Parked Vehicles Assisted Task Offloading Based on Deep Reinforcement Learning

Guangting Lu[1], Zhuojun Lv[1], Zheng Zhang[1] and Feng Zeng[1,*]

[1]School of Computer Science and Engineering, Central South University, Changsha 410083, China

Abstract

As demand continues to grow, edge servers are increasingly constrained by their limited com-puting resources. In addition, large-scale deployment of edge servers will inevitably lead to unnecessary waste of resources. To further expand the resources of Vehicular Edge Computing, in this paper, we point out that the idle resources of parked vehicles can be integrated to assist edge servers in processing offload tasks and propose a computing offloading framework for parking cluster collaboration. In this framework, the computing task of each vehicle is composed of multiple subtasks that have dependencies between each other. To efficiently manage hetero-geneous resources in the framework, a layered offloading method based on deep reinforcement learning is proposed to minimize the average completion time of all vehicles. Simulation results show that the proposed method has better performance than the other three baseline methods in terms of task processing time and task execution success rates.

Keywords

Edge Computing, Deep Reinforcement Learning, Dependent Task Offloading, Parked Vehicles

1. Introduction

In Vehicular Edge Computing, due to the very limited computing and storage resources, vehicles are often unable to process some computationally intensive and latency-sensitive intelligent applications locally, such as digital twins, augmented reality, etc. [1]. As a complementary solution, cloud computing meets the service needs of some computing-intensive tasks by offloading applications to cloud servers for execution [2]. However, cloud servers are usually located in data centers far away from vehicles, which leads to higher bandwidth consumption and increased communication latency during task offloading, thus affecting the performance of computation offloading. To this end, Vehicular Edge Computing (VEC) came into being. Its idea is to provide highly reliable and low-latency computing offload services to vehicle users by deploying edge servers on both sides of the road [3]. However, as demand continues to grow, edge servers are increasingly constrained by their limited computing resources. To optimize the resource allocation of a single-edge server, some scholars [4, 5, 6] have devoted themselves to taking one or more performance indicators as the optimization goal and modeling the computation offloading problem as the best optimization model. However, as the number of requests for computing offload services increases, the approach that simply relies on optimizing

The 6th International Symposium on Advanced Technologies and Applications in the Internet of Things (ATAIT 2024), August 19-22, 2024, Kusatsu, Shiga, Japan

*Corresponding author.

fengzeng@csu.edu.cn (F. Zeng)

the resource configuration of a single-edge server still has the problem of insufficient resources. To this end, some scholars [7, 8] have explored the resource scheduling problem across multiple edge servers and achieved resource collaborative scheduling among multiple edge servers by building a resource load balancing model. However, due to the temporal and spatial differences in the spatial distribution of vehicles, servers in the same area often face similar load pressures. Thus, migrating computing tasks to distant edge servers for collaborative processing may result in higher service delays.

Considering that idle computing resources near vehicles are ubiquitous and do not require additional deployment, some scholars have studied the use of neighboring vehicles to expand the capabilities and service scope of edge computing. Some works [9, 10] have studied how to use mobile vehicles as edge servers to assist in offloading. However, the rapid movement of vehicles may cause frequent changes in communication channels and interruptions in task offloading, which in turn affects the performance of computation offloading. Liu et al. [11] investigated the availability of parked vehicles and pointed out that parked vehicles have the characteristics of dense distribution, long parking time, and fixed location, which can provide stable network connections and computing resources, making them potential computing devices in the infrastructure [12]. Based on this, Reis et al. [13] proposed adding parked vehicles as static nodes to VEC, forming the concept of parking assistance and developing it into a new type of hybrid network. To alleviate the computing pressure on edge servers, some scholars have studied how to use the computing and communication resources of parked vehicles to collaborate with the edge for computing offloading. Kadhim et al. [14] integrated Software Defined Networks and fog computing, used parked vehicles as auxiliary nodes for fog computing, and proposed a load balancing mechanism. Pham et al. [15] studied partial computation offloading in parked vehicle-assisted multi-access edge computing and used the subgradient method to optimize the offloading ratio and resource allocation. Ma et al. [16] organized parked vehicles into parking clusters and theoretically proved the long-term stability of the number of vehicles in a parking cluster. Zhao et al. [17] organized parked vehicles into static service nodes in a scenario where edge infrastructure was limited and proposed a task offloading algorithm based on reinforcement learning.

However, the scenarios considered in the above research on parking vehicle-assisted edge computing are too idealistic, and the main scenario considered is the collaborative offloading problem between a single edge server and multiple unassociated parked vehicles. In addition, the dependencies of subtasks are not taken into account, which limits the potential of parallel processing in edge computing and makes it difficult to meet the needs for low-latency services. In the face of the shortcomings and challenges of existing research, the main contributions of this work are summarized as follows:

- We propose to integrate parked vehicles into parking clusters and design a dependent task computation offloading framework for multiple parking clusters to collaborate with a single edge server.
- We propose a deep reinforcement learning algorithm based on a multi-actor and single-critic network architecture to minimize the average completion time of the application. Guided by a single critic network, multiple actor networks efficiently divide the decision action space into two layers: the first layer determines the location of task execution

(locally, on edge servers, or in parking clusters); the second layer selects the specific parked vehicle to execute the task. This approach not only reduces the action space that each actor network handles, but also significantly improves the overall performance and efficiency of the system.

The remainder of the paper is organized as follows: Section II shows the system model studied in this paper, including: scenario modeling, task modeling, computational modeling, and the formalization of the optimization objectives established in this paper. Section III introduces the Double Actor-Layered Deep Deterministic Policy Gradient (DALDDPG) algorithm. We first model the decision-making process of the scenario studied in this paper as a Markov decision process. Then, the network structure of the DALDDPG algorithm, the update method of each network, and the DALDDPG pseudocode are introduced. Section IV evaluates the effectiveness of the proposed algorithm by comparing it with existing algorithms. Finally, we conclude this paper in Section V.

2. System model

As illustrated in Figure 1, we consider a computation offloading scenario involving multi-parked vehicles, multi-task vehicles, and a single VEC server. To effectively manage the resources of parked vehicles and facilitate task cooperation among them, we group parked vehicles into multiple Parking Vehicle Clusters (PVC) and designate a Cluster Management Vehicle (CMV) within each cluster. The primary responsibility of the CMV is to maintain basic information about the vehicles within the cluster and report this information to the Road Side Unit (RSU) of its area via V2I communication regularly. Considering that the communication cost of the CMV with exterior entities is usually greater than its communication cost within the cluster, we do not consider the communication delay between the CMV and other vehicles within the cluster. Furthermore, the CMV is regarded as a bridge for the entire cluster to communicate with the exterior, responsible for accurately forwarding messages to the targeted parked vehicles.

We posit that there are M PVCs on the road, denoted as $\{P_m|m = 1, 2, ..., M\}$. In each P_m, there are G parked vehicles, where $Q_{m,1}$ represents the CMV of the m-th PVC, and $Q_{m,g}$ represents the g-th vehicle in the m-th PVC. The computing resource set for the parked vehicles in each P_m is represented as $\{F_{m,g}|g = 1, 2, ..., G\}$, where $F_{m,g}$ signifies the CPU clock frequency of the g-th vehicle in the m-th PVC. Moreover, there are N task vehicles on the road, which can either connect to the RSU in their coverage range through V2I to access the VEC server, or connect to the PVC through V2V.

2.1. Task model

In this paper, we model the dependent subtask relationships derived from the application K_n, generated by vehicle n, as $G_n = (V_n, E_n)$. Here, $V_n = \{v_i^n|0, 1, ..., l, l + 1\}$ represents the $I + 2$ subtasks of K_n. Specifically, v_0 and v_{I+1} represent the virtual entry and exit subtasks of K_n, respectively. These two virtual tasks are established to ensure that K_n can start and end on vehicle n. Each edge within the set E_n denotes a dependency relationship between subtasks of K_n. Specifically, an edge $e(v_i^n, v_j^n) \in E_n$ indicates that the result from v_i^n must

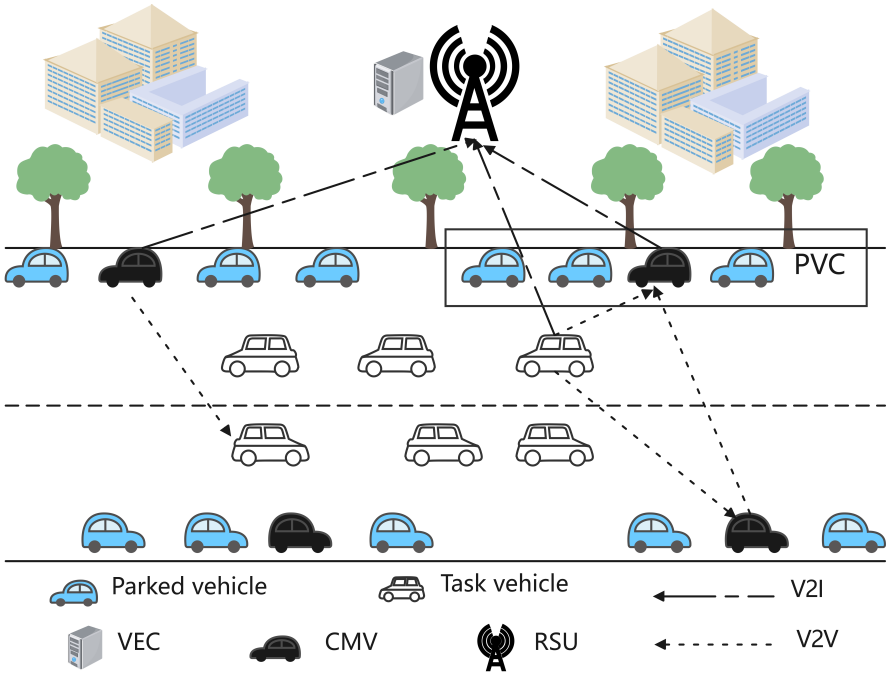

Figure 1: System Model.

be transmitted to v_j^n before v_j^n can commence its execution. The tuple $\{z_i^n, o_i^n, d_i^n, t_{i.max}^n\}$ is defined to characterize the i-th subtask v_i^n of K_n, where z_i^n is the input data volume for v_i^n, o_i^n is the output data volume resulting from executing v_i^n, d_i^n represents the number of CPU cycles required to execute v_i^n, and $t_{i.max}^n$ is the maximum tolerable delay for v_i^n.

The set of computing devices, available for offloading services within the communication range of the task vehicle, is denoted by $S = \{0, 1, ..., M, M+1\}$, where S_0 represents the task vehicle itself, S_1 to S_M represent PVCs, and S_{M+1} represents the VEC server. The decision variable $X_{i,m}^n$ is used to indicate whether subtask v_i^n is offloaded to the computing device S_m. This variable can be defined as follows:

$$X_{i,m}^n = \begin{cases} 0 & \text{if subtask } \nu_i^n \text{ is no offoaded to device } S_m, \\ 1 & \text{if subtask } \nu_i^n \text{ is offoaded to device } S_m. \end{cases} \tag{1}$$

The decision variable $Y_{i,g}^{n,m}$ indicates whether subtask v_i^n is executed on parked vehicle $Q_{m,g}$ in PVC P_m. This variable can be defined as follows:

$$Y_{i,g}^{n,m} = \begin{cases} 0 & \text{if subtask } v_i^n \text{ is executed n}ot \text{ on parked vehicle } Q_{m,g} \text{ in } P_m, \\ 1 & \text{if subtask } v_i^n \text{ is executed on parked vehicle } Q_{m,g} \text{ in } P_m. \end{cases} \tag{2}$$

2.2. Computational model

In this paper, we assume that the VEC server, parked vehicles, and local vehicles can only handle one subtask at a time and that each subtask can only be processed on a computing device.

2.2.1. Local computing model

When $X_{i,0}^n = 1$, the task v_i^n is executed locally. The local ready time, $RT_{n,i}^{loc}$, for v_i^n, is the time when all the predecessor tasks of v_i^n have been executed and their results have been trans-mitted back to the local vehicle. $RT_{n,i}^{loc}$ can be expressed as follows:

$$RT_{n,i}^{loc} = \max_{v_i^n \in pre_i^n} \{FT_{n,j} + T_{j,i}^{comm,n}\} \tag{3}$$

where pre_i^n is the set of all predecessor tasks of v_i^n; $FT_{n,j}$ refers to the completion time for v_j^n on the designated computing device based on the offloading decision; $T_{j,i}^{comm,n}$ is the time required to transmit the execution results of v_j^n back to v_i^n. When v_i^n is ready locally, it may not immediately be scheduled for execution due to the need to account for local queuing execution times. The completion time of v_i^n, when executed locally, is denoted as $FT_{n,i}^{loc}$ and can be expressed as follows:

$$FT_{n,i}^{loc} = max\{RT_{n,i}^{loc}, AT_{n,i}^{loc}\} + \frac{d_i^n}{f^{loc}} \tag{4}$$

where $AT_{n,i}^{loc}$ stands for the earliest possible scheduling time for the local execution of v_i^n, and f^{loc} denotes the computing capacity of the local terminal.

2.2.2. VEC computing model

When $X_{i,M+1}^n = 1$, the task v_i^n is carried out on the VEC server. The transmission delay for vehicle n, when uploading data z_i^n to the VEC server, can be represented as:

$$T_{n,i}^{tran,vec} = \frac{z_i^n}{r_n^{V2I,UP}} \tag{5}$$

The ready time of v_i^n on the VEC server, denoted as $RT_{n,i}^{vec}$, comprises two components: the upload time for v_i^n to the VEC server, and the time at which all precursor tasks of v_i^n get completed and their results are delivered back. Therefore, $RT_{n,i}^{vec}$ can be expressed as:

$$RT_{n,i}^{vec} = max\left\{T_{n,i}^{tran,vec}, \max_{v_j^n \in pre_i^n} \{FT_{n,j} + T_{j,i}^{comm,n}\}\right\} \tag{6}$$

Once task v_i^n is ready on the VEC server, it may not necessarily be immediately scheduled for execution due to the queuing execution time on the VEC server. The completion time of v_i^n, when executed on the VEC server, is denoted as $FT_{n,i}^{vec}$ and can be expressed as:

$$FT_{n,i}^{vec} = max\left\{RT_{n,i}^{vec}, AT_{n,i}^{vec}\right\} + \frac{d_i^n}{f^{vec}} \tag{7}$$

where $AT_{n,i}^{vec}$ is the earliest possible scheduling time for v_i^n on the VEC server, and f^{vec} denotes the computing capacity of the VEC server.

2.2.3. PVC computing model

When $X_{i,m}^n = 1$ (where $m \neq 0$ and $m \neq M+1$), task v_i^n is executed on the parked vehicle within PVC P_m. The transmission delay for vehicle n, when uploading data z_i^n to P_m, can be represented as:

$$T_{n,i,m}^{tran,pvc} = \frac{z_i^n}{r_{n,m}^{V2V}} \tag{8}$$

The ready time $RT_{n,i,m}^{pvc}$ for task v_i^n on the parked vehicle within P_m includes two parts: the time required to upload v_i^n to PVC P_m, and the time when all precursor tasks of v_i^n have been completed and their results are delivered back. Therefore, $RT_{n,i,m}^{pvc}$ can be expressed as:

$$RT_{n,i}^{vec} = max\left\{T_{n,i}^{tran,vec}, \max_{v_j^n \in pre_i^n}\{FT_{n,j} + T_{j,i}^{comm,n}\}\right\} \tag{9}$$

Once task v_i^n is ready on the parked vehicle in P_m, it may not immediately be scheduled for execution due to the queuing execution time on the parked vehicle. The completion time $FT_{n,i,m}^{pvc}$ of v_i^n, when executed on the parked vehicle in P_m, can be expressed as:

$$FT_{n,i,m}^{pvc} = max\left\{FT_{n,i,m}^{pvc}, \sum_1^G Y_{i,g}^{n,m} AT_{n,i}^{m,g}\right\} + \frac{d_i^n}{\sum_1^G Y_{i,g}^{n,m} F_{m,g}} \tag{10}$$

where $AT_{n,i}^{m,g}$ denotes the earliest scheduling time for v_i^n to be executed on the g-th parked vehicle in P_m, and $F_{m,g}$ represents the computing capacity of the g-th parked vehicle within PVC P_m.

2.3. Problem formulation

The actual completion time for subtask $v_i^n of$ application K_n, denoted as $RT_{n,i}$, based on the current offloading decisions, can be expressed as:

$$RT_{n,i} = X_{i,0}^n FT_{n,i}^l + X_{i,M+1}^n FT_{n,i}^{VEC} + \sum_1^M X_{i,m}^n FT_{n,i,m}^{PVC} \tag{11}$$

The actual completion time T_n^{total} for K_n is the actual completion time of the virtual exit subtask v_{I+1}^n and can be represented as:

$$T_n^{total} = FT_{n,\mathrm{I}+1} \tag{12}$$

The main objective of this work is to minimize the average completion time of system applications under the condition that each task is completed within its maximum tolerable delay. The optimization problem is formulated as follows:

$$\begin{cases} min \frac{1}{N} \Sigma_{n=1}^{N} T_n^{total} \\ \text{C1:} \sum_{\mathrm{m=0}}^{\mathrm{M+1}} \mathrm{X_{i,m}^{n}} \in \{0,1\}, \sum_{\mathrm{g=1}}^{\mathrm{G}} Y_{\mathrm{i,g}}^{\mathrm{n,m}} \in \{0,1\} \\ \text{C2:} T_n^{total} \leq t_{max}^n, RT_{n,i} \leq t_{\mathrm{i,max}}^n \end{cases} \tag{13}$$

where $C1$ stipulates that each task can only be executed on a single computing device, and $C2$ ensures that the actual completion time of each application and its respective subtasks remains within their maximum tolerable delay.

3. Design of algorithm

Given that the above optimization problem is a complex mixed integer linear programming problem, traditional optimization algorithms struggle to effectively solve. Moreover, to efficiently manage heterogeneous resources in proposed scenario, we propose a layered task offloading scheduling algorithm based on deep reinforcement learning with a multi-actor and single-critic network. For this purpose, we first model the offloading scheduling process for dependent tasks as a Markov Decision Process (MDP). Below, we provide the formal expressions for the state space, action space, and reward function in the MDP.

State. At time t, the Actor_1 network in the first layer is responsible for offloading the subtasks of application K_n to either the local, a PVC, or the VEC server. The local state $o_{t,n}^1$ observed by the Actor_1 network includes four main components: the position of the CMV, the available computing resources of parked vehicles, the sequence of tasks that have already been scheduled, and the collection of task priority sequences. Therefore, $o_{t,n}^1$ can be abstractly defined as follows:

$$o_{t,n}^1 = \{P^{pvc}, F^{pvc}, Prio\_Seq_t^n, Done\_Seq_t^n\} \tag{14}$$

The Actor_2 network of the second layer is responsible for offloading the subtasks of application K_n to specific parked vehicles for execution. The local state $o_{t,n}^2$ observed by the Actor_2 network includes three main parts: the computing resources available of the parked vehicles, the processing time required for tasks pending in the compute queue of the parked vehicle, and the set of task priority sequences. Therefore, $o_{t,n}^2$ can be abstractly defined as follows:

$$o_{t,n}^2 = \left\{F^{pvc}, AT^{PVC}, Prio\_Seq_t^n\right\} \tag{15}$$

Actions. In the layered action space, for task v_i^n, the first layer action space $A_{n,i}^1$ that the Actor_1 network can take is represented as:

$$A_{n,i}^1 = \{x_{n,i}^0, x_{n,i}^1, \ldots, x_{n,i}^m, \ldots, x_{n,i}^{M+1}\} \tag{16}$$

where $A_{n,i}^1$ determines the allocation layer level of task v_i^n; if $x_{n,i}^0 = 1$, v_i^n is executed locally; if $x_{n,i}^{M+1} = 1$, v_i^n is executed on the VEC server; if $X_{i,m}^n = 1$ (where $m \neq 0$ and $m \neq M+1$), v_i^n is executed on the m-th PVC. Based on the decisions of the first layer, the second layer action space $A_{n,i}^2$ that the Actor_2 network can take is defined as:

$$A_{n,i}^2 = \left\{y_{n,i}^1, y_{n,i}^2, \ldots, y_{n,i}^g, \ldots, y_{n,i}^G\right\} \tag{17}$$

where $A^2_{n,i}$ specifies that, within the layer determined by $A^1_{n,i}$,task v^n_i is further offloaded to a specific parked vehicle, and if $y^g_{n,i} = 1$, v^n_i is executed on the g-th parked vehicle.

Rewards. After executing the joint action $A_{n,i} = \left\{A^1_{n,i}, A^2_{n,i}\right\}$ under the global state $S_{n,t} = \{o^1_{t,n} \cup o^2_{t,n}\}$,the Agent receives an immediate reward R_t from the environment, which can be expressed as follows:

$$R_t = \frac{T^{total}_n(Done\_seq^n_{1:t}) - T^{total}_n(Done\_seq^n_{1:t+1})}{T^{total}_n(local)} \tag{18}$$

where $T^{total}_n(Done\_Seq^n_{1:t})$ denotes the time spent on the subgraph of tasks that have been scheduled under state $S_{n,t}$, and $T^{total}_n(local)$ represents the delay when all scheduled tasks are executed locally.

The Double Actor-Layered Deep Deterministic Policy Gradient (DALDDPG) algorithm comprises six networks: the Actor$_1$ network, $\pi^1(o^1|\theta^1)$; the Actor$_2$ network, $\pi^2(o^2|\theta^2)$; and their respective target networks, $\pi'_1(o^1|\theta'_1)$ and $\pi'_2(o^2|\theta'_2)$. Additionally, it includes a Critic network, $Q(S, A|\omega)$, and a corresponding target network, $Q'(S, A|\omega')$. In the decision-making process, the Actor$_1$ and Actor$_2$ networks independently make first-layer and second-layer decisions based on their local states. The Agent subsequently combines these two decisions (A^1_t, A^2_t) into a joint decision, A_t, which is then executed. Following this execution, the global state, S_t, and local states, o^1_t and o^2_t, move to the next state and provide the Agent with an immediate reward, R_t. Then the Agent stores the single set of experience (S_t, A_t, S_{t+1}, R_t) from the interaction with the environment in the sample pool. During the training phase, a batch of samples is periodically drawn from the experience pool, and the Q^- values for each sample i are calculated through the target network, $Q'(S, A|\omega')$.

$$Q^-_i = R_i + \gamma Q'\left(S_{i,next}, A_{i,next}|\omega'\right) \tag{19}$$

where $A_{i,next} = \pi'_1\left(o^1_{i,next}|\theta'_1\right) \cup \pi'_2\left(o^2_{i,next}|\theta'_2\right)$, and γ is the discount factor.

In this paper, we minimize the loss function $L(\omega)$ using gradient descent, based on the Temporal Difference algorithm, to update the weight parameter ω of the Critic. $L(\omega)$ can be expressed as follows:

$$L(\omega) = \frac{1}{z}\sum_{i=1}^{Z}\left(Q^-_i - Q(S_i, A_i|\omega)\right)^2 \tag{20}$$

where Z represents the number of samples drawn from the sample pool.

Under the evaluation of the $Critic$, we employ gradient ascent to update the parameters of the Actor$_1$ and Actor$_2$ networks. The policy gradients are expressed as follows:

$$\nabla_{\theta^1} J = \frac{1}{z}\sum_{i=1}^{Z}\left(\nabla_A Q(S_i, A_i|\omega)\nabla_{\theta^1}\pi^1\left(o^1_i|\theta^1\right)\right) \tag{21}$$

$$\nabla_{\theta^2} J = \frac{1}{z}\sum_{i=1}^{Z}\left(\nabla_A Q(S_i, A_i|\omega)\nabla_{\theta^2}\pi^2\left(o^2_i|\theta^2\right)\right) \tag{22}$$

To update the parameters of the target network, a soft update strategy is employed. The updates for all target networks are expressed as follows:

$$\begin{cases} \theta_1' = \lambda\theta^1 + (1-\lambda)\theta_1' \\ \theta_2' = \lambda\theta^2 + (1-\lambda)\theta_2' \\ \omega' = \lambda\omega + (1-\lambda)\omega' \end{cases} \tag{23}$$

where λ is the soft update coefficient.

Algorithm 1 DALDDPG algorithm

Randomly initialize: $\theta^1, \theta^2, \omega, \theta', \theta_2', \omega'$
Initialize the sample pool D, learning rate, λ, etc
for each episode = 1 to E **do**
  Initialize S, o^1, o^2
  for t = 1 to T **do**
    Actor$_1$ selects action A^1 based on the current o^1
    Actor$_2$ selects action A^2 based on the current o^2
    Agent combines action $A \leftarrow (A^1, A^2)$, and executes
    Compute reward R, $S \leftarrow S'$, $o^1 \leftarrow o_1'$, $o^2 \leftarrow o_2'$
    Store the transitions (S, A, R, S') in D
    if len(D) > 100 **then**
      Sample a batch of transitions (S, A, R, S') from D
      Update $Critic$ parameter ω using equation (20)
      Update Actor$_1$ parameter θ^1 using equation (21)
      Update Actor$_2$ parameter θ^2 using equation (21)
      Every C steps, update parameters θ_1', θ_2', ω' using equation (23)
    end if
  end for
end for

4. Simulation and result analysis

We use Python 3.7 and TensorFlow 2.0 for simulations. The simulation scenario considers a 400-meter road populated with $N \in [5, 30]$ task vehicles, alongside a VEC server, and $M \in [3, 6]$ PVCs, each containing $Q \in [3, 7]$ parked vehicles. The computing capacities of the task and parked vehicles are $(0, 0.5]$ and $[1, 2]$ GHz, respectively. The VEC server has a computing capacity of 6 GHz. The sample pool capacity is 2000, and the batch size for sampling is 64.

To evaluate the performance of the proposed strategy, we compare the following three offloading strategies:

- **RS:** Tasks are randomly assigned to be executed on the vehicle locally, on the VEC server, or on a parked vehicle;
- **HUDQN:** Tasks are executed according to the offloading decisions made in the first layer;

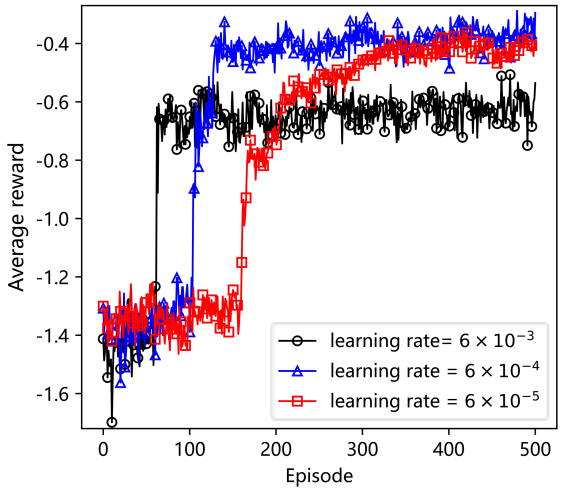

Figure 2: Convergence Performance of DALDDPG.

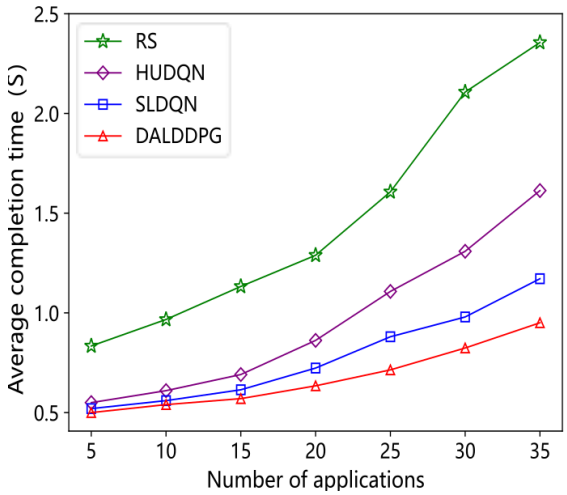

Figure 3: Average Completion Time for Different Numbers of Applications.

- **SLDQN:** The two-layer offloading decisions are consolidated into a single-layer framework.

As depicted in Figure 2, as the number of training epochs increases, the rewards under various learning rates increase and stabilize. To balance convergence speed with system stability, we adopt a learning rate of 6×10^{-4} for model training.

As shown in Figure 3, the average completion time of each strategy escalates with the increase in the number of applications, yet the proposed strategy consistently exhibits the lowest completion time. Compared to the other three strategies, our strategy reduces the average completion time by 11.47%, 25.41%, and 51.01% on average, respectively. Figure 4 shows that as the number of PVCs increases, the average completion time for each strategy decreases,

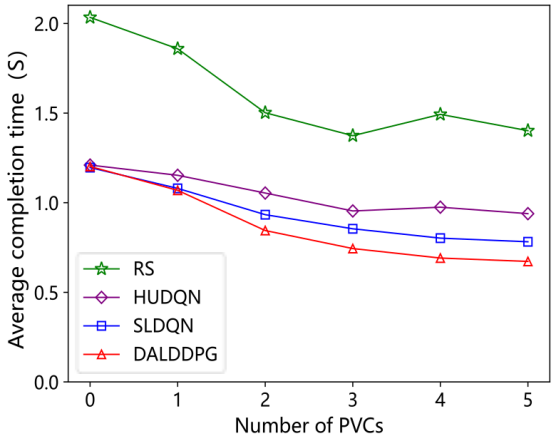

Figure 4: Average Completion Time for Different Numbers of PVCs.

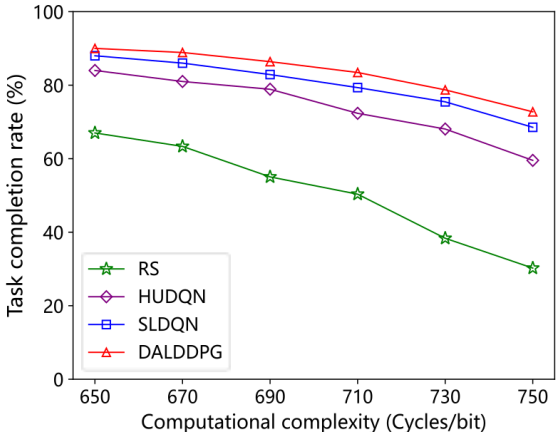

Figure 5: Task Completion Rates for Different Computational Complexities.

with the proposed strategy performing the best. Compared to the other three strategies, the proposed strategy reduces the average completion time by 15.42% to 26.58% on average. As depicted in Figure 5, with rising task computational complexity, the task completion rates for all strategies gradually decrease, but the proposed strategy maintains the highest completion rate. Compared to the other three strategies, the proposed strategy enhances the completion rate by an average of 42.52%, 13.21%, and 4.21%, respectively. The above improvements are because the proposed strategy has considered well the heterogeneity of parked vehicle resources and adopted a layered task offloading scheduling framework to optimize task allocation, which significantly improves the performance and efficiency of the system.

5. Conclusion and future work

In this paper, we design a dependent task scheduling framework, which is composed of multiple parking clusters cooperating with a single edge server. In addition, we propose a deep reinforcement learning algorithm based on a multi-actor and single-critic network architecture to minimize the average completion time of the application. Simulation results show that the proposed algorithm has better performance than the other three baseline algorithms in terms of task processing time and task execution success rates. Future work will explore task offloading and resource scheduling within a VEC system assisted by multi-parking clusters, while also considering the energy consumption cost of parked vehicles.

References

[1] Y. Peng, B. Shi, T. Jiang, X. Tu, D. Xu, K. Hua, A survey on in-vehicle time-sensitive networking, IEEE Internet of Things Journal 10 (2023) 14375–14396.

[2] S. Lu, W. Shi, Vehicle computing: Vision and challenges, Journal of Information and Intelligence 1 (2023) 23–35.

[3] M. Raeisi-Varzaneh, O. Dakkak, A. Habbal, B.-S. Kim, Resource scheduling in edge computing: Architecture, taxonomy, open issues and future research directions, IEEE Access 11 (2023) 25329–25350.

[4] H. Zhou, K. Jiang, X. Liu, X. Li, V. C. Leung, Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing, IEEE Internet of Things Journal 9 (2021) 1517–1530.

[5] J. Chen, H. Xing, Z. Xiao, L. Xu, T. Tao, A drl agent for jointly optimizing computation offloading and resource allocation in mec, IEEE Internet of Things Journal 8 (2021) 17508–17524.

[6] A. M. Seid, G. O. Boateng, S. Anokye, T. Kwantwi, G. Sun, G. Liu, Collaborative computation offloading and resource allocation in multi-uav-assisted iot networks: A deep reinforcement learning approach, IEEE Internet of Things Journal 8 (2021) 12203–12218.

[7] W. Fan, M. Hua, Y. Zhang, Y. Su, X. Li, B. Tang, F. Wu, Y. Liu, Game-based task offloading and resource allocation for vehicular edge computing with edge-edge cooperation, IEEE Transactions on Vehicular Technology 72 (2023) 7857–7870.

[8] P. Li, W. Xie, Y. Yuan, C. Chen, S. Wan, Deep reinforcement learning for load balancing of edge servers in iov, Mobile Networks and Applications 27 (2022) 1461–1474.

[9] K. Xiong, S. Leng, C. Huang, C. Yuen, Y. L. Guan, Intelligent task offloading for heterogeneous v2x communications, IEEE Transactions on Intelligent Transportation Systems 22 (2020) 2226–2238.

[10] W. Fan, Y. Su, J. Liu, S. Li, W. Huang, F. Wu, Y. Liu, Joint task offloading and resource allocation for vehicular edge computing based on v2i and v2v modes, IEEE Transactions on Intelligent Transportation Systems 24 (2023) 4277–4292.

[11] N. Liu, M. Liu, W. Lou, G. Chen, J. Cao, Pva in vanets: Stopped cars are not silent, in: 2011 Proceedings IEEE INFOCOM, IEEE, 2011, pp. 431–435.

[12] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, S. Chen, Vehicular fog computing: A viewpoint

of vehicles as the infrastructures, IEEE Transactions on Vehicular Technology 65 (2016) 3860–3873.
[13] A. B. Reis, S. Sargento, O. K. Tonguz, Parked cars are excellent roadside units, IEEE Transactions on Intelligent Transportation Systems 18 (2017) 2490–2502.
[14] A. J. Kadhim, J. I. Naser, Proactive load balancing mechanism for fog computing supported by parked vehicles in iov-sdn, China Communications 18 (2021) 271–289.
[15] X.-Q. Pham, T. Huynh-The, E.-N. Huh, D.-S. Kim, Partial computation offloading in parked vehicle-assisted multi-access edge computing: A game-theoretic approach, IEEE Transactions on Vehicular Technology 71 (2022) 10220–10225.
[16] C. Ma, J. Zhu, M. Liu, H. Zhao, N. Liu, X. Zou, Parking edge computing: Parked-vehicle-assisted task offloading for urban vanets, IEEE Internet of Things Journal 8 (2021) 9344–9358.
[17] H. Zhao, J. Hua, Z. Zhang, J. Zhu, Deep reinforcement learning-based task offloading for parked vehicle cooperation in vehicular edge computing, Mobile Information Systems 2022 (2022) 9218266.