduce the 0-1 knapsack problem which is a typical NP-hard problem. In the 0-1 knapsack problem, it is known that there are N items with the value $(v_1, v_2, ..., v_N)$. Moreover, the weights of the items are $(w_1, w_2, ..., w_N)$ and the total weight cannot exceed W. Mathematically, the problem can be expressed as

$$\begin{array}{l} Knap: \max \sum\limits_{i \in N} x_i v_i \\ \quad s.\,t. \sum\limits_{i \in N} x_i w_i \le W \qquad x_i \in \{0,1\} \end{array}. \tag{42}$$

To simplify the problem P1, we consider an RV, an MEC server in our scenario, and ignore the transmission delay and energy consumption when RV i offloads the task to the MEC server. In addition, we consider task delay minimization under energy constraints. Therefore, the problem P1 can be expressed as

$$\begin{array}{l} P1: \min \sum\limits_{j \in |T_i|} lp_{ij} \cdot \frac{c_{ij}}{f_i^{RV}} + (1 - lp_{ij}) \cdot T_{i,exe}^{V2I} \\ \quad s.t. \sum\limits_{j \in |T_i|} lp_{ij} \cdot \kappa (f_i^{RV})^2 c_{ij} \le E_i^{\max} \qquad lp \in \{0,1\} \end{array}. \tag{43}$$

For each subtask, we define $a_j = -c_{ij}/f_i^{RV}$, then P1 can be formulated as

$$\begin{array}{l} P1: \max \sum\limits_{j \in |T_i|} lp_{ij} \cdot a_j + (1 - lp_{ij}) \cdot T_{i,exe}^{V2I} \\ \quad s.t. \sum\limits_{j \in |T_i|} lp_{ij} \cdot \kappa (f_i^{RV})^2 c_{ij} \le E_i^{\max} \qquad lp_{ij} \in \{0,1\} \end{array}. \tag{44}$$

Next, we make the following assumptions

$$T_{i,exe}^{V2I} = 0, \tag{45}$$

$$\kappa (f_i^{RV})^2 c_{ij} = w_j. \tag{46}$$

Knap can be reduced to problem P1. Therefore, Theorem 1 is proved.

References

[1] Q. Luo, C. Li, T. H. Luan, W. Shi and W. Wu, “Self-Learning Based Computation Offloading for Internet of Vehicles: Model and Algorithm,” *IEEE Trans. Wirel. Commun.*, vol. 20, no. 9, pp. 5913-5925, Sept. 2021.

[2] Y. Zhai, W. Sun, J. Wu, L. Zhu, J. Shen and X. Du, “An Energy Aware Offloading Scheme for Interdependent Applications in Software-Defined IoV With Fog Computing Architecture,” *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 6, pp. 3813-3823, Jun. 2021.

[3] B. Hazarika, K. Singh, S. Biswas and C. -P. Li, “DRL-Based Resource Allocation for Computation Offloading in IoV Networks,” *IEEE Trans. Ind. Inform.*, vol. 18, no. 11, pp. 8027-8038, Nov. 2022.

[4] Z. Cheng, M. Min, Z. Gao, and L. Huang, “Joint task offloading and resource allocation for mobile edge computing in ultra-dense network,” in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2020, pp. 1-6.

[5] H. Zhou, K. Jiang, X. Liu, X. Li and V. Leung, “Deep Reinforcement Learning for Energy-Efficient Computation Offloading in Mobile-Edge Computing,” *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1517-1530, Jan. 2022.

[6] H. Liu, H. Zhao, L. Geng and W. Feng, “A Policy Gradient Based Offloading Scheme with Dependency Guarantees for Vehicular Networks,” in *Proc. IEEE Global Commun. Conf. Wkshps*, Dec. 2020, pp. 1-6.

[7] H. Ke, H. Wang, W. Sun, and H. Sun, “Adaptive Computation offloading policy for Multi-Access Edge Computing in Heterogeneous Wireless Networks,” *IEEE Trans. Netw. Serv. Manag.*, vol. 19, no. 1, pp. 289-305, Mar. 2022.

[8] J. Shi, J. Du, J, Wang, J. Wang and J. Yuan, “Priority-Aware Task Offloading in Vehicular Fog Computing Based on Deep Reinforcement Learning”, *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 16067-16081, Dec. 2021.

[9] Y. Cheng, C. Liang, Q. Chen, and F. Yu, “Energy-Efficient D2D-Assisted Computation Offloading in NOMA-Enabled Cognitive Networks,” *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13441-13446, Dec. 2021.

[10] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A survey on mobile edge computing: The communication perspective,” *IEEE Commun. Surv. Tutor.*, vol. 19, no. 4, pp. 2322–2358, 2017.

[11] S. Bi and Y. J. Zhang, “Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading,” *IEEE Trans. Wirel. Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.

[12] F. Wang, J. Xu and Z. Ding, “Multi-Antenna NOMA for Computation Offloading in Multiuser Mobile Edge Computing Systems,” *IEEE Trans. Commun.*, vol. 67, no. 3, pp. 2450-2463, Mar. 2019.

[13] C. You, K. Huang, H. Chae, and B.-H. Kim, “Energy-efficient resource allocation for mobile-edge computation offloading,” *IEEE Trans. Wirel. Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.

[14] Y. Zhan, S. Guo, P. Li and J. Zhang, “A Deep Reinforcement Learning Based Offloading Game in Edge Computing,” *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 883-893, Jun. 2020.

[15] J. Wang, T. Lv, P. Huang, and P. Mathiopoulos, “Mobility Aware Partial Computation Offloading in Vehicular Networks: A Deep Reinforcement Learning Based Scheme,” *China Commun.*, vol. 17, no. 10, pp. 31–49, Oct. 2020.

[16] Z. Ning, P. Dong, X. Wang, L. Guo, J. Rodrigues, X. Kong, J. Huang, and R. Kwok, “Deep Reinforcement Learning for Intelligent Internet of Vehicles: An Energy-Efficient Computational Offloading Scheme,” *IEEE Trans. Cogn. Commun. Netw.*, vol. 5, no. 4, pp. 1060-1072, Dec. 2019.

[17] F. Fu, Y. Kang, Z. Zhang, F. Yu and T. Wu, “Soft Actor-Critic DRL for Live Transcoding and Streaming in Vehicular Fog-Computing-Enabled IoV,” *IEEE Internet Things J.*, vol. 8, no. 3, pp. 1308-1321, Feb. 2021.

[18] W. Zhan, C. Luo, J. Wang, C. Wang, G.Min, H. Duan, and Q. zhu, “Deep-Reinforcement-Learning-Based Offloading Scheduling for Vehicular Edge Computing,” *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5449-5465, Jun. 2020.

[19] Y. Mao, J. Zhang, and K. B. Letaief, “Dynamic computation offloading for mobile-edge computing with energy harvesting devices,” *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590-3605, Dec. 2016.

[20] K. Mohamed, L. Wael, and S. Mireille, “Joint resource allocation and offloading strategies in cloud enabled cellular networks,” in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 5529-5534.

[21] X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” *IEEE-ACM Trans. Netw.*, vol. 24, no. 5, pp. 2827-2840, Oct. 2016.

[22] G. Wu and Z. Li, “Task Offloading Strategy and Simulation Platform Construction in Multi-User Edge Computing Scenario,” *Electronics*, vol. 10, no. 23, Dec. 2021.

[23] M. Guo, W. Wang, X. Huang, Y. Chen, L. Zhang and L. Chen, “Lyapunov-Based Partial Computation Offloading for Multiple Mobile Devices Enabled by Harvested Energy in MEC,” *IEEE Internet of Things J.*, vol. 9, no. 11, pp. 9025-9035, Jun. 2022.

[24] H. Ke, J. Wang, L. Deng,Y. Ge and H. Wang, “Deep Reinforcement Learning-based Adaptive Computation Offloading for MEC in Heterogeneous Vehicular Networks,” *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7916-7929, Jul. 2020.

[25] C. You and K. Huang, “Multiuser Resource Allocation for Mobile-Edge Computation Offloading,” in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1-6.

[26] Y. Chen, Z. Liu, Y. Zhang, Y. Wu, X. Chen and L. Zhao, “Deep Reinforcement Learning-Based Dynamic Resource Management for Mobile Edge Computing in Industrial Internet of Things,” *IEEE Trans. Ind. Inform.*, vol. 17, no. 7, pp. 4925-4934, Jul. 2021.

[27] Z. Zhang, C. Li, S. Peng, and X. Pei, “A new task offloading algorithm in edge computing,” *EURASIP J. Wirel. Commun. Netw.*, vol. 2021, no. 1, pp. 1-21, Jan. 2021.

[28] Y. Mao, J. Zhang, S. Song and K. B. Letaief, “Power-Delay Tradeoff in Multi-User Mobile-Edge Computing Systems,” in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, pp. 1-6, Dec. 2016.

[29] N. Zhao, P. Dong, X. Wang, J. Rodrigues, and F. Xia, “Deep Reinforcement Learning for Vehicular Edge Computing: An Intelligent Offloading System,” *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 6, pp. 1-24, Dec, 2019).

[30] 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.

[31] C. Feng, Z. Zhao, Y. Wang, T. Quek and M. Peng, “On the Design of Federated Learning in the Mobile Edge Computing Systems,” *IEEE Trans. Commun.*, vol. 69, no. 9, pp. 5902-5916, Sept. 2021.

[32] Y. Lu, X. Huang, K. Zhang, S. Maharjan and Y. Zhang, “Blockchain Empowered Asynchronous Federated Learning for Secure Data Sharing in Internet of Vehicles,” *IEEE Trans. Veh. Technol.*, vol. 69, no. 4, pp. 4298-4311, Apr. 2020.

[33] B. Zhou, S. Srirama and R. Buyya, “An auction-based incentive mechanism for heterogeneous mobile clouds,” *J. Syst. Softw.*, vol. 152, pp. 0164-12126, Jun. 2019.

[34] J. Sladana and D. Gyorgy, “Wireless and computing resource allocation for selfish computation offloading in edge computing,” in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Sept. 2019, pp. 2467-2475.

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

[36] J. Zheng, Y. Cai, Y. Wu, and X. Shen, “Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach,” *IEEE. Trans. Mob. Comput.*, vol. 18, no. 4, pp. 771–786, Apr. 2019.

[37] A. Bi and Y. Zhang, “Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading,” *IEEE Trans. Wirel. Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.

[38] G. Ahani and D. Yuan, “BS-Assisted Task Offloading for D2D Networks with Presence of User Mobility,” in *Proc. IEEE 89th Veh. Technol. Conf. (VTC2019-Spring)*, Apr. 2019, pp. 1-5.

[39] Z. Cheng, M. Min, M. Liwang, L. Huang and Z. Gao, “Multiagent DDPG-Based Joint Task Partitioning and Power Control in Fog Computing Networks,” *IEEE Internet Things J.*, vol. 9, no. 1, pp. 104-116, Jan. 2022.

[40] J. Yan, S. Bi, and Y. Zhang, “Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach,” *IEEE Trans. Wirel. Commun.*, vol. 19, no. 8, pp. 5404–5419, Aug. 2020.

[41] L. Xiao, X. Lu, T. Xu, X. Wan, W. Ji, and Y. Zhang, “Reinforcement learning-based mobile offloading for edge computing against jamming and interference,” *IEEE Trans. Commun.*, vol. 68, no. 10, pp. 6114–6126, Oct. 2020.

[42] M. Min, X. Wan, X. Liang and C. Ye, “Learning-based privacy-aware offloading for healthcare IoT with energy harvesting,” *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4307–4316, Jun. 2019.

[43] M. Hu, L. Zhuang, D. Wu, Y. Zhou, X. Chen, and L. Xiao, “Learning driven computation offloading for asymmetrically informed edge computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 8, pp. 1802–1815, Aug. 2019.

[44] J. Baek and G. Kaddoum, “Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multifog networks,” *IEEE Internet Things J.*, vol. 8, no. 2, pp. 1041–1056, Jan. 2021.

[45] J. Zhang, H. Guo, and J. Liu, “Adaptive Task Offloading in Vehicular Edge Computing Networks: a Reinforcement Learning Based Scheme,” *Mobile Netw. Appl.*, vol. 25, no. 5, pp. 1736–1745, Jun. 2020.

[46] Y. Wu, T, Dinh, Y, Fu, C, Lin, and T, Quek, “A Hybrid DQN and Optimization Approach for Strategy and Resource Allocation in MEC Networks,”*IEEE Trans. on Wirel. Commun.*, vol. 20, no. 7, pp. 4282–4295, Jul. 2021.

[47] S. Song, Z. Fang, Z. Zhang, C. Chen and H. Sun, “Semi-Online Computational Offloading by Dueling Deep-Q Network for User Behavior Prediction,” *IEEE Access*, vol. 8, pp. 118192-118204, Jul. 2020.

[48] B. R. Kiran et al., “Deep Reinforcement Learning for Autonomous Driving: A Survey,” *IEEE Trans. on Intell. Transp. Syst.*, vol. 23, no. 6, pp. 4909-4926, Jun. 2022.

[49] K. Liu and W. Liao, “Intelligent offloading for multi-access edge computing: A new actor–critic approach,” in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020. pp. 1–6.

[50] Geng, H. Zhao, H. Liu, Y. Wang, W. Feng and L. Bai, “Deep Reinforcement Learning-based Computation Offloading in Vehicular Networks,” in *Proc. IEEE International Conference on Edge Computing and Scalable Cloud*, May 2021, pp. 200-206.

[51] H. Lu, X. He, M. Du, X. Ruan, Y. Sun, and K. Wang, “Edge QoE: Computation offloading with deep reinforcement learning for Internet of Things,” *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9255–9265, Oct. 2020.

[52] Y. Hou, L. Liu, Q. Wei, X. Xu and C. Chen, “A novel DDPG method with prioritized experience replay,” in *Proc. 2017 IEEE International Conference on Systems, Man, and Cybernetics*, Oct. 2017, pp. 316-321.

Liwei Geng received the master’s degree from Beihang University, Beijing, China, in 2019, where she is currently pursuing the Ph.D. degree in electronic and information engineering with the School of Electronic and Information Engineering. She is currently doing research on mobile edge computing and deep reinforcement learning.

Hongbo Zhao is a professor in the School of Electronic and Information Engineering, Beihang University, Beijing, China. He received Ph.D. degree in communication and information system from Beihang University in 2012. He has been teaching there since 2012. His current research interests include vehicular networks, mobile edge computing and non-terrestrial communication network.

Jiayue Wang received the B.Sc. degree from the School of Electronic and Information Engineering, Beihang University, Beijing, China, in 2020. He is currently pursuing the Master's degree at Beihang University, Beijing. His current research interests include mobile edge computing.

Aryan Kaushik is an Assistant Professor at the University of Sussex, UK. He has been a Research Fellow at the University College London (UCL), UK, from 2020-21, and completed PhD degree in Communications Engineering from the University of Edinburgh, UK, in 2019. He received MSc in Telecommunications from the Hong Kong University of Science and Technology, Hong Kong, in 2015. He has held visiting appointments at the Imperial College London, UK, Shenzhen Institute of Beihang University, China, University of Luxembourg, Luxembourg, Athena Research and Innovation Center, Greece, and Beihang University, Beijing, China. He has been involved as research lead/PI or Co-I in several UK-wide and international collaborative projects. His research interests include signal processing for 5G Adv/6G wireless communications, integrated sensing and communications, reconfigurable intelligent and holographic surfaces, energy efficient communications, millimeter wave massive MIMO and satellite communication networks. He is a member of the IEEE, IEEE ComSoc and the IET. He serves as an Associate Editor in the Editorial Boards of the IEEE Open Journal of the Communications Society, IEEE Communications Letters, IET Signal Processing and IET Networks. He has been the Lead Guest Editor for Special Issues on reconfigurable intelligent and holographic surfaces topics in the IEEE Open Journal of the Communications Society and IET Signal Processing, Tutorial Speaker at IEEE WCNC 2023, Track Chair for the Backhaul/Fronthaul Networking and Communications SAC symposium at IEEE ICC 2024 and Track Co-Chair for the Emerging Technologies, Standards and Applications track at IEEE WCNC 2023. He has also been serving as General Chair for IEEE international workshops such as IEEE WCNC 2023, IEEE PIMRC 2022 and IEEE SECON 2022, TPC Member at IEEE ICC 2021-23 including RISSE SAC (ICC 2023), and Conference Champion for IEEE PIMRC 2020.

Shuai Yuan received the B.Sc. degree from the College of Information Science and Engineering, Ocean University of China, Qingdao, China, in 2020. He is currently pursuing the Master's degree at Beihang University, Beijing. His current research interests include fault diagnosis and computing offloading in the Internet of vehicles.

Wenquan Feng is a professor in the School of Electronic and Information Engineering, Beihang University, Beijing, China. He received a Ph.D. degree in communication and information system from Beihang University. He has been teaching as the dean of studies at Beihang University since 2011. His current research interests include signal processing, deep learning, and fault diagnosis.