Lin *et al. Journal of Cloud Computing: Advances, Systems and Applications* (2021) 10:33
https://doi.org/10.1186/s13677-021-00246-6

Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

# Computation offloading strategy based on deep reinforcement learning for connected and autonomous vehicle in vehicular edge computing

Bing Lin[1,2,3†], Kai Lin[1†], Changhang Lin[4*], Yu Lu[5*], Ziqing Huang[1] and Xinwei Chen[6]

**Abstract**

Connected and Automated Vehicle (CAV) is a transformative technology that has great potential to improve urban traffic and driving safety. Electric Vehicle (EV) is becoming the key subject of next-generation CAVs by virtue of its advantages in energy saving. Due to the limited endurance and computing capacity of EVs, it is challenging to meet the surging demand for computing-intensive and delay-sensitive in-vehicle intelligent applications. Therefore, computation offloading has been employed to extend a single vehicle's computing capacity. Although various offloading strategies have been proposed to achieve good computing performace in the Vehicular Edge Computing (VEC) environment, it remains challenging to jointly optimize the offloading failure rate and the total energy consumption of the offloading process. To address this challenge, in this paper, we establish a computation offloading model based on Markov Decision Process (MDP), taking into consideration task dependencies, vehicle mobility, and different computing resources for task offloading. We then design a computation offloading strategy based on deep reinforcement learning, and leverage the Deep Q-Network based on Simulated Annealing (SA-DQN) algorithm to optimize the joint objectives. Experimental results show that the proposed strategy effectively reduces the offloading failure rate and the total energy consumption for application offloading.

**Keywords:** Computation offloading, Connected and autonomous vehicle, Reinforcement learning, Simulated annealing, Offloading failure, Energy consumption, Mobility

## Introduction

With the development of artificial intelligence technology, mobile communication technology and sensor technology, the design requirements of vehicles are no longer limited to the driving function. Vehicles are gradually transformed into an intelligent, interconnected and autonomous driving system, namely, Connected and Autonomous Vehicle (CAV) [1]. The rapid increase of vehicles on the road makes traffic accidents, traffic congestion and automobile exhaust pollution become increasingly prominent. A report of the World Health Organization (WHO) pointed out that more than 1.35 million people die in traffic accidents per year [2]. By sharing information with the infrastructure and neighboring vehicles, CAVs can perceive the surrounding environment more comprehensively [3], effectively reducing traffic accidents caused by human error and alleviating traffic congestion. Electric Vehicle (EV) [4], with its advantages in energy saving, is becoming the key subject in the next generation of CAVs. Its electricity can be generated from various renewable energy sources, such as solar energy, wind energy, and geothermal energy,

*Correspondence: linchanghang@139.com; fzluyu@163.com
†Bing Lin and Kai Lin contributed equally to this work.
[4]The School of Big Data and Artificial Intelligence, Fujian Polytechnic Normal University, Fuzhou 350300, China
[5]Concord University College, Fujian Normal University, Fuzhou 350117, China
Full list of author information is available at the end of the article

etc [5]. This will greatly reduce the environmental pollution caused by automobile exhaust emissions.

The development of CAV technology has given birth to a series of computing-intensive and delay-sensitive in-vehicle intelligent applications [6], e.g., autonomous driving [7], augmented reality [8], etc. They typically require large amounts of computing resources. But it is challenging for vehicles to meet the surging demand for such emerging applications, due to the limited endurance and computing capacity of vehicles. In recent years, *computation offloading* has been employed to extend a single vehicle's computing capacity. The computation offloading methods, based on traditional cloud computing platforms [9], offload computing tasks to cloud computing centers with powerful computing capabilities, effectively alleviating the computing burden on vehicles. However, due to the long transmission distance between vehicles and cloud computing centers, it will not only cause serious service delays, but also lead to huge energy consumption, which can not meet the needs of in-vehicle intelligent applications [10].

To address the above challenges, a new networking paradigm, Vehicular Edge Computing (VEC), has been proposed. VEC deploys Mobile Edge Computing (MEC) servers with computing and storage capabilities in Roadside Units (RSU). This enables CAV applications to be either processed in the vehicles locally, or offloaded to other cooperative vehicles or RSUs within the communication range for processing. This paradigm opens up new challenges on how to manage offloading to keep the *offloading failure rate* and *overall energy consumption* low. (1) Offloading failure rate could be impacted by the application execution time and the communication link established between the vehicle and the offloading targets. If the offloaded tasks can not complete within the application's tolerance time, the offloading fails; if the communication link is broken during the offloading process, the offloading fails. This requires that the offloading strategy should minimize the overall application execution time, and minimize the communication interruption by taking into consideration the vehicle's continuous movements. (2) Energy consumption also plays an important role in offloading [11, 12]. Both communication and task execution consumes vehicle's energy. An offloading strategy that can minimize the energy consumption would benefit vehicle's endurance. Therefore, different offloading strategies can impact both objectives simultaneously, potentially in opposite directions. This necessitates the joint optimization of the two objectives.

Researchers have done a considerable amount of work on CAV computation offloading strategy in the VEC environment. Table 1 lists a group of existing research work, where we mark the objectives, conditions and offloading schemes that are considered for each approach. As we can see, they have the following limitations.

1. The optimization objective mainly focused on either execution delay [13, 14] or energy consumption [15, 16], instead of the joint optimization of offloading failure rate and energy consumption [17, 18].
2. Most work only considered computation offloading strategy in the static environment [19–21], instead of dynamic offloading strategy with the change of vehicle positions in different time slots.

**Table 1** Comparison of existing approaches and our approach (✓indicates the corresponding item is considered in the technique)

| Techniques | Objectives | | Conditions | | Schemes | | |
|---|---|---|---|---|---|---|---|
| | Latency | Energy | Time Slot | Task Dependency | Local | V2I | V2V |
| Wu [13] | ✓ | | | ✓ | ✓ | | ✓ |
| Zhang [14] | ✓ | | | | ✓ | ✓ | |
| Jang [15] | | ✓ | ✓ | | ✓ | ✓ | |
| Pu [16] | | ✓ | ✓ | | ✓ | ✓ | ✓ |
| Wang [17] | | ✓(*joint*) | ✓ | | ✓ | ✓ | ✓ |
| Khayyat [18] | | ✓(*joint*) | ✓ | | ✓ | ✓ | |
| Dai [22] | ✓ | | | | | ✓ | |
| Ke [23] | | ✓(*joint*) | ✓ | | | ✓ | |
| Zhan [24] | | ✓(*joint*) | ✓ | | ✓ | ✓ | |
| Dai [25] | ✓ | | ✓ | | ✓ | ✓ | |
| Xu [19] | ✓ | | | | | ✓ | |
| Liu [20] | ✓ | | | ✓ | | ✓ | |
| Guo [21] | ✓ | | | | ✓ | ✓ | ✓ |
| **Our work** | | ✓(*joint*) | ✓ | ✓ | ✓ | ✓ | ✓ |

3 They mostly aimed at computation offloading of independent tasks [22, 23], i.e., no data dependency among tasks of an application.
4 Most work only considered offloading tasks to RSUs or processing tasks directly on On-Board Unit (OBU) [24, 25], without utilizing the idle computing resources of cooperative vehicles.

To address the above limitations, our work establishes a new computation offloading model, which takes into consideration *task dependencies, vehicle mobility, and different computing resources to offload tasks to*. Our goal is to jointly optimize the objectives *offloading failure rate and energy consumption*. To this end, our work employs Deep Reinforcement Learning (DRL), which excels at resolving the dimension disaster problem existed in the traditional reinforcement learning methods [26, 27]. More specifically, our work designs an efficient computation offloading strategy based on Deep Q-Network Based on Simulated Annealing (SA-DQN) algorithm in the VEC environment.

The main contributions of this paper are as follows.

- A new computation offloading model for CAV applications in the VEC environment is established based on Markov Decision Process (MDP). Since the computation tasks from application decomposition can be processed locally, offloaded to RSUs or cooperative vehicles, the model introduces task queues in vehicles and RSUs to model the task transmission and processing. Moreover, vehicle mobility and temporal dependency among tasks are also considered in the model.
- The work designs a computation offloading strategy based on deep reinforcement learning, and leverages the SA-DQN algorithm to optimize the joint objectives.
- The proposed computation offloading strategy is evaluated using real vehicle trajectories. The simulation results show that the proposed strategy can effectively reduce the offloading failure rate and the total energy consumption.

The rest of this paper is organized as follows. Section II introduces the related work of computation offloading in VEC. Section III formally defines the computation offloading problem of CAV applications and the optimization goal, and analyzes the computation offloading process using an example. Section IV proposes the computation offloading strategy based on DRL, and designs the SA-DQN algorithm for the computation offloading strategy. Section V presents and analyzes the experimental results, as well as the performance differences between SA-DQN algorithm and traditional reinforcement learning algorithms. Section VI summarizes this work and looks into future work.

## Related work

As shown in Table 1, there have been a wide range of research work on CAV application offloading strategy with different objectives, conditions and offloading schemes. *Most existing studies focused on the optimization of either execution delay or energy consumption, but rarely consider joint optimization of execution delay and energy consumption*. Wu et al. [13] proposed an optimal task offloading approach using 802.11p as the transmission protocol of inter vehicle communication, in which transmission delay and computation delay are considered to maximize the long-term return of the system. Although a large number of experimental results show that the proposed optimization approach has good performance, the optimization of energy consumption is not considered in this study. Zhang et al. [14] proposed an effective combined prediction mode degradation approach considering the computation task execution time and vehicle mobility. Although the simulation results show that the approach greatly reduces the cost and improves the task transmission efficiency, it does not consider the energy consumption of communication and processing. Jang et al. [15] considered the change of communication environment, jointly optimized the offloading ratio of multiple vehicles, and optimized the total energy consumption of vehicles under the delay constraint. Although the proposed energy-saving offloading strategy significantly reduces the total vehicle energy consumption, it does not consider the processing energy consumption of the computing node. Pu et al. [16] designed an online task scheduling algorithm to minimize the energy consumption of vehicles in the network for multi-vehicle and multi-task offloading problem. Simulation results show that the proposed framework has excellent performance. Wang et al. [17] proposed a dynamic reinforcement learning scheduling algorithm to solve the offloading decision problem. Although the experimental results show that the performance of the proposed algorithm is better than other benchmark algorithms, the offloading of dependent tasks is not considered. Khayyat et al. [18] proposed a distributed deep learning algorithm to optimize the delay and energy consumption. The simulation results show that the algorithm has faster convergence speed.

*Some research work only focused on independent task offloading in the VEC environment*. Dai et al. [22] proposed a method based on utility table learning, which verified the effectiveness and scalability of the method in various scenarios. The work considers both cloud computing and edge computing platform to offload tasks. Ke et al. [23] proposed a computation offloading method

based on deep reinforcement learning in the dynamic environment.

*Some studies only considered offloading tasks to RSU or processing tasks locally.* Han et al. [24] established a MDP model for the problem, and optimized the offloading strategy with deep reinforcement learning. Although the study considers the change of vehicle's position in different time slots, it does not make full use of cooperative vehicle resources. Dai et al. [25] transformed the load balancing and offloading problem into an integer non-linear programming problem to maximize the system utility. Experiments show that the strategy is significantly better than the benchmark strategy in terms of system utility. Although the mobility of vehicles is considered in this study, the offloading mode does not consider offloading tasks to cooperative vehicles.

*Some studies did not consider the change of vehicle positions in different time slots.* Xu et al. [19] proposed an adaptive computation offloading method to optimize the delay of task offloading and resource utilization. Experimental results show the effectiveness of this method. Liu et al. [20] offloaded multiple vehicle applications to RSU, divided each application into multiple tasks with task dependencies, and proposed an efficient task scheduling algorithm to minimize the average completion time of multiple applications. This work divides the application into several tasks to effectively reduce the completion time of application. Guo et al. [21] introduced Fiber-Wireless (FI-WI) integration to enhance the coexistence of VEC network and remote cloud, and proposed two task offloading approaches. The experimental results show that the proposed approaches have advantages in reducing the task processing delay.

## Problem definition and analysis

In this section, we first define the problem by modeling the network, application, communication and computation, then analyze an example of the proposed model.

### Problem definition

#### *Network model*

The VEC network model is shown in Fig. 1. The vehicles are categorized into Task Vehicle (TaV) and Service Vehicle (SeV) [28]. Both are equipped with OBU, and hence they have certain computing capability. TaV is the user of applications, which can be offloaded to SeVs after application decomposition to utilize the computing resources of cooperative vehicles in the neighborhood. There are fixed RSUs deployed on the roadside. Each RSU is equipped with an MEC server, which is integrated with wired connection [29]. They also have certain computing capability. SeVs and RSUs are referred to as Service Nodes (SNs)[30].

In the VEC network model, there are $m$ RSUs $\{\alpha_1, \alpha_2, ..., \alpha_m\}$, one TaV $\beta_1$, and $n$ SeVs $\{\gamma_1, \gamma_2, ..., \gamma_n\}$. The coverage radiuses of RSUs are $\{r_1, r_2, ..., r_m\}$, respectively, and the communication radius of a vehicle is $r^v$. TaV can not only offload computation tasks to RSUs by Vehicle to Infrastructure (V2I) communication, but also to SeVs by Vehicle to Vehicle (V2V) communication. The two offloading schemes are referred to as remote offloading.

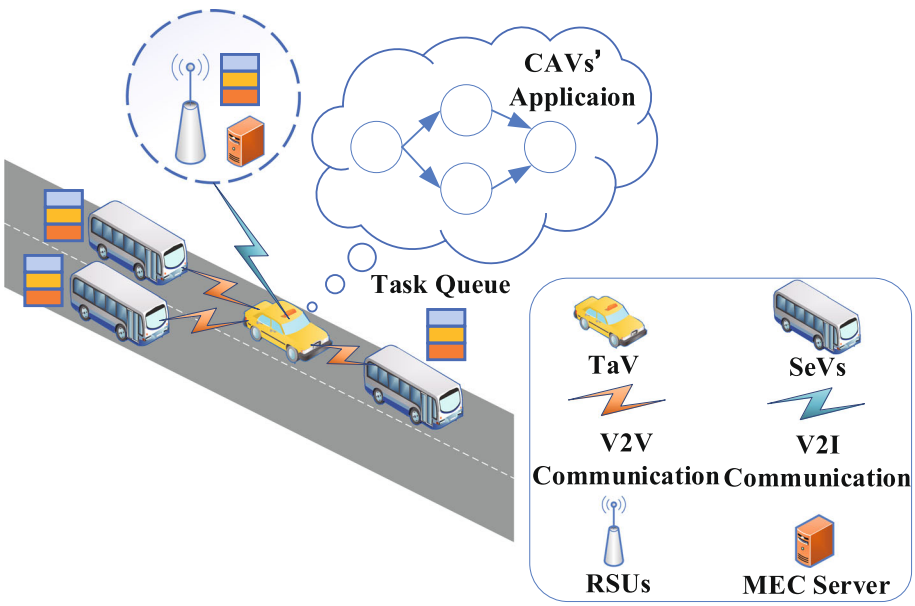


**Fig. 1** VEC network model

To better describe the generation, transmission and processing process of CAV applications, we divide the vehicle travel time into $t$ time slots, with each slot of length $\varepsilon$. In each time slot, the VEC system is quasi-static; that is, the relative position of the vehicle and the wireless channel state are stable, while they may change across different time slots [31].

### *Application model*

Most CAV applications use algorithms based on computer vision or deep learning to process a large amount of data collected by on-board sensors (cameras, radars, etc). CAV local applications and various third-party applications are usually computation-intensive or delay-sensitive applications. They typically need to use a lot of computing resources to process real-time data to meet the requirements of low execution delay [32].

The OBU on CAVs with limited computing resources cannot meet the requirements of applications. Therefore, to fully utilize the computing resources of RSUs and SeVs within CAV's communication range, CAV applications are decomposed into multiple smaller tasks, potentially with dependecies among them. Let's assume there are $z$ different CAV applications, and each of them can be generated with probability $1/z$ in each time slot. As shown in Fig. 2, each CAV application can be decomposed to multiple tasks, denoted as $\boldsymbol{A}_i \triangleq \{\boldsymbol{G}_i, l_i\}(i \in \{1,2,...,z\})$, where $\boldsymbol{G}_i$ is the temporal dependency of decomposed tasks and $l_i$ is the tolerance time for the *i-th* application. Specifically, the temporal dependency of tasks is represented by a directed acyclic graph (DAG) $\boldsymbol{G}_i = \langle \boldsymbol{N}_i, \boldsymbol{E}_i \rangle$, where $\boldsymbol{N}_i = \left\{ \boldsymbol{T}_1^i, \boldsymbol{T}_2^i, ..., \boldsymbol{T}_{|\boldsymbol{N}_i|}^i \right\}$ is the set of decomposed tasks, and $\boldsymbol{E}_i = \left\{ e_{u,v}^i | f\left(e_{u,v}^i\right) = 1, 1 \le u, v \le |\boldsymbol{N}_i|, u \ne v \right\}$ is the set of direct edges representing temporal dependency of tasks. $f\left(e_{u,v}^i\right) = 1$ indicates there is a directed edge $\boldsymbol{T}_u^i \rightarrow \boldsymbol{T}_v^i$, while $f\left(e_{u,v}^i\right) = 0$ indicates there is no edge. $\boldsymbol{T}_u^i$ is called the direct predecessor task of $\boldsymbol{T}_v^i$. The direct predecessor task $\boldsymbol{T}_u^i$ must be completed before $\boldsymbol{T}_v^i$ can be processed. The set of direct predecessors of a task can be denoted as $\boldsymbol{R}_v^i = \left\{ \boldsymbol{T}_u^i | f\left(e_{u,v}^i\right) = 1, 1 \le u, v \le |\boldsymbol{N}_i|, u \ne v \right\}$. The task $\boldsymbol{T}_v^i$ can not be processed until all tasks in the set of direct predecessors $\boldsymbol{R}_v^i$ have been completed. Tasks without any direct predecessor are called entry task, while tasks without any direct successor are called exit task. Moreover, each decomposed task can be represented as $\boldsymbol{T}_u^i \triangleq \left\{ u, Deep\left(\boldsymbol{T}_u^i\right), d_u^i \right\} (u \in \{1,2,...,|\boldsymbol{N}_i|\}$, where $u$ is the decomposed task index, $Deep\left(\boldsymbol{T}_u^i\right)$ is the task depth defined by Eq. (1), and $d_u^i$ is the task data size.

$$Deep(\boldsymbol{T}_u^i) = \begin{cases} 0, \boldsymbol{R}_u^i = \emptyset \\ 1 + MAX(Deep(\boldsymbol{R}_u^i)), otherwise \end{cases} \tag{1}$$

### *Task queue model*

The task queue model is illustrated in Fig. 3. Considering the transmission and processing of task data, we denote a task queue on TaV/SeV as $\boldsymbol{Q}^t$/ $\boldsymbol{Q}^s$, while a task queue on RSUs as $\boldsymbol{Q}^r$. Each task queue holds the tasks from the decomposition of CAV applications. Tasks in the task queue are sorted by task depth first and then by task number in the ascending order.

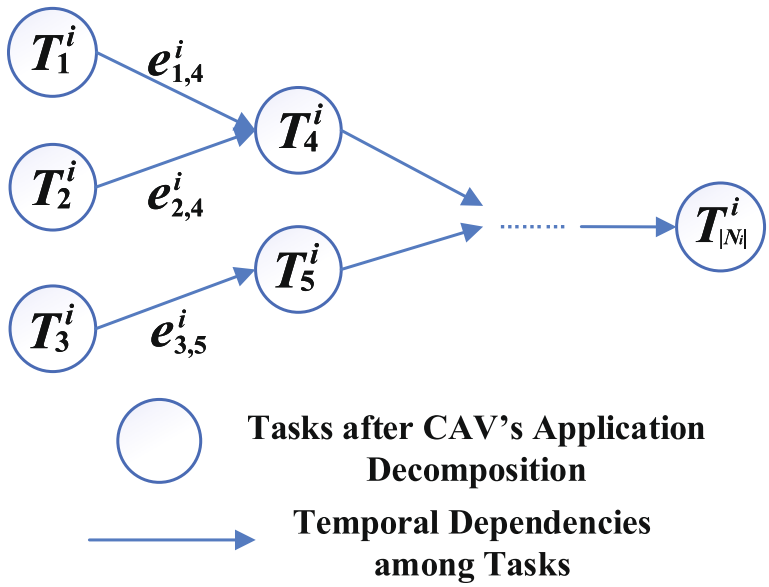


**Fig. 2** Decomposition model of CAV's application

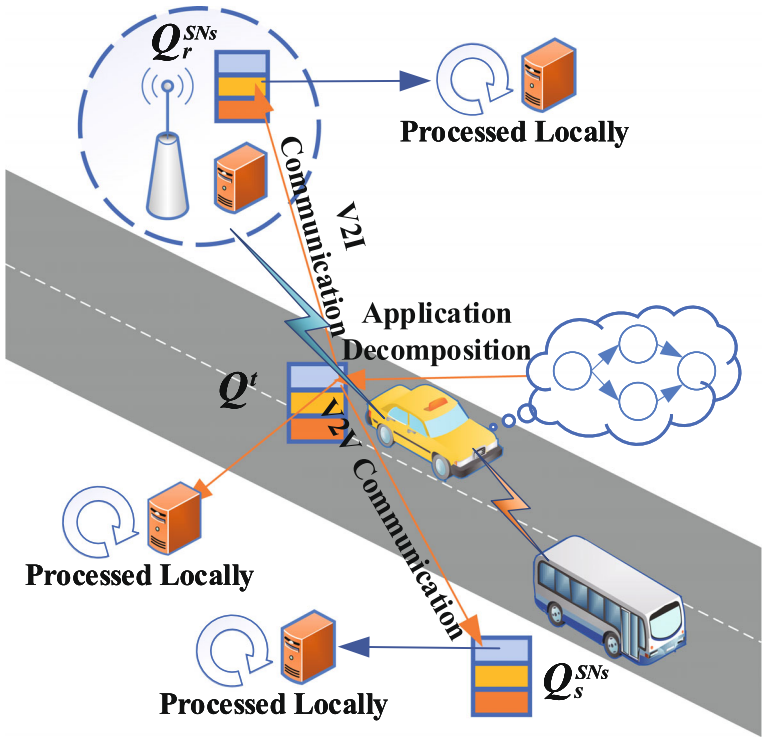


**Fig. 3** Task queue model

For the task queue $\boldsymbol{Q}^t$, we have the following definitions:

i) $\boldsymbol{Q}^t$ holds the tasks decomposed from TaV applications;
ii) TaV can only transmit or process task data at the head of $\boldsymbol{Q}^t$.

For the task queues $\boldsymbol{Q}^s$ and $\boldsymbol{Q}^r$, we have the following definitions:

i) $\boldsymbol{Q}^s$ and $\boldsymbol{Q}^r$ hold the tasks transmitted by TaV;
ii) SeVs can only process task data at the head of $\boldsymbol{Q}^s$ and RSUs can only process task data at the head of $\boldsymbol{Q}^r$.

***Communication model***

TaV can communicate with SNs to transmit task data at the head of $\boldsymbol{Q}^t$. We define channel bandwidth as $B$, transmission power of TaV as $p_{tr}$, channel fading coefficient as $h$, Gaussian white noise power as $\chi$ and path loss exponent as $\varpi$.

In the $i$-th time slot, the transmission rate from TaV to SN $j$ is expressed as

$$\tau_{i,j}^{SN} = B\log_2\left(1+\frac{|h|^2 p_{tr}}{\chi\left(\Phi_{i,j}^{SN}\right)^{\varpi}}\right) \tag{2}$$

where $\Phi_{i,j}^{SN}$ is the distance between TaV and SN $j$, defined by

$$\Phi_{i,j}^{SN} = \sqrt{\left(x_i^{tav}-x_{i,j}^{SN}\right)^2+\left(y_i^{tav}-y_{i,j}^{SN}\right)^2} \tag{3}$$

where $x_i^{tav}$ and $y_i^{tav}$ are the two-dimensional coordinates of TaV in the $i$-th time slot, $x_{i,j}^{SNs}$ and $y_{i,j}^{SNs}$ is the two-dimensional coordinates of SN $j$ in the $i$-th time slot.

In the $i$-th time slot, only when the distance between TaV and the SN $j$ is within the coverage radius of SN, the task data can be transmitted. If TaV transmits task data to the SN $j$, the amount of task data transmitted can be expressed as

$$\eta_{i,j}^{SN} = \begin{cases} \varepsilon\tau_{i,j}^{SN}, \Phi_{i,j}^{SN} \le r_j^{SN} \\ 0, \Phi_{i,j}^{SN} > r_j^{SN} \end{cases} \tag{4}$$

the data transmission between TaV and SN $j$ will cause energy consumption, which can be expressed as

$$\delta_{i,j}^{SN} = \begin{cases} \varepsilon p_{tr}, \Phi_{i,j}^{SN} \le r_j^{SN} \\ 0, \Phi_{i,j}^{SN} > r_j^{SN} \end{cases} \tag{5}$$

***Computation model***

TaV can either transmit the task at the head of $\boldsymbol{Q}^t$ to SNs, or process the task locally. SNs only process the task at the head of task queue locally.

The computation model includes two parts: tasks processed by TaV and tasks processed by SNs.

a) **Tasks processed by TaV.** The power consumption of TaV processing tasks locally is expressed as

$$p^{tav} = \kappa_{tav}(f^{tav})^3 \tag{6}$$

where $\kappa_{tav}$ is the effective switched capacitance coefficient related to the chip architecture in vehicle [33], and $f^{tav}$ is the local computing capacity of the TaV (i.e., the CPU frequency in cycles/sec). TaV processing tasks will consume a certain amount of energy, expressed as

$$\delta_{tav}^{process} = p^{tav}\varepsilon = \kappa_{tav}(f^{tav})^3\varepsilon \tag{7}$$

The data size that TaV can process in a time slot is given by

$$d^{tav} = \frac{f^{tav}\varepsilon}{c} \tag{8}$$

where $c$ is the processing density of task data (in CPU cycles/bit).

b) **Tasks processed by SNs.** The power consumption of SN $i$ processing locally is expressed as

$$p_i^{SN} = \kappa_i\left(f_i^{SN}\right)^3 \tag{9}$$

where $\kappa_i$ is the effective switched capacitance coefficient related to the chip architecture in SN $i$. $f_i^{SN}$ is the processing capability of SN $i$. SNs processing tasks will consume a certain amount of energy, expressed as

$$\delta_i^{SN} = p_i^{SN}\varepsilon = \kappa_i(f_i^{SN})^3\varepsilon \tag{10}$$

The data size that SN $i$ can process in a time slot is given by

$$d_i^{SN} = \frac{f_i^{SN}\varepsilon}{c} \tag{11}$$

In a time slot, TaV can process the task data locally or offload the task to the SNs within the communication range. The offloading decision adopted by TaV can be represented by the 0-1 decision variable as shown in Eq. (12). $v_i$ indicates whether TaV processes task data locally in the $i$-th time slot, and $o_j^i$ indicates whether TaV offloads task to SN $j$ in the $i$-th time slot. SNs process a task only when it is at the head of task queue. $\theta_j^i$ indicates whether SN $j$ processes task data in the $i$-th time slot. $\beta$ and $\zeta$ are the weight coefficients of execution delay and energy consumption respectively, where $\beta + \zeta = 1$.

$$\begin{aligned} &\textbf{Minimize}\ \ Loss = \sum_{i=1}^{t} loss_i = \\ &\sum_{i=1}^{t}\left[\beta fail_i + \zeta(\delta_i^{tav} + \delta_i^{tSN} + \delta_i^{com})\right] \\ &\textbf{subject to}\quad v_i o_j^i = 0 \end{aligned} \tag{12}$$

where $fail_i$ is offloading failure penalty in the $i$-th time slot, expressed as

$$fail_i = \sum_{j=1}^{|\boldsymbol{D}_i^{loss}|} d_{i,j}^{loss} \tag{13}$$

where $\boldsymbol{D}_i^{loss}$ is the data size set of the offloading failed tasks (unprocessed tasks belonged to offloading failed applications in task queues), $d_{i,j}^{loss}$ is data size of $j$-th offloading failed task in $i$-th time slot.

There are two cases that can lead to application offloading failure:

1 While the SNs are receiving task data, distance between TaV and SNs is out of the communication range during data transmission.
2 The completion time of application is greater than its tolerance time.

In Eq. (12), $\delta_i^{tav}$ is the energy consumption caused by TaV processing tasks, given by

$$\delta_i^{tav} = v_i\delta_{tav}^{process} \tag{14}$$

$\delta_i^{tSN}$ is the energy consumption caused by SNs processing tasks, given by

$$\delta_i^{tSN} = \sum_{1\le j\le m+n} \theta_j^i\delta_j^{SN} \tag{15}$$

and $\delta_i^{com}$ is the energy consumption of communication, given by

$$\delta_i^{com} = \sum_{1\le j\le m+n} o_j^i\delta_{i,j}^{SN} \tag{16}$$

the constraint indicates that tasks can only be processed locally or offloaded to SNs within a time slot.

#### Example analysis

Figure 4 illustrates an example of the computation offloading process in the VEC environment.

1) In the first time slot, TaV generates the first application $\boldsymbol{A}_1$ with the tolerance time of 4 time slots. It is decomposed into three tasks, which are kept in $\boldsymbol{Q}^t$. At this time slot, TaV processes task data locally. $loss_1$ is energy consumption caused by TaV processing task $\boldsymbol{T}_1^1$ locally, multiplied by the weight coefficient of energy consumption optimization.
2) In the second time slot, TaV transmits task data to $\gamma_1$. $loss_2$ is energy consumption caused by TaV transmitting task $\boldsymbol{T}_1^2$.
3) In the third time slot, TaV generates the second application $\boldsymbol{A}_2$ with the tolerance time of 4 time slots. At this time slot, TaV transmits task data to $\gamma_2$, $loss_3$ is the weight sum of energy consumption caused by

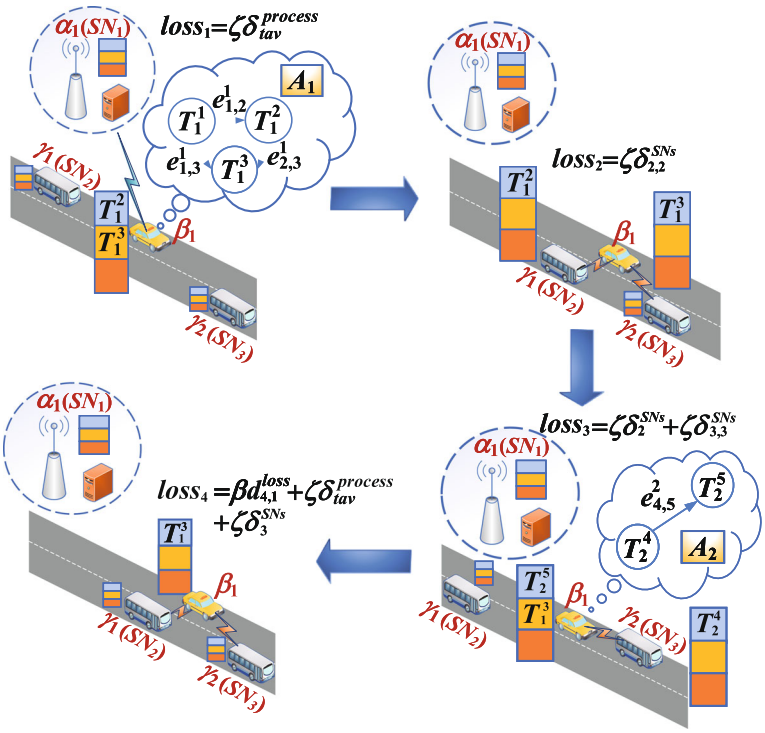


**Fig. 4** An example analysis of the computation offloading process

$\gamma_1$ processing task data and TaV transmitting $\boldsymbol{T}_2^4$ to $\gamma_2$.

4) In the fourth time slot, TaV processes task data locally. At this time slot, the task $\boldsymbol{T}_2^5$ is completed, and the task $\boldsymbol{T}_1^3$ belonged to $\boldsymbol{A}_1$ has not been processed. So application $\boldsymbol{A}_1$ offloading failed due to that the completion time of $\boldsymbol{A}_1$ is greater than its tolerance time. SeV $\gamma_2$ processes task data locally, the task $\boldsymbol{T}_2^4$ is completed, then all the tasks of $\boldsymbol{A}_2$ have been processed, so $\boldsymbol{A}_2$ are executed successfully. $loss_4$ is the weight sum of total data size of unprocessed task $d_{4,1}^{loss}$ and the energy consumption caused by TaV processing locally as well as the energy consumption caused by $\gamma_2$ processing task.

## Computational offloading strategy based on deep reinforcement learning

Reinforcement Learning (RL) algorithms have four key elements in model building: agent, environment, action and reward. It is usually modeled as Markov Decision Process (MDP) model.

In the algorithm learning process, the agent observes the current environment and chooses actions according to strategy. After executing the action, the agent observes the reward and transfers to the next environment. RL algorithms imitate the way of human learning. The purpose of RL algorithms is to maximize the total reward by adjusting the strategy appropriately when the agent interacts with the unknown environment.

In this section, we first describe the computation offloading problem by an MDP model to determine the four key elements. Secondly, we introduce Q-learning algorithm. Finally, due to the large dimension of state space in VEC environment, the traditional reinforcement learning optimization method is almost impossible to solve complex computation offloading problem in VEC. Therefore, we adopt SA-DQN to optimize the computation offloading strategy, and describe the computation offloading strategy based on SA-DQN.

### MDP model

In order to design the computation offloading strategy based on SA-DQN, we first establish an MDP model. It can fully describe the offloading scheduling model.

MDP model is the basic model of RL algorithms. Since the probability of state transition in real environment is often related to historical state, it is difficult to establish the model. Therefore, the model can be simplified according to Markov property (i.e. the next state in the environment is only related to the current state information, but not to the historical state), so that the next state is only related to the current state and the action taken [34]. Next, we will introduce each key element of MDP.

We define the state space as $\boldsymbol{S}_k \triangleq \{\boldsymbol{\Omega}_k, \boldsymbol{O}_k\}$ in $k$-th time slot, where $\boldsymbol{\Omega}_k = \{x_k^{tav}, y_k^{tav}\}$ is the two-dimensional coordinates of TaV. $\boldsymbol{O}_k$ represents the distance between TaV and SNs which can be expressed as $\boldsymbol{O}_k = \{f(\Phi_{k,1}^{SN}), f(\Phi_{k,2}^{SN}), ..., f(\Phi_{k,n+m}^{SN})\}$. If SN is out of communication range, then $f(\Phi_{k,1}^{SN}) = -1$, else $f(\Phi_{k,i}^{SN}) = \Phi_{k,i}^{SN}$. The action space can be described as $\boldsymbol{A}_k \triangleq \{ol_k, \boldsymbol{OS}_k^i, \boldsymbol{OR}_k^i\}$ in $k$-th time slot, where $ol_k$ indicates whether tasks are processed locally by TaV, $\boldsymbol{OS}_k^i = \{os_k^1, os_k^2, ..., os_k^n\}$ indicates whether tasks are offloaded to SeV $i$ and $\boldsymbol{OR}_k^i = \{or_k^1, or_k^2, ..., or_k^m\}$ indicates whether tasks are offloaded to RSU $i$. The reward can be described as $r_k \triangleq 1/loss_k$ in $k$-th time slot. The problem of computation offloading for CAV in VEC environment can be described as the following MDP model:

- **Agent**:TaV
- **State**:$\boldsymbol{S}_k$
- **Action**:$\boldsymbol{A}_k$
- **Reward**:$r_k$

#### Q-Learning algorithm

In this section, we introduce the traditional RL algorithm called Q-learning. Q-learning is a temporal difference (TD) algorithm based on stochastic process and model-free, and has no state transition probability matrix. The algorithm will select the maximum value for updating the value function, while the action selection does not necessarily follow the action corresponding to the maximum value. It will lead to an optimistic estimation of the value function. Due to this feature, Q-learning belongs to the off-line policy learning method [35].

Q-learning optimizes the value function by four tuple information $\langle S_k, A_k, R_k, S_{k+1}\rangle$, where $S_k$ represents the environmental state in $k$-th time slot, $A_k$ represents the current action chose, $R_k$ represents the immediate reward, and $S_{k+1}$ represents the environmental state of the next time slot after the state transition.

The Q-learning value function is updated as follows:

$$\begin{aligned} Q(S_k, A_k) = Q(S_k, A_k) + \alpha[\, r_k + \\ \gamma \max_{A_{k+1}} Q(S_{k+1}, A_{k+1}) - Q(S_k, A_k)] \end{aligned} \tag{17}$$

where $\alpha$ is the learning efficiency, representing the degree of value function updating; $r$ is the immediate reward, representing the reward obtained by transferring to the next state; $\gamma$ is the discount factor, representing the impact of the subsequent state's value on the current state; and $\max_{A_{k+1}} Q(S_{k+1}, A_{k+1})$ is the maximum value of next state.

The Equation (17) can be further expressed as

$$Q(S_k, A_k) = Q(S_k, A_k) + \alpha(Q_{target} - Q_{eval}) \tag{18}$$

where

$$Q_{target} = r + \gamma \max_{A_{k+1}} Q(S_{k+1}, A_{k+1}) \tag{19}$$

$$Q_{eval} = Q(S_k, A_k) \tag{20}$$

In other words, the updating of Q-learning value function can be expressed as the value function's value plus the product of the difference between target Q-value and estimated Q-value and the learning efficiency. It is also known as TD target.

#### SA-DQN algorithm

The value function in Q-learning algorithm can be designed simply by a table. But in practice, the state space of computation offloading problem in VEC is large. If we want to establish a value function table, it will lead to serious memory usage and time cost. To solve this problem, known as dimensional disaster, we describe the computation offloading problem as a DRL process, using the function approximation to combine Q-learning with Deep Neural Network (DNN), transform the value function table into Q-network, and adjust the network weight coefficient by algorithm training to fit the value function [36].

Compared with Q-learning, DQN has three main advantages:

i) The Q-network can be expressed as $Q(S_k, A_k; \theta)$. $\theta$ represents the weights of the neural network, and the Q-network fit value function by updating the parameter $\theta$ in each iteration.
ii) In order to improve the learning efficiency and remove the correlation in the subsequent training samples, DQN adopts experience replay technique in the learning process. The sample observed in $k$-th time slot $e_k = \langle S_k, A_k, R_k, S_{k+1}\rangle$ is stored into the reply memory $\boldsymbol{D}$ first, and then a sample is randomly chosen from $\boldsymbol{D}$ to train the network. It breaks the correlation among samples and makes them independent.
iii) Two neural networks with the same structure but different weights are used in DQN. One is called the target Q-network, and the other is the estimated Q-network. The estimated Q-network has the latest weights, while the weights of the target Q-network are relatively fixed. The weights of the target Q-network is only updated with the estimated Q-network every $\iota$ time slots.

The network used to calculate TD target is called TD-network. If the network parameters used in the value function are the same as those of TD network, it is easy to cause the correlation among samples and make the

training unstable. In order to solve this problem, two neural networks are introduced. The weights of the target Q-network can be expressed as $\bar{\theta}_k$ and that of estimated Q-network can be expressed as $\theta_k$, where $\bar{\theta}_k$=$\theta_{k-\iota}$, it means that $\bar{\theta}_k$ is updated with $\theta_k$ every $\iota$ time slots. In DQN algorithm, Equation (17) is transformed into:

$$\bar{y}_k = r_k + \gamma \max_{A_{k+1}} Q(S_{k+1}, A_{k+1}; \bar{\theta}_k) \tag{21}$$

In order to minimize the difference between the estimated value and the target value, we define the loss function as follows:

$$L(\theta_k) = \left[(\bar{y}_k - Q(S_k, A_k; \theta_k))^2\right] \tag{22}$$

By deriving $L(\theta_k)$ over $\theta_k$, we obtain the gradient:

$$\nabla_{\theta_k} L(\theta_k) = \left[2\left(Q(S_k, A_k; \theta_k) - y_k\right) \nabla_{\theta_k} Q(S_k, A_k; \theta_k)\right] \tag{23}$$

Therefore, the updating of value function in DQN is transformed to use gradient descent method to minimize the loss function:

$$\theta_k \leftarrow \theta_k - \nabla_{\theta_k} L(\theta_k) \tag{24}$$

In order to balance the exploration and exploitation of DQN, the Metropolis criterion [37] is used to choose the action, and cooling strategy is described as follows:

$$T_k = \theta^k T_0 \tag{25}$$

where $T_0$ is the initial temperature, $k$ is the amount of current episode, and $\theta$ is the cooling coefficient.

The computation offloading strategy based on SA-DQN algorithm is shown in Algorithm 1. In every episode, VEC network needs to be initialized, as shown in Algorithm 2. Then it needs to determine if there is communication interruption and handle it in every time slot, as shown in Algorithm 3. After TaV chooses the offloading decision, the VEC network needs to be updated, as shown in Algorithm 4. If TaV chooses to process tasks locally, the task queue of TaV needs to be updated, as shown in Algorithm 5. If TaV choose to transmit tasks to SNs, the task queue of TaV and SNs also needs to be updated, as shown in Algorithm 6. It needs to determine whether there are applications that offloading failed as it can not complete within its tolerance time in every time slot, as shown in Algorithm 7. The interaction among algorithms is shown in Fig. 5.

**Algorithm 1:** Computation offloading strategy based on SA-DQN.

**Input**: VEC network, target Q-network and estimated Q-network
**Output**: computation offloading strategy
**Initialization**:target Q-network $\theta_1$, estimated Q-network$\bar{\theta}_1 = \theta_1$, initial temperature $T_1$, the action chose in previous time slot *pre_action* $\rightarrow$ 0;
**for** *epi* $\leftarrow$ 1 ***to*** $epi_{\max}$ **do**
  ***Algorithm* 2**;
  **for** $k \leftarrow 1$ ***to*** $\varepsilon$ **do**
    TaV generates different applications, and then stored in $\boldsymbol{Q}^t$;
    Calculate exit tasks of applications and $\boldsymbol{R}_j^i$ of all tasks;
    Obtain the two-dimensional coordinates of TaV $\boldsymbol{\Omega}_k$ and that of SNs;
    $\boldsymbol{O}_k$ are calculated by Equation (3), $\boldsymbol{S}_k = \boldsymbol{\Omega}_k \cup \boldsymbol{O}_k$;
    $A_k$ = ***Algorithm* 3**;
    $loss_k$ = ***Algorithm* 4**, $R_k$=$1/loss_k$;
    Calculate $\Phi_{k+1,j}^{SN}$, obtain $\boldsymbol{S}_{k+1}$;
    Store $< \boldsymbol{S}_k, A_k, R_k, \boldsymbol{S}_{k+1} >$ to replay memory $\boldsymbol{D}$;
    Random sampling minibatch of experience $< \boldsymbol{S}_i, A_i, R_i, \boldsymbol{S}_{i+1} >$ from $\boldsymbol{D}$;
    **if** $i+1$ *is* $\varepsilon$*-th time slot* **then**
      $\bar{y}_i = R_i$;
    **else**
      $\bar{y}_i = R_i + \gamma \max_{A_{i+1}} Q(\boldsymbol{S}_{i+1}, A_{i+1}; \bar{\theta}_k)$;
    **end**
    Perform gradient descent step on $\left[(\bar{y}_i - Q(\boldsymbol{S}_i, A_i; \theta_k))^2\right]$ with respect to $\theta_k$ and update $\theta_k$;
    Every $\iota$ time-slots, update $\bar{\theta}_k$ with $\theta_k$;
    *pre_action* $= A_k$;
  **end**
  Cooling the current temperature $T_{epi}$ according to Equation (25);
**end**

**Algorithm 2:** Initialize VEC network.

1 Initialize the task queues: $\boldsymbol{Q}^t \rightarrow \emptyset$, $\boldsymbol{Q}^{SN} \rightarrow \emptyset$;
2 Initialize the direct precursor node set $\boldsymbol{R}_j^i \rightarrow \emptyset$, exit tasks of applications *exit_task*$(i) \rightarrow \emptyset$, the remaining life cycle of applications *life_cycle*$(i) \rightarrow$ *tolerance_time*$(i)$, list of tasks completed *comp_list* $\rightarrow \emptyset$, and list of applications completed *app_comp* $\rightarrow \emptyset$.

## Experimental results and analysis

### Parameter settings

All the simulation experiments were conducted on a Win10 64-bit operating system with a Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz processor and 8GB RAM. We use TensorFlow 1.15.0 with Python 3.6 to implement SA-DQN algorithm. In the experiment, we consider the real vehicle trajectory data set of two hours in the

**Algorithm 3:** Handling communication interruption.

**Input**: $\boldsymbol{Q}^t, life\_cycle, pre\_action$
**Output**: $A_k$
**if** $\boldsymbol{Q}^t(0)(2) \neq 0$ **then**
  $A_k = pre\_action$;
  **if** *the distance between TaV and SN where* $\boldsymbol{Q}^t(0)$ *offloaded exceeds its coverage range* **then**
    Find the application index according to $\boldsymbol{Q}^t(0)$;
    $life\_cycle(index) \rightarrow 0$;
  **else**
    According to simulated annealing strategy to select $A_k$;
  **end**
**end**

**Algorithm 4:** Updating VEC network.

**Input**: $\boldsymbol{Q}^{SN}, pre\_node, comp\_list, \boldsymbol{O}_k$
**Output**: $loss_k$
**Initialization**:processing energy consumption of TaV $tav\_process \leftarrow 0$, energy consumption of SNs processing $SN\_process \leftarrow 0$, communication energy consumption $com \leftarrow 0$, penalty of application offloading failure $fail\_penal \leftarrow 0$;
$SN\_process = \textbf{\textit{Algorithm 5}}(\boldsymbol{Q}^{SN}, n+m, \delta_i^{SN}, d_i^{SN})$;
**if** *TaV processes task locally* **then**
  $tav\_process = \textbf{\textit{Algorithm 5}}(\boldsymbol{Q}^t, 1, \delta_{tav}^{process}, d^{tav})$;
**else if** *TaV transmit task to SN j and distance between them* $\leq r_j$ **then**
  Calculate data size transmitted $trans\_amount = \eta_{k,j}^{SN}$;
  $com = \textbf{\textit{Algorithm 6}}(\boldsymbol{Q}^t, \boldsymbol{Q}_j^{SN}, trans\_amount)$;
**end**
$life\_cycle(i) -= 1$;
$fail\_penal = \textbf{\textit{Algorithm 7}}$;
$loss_k = \beta fail\_penal + \zeta(SN\_process + tav\_process + com)$

**Algorithm 5:** Updating task queue while processing tasks locally.

**Input**: $\boldsymbol{Q}, pre\_node, comp\_list, amount, exec\_energy, comp\_power$
**Output**: $process\_energy$
**Initialization**:$total\_energy \rightarrow 0$;
**for** $i \leftarrow 1$ ***to*** $amount$ **do**
  **if** $\boldsymbol{Q} \neq \emptyset$ *and* $pre\_node(\boldsymbol{Q}(0)(0)) \subseteq comp\_list$ **then**
    $total\_energy += exec\_energy(i)$;
    **if** $comp\_power(i) \geq \boldsymbol{Q}(0)(2)$ **then**
      $comp\_list.append(\boldsymbol{Q}(0)(0))$;
      $\boldsymbol{Q}.remove(0)$;
    **else**
      $\boldsymbol{Q}(0)(2) -= comp\_power(i)$;
    **end**
  **end**
**end**

**Algorithm 6:** Updating task queue while offloading tasks remotely.

**Input**: $\boldsymbol{Q}^t, \boldsymbol{Q}_{to}, pre\_node, comp\_list, trans\_amount$
**Output**: $trans\_energy$
**Initialization**:$trans\_energy \rightarrow 0$;
**if** $\boldsymbol{Q}^t \neq \emptyset$ *and* $pre\_node(\boldsymbol{Q}(0)(0)) \subseteq comp\_list$ **then**
  $trans\_energy += \varepsilon p_{tr}$;
  **if** $trans\_amount \geq \boldsymbol{Q}^t(0)(2)$ **then**
    $\boldsymbol{Q}_{to}.append(\boldsymbol{Q}^t(0))$;
    sort tasks in $\boldsymbol{Q}_{to}$;
    $\boldsymbol{Q}^t.remove(0)$;
  **else**
    $\boldsymbol{Q}(0)(2) -= trans\_amount$
**end**

morning (7.00-9.00) on a circular road in cretey, France [38]. The communication radius of the vehicle is 130 meters. The road conditions include a roundabout with 6 entrances and exits, multiple two lane or three lane roads, one bus road, four lane change points, and 15 traffic lights. In the data set, the trajectory of vehicle with vehicle ID "BusFlowWestEast0.0" is considered as the trajectory of TaV. The trajectory of vehicle with vehicle ID "VehicleFlowWestToEast.0" is considered as the trajectory of the first SeV, and the trajectory of vehicle with vehicle ID "VehicleFlowWestToEast_0.0" is considered as the trajectory of the second SeV. In the center of the TaV's trajectory, i.e. the two-dimensional coordinate (1250,600), we place an RSU with a coverage radius of 300 meters. $\beta$ is set to 0.4 and $\zeta$ is set to 0.6. There are six CAV's applications, each application can be divided into three tasks. The length of the time slot is set to 10 ms. The range of data size is distributed uniformly from 1 to 2, and the range of tolerance time is distributed uniformly from 50 to 100 time slots. Table 2 is the detailed setting of simulation parameters. After a number of experiments on parameter adjustment of the RL algorithms to achieve good convergence, parameters setting of RL algorithms is shown in Table 3.

#### Comparative offloading strategies

In order to verify the effectiveness of our proposed computation offloading strategy, we designed comparative

**Algorithm 7:** Handling application offloading failure.

**Input**: $life\_cycle$, $\boldsymbol{Q}^t$, $\boldsymbol{Q}^{SN}$
**Output**: $fail\_penal$
**Initialization**: $fail\_penal \rightarrow 0$;
**for** $i \leftarrow 1$ ***to*** $len(life\_cycle)$ **do**
  **if** $life\_cycle(i) = 0$ *and* $i \notin app\_comp$ **then**
    $app\_comp.append(i)$;
    **if** $exit\_task(i) \not\subset comp\_list$ **then**
      $fail_penal+ =$sum of data size of all tasks belonging to application $i$ in $\boldsymbol{Q}^t$, $\boldsymbol{Q}^{SN}$;
      delete all tasks belonged to application $i$ in $\boldsymbol{Q}^t$, $\boldsymbol{Q}^{SN}$;
    **end**
  **end**
**end**

offloading strategies from two perspectives: different RL algorithms and different offloading schemes.

In the first part, we select TD(0) algorithm combined with simulated annealing: Q-learning [39], Sarsa [40] and TD(λ) algorithm [41] with that: Sarsa(λ), Q-learning(λ) as comparative algorithms.

In the second part, we select four schemes for comparison. It is described as follows: Scheme 1 is our proposed strategy; Scheme 2 only considers tasks processed by TaV; Scheme 3 only considers tasks processed by TaV or offloaded to RSU; Scheme 4 only considers tasks processed by TaV or offloaded to cooperative vehicle.

## Experimental results

### *Offloading strategies with different algorithms*

Figure 6 shows the average reward of computation offloading strategy based on SA-DQN and comparative algorithms in every 20 episodes. It can be seen that, in the process of optimizing the offloading strategy, SA-DQN continuously interacts with the environment in every episode, updates the weights of neural network, and approaches the optimal value function. With the amount of episodes increasing, the reward increased. Around the 80th episode, the average reward obtained by SA-DQN tends to be optimal and stable, and remained at about 978. Compared with the comparative algorithms, SA-DQN has faster convergence speed. TD(λ) algorithms converged around the 100th round, while TD(0) algorithm converged around the 120th round. It indicates that TD(λ) algorithm converges faster than TD(0) algorithms. The possible reason is that TD(λ) algorithms introduces eligibility trace and adopts multi-step updating strategy. Therefore, it can accelerate the convergence speed. In the experiment, SA-DQN does not encounter the problems of divergence and oscillation, which proves the feasibility computation offloading strategy based on SA-DQN proposed in this paper.

Figure 7 shows the average total offloading energy consumption of strategy optimized by SA-DQN and comparative algorithms in every 20 episodes. It can be seen that

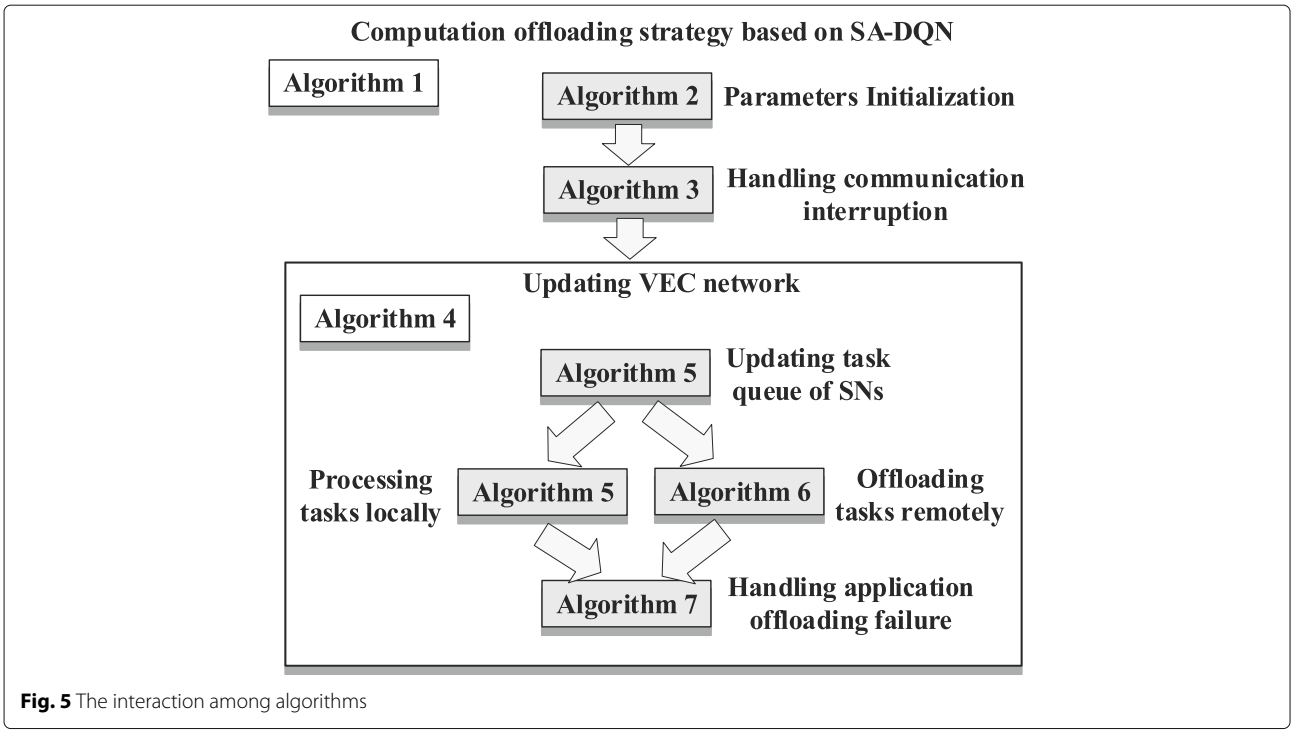


**Fig. 5** The interaction among algorithms

**Table 2** Parameters setting about simulation

| Description | Parameter | Value |
|---|---|---|
| Maximum episodes | $epi_{\max}$ | 200 |
| Length of time slot | $\varepsilon$ | 100 ms |
| Maximum time slots | $t$ | 186 |
| Number of RSUs | $m$ | 1 |
| Number of SeV | $n$ | 2 |
| Coverage radius of RSU | $r_1$ | 300 m |
| Communication radius of vehicle | $r^v$ | 130 m |
| Number of applications | $z$ | 6 |
| Tolerance time of applications | $l_i$ | [50,100] slots |
| Data size of tasks | $d_j^i$ | [1,2] Mb |
| Channel bandwidth | $b$ | 1 MHz |
| Transmission power of vehicles | $p_{tr}$ | 30 dBm |
| Channel fading coefficient | $h$ | 1 |
| White Gaussian noise power | $\gamma$ | -100 dBm |
| Path loss exponent | $\varpi$ | 2 |
| Switched capacitance coefficient of vehicle | $\kappa_{tav}$ | $10^{-24}$ |
| Switched capacitance coefficient of RSUs | $\kappa_i$ | $10^{-29}$ |
| Processing capability of vehicle | $f_i^l$ | 1.4 G cycles/s |
| Processing capability of RSU | $f_1^r$ | 10 G cycles/s |
| Processing density of data | $c$ | 100 cycles/bit |

**Table 3** Parameters setting about RL algorithms

| Description | Parameter | Value |
|---|---|---|
| Learning rate | $\alpha$ | 0.01 |
| Discount factor | $\gamma$ | 0.9 |
| Trace decay rate | $\lambda$ | 0.5 |
| Initial temperature | $\theta$ | 0.9 |
| Number of hidden layers | | 2 |
| Number of nodes for the first hidden layer | | 20 |
| Number of nodes for the second hidden layer | | 20 |
| Activation function for hidden layers | | ReLU |
| Maximum replay memory size | $\|\boldsymbol{D}\|$ | 500 |
| Minibatch size | | 300 |
| Parameter updating frequency for target DQN | $\iota$ | 10 |

the average offloading total energy consumption of SA-DQN and comparative algorithms is decreasing. Around the 160th episode, the average total offloading energy consumption of each algorithm tends to be optimal and stable. Compared with the comparative algorithms, the average total offloading energy consumption of SA-DQN can be maintained at about 30, reaching a lowest energy consumption level. In the comparative algorithms, the average total offloading energy consumption of Sarsa and Sarsa($\lambda$) algorithms maintained at about 35, while that of Q-learning and Q-learning($\lambda$) algorithms maintained at about 40. This shows that the online learning method can converge to a lower level than the offline learning method in optimizing the average total offloading energy consumption. This is because the online learning method updates the value function by the samples generated by the current strategy. Therefore it can converge faster, but the disadvantage is that it is easy to fall into the local optimal solution.

Figure 8 shows the average offloading failure rate of applications optimized by SA-DQN and comparative algorithms in every 20 episodes. It can be seen that with the increase of episodes, the average offloading failure rate of every algorithm decreased. In the 160th episode, except for Q-learning, the average offloading failure rate obtained by other algorithms tends to converge, and the average failure rate of SA-DQN can reach a lower level faster than other algorithms. Compared with the offline learning method, the average offloading failure rate of online learning method was lower than that of offline learning method. This shows that the online learning method can converge to a lower level than the offline learning method in optimizing the average offloading failure rate. This is because the online learning method is a conservative strategy, and it can converge to a lower level faster by following the current strategy.

#### *Offloading strategies with different offloading schemes*

Figure 9 shows the average application offloading failure rate of offloading strategies based on different schemes varying from data size. It can be seen that with the increase of data size, the average application offloading failure rate of all strategies increased continuously. The average application offloading failure rate obtained by our proposed strategy reached lower level compared with other strategies when data size is 32. It is because Scheme 1 has various offloading methods, and high offloading flexibility. The average offloading failure rate of Scheme 3 is the closest to that of Scheme 1, because the computing capacity of RSU is higher than that of vehicle. Although it requires certain communication energy consumption to offload to RSU, the completion time of tasks can be greatly reduced, and hence the completion time of application is not easy to exceed its tolerance time, so the penalty

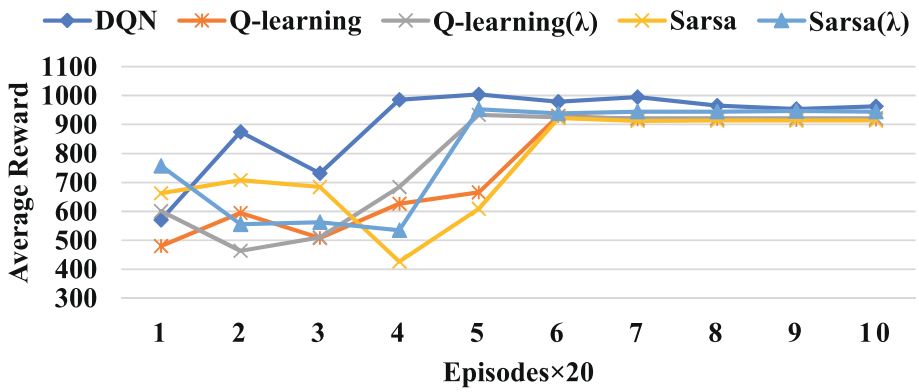


**Fig. 6** Average reward in every 20 episodes

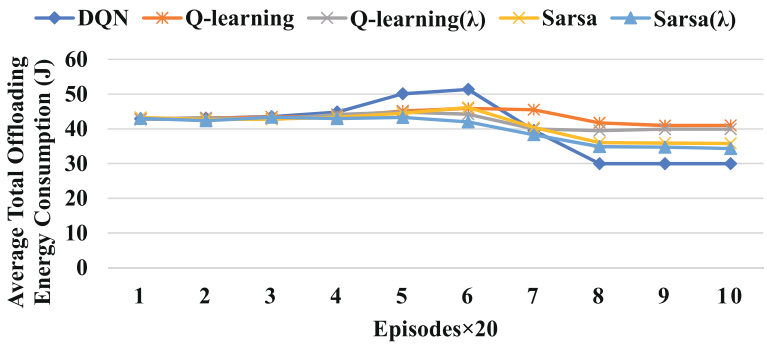


**Fig. 7** Average total offloading energy consumption in every 20 episodes

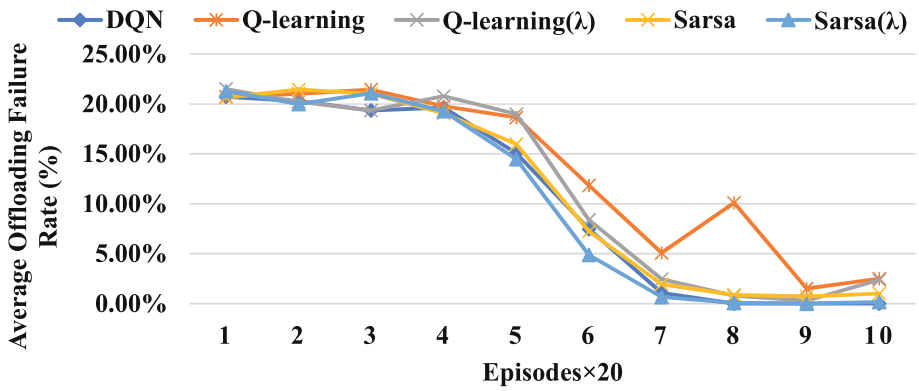


**Fig. 8** Average offloading failure rate in every 20 episodes

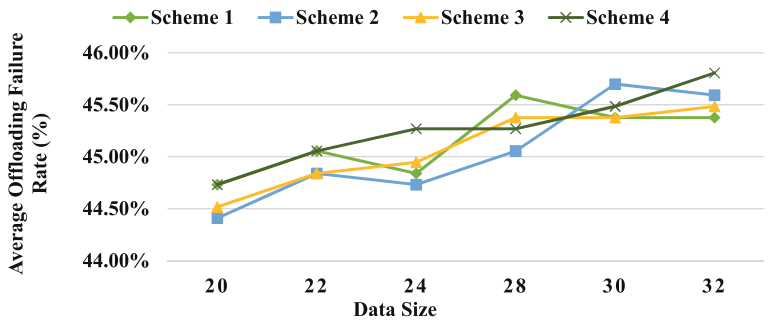


**Fig. 9** Average offloading failure rate of different strategies with varying data size

of offloading failure decreased. The average application offloading failure rate of Scheme 4 is the highest, because its offloading targets include not only local processing, but also cooperative vehicles, which requires a certain transmission time, and the processing capacity of vehicles is limited, which is not enough to process large amount of data. With the increase of data size, the completion time of application is more likely to exceed its tolerance time, and the penalty for application offloading failure will increase significantly. Therefore, it is obvious that the average application offloading failure rate is rising.

Figure 10 shows the average application offloading failure rate of offloading strategies with varying from tolerance time. It can be seen that with the increase of tolerance time, the average application offloading failure rate of all strategies decreased. Compared with other strategies, the proposed strategy reached lower level when tolerance time is 90. This is because when the application tolerance time increases, the application can have more time to offload, and the application completion time is less likely to exceed its tolerance time, and hence the application offloading failure rate decreased. The average offloading failure rate of Scheme 3 is the closest to that of Scheme 1. This is because RSU has strong computing capacity, which can greatly reduce the task completion time. With the increase of the application tolerance time, Scheme 3 can make full use of the computing power of RSU. Therefore, it can be seen that the average application offloading failure rate of Scheme 3 is significantly reduced. Compared with other strategies, the average offloading failure rate of Scheme 2 and Scheme 4 stay at a high level. One possible reason is that both Scheme 2 and Scheme 4 offload tasks to the vehicles with the limited processing capacity. with the increase of application tolerance time, the task offloading with large amount of data size may fail. Thus, the application offloading failure rate is high. On the contrary, both Scheme 1 and Scheme 3 can offload tasks to RSU with stronger computing capacity. Therefore, the number of successful offloading applications increases, and the failure rate of application offloading is lower.

## Conclusion

In order to solve the problem of computation offloading for CAV's applications in VEC environment, this paper proposed an computation offloading strategy based on SA-DQN algorithm. In the simulation experiment, the proposed strategy was evaluated based on the real vehicle trajectory. The experimental results show that our proposed computation offloading strategy based on SA-DQN algorithm has good performance, and further indicates that the strategy proposed can effectively reduce the total offloading energy consumption and offloading failure rate of CAV.

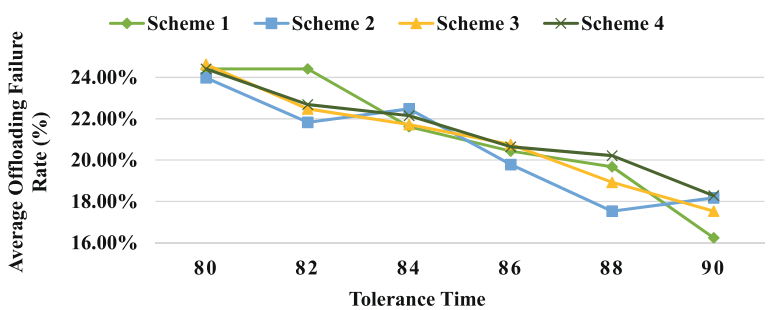


**Fig. 10** Average offloading failure rate of different strategies with varying tolerance time

In the future work, we will further consider to design collaborative computation offloading strategy in End-Edge-Cloud orchestrated architecture, which can transfer complicated computation tasks to remote cloud for further processing, and it can prompt the flexibility of computation offloading. We will consider more dynamic factors in the VEC environment to make it more suitable for the real world model. In addition, we will take on-board applications in real world into account.

**Acknowledgements**
Both Changhang Lin and Yu Lu are corresponding authors.

**Authors' contributions**
Both Bing Lin and Kai Lin drafted the original manuscript and designed the experiments. Changhang Lin provide ideas and suggestions. Yu Lu provided critical review and helped to draft the manuscript. Both Ziqing Huang and Xinwei Chen designed partial experimental work in the preliminary work and contributed to the revised manuscript. The authors read and approved the final manuscript.

**Authors' information**
**Bing Lin** received the B.S. and M.S degrees in Computer Science from Fuzhou University, Fuzhou, China, in 2010 and 2013, respectively, and the Ph.D. degree in Communication and Information System from Fuzhou University in 2016. He is currently an associate professor with the College of Physics and Energy at Fujian Normal University. Now he is the deputy director of the Department of Energy and Materials, and leads the Intelligent Computing research group. His research interest mainly includes parallel and distributed computing, computational intelligence, and data center resource management.
**Kai Lin** received the B.S. in Software Engineering from Fujian Agriculture and Forestry University, he is currently pursuing the M.S degree in Fujian Normal University. His main research interests include vehicular edge computing and cloud computing.
**Changhang Lin** received the B.S. and M.S degrees in Computer technology from Fuzhou University. He is currently an professor of the School of Big Data and Artificial Intelligence, Fujian Polytechnic Normal University. His research interest mainly includes Resource scheduling of cloud computing, Edge computing, Federated learning.
**Yu Lu** is currently an professor the College of Physics and Energy at Fujian Normal University. His research interest mainly includes Intelligent measurement and control technology in new energy technology.
**Ziqing Huang** is currently pursuing the B.S. degree in Fujian Normal University. Her main research interests include vehicular edge computing and mobile edge computing.
**Xinwei Chen** received the B.S., M.S. and Ph.D both in Computer Science and Intelligent Robot System from Nankai University, in 2006,2009 and 2012, and he is with College of Computer and Control Engineering, Minjiang University, Fujian Provincial Key Laboratory of Information Processing and Intelligent Control, Fuzhou, 351008, China. His current research interests include Intelligent Robot System, Embeded System and Computer Vision.

**Funding**
This work is partly supported by the Intelligent Computing and Application Research Team of Concord University College, Fujian Normal University under Grant No.2020TD001, the Natural Science Foundation of China under Grant No.62072108, the Industry and Science guide Foundation of Fujian Province under Grant No.2017H0011, the Natural Science Foundation of Fujian Province under Grant No.2019J01286, and the Young and Middle-aged Teacher Education Foundation of Fujian Province under Grant No.JT180098, Fujian Provincial Key Laboratory of Information Processing and Intelligent Control (Minjiang University) under Grant No.MJUKF-IPIC201907, Open Foundation of Engineering Research Center of Big Data Application in Private Health Medicine, Fujian Province University under Grant No.KF2020001.

**Availability of data and materials**
The data set of vehicle trajectory is available on http://vehicular-mobility-trace.github.io/.

## Declarations

**Competing interests**
The authors declare that they have no competing interests.

**Author details**
[1]College of Physics and Energy, Fujian Normal University, Fujian Provincial Key Laboratory of Quantum Manipulation and New Energy Materials, Fujian Provincial Collaborative Innovation Center for Advanced High-Field Superconducting Materials and Engineering, Fuzhou 350117, China. [2]Fujian Provincial Collaborative Innovation Center for Optoelectronic Semiconductors and Efficient Devices, Xiamen 361005, China. [3]Engineering Research Center of Big Data Application in Private Health Medicine, Putian University, Putian 351100, China. [4]The School of Big Data and Artificial Intelligence, Fujian Polytechnic Normal University, Fuzhou 350300, China. [5]Concord University College, Fujian Normal University, Fuzhou 350117, China. [6]Engineering Research Center of Big Data Application in Private Health Medicine, Fujian Province University, Minjiang University, Fuzhou 351008, China.

Received: 8 January 2021 Accepted: 18 May 2021
Published online: 08 June 2021

**References**

1. Wang Y, Liu S, Wu X, Shi W (2018) CAVBench: A Benchmark Suite for Connected and Autonomous Vehicles. In: 2018 IEEE/ACM Symposium on Edge Computing (SEC). pp 30–42. https://doi.org/10.1109/sec.2018.00010
2. Organization WH, et al (2018) Global status report on road safety 2018: Summary. World Health Organization, Geneva
3. Lin K, Lin B, Chen X, Lu Y, Huang Z, Mo YA (2019) Time-Driven Workflow Scheduling Strategy for Reasoning Tasks of Autonomous Driving in Edge Environment. In: 2019 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom). pp 124–131. https://doi.org/10.1109/ispa-bdcloud-sustaincom-socialcom48970.2019.00028
4. Wang M, Liang H, Deng R, Zhang R, Shen XS (2013) VANET based online charging strategy for electric vehicles. In: 2013 IEEE Global Communications Conference (GLOBECOM). pp 4804–4809. https://doi.org/10.1109/glocomw.2013.6855711
5. Kadav P, Asher ZD (2019) Improving the Range of Electric Vehicles. In: 2019 Electric Vehicles International Conference (EV). pp 1–5. https://doi.org/10.1109/ev.2019.8892929
6. Dhirani L, Newe T (2020) 5G security in smart manufacturing. ResearchGate. https://doi.org/10.13140/RG.2.2.27292.72320
7. Feng J, Liu Z, Wu C, Ji YAVE (2017) Autonomous Vehicular Edge Computing Framework with ACO-Based Scheduling. IEEE Trans Veh Technol 66(12):10660–10675
8. Zhao J, Li Q, Gong Y, Zhang K (2019) Computation Offloading and Resource Allocation For Cloud Assisted Mobile Edge Computing in Vehicular Networks. IEEE Trans Veh Technol 68(8):7944–7956
9. Adiththan A, Ramesh S, Samii S (2018) Cloud-assisted control of ground vehicles using adaptive computation offloading techniques. In: 2018 Design, Automation Test in Europe Conference Exhibition (DATE). pp 589–592. https://doi.org/10.23919/date.2018.8342076
10. Liu L, Chen C, Pei Q, Maharjan S, Zhang Y (2020) Vehicular Edge Computing and Networking: A Survey. Mob Netw Appl. https://doi.org/10.1007/s11036-020-01624-1
11. Wu H, Wolter K, Jiao P, Deng Y, Zhao Y, Xu MEEDTO (2021) An Energy-Efficient Dynamic Task Offloading Algorithm for Blockchain-Enabled IoT-Edge-Cloud Orchestrated Computing. IEEE Internet Things J 8(4):2163–2176
12. Wu H (2018) Multi-Objective Decision-Making for Mobile Cloud Offloading: A Survey. IEEE Access 6:3962–3976
13. Wu Q, Liu H, Wang R, Fan P, Fan Q, Li Z (2020) Delay-Sensitive Task Offloading in the 802.11p-Based Vehicular Fog Computing Systems. IEEE Internet Things J 7(1):773–785
14. Zhang K, Mao Y, Leng S, He Y (2017) ZHANG Y. Mobile-Edge Computing for Vehicular Networks: A Promising Network Paradigm with Predictive Off-Loading. IEEE Veh Technol Mag 12(2):36–44
15. Jang Y, Na J, Jeong S, Kang J (2020) Energy-Efficient Task Offloading for Vehicular Edge Computing: Joint Optimization of Offloading and Bit

Allocation. In: 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring). pp 1–5. https://doi.org/10.1109/vtc2020-spring48590.2020.9128785
16. Pu L, Chen X, Mao G, Xie Q, Xu J (2019) Chimera: An Energy-Efficient and Deadline-Aware Hybrid Edge Computing Framework for Vehicular Crowdsensing Applications. IEEE Internet Things J 6(1):84–99
17. Wang Y, Wang K, Huang H, Miyazaki T, Guo S (2019) Traffic and Computation Co-Offloading With Reinforcement Learning in Fog Computing for Industrial Applications. IEEE Trans Ind Inform 15(2):976–986
18. Khayyat M, Elgendy IA, Muthanna A, Alshahrani AS, Alharbi S, Koucheryavy A (2020) Advanced Deep Learning-Based Computational Offloading for Multilevel Vehicular Edge-Cloud Computing Networks. IEEE Access 8:137052–137062
19. Xu X, Zhang X, Liu X, Jiang J, Qi L, Bhuiyan MZA (2020) Adaptive Computation Offloading With Edge for 5G-Envisioned Internet of Connected Vehicles. IEEE Trans Intell Transport Syst:1–10
20. Liu Y, Wang S, Zhao Q, Du S, Zhou A, Ma X, et al (2020) Dependency-Aware Task Scheduling in Vehicular Edge Computing. IEEE Internet Things J 7(6):4961–4971
21. Guo H, Zhang J, Liu J (2019) FiWi-Enhanced Vehicular Edge Computing Networks: Collaborative Task Offloading. IEEE Veh Technol Mag 14(1):45–53
22. Dai P, Hang Z, Liu K, Wu X, Xing H, Yu Z, et al (2020) Multi-Armed Bandit Learning for Computation-Intensive Services in MEC-Empowered Vehicular Networks. IEEE Trans Veh Technol 69(7):7821–7834
23. Ke H, Wang J, Deng L, Ge Y, Wang H (2020) Deep Reinforcement Learning-Based Adaptive Computation Offloading for MEC in Heterogeneous Vehicular Networks. IEEE Trans Veh Technol 69(7):7916–7929
24. Zhan W, Luo C, Wang J, Wang C, Min G, Duan H, et al (2020) Deep-Reinforcement-Learning-Based Offloading Scheduling for Vehicular Edge Computing. IEEE Internet Things J 7(6):5449–5465
25. Dai Y, Xu D, Maharjan S, Zhang Y (2019) Joint Load Balancing and Offloading in Vehicular Edge Computing and Networks. IEEE Internet Things J 6(3):4377–4387
26. Lee SS, Lee S (2020) Resource Allocation for Vehicular Fog Computing Using Reinforcement Learning Combined With Heuristic Information. IEEE Internet Things J 7(10):10450–10464
27. Dong P, Wang X, Rodrigues J (2019) Deep Reinforcement Learning for Vehicular Edge Computing: An Intelligent Offloading System. ACM Trans Intell Syst Technol 10(6)
28. Sun Y, Song J, Zhou S, Guo X, Niu Z (2018) Task Replication for Vehicular Edge Computing: A Combinatorial Multi-Armed Bandit Based Approach. In: 2018 IEEE Global Communications Conference (GLOBECOM). pp 1–7. https://doi.org/10.1109/glocom.2018.8647564
29. Zeng F, Chen Q, Meng L, Wu J (2020) Volunteer Assisted Collaborative Offloading and Resource Allocation in Vehicular Edge Computing. IEEE Trans Intell Transport Syst:1–11
30. Qin Y, Huang D, Zhang X (2012) VehiCloud: Cloud Computing Facilitating Routing in Vehicular Networks. In: 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications. pp 1438–1445. https://doi.org/10.1109/trustcom.2012.16
31. Luo Q, Li C, Luan TH, Shi W (2020) Collaborative Data Scheduling for Vehicular Edge Computing via Deep Reinforcement Learning. IEEE Internet Things J 7(10):9637–9650
32. Wang L, Zhang Q, Li Y, Zhong H, Shi W (2019) MobileEdge: Enhancing On-Board Vehicle Computing Units Using Mobile Edges for CAVs. In: 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS). pp 470–479. https://doi.org/10.1109/icpads47876.2019.00073
33. Mao Y, Zhang J, Song SH, Letaief KB (2016) Power-Delay Tradeoff in Multi-User Mobile-Edge Computing Systems. In: 2016 IEEE Global Communications Conference (GLOBECOM). pp 1–6. https://doi.org/10.1109/glocom.2016.7842160
34. Sidford A, Wang M, Wu X, Ye Y (2018) Variance Reduced Value Iteration, Faster Algorithms for Solving Markov Decision Processes. Society for Industrial and Applied Mathematics, USA
35. Li J, Xiao Z, Li P (2019) Discrete-Time Multi-Player Games Based on Off-Policy Q-Learning. IEEE Access 7:134647–134659
36. Li R, Zhao Z, Sun Q, I C, Yang C, Chen X, et al (2018) Deep Reinforcement Learning for Resource Management in Network Slicing. IEEE Access 6:74429–74441
37. Su R, Wu F, Zhao J (2019) Deep reinforcement learning method based on DDPG with simulated annealing for satellite attitude control system. In: 2019 Chinese Automation Congress (CAC). pp 390–395. https://doi.org/10.1109/cac48633.2019.8996860
38. Lèbre MA, Le Mouël F, Ménard E (2015) On the Importance of Real Data for Microscopic Urban Vehicular Mobility Trace. In: Proceedings of the 14th International Conference on ITS Telecommunications (ITST'2015), Copenhagen, Denmark. pp 22–26. https://doi.org/10.1109/itst.2015.7377394
39. Jiang K, Zhou H, Li D, Liu X, Xu S (2020) A Q-learning based Method for Energy-Efficient Computation Offloading in Mobile Edge Computing. In: 2020 29th International Conference on Computer Communications and Networks (ICCCN). pp 1–7. https://doi.org/10.1109/icccn49398.2020.9209738
40. Alfakih T, Hassan MM, Gumaei A, Savaglio C, Fortino G (2020) Task Offloading and Resource Allocation for Mobile Edge Computing by Deep Reinforcement Learning Based on SARSA. IEEE Access 8:54074–54084
41. Altahhan A (2020) True Online TD(λ)-Replay An Efficient Model-free Planning with Full Replay. In: 2020 International Joint Conference on Neural Networks (IJCNN). IEEE, Glassglow. pp 1–7

## Publisher's Note