Double Deep Q-Network based Dynamic Framing Offloading in Vehicular Edge Computing

Huijun Tang, Huaming Wu, *Senior Member, IEEE*, Guanjin Qu, and Ruidong Li, *Senior Member, IEEE*

***Abstract*—With the rapid development of Artificial Intelligence (AI) and the Internet of Vehicles (IoV), there is an increasing demand for deploying various intelligent applications on vehicles. Vehicular Edge Computing (VEC) is receiving extensive attention from both the industry and academia due to its benefits from the edge computing paradigm, which pushes computing tasks from the core of the network to the edge of the network. However, in the VEC environment considering vehicles to Road Side Units (RSUs), due to the mobility of vehicles, it is still a challenge to make dynamic and efficient offloading decisions for compute-intensive tasks, especially in the congestion situation. In order to minimize the total delay and waiting time of tasks from moving vehicles, we establish a dynamic offloading model for multiple moving vehicles whose tasks can be divided into sequential subtasks, so that the offloading decisions are more refined. Moreover, the proposed model is frame-based to avoid unnecessary waiting time, which makes offloading decisions when the subtasks of each vehicle are generated rather than offloading subtasks after gathering subtasks of vehicles for a time slot. Aiming to find the optimal offloading decision for sequential subtasks, we propose a Dynamic Framing Offloading algorithm based on Double Deep Q-Network (DFO-DDQN). Extensive experimental results demonstrate the effectiveness and superiority of the proposed DFO-DDQN when compared with other DRL-based methods and greedy-based methods.**

***Index Terms*—Vehicular Edge Computing, Internet of Vehicles, Task Offloading, Deep Reinforcement Learning**

I. Introduction

ALONG with the fast development of Internet of Things (IoT) and communication technologies, the number of in-vehicle applications, e.g., online gaming, Augmented Reality (AR), and Deep Neural Network (DNN)-based intelligent applications [1], [2], has risen significantly in recent years. The upcoming sixth-generation (6G) wireless communication brings together various enabling technologies and opens a new era of 'Internet of Intelligence'. Unfortunately, there is still a contradiction between the huge demand to perform delay-sensitive and compute-intensive tasks and the severely constrained computing resources of vehicles [3]. In the meanwhile, with the continuous improvement of 6G networks and edge computing paradigm, these tasks can be either performed in vehicles, roadside infrastructure, or the cloud, which makes it possible to make full use of the computing resources in Vehicular Edge Computing (VEC) environments.

H. Tang, H. Wu and G. Qu are with the Center for Applied Mathematics, Tianjin University, Tianjin 300072, China. E-mail: {tanghuijune, whming, guanjinqu}@tju.edu.cn

R. Li is with the Institute of Science and Engineering, Kanazawa University, Kanazawa 920-1192, Japan (e-mail: liruidong@ieee.org).

(Corresponding author: Huaming Wu)

Many efforts have been devoted to task offloading in Mobile Edge Computing (MEC) environments, where tasks can be divided into several subtasks that can be executed locally or be partially offloaded to other devices [4]–[7]. In order to perform better task offloading under poor wireless channel conditions, partial offloading is believed to be more suitable for tasks with stringent latency requirements, rather than binary offloading whose tasks will be completely offloaded or not in edge computing [8], [9]. Compared with the MEC environment, the conditions of VEC environments are even more complicated due to the mobility of vehicles, especially for compute-intensive tasks which need more execution time. To simplify the offloading-decision problems, numerous studies [10]–[13] have divided the road into several service sections of RSUs that do not intersect and assume vehicles in the coverage of a certain RSU can only offload tasks to the edge server of this RSU. This is called whether-to-offload setting of RSUs, where the offloading decision is whether to offload or not instead of where to offload to. However, the whether-to-offload setting of edge servers ignores the computing resources of other available edge servers and may cause more waiting time when there is traffic congestion in the coverage of the current RSU. Unlike the where-to-offload-to setting, vehicles can choose multiple edge servers.

In addition, people are in great demand of Artificial Intelligence (AI), which is pervading every aspect of life and traffic is no exception. For example, the amount of computation required per task for computing a 1024×768 image can reach 2,640 cycles in [20], which means that executing AR tasks requires a large amount of computation. However, executing compute-intensive tasks in VEC environment with high speeds is still a great challenge. It is important to note that certain compute-intensive tasks such as Apple ARkit [21] and Google ARCore [22], which are well-modularized software development kits that can split tasks into multiple independent subtasks, which can be processed in sequence. Considering the limitation of the computing abilities of local devices, such compute-intensive tasks are usually partially offloaded to edge or cloud servers. However, dynamic offloading of tasks with sequential dependencies in a VEC environment considering both Vehicle-to-RSU (V2R) and Vehicle-to-Network (V2N) remains unresolved. There are few studies discussing the offloading of sequential subtasks in the VEC environment, which is important for applying edge computing to compute-intensive tasks in the VEC environment. As shown in Table I, due to the consideration of resource competition and decision-making cooperation between different vehicles, the previous VEC models were built in the form of time slots rather than

TABLE I: The Qualitative Comparison of the Current Literature on Deep Learning-based VEC Schemes.

Approaches	Partial Offloading	Multiple Vehicles	Dynamic Offloading	Task Dependency	Infrastructures Intercommunication	Granularity
Huang *et al.* [14]	✗	✓	✓	✗	✗	Slot-based
Huang *et al.* [15]	✗	✗	✗	✗	✗	Slot-based
Tang *et al.* [16]	✓	✓	✓	✓	✗	Slot-based
Wu *et al.* [17]	✗	✗	✗	✗	✓	Slot-based
Luo *et al.* [18]	✗	✓	✗	✗	✗	Slot-based
Ke *et al.* [19]	✓	✗	✗	✗	✗	Slot-based
Ours	✓	✓	✓	✓	✓	Frame-based

in the form of frames, where different frames correspond to different timestamps. Slot-based offloading decisions are more coarse-grained than frame-based offloading decisions.

In this paper, we propose a novel dynamic offloading-decision algorithm based on DDQN to optimize the total delay of subtasks with sequential dependency, including the waiting time caused by the subtasks congestion when the number of subtasks received by the same edge server increases sharply. The key contributions of this paper are summarized as follows:

- *Where-to-Offload-to Setting*: In this paper, the subtask of vehicles can choose different edge servers to offload instead of choosing the edge server by the location of vehicles and the coverage of RSUs.
- *Sequential Subtasks Offloading*: We divide compute-intensive tasks into several sequential subtasks, which means the latter subtask is generated after completing the previous subtask of the same vehicle, and the results of the last subtask need to be transmitted to vehicles. Therefore, sequential subtasks can be executed in different edge servers to cope with the uncertainty of communication conditions caused by the mobility of vehicles and optimize the waiting time caused by the limited resources in a traffic congestion environment.
- *Frame-based Offloading*: To further find the optimal space of offloading decision-making, we build the VEC model by frame-based form, so that the VEC system can make decisions immediately once subtasks of vehicles are generated, instead of being aggregated into a set of subtasks in a time slot. To the best of our knowledge, this work is the first one to explore the frame-based offloading in VEC environments.
- *Algorithm Design*: We design a novel Dynamic Framing Offloading algorithm based on Double Deep Q-Network (DFO-DDQN) to solve the MDP problems and further optimize the time delay of tasks of the multiple moving vehicles in the VEC system. Based on simulation experiments conducted under different VEC environmental conditions, our DFO-DDQN outperforms the greedy-based method without task segmentation (NoSeg-Greedy) by over 46.44% at least.

The remainder of this paper is organized as follows. Related works are provided in Section II. The system model and problem formulation are provided in Section III. The proposed algorithm is presented in Section IV. Extensive simulation experiments are conducted and discussed in Section V. Section VI concludes this paper and points out future directions.

II. Related work

In recent years, with the rapid development of AI and autonomous driving technology, a number of studies focus on offloading tasks in VEC environments. Huang *et al.* [12] proposed a Lyapunov-based dynamic offloading algorithm, which considers the uplink transmission from vehicles to Road Side Units (RSUs) to optimize the tradeoff between energy consumption, packet drop rate, and queue stability. Wang *et al.* [10] proposed a dynamic offloading algorithm considering limited resources and variable speeds of the vehicle for MEC-enabled vehicular networks. However, the shortcomings of the offloading methods based on classic methods are slow calculation and weak generalization ability.

Apart from conventional offloading-decision approaches, deep learning-based methods such as Federal Learning (FL) [23], [24], Deep Imitation Learning (DIL) [25], [26], Deep Reinforcement Learning (DRL) [14], [16]–[19], [27], [28], Multi-agent Learning [29], [30] and Deep Meta-Learning (DML) [15], [31], [32] have been widely applied to cope with the challenges of dynamic offloading decision-making in VEC environments. As far as we know, the offloading decision-making process of sequential subtasks can be regarded as a Markov Decision Process (MDP), where the next state is only related to the current state. Thus, DRL is particularly suitable for solving MDP in a complex interactive environment, where DNNs need to learn how to represent a complicated relationship (or policy π) between the state s and the action a. After inputting the current state, the network generates an action that causes the environment to generate a new state and a reward value of r, as a feedback for adjusting the network. Deep Q-Network (DQN) [33] is a classic DRL method that is widely applied in the field of human-computer interaction. Wu *et al.* [17] considered a VEC environment, where the RSU can be switched to a sleeping or working state and proposed a DQN-based method to find the optimal offloading decision to minimize the total delay of tasks. However, DQN still suffers from Q-value overestimation. To solve the overestimate problem, Double Deep Q-Network (DDQN) is further proposed in [34] by choosing actions by MainNet parameters instead of TargetNet parameters. In addition, Huang *et al.* [14] proposed a speed-aware offloading algorithm based on Deep Deterministic Policy Gradient (DDPG) [35] to minimize the energy cost within delay constraints. Wu *et al.* [17] proposed a DQN based method to learn the optimal scheduling policy for minimizing the total delay of tasks. Luo *et al.* [18] propose a collaborative data scheduling scheme based on DQN to minimize the processing cost with ensured delay constraints

of applications. The aim of DRL-based methods is to achieve long-term benefits than greedy-based methods, since greedy-based methods always choose the current optimal solution.

Unfortunately, the aforementioned studies ignore the dependencies between subtasks, which are generally required to be considered in AR, video stream, and other DNN-based applications. For example, we need to perform data acquisition, image rendering, encoding, transmitting, decoding, and display in a virtual reality system [36], whether to consider the order of tasks has a great impact on the offloading decision and the overall execution time. Tang *et al.* [16] utilized several service vehicles to provide computing services and designed a DQN-based algorithm to offload subtasks that have order dependencies. However, it still ignores the computing services of edge servers and roadside infrastructure. In addition, due to the mobility of the vehicle, the transmission rate will vary with the distance between the devices, allowing more room for optimization during partial offloading, which is much more efficient than binary offloading in meeting stringent Quality of Service (QoS) requirements.

III. System Model and Problem Formulation

In this paper, we consider a VEC system model as depicted in Fig. 1, which consists of multiple vehicles (blue, green, and orange represent different vehicles) and multiple RSUs, and the tasks of vehicles are divided into several sequential subtasks.

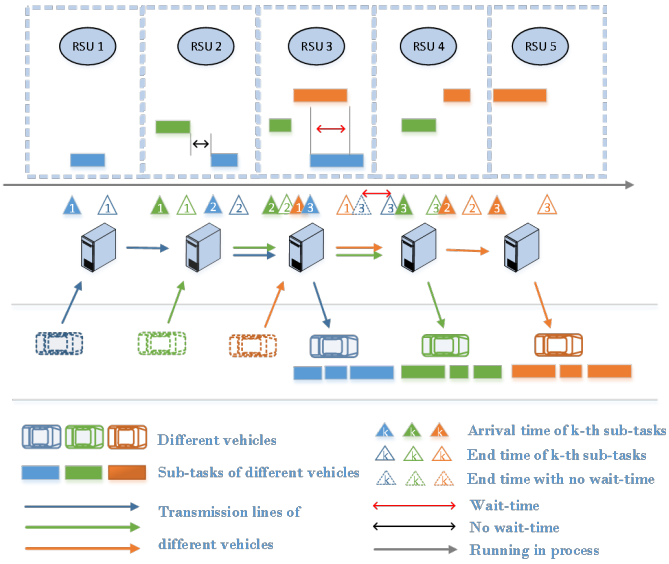


Fig. 1: An illustration of our VEC system model.

Once a task is generated in a vehicle, it is common to be executed on the vehicle or executed on edge servers in RSUs in previous studies. However, when the task needs to occupy a large number of computing resources, executing on the vehicle will occupy computing resources of the vehicle for a long time, which may cause a safety hazard, while executing on edge servers in RSUs will fail into two situations. One is the congestion situation, where multiple vehicles may offload tasks to the same edge server and some tasks need to wait for a while because of the congestion. The other is a high-speed situation, where the vehicle may move out of the receiving range with a high transmission rate during execution and transmission, causing downlink transmission to fail. To cope with the aforementioned challenges, we segment each task into several sequential subtasks and execute them on different devices. The task of the m-th vehicle is divided to K parts $\Phi_m = \{\phi_{m1}, \phi_{m,1}, \cdots, \phi_{m,K}\}$, where Φ_m is the subtasks set of the m-th vehicle, $m = 1, 2, \cdots, M$ and the subtask $\phi_{m,k+1}$ is generated on the l-th edge server when the subtask $\phi_{m,k}$ is completed on the l-th edge server, where $l = 1, 2, \cdots, L$ and $k = 1, 2, \cdots, K$. The major notations used in this paper are defined in Table II.

In the case when the edge server is not occupied when the subtask arrives at it, the subtask can be executed immediately and there is no need to wait (black double arrow in Fig. 1). Otherwise, if the edge server is occupied by subtasks of other vehicles when the subtask arrives, the subtask needs to wait for other subtasks (red arrow in Fig. 1). Solid triangle k of different colors corresponds to the arrival time of the k-th subtask of different vehicles, dotted hollow triangle k of different colors corresponds to the end time of k-th subtask of different vehicles without considering waiting time, and hollow triangle k in different colors represents the actual end time of the k-th subtask of different vehicles, which is also the generation time of the $k+1$-th subtask. The transmission time can be regarded as the gray line between the hollow triangle of k-th and the solid triangle of $k+1$-th. The gray line represents the transmission and execution process of the VEC system, and it only can represent chronological order instead of the length of time. When the tasks that can be divided into several sequential subtasks are partially offloaded, their subtasks can be executed on different edge servers, so that we can make more refined offloading decisions to cope with the mobility of vehicles and traffic congestion. By doing this, we can get less time delay and waiting time compared with offloading the whole task to the edge. The last subtasks need to be transmitted back to vehicles, whose amount of computation can be 0 or not. The time delay of our VEC system consists of three parts, i.e., execution time, transmission time, and waiting time.

A. Execution Time

We denote $t^E_{m,k,l}$ as the execution time when the current to-do subtask $\phi_{m,k}$ of vehicle m is executed on the edge server l in RSU_l, which can be expressed as:

$$t^E_{m,k,l} = \frac{w_{m,k}}{f_l}, \tag{1}$$

where $w_{m,k}$ is the amount of computation of the subtask $\phi_{m,k}$, and f_l (cycle/s) is the computing capability of the edge server l in RSU_l, where $l = 1, 2, \cdots, L$ and $k = 1, 2, \cdots, K$. We denote $t^E_{m,k,0}$ as the execution time when the subtask $\phi_{m,k}$ is executed on vehicle m, which is calculated by:

$$t^E_{m,k,0} = \frac{w_{m,k}}{f_m}, \tag{2}$$

TABLE II: Notations and their definitions.

Notations	Definitions	Notations	Definitions
M	The number of vehicles	K	The number of subtasks of each vehicle
L	The number of the edge servers	v	The speed of vehicles
Φ_m	The subtasks set of vehicle m	$\phi_{m,k}$	The k-th subtasks of vehicle m
w	The amount of computation	D	The data size
P	The transmission power	h	The channel fading coefficient
ϑ	The path loss exponent	ω_0	The white Gaussian noise power
$B_{m,l}$	The bandwidth from vehicle m to edge server l	$d_{m,l}$	The distance between vehicle m to edge server l
$B_{l',l}$	The bandwidth from edge server l' to l	$d_{l',l}$	The distance between edge server l' to l
$B_{l,m}$	The bandwidth from edge server l to vehicle m	$d_{l,m}$	The distance between edge server l to vehicle m
f_l	The computing capability of the edge server l	f_m	The computing capability of the vehicle m
$t^E_{m,k,l}$	The execution time of subtask $\phi_{m,k}$ on edge server l	$t^E_{m,k,0}$	The execution time of subtask $\phi_{m,k}$ on vehicle m
$I^E_{m,k,l}$	The indicators describing whether $\phi_{m,k}$ is executed on device l	$T^W_{m,k,l}$	The waiting time before $\phi_{m,k}$ executed on edge server l
$t^{Tr}_{m,k,l}$	The transmission time from vehicle m to edge server l	$t^{Tr}_{m,l',k,l}$	The transmission time of $\phi_{m,k}$ from edge server l' to l
$t^{Tr}_{l,k,m}$	The transmission time from edge server l to vehicle m	$t^{Tr}_{m,k}$	The transmission time of subtask $\phi_{m,k}$
$R_{m,k,l}$	The transmission rate from vehicle m to edge server l	$R_{l',k,l}$	The transmission rate from edge server l' to l
$R_{l,k,m}$	The transmission rate from edge server l to vehicle m	$T^{End}_{m,k}$	The end time of subtask $\phi_{m,k}$
$I^{Tr}_{m,k,l}$	The indicators describing whether $\phi_{m,k}$ is transmitted from vehicle to edge server l	$I^{Tr}_{l',k,l}$	The indicators describing whether $\phi_{m,k}$ is transmitted from edge server l' to edge server l
$I^{Tr}_{l',k,l}$	The indicators describing whether $\phi_{m,k}$ is transmitted from edge server l' to vehicle m	$T^{Occ}_{m,k,l}$	The value describing the occupied time of the edge server l when subtask $\phi_{m,k}$ arrives at RSU_j
$CD_{m,k}$	The indicator describing which devices that $\phi_{m,k}$ executed on	T^{Sum}	The sum of the end time of all vehicles in the VEC system
$T^A_{m,k,l}$	The arrival time of subtask $\phi_{m,k}$	T^{Start}_m	The start time of subtask $\phi_{m,1}$
$T^E_{m,k}$	The execution time of $\phi_{m,k}$	$T^W_{m,k}$	The waiting time of subtask $\phi_{m,k}$
$s(t_p)$	The state when the current frame is t_p	$T^G_{m,k}$	The generation time of subtask $\phi_{m,k}$
t_p	The p-th frame	x_{m,t_p}	The location of vehicle m when the current frame is t_p
$\phi(t_p)$	The subtask which should be offloaded in frame t_p	$a(t_p)$	The action when the current frame is t_p
$\triangle T_{\phi(t_p)}$	The time required when the subtask $\phi(t_p)$ is executed		

where f_m (cycle/s) is the computing capability of the vehicle m, and $m = 1, 2, \cdots, M$. For subtask $\phi_{m,k}$, the execution time $T^E_{m,k}$ is:

$$T^E_{m,k} = \sum_{l=0}^{L} I^E_{m,k,l} t^E_{m,k,l}, \tag{3}$$

where $I^E_{m,k,l} \in \{0, 1\}$ is an indicator value that describes whether $\phi_{m,k}$ is executed on the device l ($l = 0, 1, \cdots, L$). For instance, $I^E_{m,k,0} = 1$ means the subtask $\phi_{m,k}$ is executed on vehicles, otherwise, it is offloaded. In addition, we have $\sum_{l=0}^{L} I^E_{m,k,l} = 1$.

B. Transmission Time

There are three types of transmission time, namely, $t^{Tr}_{m,k,l}$, $t^{Tr}_{m,l',k,l}$ and $t^{Tr}_{l,k,m}$, where $t^{Tr}_{m,k,l}$ is the transmission time from vehicle m to the edge server l in RSU_l, $t^{Tr}_{m,l',k,l}$ is the transmission time of $\phi_{m,k}$ from the edge server l' in $RSU_{l'}$ to the l-th edge server in RSU_l, and $t^{Tr}_{l,k,m}$ is the transmission time from the edge server l to the vehicle m. Moreover, we assume that the channel is a frequency-flat block-fading Rayleigh channel [37].

1) Vehicle m to RSU_l: When the subtask $\phi_{m,k}$ is transmitted from the vehicle m to the RSU_l, the transmission rate $R_{m,k,l}$ is:

$$R_{m,k,l} = B_{m,l} \log_2 \Big(1 + \frac{P|h|^2}{\omega_0 (d_{m,l})^{\vartheta}}\Big), \tag{4}$$

where $B_{m,l}$ and $d_{m,l}$ are the bandwidth and the distance from vehicle m to RSU_l, respectively. P is the transmission power, h is the channel fading coefficient, ϑ is the path loss exponent, and ω_0 is the white Gaussian noise power. The transmission delay $t^{Tr}_{m,k,l}$ from vehicle m and the RSU_l is calculated by:

$$t^{Tr}_{m,k,l} = \frac{D_{m,k}}{R_{m,k,l}}, \tag{5}$$

where $D_{m,k}$ is the data size of $\phi_{m,k}$.

Furthermore, we define $CD_{m,k}$ as an indicator that describes on which device the subtask $\phi_{m,k}$ is executed. For example, $CD_{m,k} = l$ means $\phi_{m,k}$ is executed on the edge server l.

2) $RSU_{l'}$ to RSU_l: When the subtask $\phi_{m,k}$ is transmitted from the edge server l' in $RSU_{l'}$ to the edge server l in RSU_l, which means $CD_{m,k-1} = l'$ and $CD_{m,k} = l$, the transmission rate $R_{l',k,l}$ is:

$$R_{l',k,l} = B_{l',l} \log_2 \Big(1 + \frac{P|h|^2}{\omega_0 (d_{l',l})^{\vartheta}}\Big), \tag{6}$$

where $B_{l',l}$ and $d_{l',l}$ are the bandwidth and the distance from the edge server l' in $RSU_{l'}$ to the edge server l in RSU_l, respectively. The transmission delay $t^{Tr}_{l',k,l}$ from $RSU_{l'}$ to RSU_l is calculated by:

$$t^{Tr}_{m,l',k,l} = \frac{D_{m,k}}{R_{l',k,l}}. \tag{7}$$

3) RSU_l to Vehicle m: When the subtask $\phi_{m,k}$ is transmitted from the edge server l in RSU_l to vehicle m, which means $CD_{m,k-1} = l$ and $CD_{m,k} = 0$, the transmission rate $R_{l,k,m}$ is calculated by:

$$R_{l,k,m} = B_{l,m} \log_2 \Big(1 + \frac{P|h|^2}{\omega_0 (d_{l,m})^{\vartheta}}\Big), \tag{8}$$

where $B_{l,m}$ and $d_{l,m}$ are the bandwidth and the distance from the edge server l in RSU_l to vehicle m, respectively.

The transmission delay $t^{Tr}_{l,k,m}$ from RSU_l to vehicle m is calculated by:

$$t^{Tr}_{l,k,m} = \frac{D_{m,k}}{R_{l,k,m}}. \tag{9}$$

For subtask $\phi_{m,k}$, the transmission time $T^{Tr}_{m,k}$ is:

$$T^{Tr}_{m,k} = \begin{cases} \sum_{l=1}^{L} I^{Tr}_{m,k,l} t^{Tr}_{m,k,l}, & \text{if } CD_{m,k-1} = 0 \\ \sum_{l=1}^{L} I^{Tr}_{l',k,l} t^{Tr}_{m,l',k,l} + I^{Tr}_{l',k,m} t^{Tr}_{l',k,m}, & \text{if } CD_{m,k-1} = l' \end{cases} \tag{10}$$

where $I^{Tr}_{m,k,l}$ and $I^{Tr}_{l',k,l}$ are the indicator values that both describe whether $\phi_{m,k}$ is execute on the l-th device ($l = 1, 2, \cdots, L$), $I^{Tr}_{l',k,m} = 1$ means the subtask $\phi_{m,k}$ is executed on vehicles. $\sum_{l=0}^{L} I^{Tr}_{m,k,l} + \sum_{l=0}^{L} I^{Tr}_{l',k,l} + I^{Tr}_{l',k,m} = 1$ means that the subtask $\phi_{m,k}$ can only be executed on one device and $I^{Tr}_{m,k,l}, I^{Tr}_{l',k,l}, I^{Tr}_{l',k,m} \in \{0, 1\}$. $I^{Tr}_{m,k}$ is the transmission decision indicator of subtasks $\phi_{m,k}$.

C. Waiting Time

Vehicles may offload subtasks to the same edge server, especially when vehicles drive on a low speed-limit road or meet the traffic congestion situation so that the waiting time is an inevitable part when the objective is optimizing the total delay of the VEC system. The order of executing subtasks that are offloaded to the same edge server operates on a first-come-first-serve rule, which means that the execution of the current subtask needs to wait until the end of the earlier arriving subtask.

When the subtask $\phi_{m,k}$ is chosen to be executed on the edge server l, the transmission time is:

$$T^{Tr}_{m,k,l} = \sum_{l'=0}^{L} I^{Tr}_{l',k,l} t^{Tr}_{m,l',k,l} + I^{Tr}_{m,k,l} t^{Tr}_{m,k,l}. \tag{11}$$

We denote $T^{Occ}_{m,k,l}$ as the occupied time of the edge server l when the subtask $\phi_{m,k}$ arrive at RSU_j and executed on the edge server l, which is calculated by:

$$T^{Occ}_{m,k,l} = \max\left\{T^{End}_{m',k'}, T^{End}_{m,k}\right\}, \phi_{m',k'} \in \Phi^{-1}_{m,k,l}, \forall k \geq 2, \tag{12}$$

where $\Phi^{-1}_{m,k,l}$ is the set of subtasks that are executed on RSU_l and the offloading decision is made before $\phi_{m,k}$.

The generation time of the whole tasks for vehicles are different, that is, the start time of subtasks, which is denoted as T^{Start}_m. The end time of the subtask $\phi_{m,k-1} (k \geq 2)$ is the sum of four parts, which is defined as follows:

$$\begin{aligned} T^{End}_{m,k-1} &= T^{Start}_m + \sum_{j=1}^{k-1} \triangle T_{m,j} \\ &= T^{Start}_m + \sum_{j=1}^{k-1} T^{E}_{m,j} + \sum_{j=1}^{k-1} T^{Tr}_{m,j} + \sum_{j=1}^{k-1} T^{W}_{m,j}, \end{aligned} \tag{13}$$

where $T^{W}_{m,j}$ is the waiting time of the subtask $\phi_{m,j}$ and $\triangle T_{m,j}$ is the total time of subtask $\phi_{m,j}$, which is equal to the sum of the execution time $T^{E}_{m,j}$, the transition time $T^{Tr}_{m,j}$, and the waiting time $T^{W}_{m,j}$.

The arrival time $T^{A}_{m,k,l}$ of the subtask $\phi_{m,k}$ arriving at RSU_l is calculated by:

$$T^{A}_{m,k,l} = T^{End}_{m,k-1} + T^{Tr}_{m,k,l}. \tag{14}$$

When some subtasks arrive at the edge server l earlier than subtask $\phi_{m,k}$, then the waiting time $T^{W}_{m,k,l}$ is:

$$T^{W}_{m,k,l} = \begin{cases} T^{Occ}_{m,k,l} - T^{A}_{m,k,l} & \text{if } T^{Occ}_{m,k,l} > T^{A}_{m,k,l} \\ 0 & \text{else} \end{cases} \tag{15}$$

Therefore, the waiting time of $\phi_{m,k}$ can be calculated as:

$$T^{W}_{m,k} = \sum_{l=1}^{L} I^{E}_{m,k,l} T^{W}_{m,k,l}. \tag{16}$$

where $I^{E}_{m,k,l}$ is the indicator value describing whether $\phi_{m,k}$ is executed on the edge server l.

D. Problem Formulation

The total delay of the task of vehicle m is $T^{End}_{m,K}$, and the total delay of the VEC system is:

$$T^{Sum} = \sum_{m=1}^{M} T^{End}_{m,K}. \tag{17}$$

To avoid occupying the limited computing resource of vehicles and taking too much transmission time between the edge server and vehicles, we offload almost all subtasks to the edge server except the last subtasks that should be transmitted back to vehicles, which means $I^{Tr}_{l',K,m} = 1$ for vehicle m when $CD_{m,K-1} = l'$. The last subtask may include result data or even a small amount of computation that needs to be executed in the vehicle. Accordingly, the dynamic offloading problem in the VEC environment with sequential subtask dependency is formulated as:

$$(\boldsymbol{\mathcal{P}_1}) \min_{I^{E}_{m,k,l}, I^{Tr}_{m,k}} : T^{Sum} = \sum_{m=1}^{M} T^{End}_{m,K}, \tag{18}$$

$$\text{s.t.} : \sum_{l=0}^{L} I^{E}_{m,k,l} = 1, \tag{19}$$

$$\sum_{l=0}^{L} I^{Tr}_{m,k,l} + \sum_{l=0}^{L} I^{Tr}_{l',k,l} + I^{Tr}_{l',k,m} = 1 \tag{20}$$

$$I^{E}_{m,k,l}, I^{Tr}_{m,k,l}, I^{Tr}_{l',k,l}, I^{Tr}_{l',k,m} \in \{0, 1\} \tag{21}$$

$$I^{Tr}_{l',K,m} = 1 \tag{22}$$

where Eq. 19 indicates that $\phi_{m,k}$ can only be executed on one device, Eq. 20 indicates that $\phi_{m,k}$ can only be transmitted to one device, and Eq. 22 indicates that the last subtask need to be transmitted back to vehicles.

IV. Dynamic Framing Offloading Algorithm based on Double-DQN

In this part, we make offloading decisions in the order of the generation time of each subtask of multiple vehicles. The generation time $T^{G}_{m,k}$ of subtasks $\phi_{m,k}$ is calculated as:

$$T^{G}_{m,k} = T^{End}_{m,k-1}, \quad \forall k \geq 2. \tag{23}$$

As shown in Fig. 2, the process of making the offloading decision of whole tasks in the current road is divided into different offloading frames t_p of subtasks, where $p = 1, 2, \cdots, MK$.

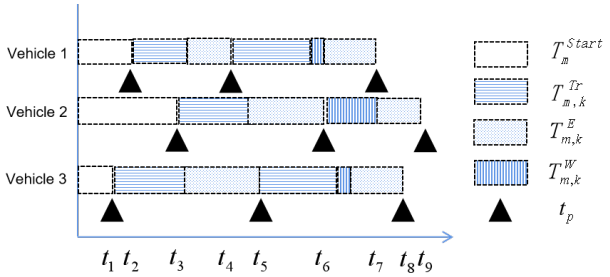


Fig. 2: An illustration of the frame-based offloading process. The execution time is necessary for a subtask, while the waiting time and transmitting time may be zero.

The proposed Dynamic Framing Offloading algorithm based on Double Deep Q-Network (DFO-DDQN) is as shown in Fig. 3. The state s, action a, and reward r are set as follows:

1) State Space: The state in our model consists of the following two parts, namely, the information of subtasks and the environmental information.

- **Subtask Feature (SF)**: It includes the amount of computation w and the data size D of subtasks. After the current subtask corresponding to the current t_p is executed, the locations of the subtask feature with $w_{m,k}$ and $D_{m,k}$ are updated with zeros, and the new subtask feature is generated. It is worth mentioning that the number of subtasks of a vehicle can be less than K and the empty position is filled up with zero. The p-th SF state $SF(t_p)$ is as shown in Eq. 24.

$$SF(t_p) = \begin{pmatrix} sf_1^p, & sf_2^p, & \cdots, & sf_M^p \end{pmatrix} = \begin{pmatrix} w_{1,1}^p & w_{2,1}^p & \cdots & w_{M,1}^p \\ D_{1,1}^p & D_{2,1}^p & \cdots & D_{M,1}^p \\ w_{1,2}^p & w_{2,2}^p & \cdots & w_{M,2}^p \\ D_{1,2}^p & D_{2,2}^p & \cdots & D_{M,2}^p \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,K}^p & w_{2,K}^p & \cdots & w_{M,K}^p \\ D_{1,K}^p & D_{2,K}^p & \cdots & D_{M,K}^p \end{pmatrix} \quad (24)$$

where $sf_m^p = [w_{m,1}^p, D_{m,1}^p, w_{m,2}^p, D_{m,2}^p, \cdots, w_{m,K}^p, D_{m,K}^p]^T$ is the subtask information set of m-th vehicle in frame t_p, and $w_{m,k}^p$ and $D_{m,k}^p$ are the amounts of computation and the data size of subtask $\phi_{m,k}$ when in frame t_p, which are computed by Eq. 25 and Eq. 26, respectively.

$$w_{m,k}^p = \begin{cases} 0, & \text{if } \phi_{m,k} \text{ has been executed before } t_p \\ w_{m,k}, & \text{else} \end{cases} \quad (25)$$

$$D_{m,k}^p = \begin{cases} 0, & \text{if } \phi_{m,k} \text{ has been executed before } t_p \\ D_{m,k}, & \text{else} \end{cases} \quad (26)$$

- **Vehicle Feature (VF)**: It includes four aspects of information, namely, the start time of the whole tasks of vehicles, the computing ability of vehicles, the speed of vehicles, and the location information of vehicles in current t_p. After the subtask $\phi_{m,k}$ is executed, the environmental characteristics are updated as the situation changes, e.g., the location information of the vehicle m changes. The p-th VF state $VF(t_p)$ is as shown in Eq. 27.

$$VF(t_p) = \begin{pmatrix} vf_1^p, & vf_2^p, & \cdots, & vf_M^p \end{pmatrix} = \begin{pmatrix} T_1^{Start} & T_2^{Start} & \cdots & T_M^{Start} \\ f_1 & f_2 & \cdots & f_M \\ v_1 & v_2 & \cdots & v_M \\ x_1^p & x_2^p & \cdots & x_M^p \end{pmatrix} \quad (27)$$

where $vf_m^p = [T_m^{Start}, f_m, v_m, x_m^p]^T$ is the vehicle information set of m-th vehicle in frame t_p. x_m^p is the distance from the starting point of the road of m-th vehicle when in frame t_p, which can be calculated by T_m^{Start}, v_m, and T^{End} or obtained from the monitoring system of the reality.

The state in frame t_p is as Eq. 28:

$$s(t_p) = [sf_1^{pT}, vf_1^{pT}, sf_2^{pT}, vf_2^{pT}, \cdots, sf_M^{pT}, vf_M^{pT}]^T. \quad (28)$$

2) Action Space: In t_p frame, we need to compare which subtask among the current to-do subtasks is the earliest generated one, denoted as $\phi(t_p) = \arg\min_m T_m^{End,p}$, where $T_m^{End,p}$ is the generation time of the current to-do subtasks of m-th vehicle in frame t_p, which is calculated by Eq. 29:

$$T_{m,k_p}^{End,p} = \begin{cases} T_m^{Start}, & k_p = 0 \\ T_{m,k_p}^{End}, & k_p \geq 1 \end{cases} \quad (29)$$

where (m, k_p) is the index tuple of the current to-do subtask of m-th vehicle in frame t_p.

The action space is $\mathscr{A} = [a_1, a_2, \cdots, a_L]$, each of whose element corresponds an edge server, where $a_l \in \{0, 1\}$ and $\sum_l^L a_l = 1$, $l = 1, 2, \cdots, L$. $\phi(t_p) = \phi_{m,k}$ in frame t_p means the current action choice $a(t_p)$ is the offloading decision of the k-th subtask of the m-th vehicle, where $a(t_p)$ is the action chosen in t_p frame and $a(t_p) \in \mathscr{A}$. When the current subtask is $\phi_{m,K}$, the action is a_0, indicating that the last subtask will be transmitted back to the vehicle m.

3) Reward: We use $T_{m,k}^{End}$ to evaluate the action, which is the time delay required by the subtask $\phi_{m,k}$.

$$\begin{aligned} r(s(t_p), a(t_p)) &= \triangle T_{\phi(t_p)} \\ &= \min_m T_m^{End,p} - \min_m T_m^{End,p-1} \end{aligned} \quad (30)$$

where $p \geq 1$ and $\phi(t_p)$ are the subtasks that need to be offloaded when the frame is t_p. $\triangle T_{\phi(t_p)}$ is also equal to the sum of the execution time, the transmission time, and the waiting time of $\phi(t_p)$.

The Q-function of our model is updated as:

$$\hat{Q}_M(s(t_p), a(t_p)) = r(\triangle T_{\phi(t_p)}) + \gamma Q_T\Big(s(t_{p+1}), \arg\max_{a(t_{p+1})} Q_M(s(t_{p+1}), a(t_{p+1}))\Big), \quad (31)$$

where $s(t_p)$ and $a(t_p)$ are the state and the action of the current frame t_p, respectively; $s(t_{p+1})$ and $a(t_{p+1})$ are the

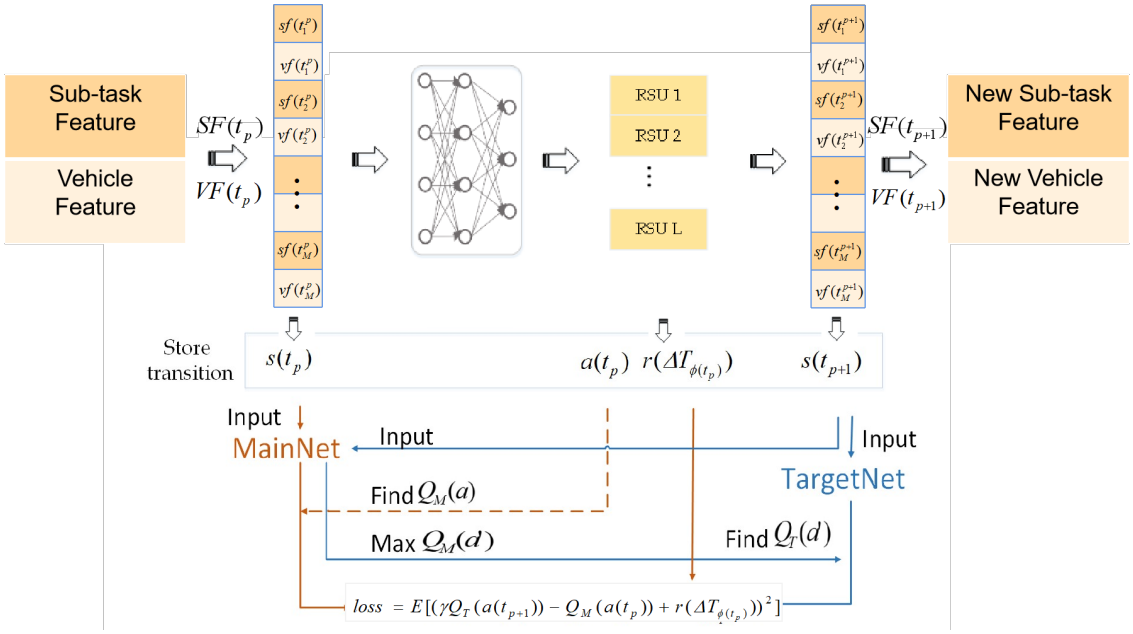


Fig. 3: An illustration of the proposed DFO-DDQN algorithm.

TABLE III: Parameter Settings.

Parameter	Value
v	$[10, 15, 20, 25]$ m/s
M	$[10, 15, 20, 25, 30]$
K	6
L	5
P	1.3
ϑ	2
ω_0	3×10^{-13}
h	4
ρ	40
w	$[0, 100] \times 10^8$ cycles
f_l	$[20, 10, 10, 20, 10]$ GHz
f_m	$[10, 50]$ MHz
B_{lm}, B_{ml}	5 MHz
$R_{l',k,l}$	1 Gbps
RSU location	$[20, 50, 80, 110, 140]$

state and the action of the next frame t_{p+1}, respectively; $Q_M(s(t_p), a(t_p))$ is the Q-value of the main network when choosing action $a(t_p)$ under state $s(t_p)$, $\hat{Q}_M(s(t_p), a(t_p))$ is the prediction of the Q-value of the main network when choosing action $a(t_p)$ under state $s(t_p)$ and $Q_T(s(t_{p+1}), a(t_{p+1}))$ is the Q-value of the target network when choosing action $a(t_{p+1})$ under state $s(t_{p+1})$. The detailed algorithmic process is as described in **Algorithm 1**.

V. Performance Evaluation

In this section, we conduct extensive experiments on the proposed algorithm under various parameter settings, and to verify its effectiveness, we further compare it with several existing offloading schemes in VEC environments.

A. Experimental Settings

The simulation settings are as shown in Table III, which mainly refer to [10]. We have $w = \rho D$, where ρ is the computational complexity. Regardless of whether the vehicle position changes or not, the transmission rate between different RSUs remains the same because the distance between different RSUs remains the same. We set $R_{l',k,l}$ to 1 Gbps, while the transmission rates $R_{m,k,l}$ and $R_{l,k,m}$ are calculated according to Eq. 4 and Eq. 8, respectively, since they are severely affected by the distance. We set the number of subtasks $K = 6$, the number of RSUs $L = 5$, and the speed of vehicle $v \in [10, 15, 20, 25]$ m/s. For training, we set the maximum training episode $Max_{iter} = 10,000$, the learning rate 0.01, the reward decay 0.9, the ϵ-greedy rate 0.98, the replace target iter 500, and the memory size 13,000.

B. Baselines

In order to validate the effectiveness of the proposed DFO-DDQN algorithm, we compare it with the other four offloading schemes in VEC environments, as shown below:

- **Dynamic Framing Offloading algorithm based on Deep Q-learning Network (DFO-DQN)**: As the earliest DRL frame, Deep Q-learning Network (DQN) [33] is widely used to solve Markov decision processes in complex environments. Similar to DFO-DDQN, the main difference lies in Q-value [33], i.e., $\hat{Q}_M(s, a) = r + \gamma \max_{a'} Q_T(s', a')$. The deep Q network of DFO-DQN consists of three layers: the first layer is an input layer with $M \times (2 \times K + 4)$, the following layer is the hidden layer with 20 nodes, and the last layer is the output layer with K nodes.
- **Dynamic Framing Offloading algorithm based on Dueling Deep Q-learning Network (DFO-DuelingDQN)**: Dueling Deep Q-learning Network (Dueling DQN) [38] are proposed based on DQN and aims to guide the network to distinguish the impacts of state and action on reward. Similar to DFO-DQN, we implement DFO-DuelingDQN through the algorithm proposed in [38]. Compared with DFO-DQN, the output layer of DFO-DuelingDQN has been improved, which is divided into

Algorithm 1 Dynamic framing Offloading algorithm based on DDQN (DFO-DDQN) for sequential subtasks in the VEC environment

Input: The initial subtask feature and vehicle feature
Output: optimal offloading decision for input

Initialize network parameters θ, update steps n, $\mathcal{D}$
for episode **do**
  $T_{m,0}^{End} = T_m^{Start}$, $t_p = 1$.
  while not all subtasks is executed **do**
    Get the earliest generation time of all subtasks that need to be executed in the current frame t_p, and the corresponding vehicle and its subtask are m and $\phi_{m,k}$, respectively, where $\phi_{m,k} = \phi(t_p)$.
    Calculate the current location $x(t_p)$ of vehicle m
    Input $s(t_p)$ to MainNet and get $Q_M(s(t_p), a_l)$, $l = 1, 2, \cdots, L$
    if $k = K$ **then**
      Choose $a(t_p) = a_0$
      $done = 1$
    else
      Choose $a(t_p) = \arg\max_a Q_M(s(t_p), a)$ according to $\epsilon-$greedy policy
      Get the occupied time $T_{m,k,a(t_p)}^{Occ}$ of $RSU_{a(t_p)}$ before executing the subtask $\phi(m,k)$ by Eq. 12
      Get the arrival time $T_{m,k,a(t_p)}^{A}$ of the current subtask $\phi(t_p)$, which is transmitted to $RSU_{a(t_p)}$ by Eq. 14
      Compute the waiting time $T_{m,k,a(t_p)}^{W}$ by Eq. 15
    end if
    Compute $T_{m,k}^{End}$
    Compute $r(\phi(t_p)) = \triangle T_{\phi(t_p)} = T_{m,k}^{End} - T_{m,k-1}^{End}$
    Generate the new state $s(t_{p+1})$
    Save $(s(t_p), a(t_p), r(t_p), s(t_{p+1}), done)$ in $\mathcal{D}$
    if Training Step **then**
      Sample memories from $\mathcal{D}$
      Input $s(t_p)$ to MainNet and get $Q_M(s(t_p), a(t_p))$
      Input $s(t_{p+1})$ to MainNet and get $a(t_{p+1}) = \arg\max_{a_l} Q_M(s(t_{p+1}), a_l)$
      Input $s(t_{p+1})$ to TargetNet and get $Q_T(s(t_{p+1}), a(t_{p+1}))$
      Compute the prediction Q-value by Eq. 31: $\hat{Q}_M(s,a) = r(t_p) + \gamma(1 - done)Q_T(s(t_{p+1}), a(t_{p+1}))$
      Update the parameter of the main network by minimizing $|Q_M(s,a) - \hat{Q}_M(s,a)|^2$
      Copy the parameter of the main network to the target network, every n steps
    end if
  end while
end for
return optimal offloading decision for input

two parts: one is the value of the state, called $V(s)$, and the other one is the advantages for each action, called $A(s,a)$. The Q-value of the target net is $Q_T(s,a) = V(s) + A(s,a)$ and $Q_M(s,a) = r + \gamma \max_{a'} Q_T(s', a')$.

- **Greedy algorithm with segmented tasks (Seg-Greedy)**: Greedy algorithm is a classic optimization method that chooses the best decision of each step and can get the local optimal solution of the system. First, we make offloading decisions in the order of the generation time $T_{m,k}^{G}$ of sub-tasks $\phi_{m,k}$ and compare the time cost $\triangle T_{\phi_{m,k}}$ among decisions executing on different RSUs of the current to-offload subtask, $m = 1, 2, \cdots, M, k = 1, 2, \cdots, K-1$. Then transfer the current subtask $\phi_{m,k}$ to the RSU with the lowest delay, where $l = \arg\min_l \triangle T_{m,k,l}$. The last subtask whose $k = K$ may include result data or even a small amount of computation so that it needs to be executed in the vehicle.
- **Greedy algorithm without segmented tasks (NoSeg-Greedy)**: The above methods are all based on segmented tasks and we need to compare our method to the task offloading method without task segmentation. We make offloading decisions in the order of the generation time of tasks and compare the time cost of the whole tasks of vehicles, including $\phi_{m,1}, \phi_{m,1}, \cdots, \phi_{m,K-1} (m = 1, 2, \cdots, M)$ among offloading decisions that executing on different RSUs. Then transfer the current to-offload the whole task to the RSU with the lowest delay and transfer the last task $\phi_{m,K}$ to local.

Among them, DFO-DDQN, DFO-DQN and DFO-DuelingDQN are three DRL-based methods, while Seg-Greedy and NoSeg-Greedy are two representative greedy-based methods.

C. Evaluation Indicators

- **Average Delay**: The average finish time of tasks of vehicles, which is as follows:

$$\overline{T} = \frac{T^{Sum}}{M} = \frac{\sum_{m=1}^{M} T_{m,K}^{End}}{M}. \tag{32}$$

- **Average Reward**: The average reward of the frames. The higher the value, the better the performance of the DRL-based method is.
- **Average Start Time**: The start time of the m-th vehicle is T_m^{Start}, which is the generation time of the task of vehicle m. The start time is entirely unrelated to different methods, and we simulate it by some start tasks ϕ^{Start} whose computations $w^{Start} = [w_1^{Start}, w_2^{Start}, \cdots, w_M^{Start}]$ is in $[0, 100] \times 10^8$ cycles. Then we can get T_m^{Start} as follows:

$$T_m^{Start} = \frac{w_m^{Start}}{f_m}. \tag{33}$$

- **Average Waiting Time**: The average waiting time is calculated as follows:

$$\overline{T}^W = \frac{1}{M} \sum_{m=1}^{M} \sum_{k=1}^{K} T_{m,k}^{W}. \tag{34}$$

- **Average Transmission Time**: The average transmission time is calculated as follows:

$$\overline{T}^{Tr} = \frac{1}{M}\sum_{m=1}^{M}\sum_{k=1}^{K} T_{m,k}^{Tr}. \tag{35}$$

- **Average Execution Time**: The average execution time is as follows:

$$\overline{T}^{E} = \frac{1}{M}\sum_{m=1}^{M}\sum_{k=1}^{K} T_{m,k}^{E}. \tag{36}$$

- **The Average Delay excluding Start Time** $\triangle\overline{T}$: It includes the average waiting time, the average transmission time, and the average execution time. The relationship among different types of time is as follows:

$$\begin{aligned}\triangle\overline{T} &= \frac{\sum_{m=1}^{M}\sum_{j=1}^{K}\triangle T_{m,j}}{M} \\ &= \frac{\sum_{m=1}^{M}\sum_{j=1}^{K}(T_{m,j}^{E}+T_{m,j}^{Tr}+T_{m,j}^{W})}{M} \\ &= \overline{T}^{E}+\overline{T}^{Tr}+\overline{T}^{w} \\ &= \overline{T}-\frac{\sum_{m=1}^{M}T_{m}^{Start}}{M}.\end{aligned} \tag{37}$$

- **The Improvement Rate**: The improvement rate IR_{A_1/A_2} of algorithm A_1 compared with A_2 is:

$$IR_{A_1/A_2} = \frac{\overline{T}_{A_1}-\overline{T}_{A_2}}{\overline{T}_{A_2}}, \tag{38}$$

where $\overline{T}_{A_1}$ and $\overline{T}_{A_2}$ are the average delays of the A_1 algorithm and the A_2 algorithm, respectively.

D. Experimental Results

1) Impact of the Speeds of Vehicles: As shown in Fig. 4, DFO-DDQN achieves significantly better results than other approaches. For instance, when the speed of the vehicle is 15 m/s and the number of vehicles is 15, it achieves 10.28%, 25.64%, 43.65%, and 46.45% improvements when compared with DFO-DQN, DFO-DuelingDQN, Seg-Greedy, NoSeg-Greedy, respectively. The Seg-Greedy algorithm achieves a relatively good offloading decision, which is better than the NoSeg-Greedy algorithm. The three DRL-based algorithms DFO-DDQN, DFO-DQN, and DFO-DuelingDQN are always significantly better than two greedy algorithms under different speeds.

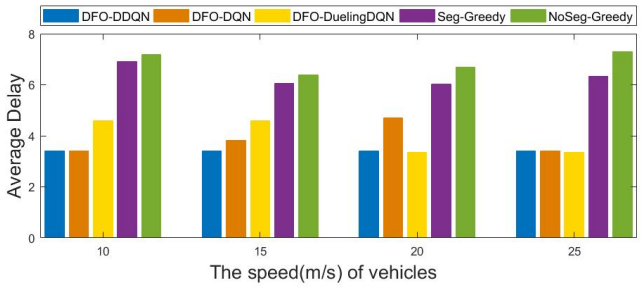


Fig. 4: Average time delay in different speeds under different offloading algorithms.

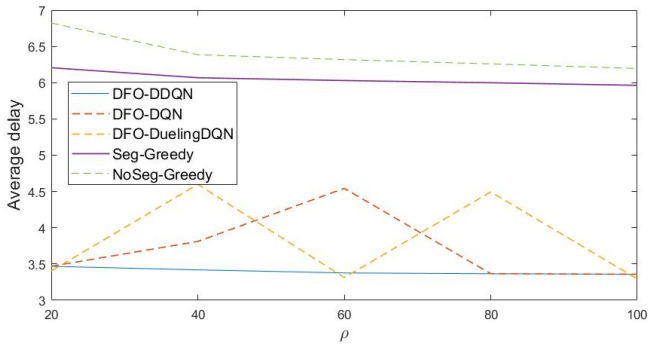
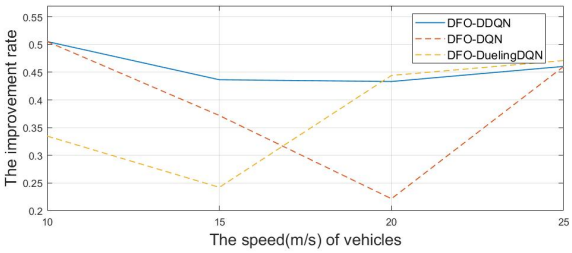


Fig. 5: Average time delay of tasks with different ρ under different offloading algorithms.

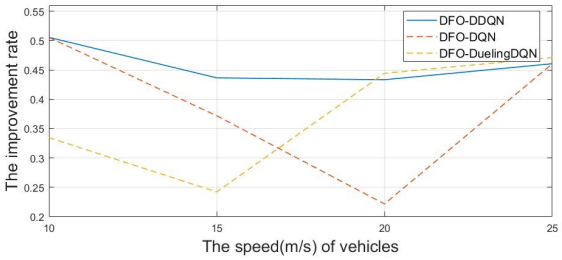
2) Impact of ρ: In this part, we set $\rho \in [20, 40, 60, 80, 100]$, which corresponds to the computational complexities of different types of tasks. Each of ρ is larger than one to make sure that the tasks are compute-intensive and have high computational complexity. As shown in Fig. 5, DFO-DDQN performs best when tasks with different ρ. when ρ is increased, the average delay of DFO-DDQN method and two greedy-based methods are decreased although the decreasing trends are not pronounced, indicating that these methods are insensitive to the value of ρ.

3) Comparison at Different Speeds: As shown in Fig. 6(a), the values of $IR_{DFO-DDQN/Seg-Greedy}$ at different speeds are relatively good, which are mostly higher than $IR_{DFO-DQN/Seg-Greedy}$ and $IR_{DFO-DuelingDQN/Seg-Greedy}$. Compared with Seg-Greedy, the improvement rates of DFO-DDQN are always higher than 43.3% at different speeds. As shown in Fig. 6(b), the values of $IR_{DFO-DDQN/NoSeg-Greedy}$ at different speeds are relatively good, which are mostly higher than $IR_{DFO-DQN/NoSeg-Greedy}$ and $IR_{DFO-DuelingDQN/NoSeg-Greedy}$. The improvement rates of DFO-DDQN are always higher than 46.4% at different speeds compared with NoSeg-Greedy. Overall, for the performance of DFO-DDQN at different speeds, DFO-DDQN is better than DFO-DQN and DFO-DuelingDQN.

4) Convergence Performance: Considering the randomness of results of different random seeds, we set three same seeds 0, 50, 100 for DFO-DDQN, DFO-DQN, and DFO-DuelingDQN to fix the results so that the comparisons are equitable. As depicted in Fig. 7, the red scope indicates the average rewards of different seeds of DFO-DDQN. The narrower the scope, the smaller the influence by different seeds, which means that the algorithm is more stable. DFO-DuelingDQN performs poorly in the low-speed situation, while DFO-DQN utperforms it in in each speed situation, but not better than DFO-DDQN. DFO-DDQN is more stable than DFO-DQN and DFO-DuelingDQN. The average rewards of DFO-DDQN at different speeds go beyond -0.2 when the episodes are in $[4000, 6000]$, while the average rewards of DFO-MDQN and DFO-DuelingDQN are always under -0.2 after training for 10,000 episodes, which indicates that DFO-DDQN achieves significantly better results than DFO-DQN and DFO-DuelingDQN.



(a) $IR_{DRL/Seg-Greedy}$



(b) $IR_{DRL/NoSeg-Greedy}$

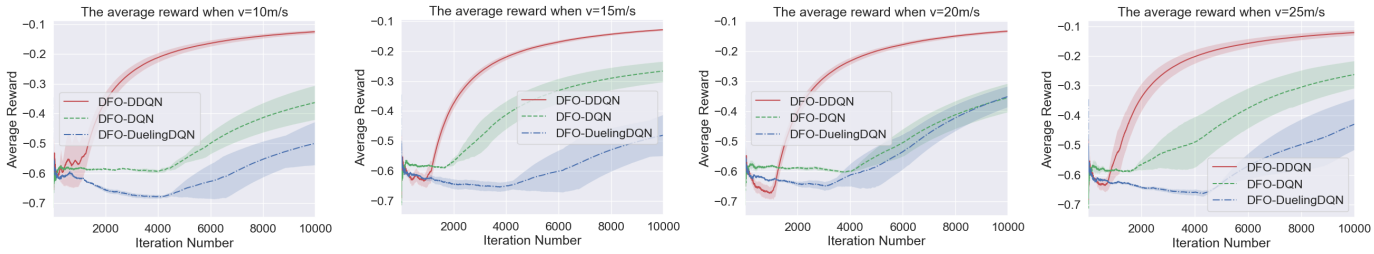
Fig. 6: Performance improvement of different DRL-based algorithms compared with Seg-Greedy and NoSeg-Greedy at different speeds when $M = 15$.

5) Comparison under Different Numbers of Vehicles: In order to further evaluate our DFO-DDQN, we conduct comparison experiments under different numbers of vehicles when $v = 15$ m/s, and obtain the results as follows:

- **Average Waiting Time** $\overline{T}^W$: The waiting time is generated from the chosen edge server when it is occupied by other subtasks. As shown in Fig. 8(a), DRL-based methods including DFO-DDQN, DFO-DQN, DFO-DuelingDQN perform obviously better than greedy-based methods including Seg-Greedy and NoSeg-Greedy. As the number of vehicles increases, the subtasks congestion is more and more serious, the waiting time of greedy-based method increases significantly, while the waiting time of DRL-based method has slow growth. It shows that even in the congestion situation, DRL-based methods are better suited to dynamic sequential subtasks offloading decisions. This is because DRL-based methods consider long-term returns, while greedy-based methods only consider the immediate rewards.
- **Average Transmission Time** $\overline{T}^{Tr}$: As shown in Fig. 8(b), the transmission time of greedy-based methods is significantly higher than that of DRL-based methods, which means that DRL-based methods can find edge servers that are more suitable for offloading. It is worth mentioning that the transmission time of the NoSeg-Greedy method is higher than that of the Seg-Greedy method. This is because the Seg-Greedy method can refine the offloading decision and choose different edge servers to execute subtasks, while the NoSeg-Greedy method can only choose one edge server because of the need to offload the entire task.
- **Average Execution Time** $\overline{T}^E$: As shown in Fig. 8(c), the execution time of the NoSeg-Greedy method is significantly higher than other methods. This is because offloading the whole task may lose the wide optimization space of choosing edge servers. The average execution time of Seg-Greedy is close to but higher than that of DRL-based methods, which means that task segmentation is an effective method of optimizing execution time. This is because partial offloading can offload subtasks according to the combination of a more suitable decision, instead of offloading the whole task to an edge server, especially in the dynamic situation of vehicle movement.
- **Average Delay except Start Time** $\triangle\overline{T}$: As shown in Fig. 8(d), the $\triangle\overline{T}$ of greedy-based methods are significantly higher than that of DRL-based methods under different numbers of vehicles. The seg-Greedy method performs better than the NoSeg-Greedy method when $M = 10, 15, 20$, while it performs worse than the NoSeg-Greedy method when $M = 25, 30$, indicating that task segmentation is not a good choice when the traffic congestion is heavy. This is because partial offloading may enlarge the shortcomings of greedy-based methods when applied to solve the MDP problem, while this problem can be well solved by DRL-based methods.
- **Average Start Time**: The average simulation value of T^{Start} is shown in Fig. 8(e). The average start time is not flat but instead rises and falls with the random settings of the computing capability of vehicles. Different methods share the same start time to be fair.
- **Average Finish Time** $\overline{T}$: As shown in Fig. 8(f), the average delay of different methods increases with the increase in the number of vehicles, and the NoSeg-Greedy method and DRL-based methods perform diminishing growth, which may mainly be because of the influence of the random start time. The average delay of greedy-based methods is significantly higher than that of DRL-based methods under different numbers of vehicles.

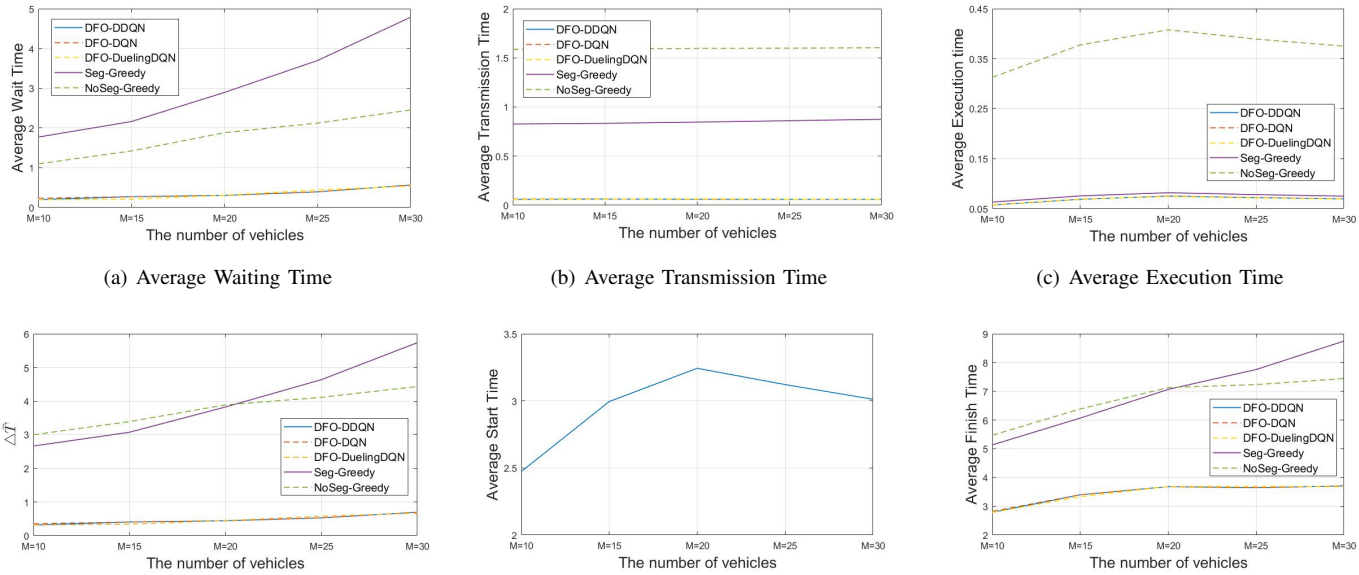
6) Impact of the Number of Vehicles at Different Speeds: To further study the impact of congestion at different speeds, we conduct experiments under different numbers of vehicles at different speeds and compare the values of $IR_{DFO-DDQN/Seg-Greedy}$ and $IR_{DFO-DDQN/NoSeg-Greedy}$. As shown in Fig. 9(a), the values of $IR_{DFO-DDQN/Seg-Greedy}$ at different speeds increase with the number of vehicles. When $M = 10$, DFO-DDQN performs best in lowest speed 10 m/s because it is more likely to occur congestion under lower speed than under higher speed and DFO-DDQN performs well, especially in the congestion situation. When $M = 20$, DFO-DDQN performs best in highest speed 25 m/s because the Seg-Greedy method performs worse in the high-speed situation, while DFO-DDQN is less affected by speed than the Seg-Greedy method. As shown in Fig. 9(b), DFO-DDQN performs better in high speed and heavier task congestion than in low speed and lighter task congestion when compared with the NoSeg-Greedy method.

7) Convergence Performance at Different Speeds and Different Numbers of Vehicles: As shown in Fig. 10, the difference in rewards at different speeds when $M = 10$ is greater than that when $M = 30$, and the difference in rewards



(a) $v = 10$ m/s (b) $v = 15$ m/s (c) $v = 20$ m/s (d) $v = 25$ m/s

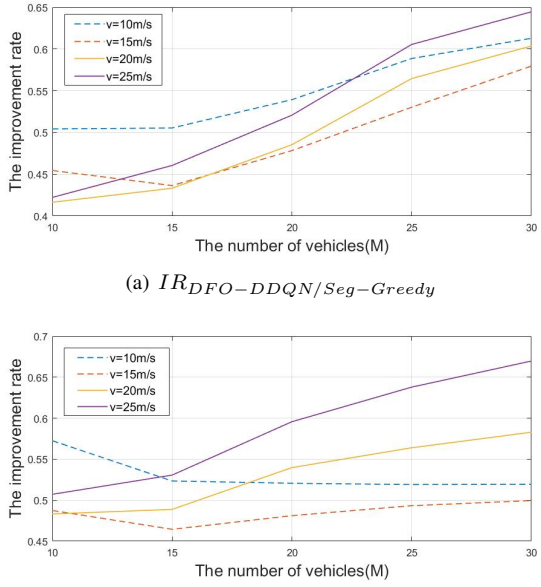
Fig. 7: Comparison of the average rewards under DF0-DDQN, DFO-DQN and DFO-DuelingDQN when $v = [10, 15, 20, 25]$ m/s



(a) Average Waiting Time (b) Average Transmission Time (c) Average Execution Time

(d) $\triangle\overline{T}$ (e) Average Start Time (f) Average Finish Time

Fig. 8: Comparison of different types of delay under different $M = [10, 15, 20, 25, 30]$ when $v = 15$ m/s.



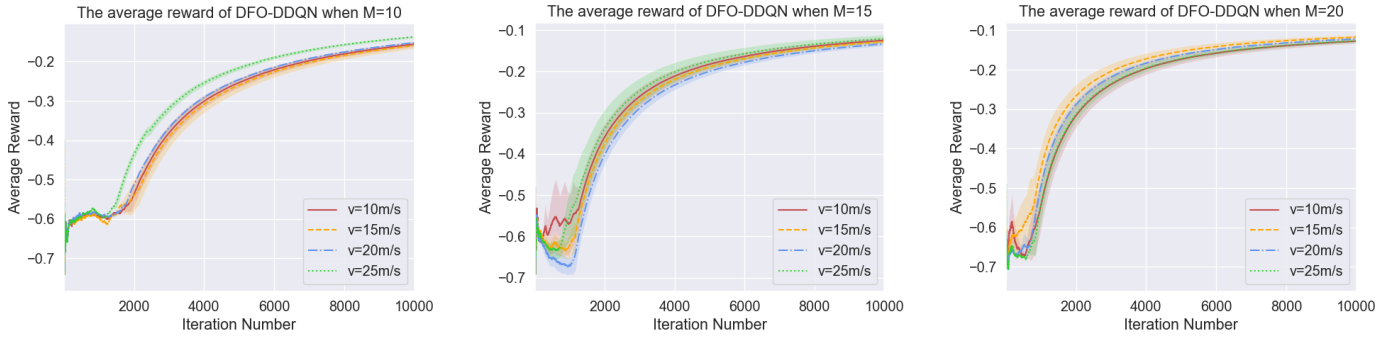
(a) $IR_{DFO-DDQN/Seg-Greedy}$

(b) $IR_{DFO-DDQN/NoSeg-Greedy}$

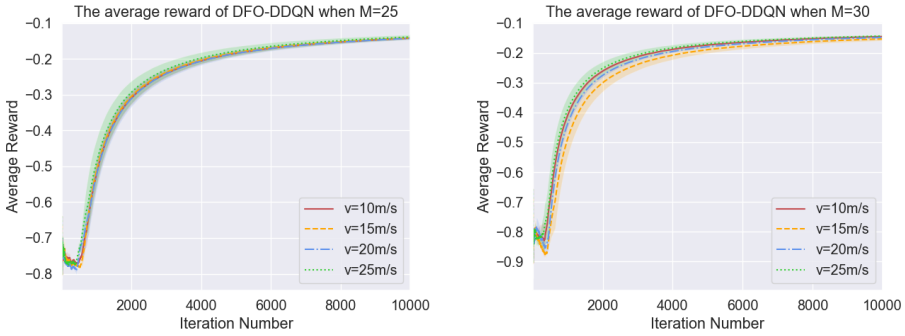
Fig. 9: Performance improvement compared with Seg-Greedy and NoSeg-Greedy at different speeds when $M = [10, 15, 20, 25]$.

decreases as the number of vehicles increases. This is because the congestion is mainly caused by the number of vehicles rather than the speed when the number of vehicles is large, while the congestion of a small number of vehicles is mainly affected by the speeds of vehicles because vehicles are more likely to be scattered at high speeds.

8) Convergence Performance at High Speeds: In this part, we set $M = 15$, $seed \in [0, 50, 100]$ and $v \in [10, 15, 20, 25, 30, 35, 40, 45]m/s$. As shown in Fig. 11, DFO-DDQN performs best when it is compared with other algorithms. Tasks cost an initial decreased average delay which was followed by a subsequent increase in average delay when the speeds are increased under the NoSeg-Greedy algorithm. Because the congestions are more serious when in lower speeds which cause higher waiting time, while the transmission distances are farther when in higher speeds which cause higher transmission time. However, when the task is divided into several subtasks, the trends of increasing average delay are slow down, which means task segmentation contributes to optimizing the average delay of the computation-intense task when in high speeds environment. In addition, the average delays of DRL-based methods are all lower than the average delays of greedy-based methods, which have again proved that DRL-based methods are better at solving the MDP problem



(a) $M = 10$ (b) $M = 15$ (c) $M = 20$



(d) $M = 25$ (e) $M = 30$

Fig. 10: Comparison of the average rewards of DFO-DDQN at different speeds when $M = [10, 15, 20, 25, 30]$.

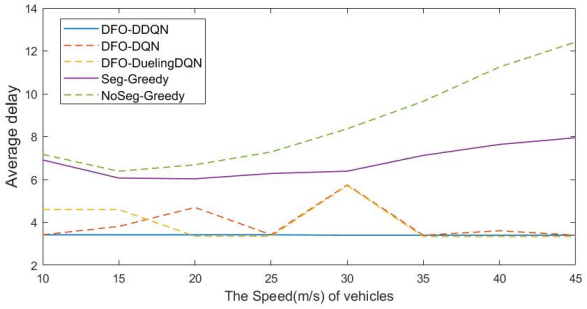


Fig. 11: Average time delay in high speeds under different offloading algorithms.

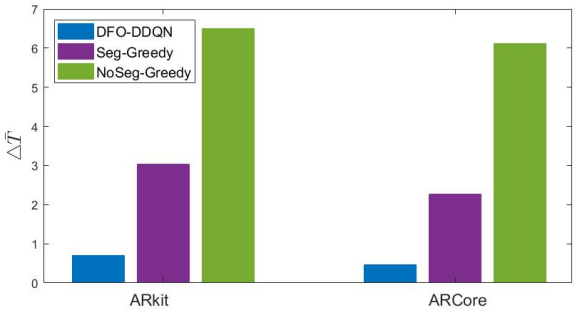


Fig. 13: $\triangle\overline{T}$ of ARkit and ARCore.

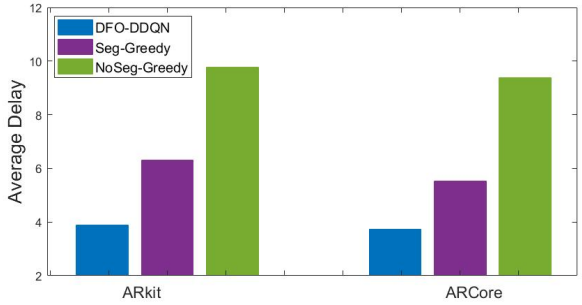


Fig. 12: Average time delay of ARkit and ARCore.

than greedy-based methods.

E. Results on ARkit and ARCore

ARkit can be divided into six sequential modules, namely, UIView, ARSCNView, SCNScene, ARCamera, ARSession and ARFrame [21], while ARCore can be divided into five sequential modules, namely, SCNScene, Environment Tracking, ARSession, ARFrame and Render [22]. AR tasks of vehicles are generated by initial subtasks on local where different vehicles have different initial subtasks, so that the generation time of AR tasks are different.

Considering the speeds of vehicles are different in the reality, we set different speeds for different vehicles, whose values are randomly taken from [10, 20] m/s. The amount of computation and data size of each subtask are set as the former simulations. The entire task of ARCore is set to the same amount of computation and data size as ARkit. The number of M is unfixed, it is randomly taken from $[10, 20]$, and the number of subtasks L is set as 7 for ARkit, where the former 6 subtasks correspond to the 6 modules of ARkit, while the last subtask is the result of the last module of ARkit that needs to be transmitted back to the vehicle. ARCore has fewer modules than ARkit, which means that the state set of ARCore can be

regarded as a subset of ARkit's state, so we can directly utilize the trained network of ARkit to make offloading decisions of ARCore. As shown in Fig. 12 and Fig. 13, DFO-DDQN is obviously much better than two greedy-based methods.

VI. Conclusion and Future Work

In this paper, aiming to minimize the total delay and waiting time of tasks from moving vehicles, we build a dynamic offloading model for multiple moving vehicles whose tasks can be divided into sequential subtasks, and further propose a DFO-DDQN algorithm to offload these sequential subtasks. By doing this, fine-grained offloading decisions can be achieved based on the frame-based process for offloading compute-intensive tasks in complex VEC environments considering V2R. Experimental results demonstrate that the proposed DFO-DDQN algorithm is far superior to other DRL-based offloading algorithms and greedy-based offloading algorithms, regardless of whether the congestion is mainly caused by the number of vehicles or the low speed.

For future work, we will introduce cloud computing into our model and consider the intelligent optimization of end-edge-cloud collaboration among different vehicles [39]–[41]. In addition, introducing content caching into task offloading may further reduce delay and save energy, which is in favor of a low-carbon VEC system.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant Number 62071327 and JSPS KAKENHI under Grant Number 19H04105.

References

[1] M. Xue, H. Wu, G. Peng, and K. Wolter, "Ddpqn: An efficient dnn offloading strategy in local-edge-cloud collaborative environments," *IEEE Transactions on Services Computing*, vol. 15, no. 2, pp. 640–655, 2022.

[2] Z. Xu, L. Zhao, W. Liang, O. F. Rana, P. Zhou, Q. Xia, W. Xu, and G. Wu, "Energy-aware inference offloading for dnn-driven applications in mobile edge clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 4, pp. 799–814, 2021.

[3] H. Tang, H. Wu, Y. Zhao, and R. Li, "Joint computation offloading and resource allocation under task-overflowed situations in mobile edge computing," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2021.

[4] X. Cao, F. Wang, J. Xu, R. Zhang, and S. Cui, "Joint computation and communication cooperation for energy-efficient mobile edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4188–4200, 2019.

[5] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6774–6785, 2019.

[6] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 37–45, 2018.

[7] H. Huang, K. Peng, and P. Liu, "A privacy-aware stackelberg game approach for joint pricing, investment, computation offloading and resource allocation in mec-enabled smart cities," in *2021 IEEE International Conference on Web Services (ICWS)*, pp. 651–656, 2021.

[8] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.

[9] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu, "EEDTO: An energy-efficient dynamic task offloading algorithm for blockchain-enabled iot-edge-cloud orchestrated computing," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2163–2176, 2021.

[10] H. Wang, X. Li, H. Ji, and H. Zhang, "Dynamic offloading scheduling scheme for mec-enabled vehicular networks," in *2018 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, pp. 206–210, 2018.

[11] S. Raza, S. Wang, M. Ahmed, M. R. Anwar, M. A. Mirza, and W. U. Khan, "Task offloading and resource allocation for IoV using 5G NR-V2X communication," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[12] X. Huang, K. Xu, C. Lai, Q. Chen, and J. Zhang, "Energy-efficient offloading decision-making for mobile edge computing in vehicular networks," *Journal on Wireless Communications and Networking*, vol. 2020, feb 2020.

[13] W. Zhan, C. Luo, J. Wang, C. Wang, and Q. Zhu, "Deep reinforcement learning-based offloading scheduling for vehicular edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 5449–5465, 2020.

[14] X. Huang, L. He, and W. Zhang, "Vehicle speed aware computing task offloading and resource allocation based on multi-agent reinforcement learning in a vehicular edge computing network," in *2020 IEEE International Conference on Edge Computing (EDGE)*, pp. 1–8, 2020.

[15] L. Huang, L. Zhang, S. Yang, L. P. Qian, and Y. Wu, "Meta-learning based dynamic computation task offloading for mobile edge computing networks," *IEEE Communications Letters*, vol. 25, no. 5, pp. 1568–1572, 2021.

[16] D. Tang, X. Zhang, M. Li, and X. Tao, "Adaptive inference reinforcement learning for task offloading in vehicular edge computing systems," in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–6, 2020.

[17] Y. Wu, J. Wu, L. Chen, J. Yan, and Y. Luo, "Efficient task scheduling for servers with dynamic states in vehicular edge computing," *Computer Communications*, vol. 150, pp. 245–253, 2020.

[18] Q. Luo, C. Li, T. H. Luan, and W. Shi, "Collaborative data scheduling for vehicular edge computing via deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9637–9650, 2020.

[19] 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 Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 7916–7929, 2020.

[20] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wireless Communications Letters*, vol. 6, pp. 398–401, June 2017.

[21] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4804–4814, 2019.

[22] J. Wang, T. Lv, P. Huang, and P. T. Mathiopoulos, "Mobility-aware partial computation offloading in vehicular networks: A deep reinforcement learning based scheme," *China Communications*, vol. 17, no. 10, pp. 31–49, 2020.

[23] G. Qu, N. Cui, H. Wu, R. Li, and Y. Ding, "Chainfl: A simulation platform for joint federated learning and blockchain in edge/cloud computing environments," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3572–3581, 2022.

[24] Z. Zhou, S. Yang, L. J. Pu, and S. Yu, "CEFL: Online admission control, data scheduling and accuracy tuning for cost-efficient federated learning across edge nodes," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9341–9356, 2020.

[25] X. Wang, Z. Ning, and S. Guo, "Multi-agent imitation learning for pervasive edge computing: A decentralized computation offloading algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 411–425, 2021.

[26] S. Yu, X. Chen, L. Yang, D. Wu, M. Bennis, and J. Zhang, "Intelligent edge: Leveraging deep imitation learning for mobile edge computation offloading," *IEEE Wireless Communications*, vol. 27, no. 1, pp. 92–99, 2020.

[27] Z. Zhou, K. Luo, and X. Chen, "Deep reinforcement learning for intelligent cloud resource management," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1–6, 2021.

[28] X. Zhang, Y. Xiao, Q. Li, and W. Saad, "Deep reinforcement learning for fog computing-based vehicular system with multi-operator support," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2020.

[29] T. Cai, Z. Yang, Y. Chen, W. Chen, Z. Zheng, Y. Yu, and H.-N. Dai, "Cooperative data sensing and computation offloading in uav-assisted crowdsensing with multi-agent deep reinforcement learning," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2021.

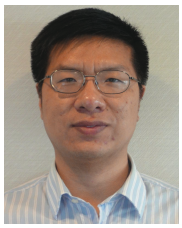
[30] H. Huang, Q. Ye, and Y. Zhou, "Deadline-aware task offloading with partially-observable deep reinforcement learning for multi-access edge

computing," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2021.
[31] G. Qu, H. Wu, R. Li, and P. Jiao, "DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3448–3459, 2021.
[32] Z. Zhang, N. Wang, H. Wu, C. Tang, and R. Li, "Mr-dro: A fast and efficient task offloading algorithm in heterogeneous edge/cloud computing environments," *IEEE Internet of Things Journal*, pp. 1–1, 2021.
[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *Computer Science*, 2013.
[34] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, p. 2094–2100, AAAI Press, 2016.
[35] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, vol. 30, p. 6382–6393, 2017.
[36] L. Liu, R. Zhong, W. Zhang, Y. Liu, J. Zhang, L. Zhang, and M. Gruteser, "Cutting the cord: Designing a high-quality untethered vr system with low latency remote rendering," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '18, (New York, NY, USA), p. 68–80, 2018.
[37] Y. Wang, S. Min, X. Wang, W. Liang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.
[38] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*, pp. 1995–2003, PMLR, 2016.
[39] K. Peng, H. Huang, B. Zhao, A. Jolfaei, X. Xu, and M. Bilal, "Intelligent computation offloading and resource allocation in IIoT with end-edge-cloud computing using NSGA-III," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2022.
[40] M. Xue, H. Wu, R. Li, M. Xu, and P. Jiao, "Eosdnn: An efficient offloading scheme for dnn inference acceleration in local-edge-cloud collaborative environments," *IEEE Transactions on Green Communications and Networking*, vol. 6, no. 1, pp. 248–264, 2022.
[41] I. Attiya, M. A. Elaziz, L. Abualigah, T. N. Nguyen, and A. A. Abd El-Latif, "An improved hybrid swarm intelligence for scheduling iot application tasks in the cloud," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2022.

Huijun Tang received the BSc degree from Jinan University, China in 2016 and the M.S. degree from Tianjin University, China in 2018. She is currently pursuing the PhD degree at the Center for Applied Mathematics, Tianjin University, China. Her research interests include internet of things, mobile edge computing and deep learning.

Huaming Wu received the B.E. and M.S. degrees from Harbin Institute of Technology, China in 2009 and 2011, respectively, both in electrical engineering. He received the Ph.D. degree with the highest honor in computer science at Freie Universität Berlin (FU Berlin), Germany in 2015. He is currently an Associate Professor in the Center for Applied Mathematics, Tianjin University, China. His research interests include wireless networks, mobile edge computing, internet of things and deep learning.

Guanjin Qu received the bachelor's degree from Taiyuan University of Technology, China in 2019. He is currently working towards the Master's degree at the Center for Applied Mathematics, Tianjin University, China. His research interests include distributed deep learning and edge computing.



Ruidong Li is an associate professor at Kanazawa University, Japan. Before joining this university, he was a senior researcher at the National Institute of Information and Communications Technology (NICT), Japan. He received the M.Sc. degree and Ph.D. degree in computer science from the University of Tsukuba in 2005 and 2008, respectively. He serves as the secretary of IEEE ComSoc Internet Technical Committee (ITC), and are the founders and chairs of IEEE SIG on Big Data Intelligent Networking and IEEE SIG on Intelligent Internet Edge. He is the associate editor of IEEE Internet of Things Journal, and also served as the guest editors for a set of prestigious magazines, transactions, and journals, such as IEEE communications magazine, IEEE network, IEEE TNSE. He also served as chairs for several conferences and workshops, such as the general co-chair for IEEE MSN 2021, AIVR2019, IEEE INFOCOM 2019/2020/2021 ICCN workshop, TPC co-chair for IWQoS 2021, IEEE MSN 2020, BRAINS 2020, IEEE ICDCS 2019/2020 NMIC workshop, and ICCSSE 2019. His research interests include future networks, big data, intelligent Internet edge, Internet of things, network security, information-centric network, artificial intelligence, quantum Internet, cyber-physical system, and wireless networks. He is a senior member of IEEE and a member of IEICE.