Dependency-Aware Task Offloading for Vehicular Edge Computing with End-Edge-Cloud Collaborative Computing

Guozhi Liu
Southwest Forestry University
Fei Dai (daifei@swfu.edu.cn)
Southwest Forestry University
Bi Huang
Southwest Forestry University
Zhenping Qiang
Southwest Forestry University
Shuai Wang
Southwest Forestry University
Lecheng Li
Southwest Forestry University

Research Article

Keywords: Task Offloading, Dependency-Aware, Vehicular Edge Computing, End-Edge-Cloud collaborative computing, Deep Deterministic Policy Gradient, Deep Reinforcement Learning

Posted Date: August 2nd, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1905904/v1

RESEARCH

Dependency-Aware Task Offloading for Vehicular Edge Computing with End-Edge-Cloud Collaborative Computing

Guozhi Liu[1], Fei Dai[1*], Bi Huang[1], Zhenping Qiang[1], Shuai Wang[1] and Lecheng Li[1]

Abstract

Vehicular edge computing (VEC) is emerging as a new computing paradigm to improve the quality of vehicular services and enhance the capabilities of vehicles. It aids vehicles execution tasks with low latency by deploying computing and storage resources close to the vehicle side. However, the traditional task offloading schemes waste the fine-grained offloading opportunities for subtasks by neglecting the decomposability and dependency of tasks. Furthermore, the continuous control problem caused by the learning-based offloading schemes should be taken into account. In this paper, we proposed an efficient dependency-aware task offloading scheme, which minimizes the average processing delay of tasks by fully utilizing the end-edge-cloud collaborative computing. Specifically, first, the directed acyclic graph (DAG) technique is utilized to model the inter-subtask dependency, which establishes the priority of subtask execution. Second, a task offloading algorithm based on Deep Deterministic Policy Gradient (DDPG) in Reinforcement Learning (RL) was proposed to obtain the optimal offloading strategy in a dynamic environment, which efficiently solves continuous control problems and helps reach fast convergence. Finally, we conduct extensive simulation experiments, and the experimental results show that the proposed dependency-aware task offloading scheme can achieve a good performance.

Keywords: Task Offloading; Dependency-Aware; Vehicular Edge Computing; End-Edge-Cloud collaborative computing; Deep Deterministic Policy Gradient; Deep Reinforcement Learning

Introduction

The Internet of Vehicles (IoV) is an evolution of conventional Vehicle Ad-hoc Networks, which goals to effectively improve the safety and comfort of travel [1, 2]. In IoV, an intelligent vehicle is capable of vehicle-to-everything communication and can run various applications [2], such as collision warning [3], automatic driving [4], and auto navigation [4]. Unfortunately, these applications are inherently computation-intensive and latency-critical tasks [5, 6]. As a result, it is challenging to execute them on vehicles with low latency that have limited resources.

In response to the on-board equipment cannot meet the service requests with low latency in the IoV, traditional wisdom resorts to the powerful cloud server for intensive task computation [7]. In this case, cloud computing reduces service processing delay by leveraging abundant computing resources. Unfortunately, this approach may result in unpredictable communication latency caused by the long wide-area massive data transmission between vehicles and the cloud server.

To compensate for the shortcomings of could computing in data transmission, vehicular edge computing (VEC) is emerging as a novel computing paradigm [8, 9], which greatly reduces the distance of data transmission by deploying edge servers in roadside units or base stations relatively close to roads, thus largely enhancing the users' experience.

Although the VEC can further reduce data transfer latency and enhance the quality of service, there are still defects in only using mobile edge computing (MEC) servers to process tasks. The limited computing and storage ability of MEC servers cannot ensure load balancing [10, 11]. Therefore, current research suggests that efficient end-edge-cloud collaborative computing by fully utilizing vehicles, MEC servers, and the cloud server resources is necessarily taken into account [12]. Figure 1 shows three different offloading approaches for five tasks at the current time epoch. As shown in Figure 1a, processing the five

*Correspondence: daifei@swfu.edu.cn
[1]School of Big Data and Intelligent Engineering, Southwest Forestry University, Kunming, China
Full list of author information is available at the end of the article

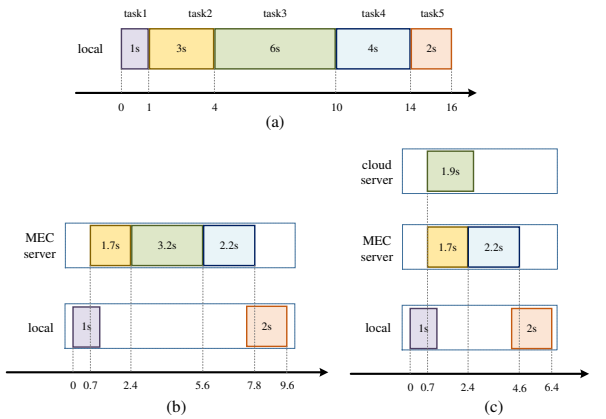

Figure 1 Three different offloading strategies for five tasks

tasks on a vehicle may cause a long processing delay due to the limited computation capability of the vehicle. Then, to further reduce the delay, we offload some tasks (task2, task3, and task4) to a MEC server for execution. As shown in Figure 1b, it is a better offloading approach due to the powerful MEC server. In addition, we can see that a more reasonable offloading scheme is shown in Figure 1c due to the scheme fully utilizing vehicles, edge servers, and cloud server computing resources. In other words, we offload some tasks to the cloud server and some to the MEC server for execution at the same time. As shown in Figure 1c, when task2 and task4 are processed on MEC server, task3 is processed on the cloud server. As a result, the minimum processing delay of these five tasks is obtained. Therefore, the end-edge-cloud collaborative computing needs to be fully considered in our task offloading approach.

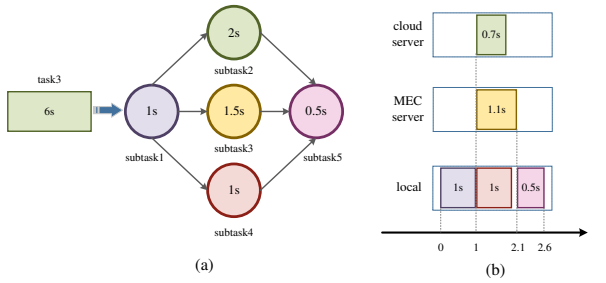

Figure 2 (a) DAG of a task. (b) Fine-grained offloading

Furthermore, it is well known that a typical application task consists of a series of subtasks [13, 14], which are originally designed to enable multithread processing [14]. As shown in Figure 2a, we divide the task3 of Figure 1 into five subtasks and use a directed acyclic graph (DAG) to describe the inter-subtask dependency. Figure 2b illustrates the dependency-aware task offloading approach, and we can see that this approach significantly reduces processing delay as subtask 2, subtask 3, and subtask 4 are processed in parallel on the cloud, an edge server, and a vehicle. Nevertheless, in the mast existing offloading works [15, 16, 17], these five subtasks are often packed and treated as an entire task, namely coarse-grained task offloading. The reason is that they are considered as an "inseparable" unit due to the dependency [14, 18]. As a result, the fine-grained offloading opportunities for subtasks are also wasted. As shown in Figure 2b, subtask 2, subtask 3, and subtask 4 are three parallel subtasks that can be processed concurrently at the vehicle, edge servers, and cloud server and the delay could be reduced. Therefore, to ensure that vehicular tasks can be completed in time, it is necessary to take subtask dependency into account for the task offloading problem considered.

The task offloading problem is a long-term non-convex optimization problem. Thus, the task offloading problem can be formulated as a Markov Decision Process (MDP) and solved by reinforcement learning (RL) or Deep reinforcement learning (DRL) approaches. Such as in [9, 19]. In [19], the authors proposed a task offloading approach based on Q-learning to jointly optimize processing order and computing mode selection for minimizing the total processing delay. In [9], the authors proposed an efficient task offloading approach based on the deep Q-network (DQN) to minimize the processing delay of tasks, which jointly considered the edge-cloud opportunities and the convergence of deep reinforcement learning. Nevertheless, most of the existing works [9, 19] quantized the decision variables into a discrete action space to achieve optimal computation offloading strategies due to Q-learning and DQN cannot solve the problem of continuous action space. As a result, it is not possible to obtain an optimal offloading strategy. Therefore, we use Deep Deterministic Policy Gradient (DDPG) [20] as an optimization algorithm. It can deal with high-dimensional input, disassociate data between tasks, and can output continuous actions, making DRL usable in more complex scenarios with large action spaces and continuous action spaces.

We can see that for the IoV scenarios with constrained resources [14], we need to identify an optimal offloading approach to fully utilize vehicles, MEC servers, and cloud server resources. Meanwhile, it is necessary to take subtask dependency into account to further minimize execution delays when designing task offloading strategies. However, it remains a challenge

to reveal the most appropriate fine-grained offloading opportunities in the dynamic environment. The main issues to be addressed for this is that: a) how to model the dependencies of subtasks, b) how to coordinate all the contending subtasks in the changing environment, c) which subtask to offload, d) which subtask to offload the MEC server, e) which subtask to offload to the cloud server, f) how to effectively allocate computation and communication resources of MEC servers, and g) which combinations of parallel subtasks provide more potential performance gains.

To tackle the above challenges, an efficient dependency-aware task offloading scheme for IoV with end-edge-cloud collaborative computing was proposed in this paper. Specifically, we first model the task as a directed acyclic graph (DAG). We then formulate the dependency-aware task offloading problem as an optimization problem. To minimize the average processing delay of subtasks, we then leverage deep reinforcement learning (DRL) to adaptively identify the optimal offloading strategy in the changing environment. In particular, DDPG is adopted to explore the topologies of inter-subtask dependency, which can effectively support a continuous action space.

The main contributions of this article are summarized as follows:

- Model inter-subtask dependency with directed acyclic graph (DAG) techniques to define the execution order of subtasks, in which the node denotes subtasks and the directed edges represent the inter-task dependency.
- Propose the policy-based DDPG approach to obtaining the optimal offloading strategy, which considers not only the dynamic environment but also the dependency among subtasks in the task DAGs. The DDPG method that adopts a double actor-critic framework can efficiently solve the continuous control problems. With this algorithm, we can find out the potential benefits of offloading specific subtasks and possible combinations.
- Conduct extensive experiments to evaluate the performance of our proposed scheme. The experimental results show that our scheme can greatly reduce the average processing time compared with baseline schemes.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 induces the system model and problem definition. Section 4 presents the design of the dependency-aware task offloading algorithm based on DDPG. Section 5 evaluates the performance of our proposed offloading scheme. Section 6 concludes this paper and outlines future work.

Related work

As a critical issue to be addressed in MEC, computation offloading has recently become a research hotspot [21], especially for the internet of vehicles. In the following, we introduce the most related research, which can be roughly classified into coarse-grained task offloading [22] and fine-grained task offloading [23].

Coarse-grained task offloading refers to the task offloading schemes without the consideration of the decomposability and dependency of tasks. Dai et al. [9] proposed an efficient task offloading approach based on DQN to minimize the processing delay of tasks, which jointly considered the edge-cloud opportunities and the convergence of deep reinforcement learning. Xu et al. [8] presented a game theory-based service offloading approach to minimize tasks processing latency of users, where both predictions of traffic flow and the allocation of resources are considered. Qiu et al. [24] propose a distributed and collective DRL algorithm, namely DC-DRL, to solve the multi-user computation offloading problem. He et al. [25] first introduce a novel Quality of Experience (QoE), and then proposed a task offloading algorithm based on DRL to improve QoE for IoV. Yao et al. [26] considered both energy consumption and processing delay and proposed a twin delayed deep deterministic policy gradient algorithm to achieve the optimal offloading strategy. However, these works merely considered entirely task offloading and ignored the parallelism among subtasks due to not considering the inter-subtask dependency, and these works cannot fully solve continuous control problems.

Fine-grained task offloading refers to the task offloading schemes with the consideration of the inter-subtask dependency. Liao et al. [23] consider both the dependency of applications and processing latency of applications and designed an algorithm to select the optimal offloading strategy. Shu et al. [14] investigate the problem of fine-grained task offloading in edge computing, where a distributed consensus algorithm was proposed to minimize the end-to-end task execution time. Liu et al. [27] considered both the processing latency of applications and the dependency of multiple tasks, where an efficient task offloading scheme was designed to reduce the average completion time of applications. Awada et al. [28] proposed a dependency-aware intelligent resource scheduling scheme to minimize the processing delay of tasks in a federated aerial edge computing system. Wang et al. [29] presented an intelligent task offloading scheme based on off-policy reinforcement learning to maximize the Quality-of-Service. However, these works only considered the resources of vehicles and MEC servers and did not fully utilize the rich resources of the cloud.

Unlike these above works, we investigate dependency-aware task offloading for IoV in end-edge-cloud collaborative computing environment. Our work is different from these works in three aspects: 1) we take task dependency into account to further minimize execution delays when designing task offloading strategies. 2) we fully utilize the resources of vehicles, MEC servers, and the cloud server. 3) we efficiently solve the continuous control problems.

System model and problem formulation

In this section, the system model of dependency-aware task offloading is designed, and then formulate the problem of dependency-aware task offloading as an optimization problem.

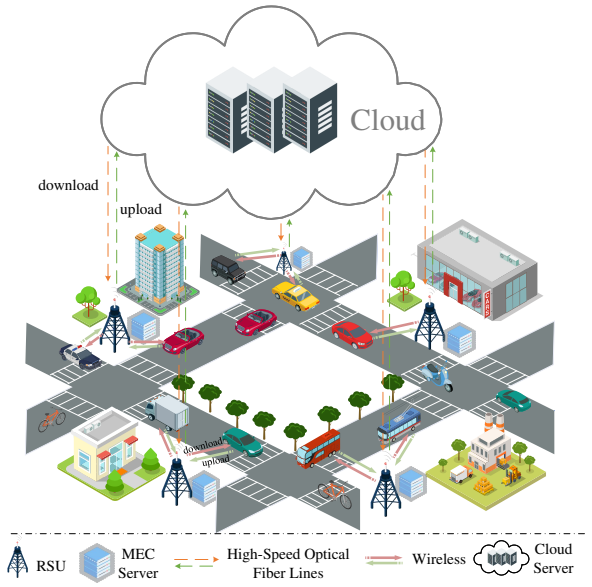

Figure 3 The architecture of VEC

Network model

As illustrated in Figure 3, we present the network model of dependency-aware task offloading for VEC, which includes RSUs, vehicles, MEC servers, and one remote cloud server.

We consider that there are $R = \{1, 2, ..., n, ..., N\}$ RSUs equally located along the road and let a set of denote the index set of RSUs. For each RSU has a fixed coverage area and is equipped with a MEC server. Each MEC server provides computing services for resource-constrained vehicles within the RSU's coverage area. In addition, each MEC server is connected to the remote cloud server through a high-speed optical fiber line.

We consider a set of vehicles, i.e., $V = \{1, 2, ..., k, ..., K\}$, and each vehicle is equipped with a limited amount of computing resources and access to an RSU via wireless networks. Furthermore, each vehicle has a set of delay-sensitive tasks to process at the current time epoch, and tasks either be performed on the vehicle or offloaded to the MEC server or cloud server for execution.

Task model

In this section, we consider the task model of the dependency-aware task offloading for VEC. Let $\mathcal{Q} = \{Q_1^1, Q_2^1, Q_3^2, ..., Q_m^k\}$ denote the set of tasks. The Q_m^k denotes mth task of the kth vehicle and we assume that each task Q_m^k can be divided into several interdependent subtasks and are modeled by a directed acyclic graph (DAG), i.e., $G = (\mathcal{I}, \mathcal{E})$, where $\mathcal{I}$ is the set of subtasks, $\mathcal{E}$ is the set of directed edges which defines the dependencies between subtasks, $I = |\mathcal{I}|$ denote the total number subtasks of task Q_m^k. Specifically, the successor subtask can only be executed after the predecessor subtask is executed. For example, a directed edge $(Q_{m,i}^k, Q_{m,j}^k)$ indicates that subtask $Q_{m,j}^k$ relies on the output of subtask $Q_{m,i}^k$.

To accurately illustrate the task dependencies, we show an example in Figure 4. The figure shows that task Q_m^k is divided into 9 subtasks (i.e., $Q_{m,1}^k$, $Q_{m,2}^k$, $Q_{m,3}^k$, $Q_{m,4}^k$, $Q_{m,5}^k$, $Q_{m,6}^k$, $Q_{m,7}^k$, $Q_{m,8}^k$, $Q_{m,9}^k$). Specifically, subtask $Q_{m,1}^k$ is entry task of task Q_m^k. Subtask $Q_{m,3}^k$ is predecessor task of $Q_{m,4}^k$ and $Q_{m,5}^k$. Subtask $Q_{m,4}^k$ and $Q_{m,5}^k$ are successor tasks of $Q_{m,3}^k$. So, subtask $Q_{m,4}^k$ and $Q_{m,5}^k$ are start executed until subtask $Q_{m,3}^k$ has been completed. Similarly, subtask $Q_{m,9}^k$ is exit task of task Q_m^k. Therefore, $Q_{m,9}^k$ is start executed until all predecessor subtasks (i.e., $Q_{m,1}^k$, $Q_{m,2}^k$, $Q_{m,3}^k$, $Q_{m,4}^k$, $Q_{m,5}^k$, $Q_{m,6}^k$, $Q_{m,7}^k$, $Q_{m,8}^k$, $Q_{m,9}^k$) has been completed. In addition, subtask $Q_{m,2}^k$ and $Q_{m,3}^k$ or $Q_{m,7}^k$ and $Q_{m,8}^k$ can execute in parallel, Similarly, subtask $Q_{m,4}^k$, $Q_{m,5}^k$, and $Q_{m,6}^k$ can also be executed in parallel.

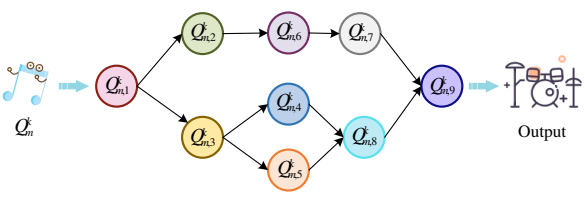

Figure 4 An example of a directed task graph

Each subtask $Q_{m,i}^k$ contains two portions, i.e., $Q_{m,i}^k = \langle d_{m,i}^k, c_{m,i}^k \rangle$, where $Q_{m,i}^k$ denotes ith subtask

of the mth task of the vehicle k, $k \in V$ represents one vehicle, $m \in \mathcal{Q}$ denotes a task, $i \in I$ denotes a subtask, $d_{m,i}^k$ denotes the input data size of the subtask ith of the mth task, and $c_{m,i}^k$ denotes the expected number of CPU cycles allocated to the subtask ith of the mth task.

Communication model

In this section, we introduce the communication model of the task offloading for VEC, which consists of the communication model of vehicle to MEC servers and the communication model of MEC servers to the cloud server.

1) Vehicles to MEC servers: when the task has been offloaded to MEC servers or the cloud server, network latency occurs in the data transmission. We assume that the vehicle transmit data to MEC servers connected to RSUs through wireless networks. Furthermore, we consider that vehicles communicate with MEC servers through a one-hop ad hoc network. Specifically, we let $\lambda_{k,e}$ denote the data transmission rate between vehicle k and MEC server e, which can be calculated as

$$\lambda_{k,e} = n_k^e.w\log_2(1 + p_k.\delta_{k,e}.\sigma^{-2}). \tag{1}$$

where n_k^e denotes the number of sub-channels allocated to the vehicle k, w denotes the bandwidth of each subchannel, p_k denotes the transmission power of vehicle k, $\delta_{k,e}$ denotes the channel gain between vehicle k and MEC server e, σ^{-2} denotes the surrounding noise power.

2) MEC servers to cloud server: we consider that MEC servers communicate with the cloud server through high-speed optical fiber lines. In the case MEC servers cannot serve vehicles, they transmit the tasks to the remote cloud server for execution. Especially, it should be noted that the transmission latency of the MEC server transmitting data to the cloud server is equal the transmission latency of the cloud server returns the result to the MEC server and this transmission latency is independent of the size of the data, due to the long geographical distances between cloud servers and MEC servers. We let $t_{e,c}$ denote the round-trip time of transmission data between MEC server e and the cloud server, which can be calculated as

$$t_{e,c} = 2t_{off}^{cloud}. \tag{2}$$

where t_{off}^{cloud} denotes the transmission latency of MEC server transmitting data to the cloud server.

Computation model

The subtasks either be performed on the vehicle or be offloaded to the MEC server or cloud server for execution. Thus, in this section, we will discuss the execution time of subtasks in the vehicle, MEC server, and cloud server, respectively.

1) Local computing: when the subtask is assigned to be executed on the vehicle, we consider that there may be subtasks being executed locally, so the waiting delay cannot be ignored. In addition, when the locally executing tasks are none, the waiting delay is 0. Therefore, the total local completion delays of subtask $Q_{m,i}^k$ consist of the local execution delay and the local waiting delay, which is given by

$$t_{k,m,i}^{local,completion} = \frac{c_{m,i}^k}{f_k} + t_{k,m,i}^{local,wait}. \tag{3}$$

where $\frac{c_{m,i}^k}{f_k}$ denotes the local execution delay, f_k denotes the computation capability of vehicle k, and $t_{k,m,i}^{local,wait}$ denotes the local waiting delay.

2) Edge computing: when the subtask is offloaded to MEC servers for processing, we consider that the whole process can be broken down into two parts: the transmission delay and the execution delay of the subtask on the MEC server.

Firstly, the raw data of the subtask $Q_{m,i}^k$ is transmitted from vehicle k to MEC server e via wireless communication. According to the communication mode (1), the transmission delay of the subtask $Q_{m,i}^k$ is defined as

$$t_{k,m,i}^{mec,trans} = \frac{d_{m,i}^k}{\lambda_{k,e}}. \tag{4}$$

Secondly, similar of local execution, the processing delay of the subtask $Q_{m,i}^k$ on the MEC server e is defined as

$$t_{k,m,i}^{mec,com} = \frac{c_{m,i}^k}{f_e^{mec}} + t_{k,m,i}^{mec,wait}. \tag{5}$$

where $\frac{c_{m,i}^k}{f_e^{mec}}$ represents the MEC server execution delay, f_e^{mec} represents the computation capability of the MEC server e, and $t_{k,m,i}^{mec,wait}$ represents the MEC server waiting delay.

According to equations (4) and (5), the total completion delays for offloading subtask $Q_{m,i}^k$ to the MEC server e can be defined as

$$\begin{aligned} t_{k,m,i}^{mec,completion} &= t_{k,m,i}^{mec,trans} + t_{k,m,i}^{mec,com} \\ &= \frac{d_{m,i}^k}{\lambda_{k,e}} + \frac{c_{m,i}^k}{f_e^{mec}} + t_{k,m,i}^{mec,wait}. \end{aligned} \tag{6}$$

3) Cloud computing: when the subtask is offloaded to the cloud server for processing, the raw data of subtask $Q_{m,i}^k$ is first transmitted from vehicle k to MEC server e, and then is transmitted from MEC server e to the cloud server. In addition, considering the enormous computation capability of the cloud server, the execution delay is negligible compared with the transmission delay. Therefore, the total completion delays of offloading subtask $Q_{m,i}^k$ to the cloud server can be broken down into two parts: the MEC server transmission delay and the cloud transmission delay, which is defined as

$$\begin{aligned} t_{k,m,i}^{cloud,completion} &= t_{k,m,i}^{mec,trans} + t_{e,c} \\ &= \frac{d_{m,i}^k}{\lambda_{k,e}} + t_{e,c}. \end{aligned} \tag{7}$$

where $\frac{d_{m,i}^k}{\lambda_{k,e}}$ denotes the MEC server transmission delay, and according to (2) $t_{e,c}$ denotes the cloud transmission delay.

Problem formulation

In this part, we formalize the problem of dependency-aware task offloading for VEC as an optimization problem. This optimization problem aims to minimize the average processing latency under the constraints of computing and communication resources of MEC servers. Specifically, the optimization problem is defined as

$$\min_{(p_{m,i}^k, r_{m,i}^k)} \sum_{k \in V, m \in \mathcal{Q}} t_{m,i}^k \tag{8}$$

$$\text{s.t.} \quad \mathrm{p}_{\mathrm{k,m,i}}^{\mathrm{local}} + \mathrm{p}_{\mathrm{k,m,i}}^{\mathrm{edge}} + \mathrm{p}_{\mathrm{k,m,i}}^{\mathrm{cloud}} = 1 \tag{9a}$$

$$\sum_{k \in V} \beta_{k,e} \leq B \tag{9b}$$

$$\sum_{k \in V} x_{k,e} \leq C \tag{9c}$$

where $r_{m,i}^k = \{\beta_{k,m,i}^E, x_{k,m,i}^E\}$ denotes the allocated computation and communication resources, $t_{m,i}^k = t_{k,m,i}^{local} \alpha_{k,m,i}^{local} + t_{k,m,i}^{mec,offload} \alpha_{k,m,i}^{mec} + t_{k,m,i}^{cloud,offload} \alpha_{k,m,i}^{cloud}$, B denotes the maximum communication capability of MEC servers, and C denotes the maximum computation capability of MEC servers.

Task offloading algorithm based on DDPG

In this section, an efficient dependency-aware task offloading algorithm based on DDPG, named DA-TODDPG, is proposed. Different from the value-based reinforcement learning (RL) approach (e.g., DQN) [30, 31], DDPG combines the characteristics of DQN and the actor-critic (AC) algorithm to learning the Q value and the deterministic policy by the experience relay and the frozen network [32], thereby efficiently solving continuous control problems and helping reach the fast convergence. Figure 5 illustrates the framework of DA-TODDPG. First, the algorithm settings of DA-TODDPG are defined. Second, we present the details of DA-TODDPG. Third, the action selection based on the $\mathcal{E}$-greedy policy is described. Finally, the DDPG network update is detailed presented.

Algorithm setting

In this section, DDPG is introduced to address the proposed optimization problem in (8). i.e., obtaining the optimal offloading strategy (i.e., the subtask executed on local, or offloading to edge server, or offloading to cloud server) through exploring the dynamic environment at the beginning of each subtask offloading round.

Similar to [33], we assume that there is one centralized MEC server as the agent in DA-TODDPG that selects the optimal offloading strategy by interacting with the environment through a sequence of observations, actions, and rewards [34]. Specifically, when the agent receives a subtask request from a vehicle, it selects an action based on the current environment state. Then the vehicles choose where to execute the task according to the selected action. After the action is executed, the agent receives a reward that indicates the benefits of the selected action. Finally, the environment evolves into the next state.

There are four key elements in the DDPG, namely environment, state, action, and reward, which are specified as follows:

Environment. The environment env reflects the internet of vehicles environment, including the set of vehicles, the set of tasks, the transmission power of the vehicle, the channel gain between the vehicle and MEC server, and the computation and communication resources of MEC servers. The environment is defined as

$$env = \{V, \mathcal{Q}, p, \delta, B, C\}. \tag{10}$$

State. The state s reflects the observation the available resources of MEC servers, which is defined as

$$s_t = \{B_a, C_a\}. \tag{11}$$

where the subscript t denotes the t-th time step, B_a denotes the available communication resources of MEC servers, and C_a denotes the available computation resources of MEC servers.

Action. Based on the observed states, the agent decides the allocation of computing and bandwidth resources of the MEC server for executing the subtask. The action a is defined as

$$a_{k,m,i}^{t} = \{b_{k,m,i}, c_{k,m,i}\}. \tag{12}$$

where $a_{k,m,i}^{t}$ denotes the action of subtask i of task $Q_{m,i}^{k}$ at the t-th time step, $b_{k,m,i}$ denotes the allocated communication resources for subtask i of task $Q_{m,i}^{k}$, and $c_{k,m,i}$ denotes the allocated computation resources for subtask i of task $Q_{m,i}^{k}$.

Reward. According to the state and action, the agent calculates the offloading strategy benefits, which can be defined as

$$r_{k,m,i}^{t} = \begin{cases} 0, \ \text{if subtask } i \text{ is executed locally} \\ t_{k,m,i}^{lcoal,completion} - t_{k,m,i}^{mec,completion}, \text{if } s\text{ubtask} \\ i \text{ is offloaded to the MEC server} \\ t_{k,m,i}^{local,completion} - t_{k,m,i}^{cloud,completion}, \text{if subtask} \\ i \text{ is offloaded to the cloud server} \end{cases} \tag{13}$$

where $r_{k,m,i}^{t}$ denotes the benefits of the action $a_{k,m,i}^{t}$ of subtask i of task $Q_{m,i}^{k}$ at the t-th time step.

Task offloading algorithm

The illustration of DA-TODDPG is shown in Figure 5, which combines the characteristics of DQN and the actor-critic (AC) algorithm [32]. Therefore, it consists of three components (i.e., evaluation network, target network, and replay memory).

The evaluation network consists of two deep neural networks, namely an evaluation actor network Critic-E and an evaluation critic network Action-E. The evaluation actor network is utilized to explore the offloading strategy. The evaluation critic network estimates the offloading strategy and provides the critic value which helps the evaluation actor to learn the gradient of the policy. In addition, the input of the evaluation network is the current state s_t, the output is the action a_t, the training state and the training action from replay memory.

The target network can be understood as an older version of the evaluation network due to their having the same network structure but different parameters, which is utilized to produce the target value for training Critic-E. It consists of a target actor network Actor-T and a target critic network Critic-T. The input of the target network is the next state s_{t+1} from replay memory and the output is a critic value for computing loss of Critic-E.

The replay memory stores experience tuples, consisting of the current state, selected action, reward, and next state. The stored experience tuples can be randomly sampled for training the evaluation network and the target network. The randomly sampled experience tuples are intended to reduce the effect of data correlation.

The DA-TODDPG algorithm based on DDPG is as described in Algorithm 1, there are two main parts including action selection (Line 7), reward evaluation (Line 9), and DDPG network update (Line 14).

Algorithm 1: DA-TODDPG Algorithm

Input: the size of replay memory N;
the current state s_t
Output: an action a_t
1 Initialize replay memory D with the size of N
2 Randomly initialize the weight of evaluation actor network θ^{μ} and critic network θ^{Q}, respectively
3 Initialize target actor network with weights $\theta^{\mu'} \leftarrow \theta^{\mu}$ and target critic network with weights $\theta^{Q'} \leftarrow \theta^{Q}$
4 **for** $episode = 1$ ***to*** M **do**
5 $s = st$
6 **for** $t = 1$ ***to*** T **do**
7 Obtain the action a_t with evaluation actor network θ^{μ} and the behavior noise n_t: $a_t = \min(\max(\mu(s_t|\theta^{\mu}) + n_t, -1), 1)$
8 Preform action a_t
9 $r_t = RewardEvaluation(a_t)$
10 observe next state s_{t+1}
11 store transition (s_t, a_t, r_t, s_{t+1}) in D
12 **if** *the replay memory* D ***is full*** **then**
13 $\theta^{Q}, \theta^{\mu}, \theta^{Q'}, \theta^{\mu'} = DDPGNetworkUpdate(D)$
14 **end**
15 **end**
16 **end**

Reward evaluation algorithm

Evaluating the reward of each offloading decision not only enables us to obtain an optimal offloading strategy but is also one of the key factors in the convergence of the DDPG algorithm. The goal of the DDPG algorithm is to maximize the reward after performing an action. Therefore, the reward is negatively related to the execution time. The algorithm for evaluating reward is shown in Algorithm 2. Specifically, when the subtask is executed locally, the reward is equal to 0 (Lines 4-5). When the subtask is offloading to the MEC server for execution, the reward is equal to the subtask's local processing delay minus the subtask's processing delay on the edge server (Lines 6-7). Similarly, when a subtask is offloaded to a cloud server for execution, the reward is equal to the subtask's local processing delay minus the subtask's processing delay on the cloud server (Lines 8-9).

DDPG Network algorithm

The DDPG network update is outlined in Algorithm 3. Specifically, in each training step, a minibatch of ex-

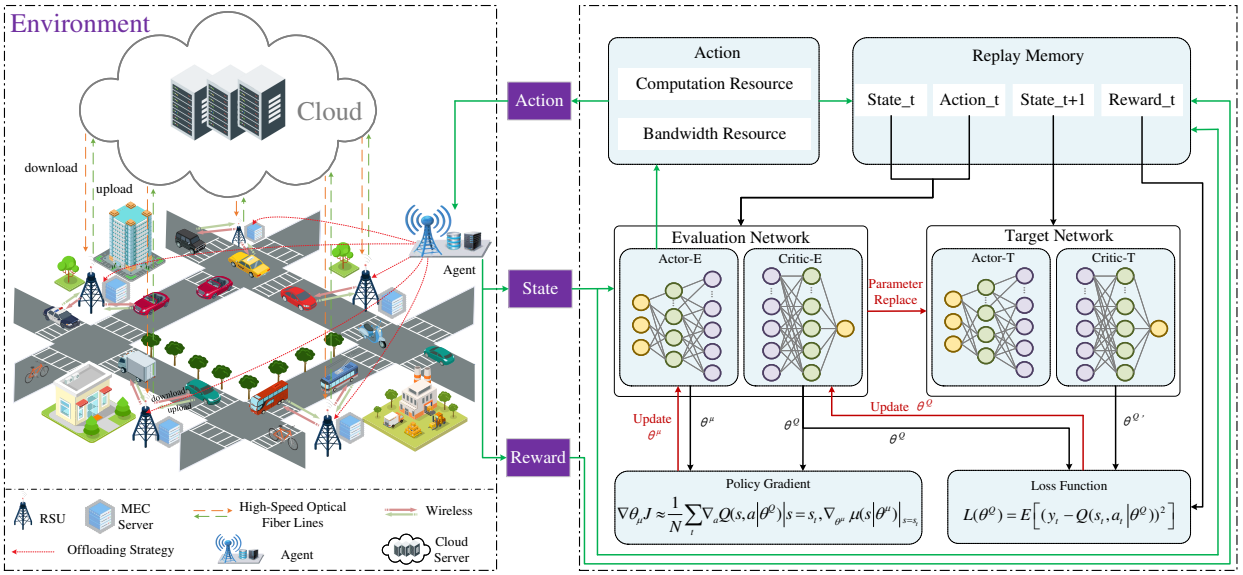

Figure 5 The framework of DA-TODDPG

Algorithm 2: DA-TODDPG Algorithm

Input: the current state st, the action $a_t = \{b_{m,i}, c_{m,i}\}$; subtask $Q^k_{m,i}$
Output: the reward r_t

1 execute on the local completion time $t^{local,completion}_{k,m,i}$ of subtask $Q^k_{m,i}$ with Eq. (3)
2 execute on the MEC server completion time $t^{mec,completion}_{k,m,i}$ of subtask $Q^k_{m,i}$ with Eq. (6)
3 execute on the cloud completion time $t^{cloud,completion}_{k,m,i}$ of subtask $Q^k_{m,i}$ with Eq. (7)
4 **if** $b_{m,i} = 0$ **then**
5 $\quad r_t = t^{local,completion}_{k,m,i} - t^{local,completion}_{k,m,i}$
6 **end**
7 **if** $b_{m,i} > 0$ *and* $c_{m,i} > 0$ **then**
8 $\quad r_t = t^{local,completion}_{k,m,i} - t^{mec,completion}_{k,m,i}$
9 **end**
10 **if** $b_{m,i} = 0$ *and* $c_{m,i} > 0$ **then**
11 $\quad r_t = t^{local,completion}_{k,m,i} - t^{cloud,completion}_{k,m,i}$
12 **end**
13 output r_t

perience tuples D_t are randomly sampled from replay memory D (Line 1-2). Then, the target critic network Critic-T calculates the target value y_t and transmits y_t to evaluation critic network Critic-E (Line 3). After receiving y_t Critic-E updates θ^Q by minimized the loss function $L(\theta^Q)$ (Line 4). On the other hand, Utilizing the sampled policy gradient $\nabla\theta_\mu J$ to update the weights θ^μ of evaluation actor network (Line 5). Finally, the parameters $\theta^{Q'}$ and $\theta^{\mu'}$ of target actor network and target critic network are updated after each C step, respectively (Line 6-8).

Algorithm 3: DDPG Network Update Algorithm

Input: the replay memory D
Output: θ^Q, θ^μ, $\theta^{Q'}$, $\theta^{\mu'}$

1 Sample random minibatch of transitions from D and get D_t
for (s_t, a_t, r_t, s_{t+1}) *in* D_t **do**
2 $\quad$ Calculate the target value:$y_t = r_t + \gamma Q(s_{t+1}, \mu(s_t \,|\, \theta^{\mu'}) \,|\, \theta^{Q'})$
3 $\quad$ Update the θ^Q in evaluation critic network by minimizing the loss: $L(\theta^Q) = E\left[(y_t - Q(s_t, a_t \,|\, \theta^Q))^2\right]$
4 $\quad$ Update the weights $\theta^{Q'}$ of target actor network every C step: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
5 $\quad$ Update the weights $\theta^{\mu'}$ of target critic network every C step: $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
6 $\quad$ output θ^Q, θ^μ, $\theta^{Q'}$, $\theta^{\mu'}$
7 **end**

Experimental performance

In this section, extensive experiments are carried out to evaluate the performance of our proposed scheme. The experimental setting details explained first, and then the convergence performance of the DA-TODDPG scheme is analyzed. Finally, DA-TODDPG is compared with existing task offloading schemes to prove the effectiveness of DA-TODDPG scheme.

Experimental setting

We simulate a VEC system, which consists of 7 vehicles, 40 tasks, and 4 RSUs. Each RSU is equipped with one MEC server and each MEC server is equipped with several CPUs. The size of input data of tasks is randomly generated from the set {25, 30, 40, 45, 60} MB. The computation resource requirements of tasks are randomly assigned from the set {0.5, 0.6, 0.7, 0.8, 1.2} Gigacycle/s. Each task is randomly divided into 4 to 8 subtasks, and the size of input data and computation resource requirements of the task are randomly assigned to subtasks. In addition, we make the other parameters in the experiments in Table 1. To demonstrates the effectiveness of DA-TODDPG, three baselines are selected to compare the DA-TODDPG as follows.

Table 1 Parameter values

Parameter	Value
the transmission power p	2 ± 0.2 Watt[35]
the channel gain δ	144 ± 14.4 dB[36]
noise power σ^{-2}	1.5×10^{-8} Watt[36]
the computation capacity of a vehicle	0.3 ± 0.03 Gigacycles/s[9]
the size of the experience pool D	2000
min-batch size of D_t	32
the learning rate	0.01
the communication capability of a MEC server	40 MHz
The computation capability of a MEC server	5 Gigacycle/s
t_{off}^{cloud}	1000ms

- **Dependency-aware random offloading (DA-RO).** The DA-RO is a traditional offloading approach without utilizing optimization algorithms, where the edge server randomly assigns subchannels and computational resources to vehicles for the corresponding task offloading operations.
- **Dependency-aware task offloading scheme based on DQN (DA-TODQN).** The DA-TODQN is a fine-grained task offloading approach, which considers the decomposability and dependencies of the task. Unlike our method DA-TODDPG, DA-TODQN uses a value-based reinforcement learning approach to allocate subchannels and computational resources. It is an implementation of MORL-ODT [37].
- **Entire task offloading scheme based on DDPG (E-TODDPG).** The E-TODDPG is a coarse-grained task offloading approach, which without the consideration of the decomposability and dependency of tasks. It is a realization of [38].

Evaluation of train performance

We first evaluate the train performance of the DA-TODDPG in the section. Specifically, to prove the advantage of DA-TODDPG, we use the average reward as a metric. On the one hand, we compare the train performance of DA-TODDPG at different learning rates, and on the other hand, we compare DA-TODDPG with three baseline methods.

1) Convergence with Different Learning Rates: Figure 6 presents the train performance of average reward under different learning rates (i.e., Learning rate=0.005, Learning rate=0.01, Learning rate=0.05). It is seen that the learning rate affects the learning rewards, and although the convergence performance is ultimately optimal, there are differences in reward performance. When the learning rate is too low, the convergence speed is plodding, and the reward value is a little small compared to learning late is 0.01 due to slow learning speed. When the learning rate is 0.05 and above, the DA-TODDPG cannot well converge due to a too large learning rate leads to our method easily getting trapped into local optimal. Thus, the learning rate is set to 0.01 in the following system simulations and evaluations.

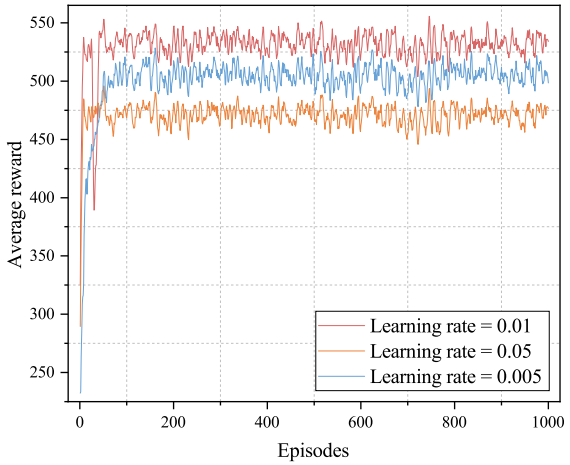

Figure 6 The convergence performance of average reward under different learning rates

2) Convergence with Different Baselines: Figure 7 shows the train performance of different baselines (i.e., E-TODDPG, DA-TODQN). It is seen that the DA-TODDPG converges faster and has a higher reward value than the E-TODDPG and the DA-TODQN. The reason is that DA-TODDPG both considers the inter-subtask dependency and the dynamic environment, and utilizes DDPG to solve the continuous control problems. Thus, fine-grained offloading opportunities can be well obtained. Specifically, the E-TODDPG is a

coarse-grained task offloading approach that does not consider task decomposability and dependencies thus the fine-grained offloading opportunities for subtasks are wasted and cannot obtain optimization strategy. The DA-TODQN demonstrates slow convergence and unstable performance caused by that DQN shows the inefficiency on the network with a high dimension in the action space.

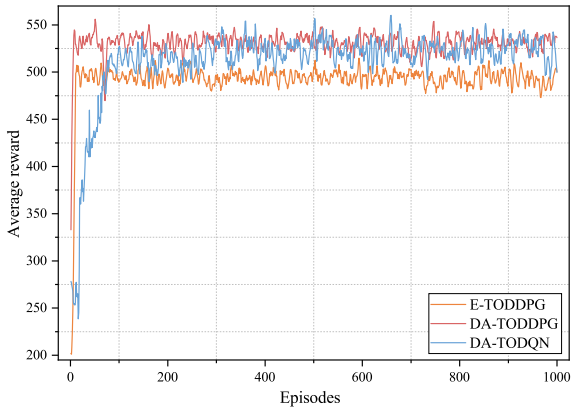

Figure 7 The convergence performance of average reward under different baselines

Performance evaluation and analysis

To verify the adaptability and effectiveness of DA-TODDPG, three sets of simulation experiments with diversity in environments are conducted, and the performance of DA-TODDPG is evaluated.

The control values for the comparative analysis variables are listed in Table 2. In each set of experiments, the value of one variable fluctuated around the control value and the other variables remained constant.

Table 2 Controlled Variables setting

Variable description	Controlled value
Number of MEC servers	4
Number of tasks	40
Average size of raw data	50MB

1) Analysis on the Variety of Task Number: When the number of Tasks in the offloading system are different, the Average processing delay of different baselines are shown in Figure 8. With other variables unchanged, the number of tasks ranges from 20 to 70 in this set of experiments. It is seen that the average processing delay increase with the rise in the number of tasks. As the number of tasks grows from 20 to 70, DA-TODDPG perpetually outperforms DA-RO, DA-TODQN and E-TODDPG. This is because DA-TODDPG gains the most appropriate fine-grained offloading opportunities for subtasks. More specifically, when the task number increases from 20 to 70, the DA-TODDPG outperforms the DA-RO, DA-TODQN, and E-TODDPG by the improvements of 21.3% to 11.4%, 8.8% to 7.6%, and 13.2% to 17.8%. In addition, when the task number increases to 70, the performance of E-TODDPG drops sharply. This is because E-TODDPG does not take into account the inter-task dependency, thus MEC servers suffer from the computation resource contention caused by a large number of tasks.

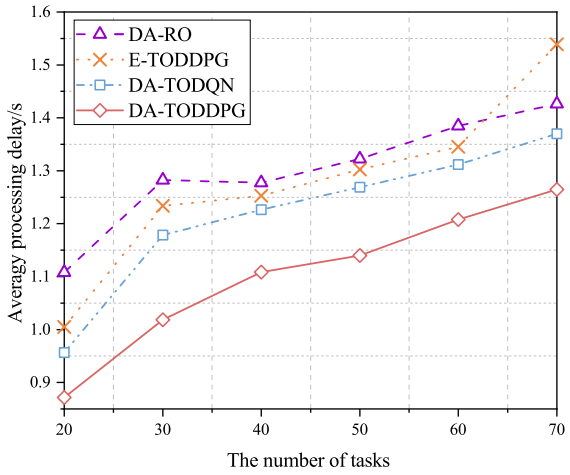

Figure 8 Compare of average processing delay with variety in task number

2) Analysis on the Variety of MEC Server Number: In Figure 9 illustrates the impact on the average processing delay by the number of MEC servers. Experiments are conducted with the number of MEC servers ranging from 2 to 6, while the other variables remain unchanged. The figure shows that the average processing delay goes down as the number of MEC servers increases. This is because the increasing MEC servers can introduce more computing resources and communication bandwidth into the system. Especially, when the number of MEC servers is 2, 3, 4, 5, and 6, the improvement of DA-TODDPG compared to the DA-RO is 12.4%, 16.1%, 20.5%, 15.2%, and 21%, the improvement of DA-TODDPG compared to the DA-TODQN is 7.8%, 11.6%, 13.6%, 12.4%, and 12.1%, the improvement of DA-TODDPG compared to the E-TODDPG is 10.1%, 13.4%, 17.4%, 16.1%, and 18.8%, respectively. The performance of DA-TODDPG is higher than other baselines due to DA-TODDPG both considering the dynamic environment and inter-task de-

pendency, and utilizing DDPG to solve the problem of continued action space.

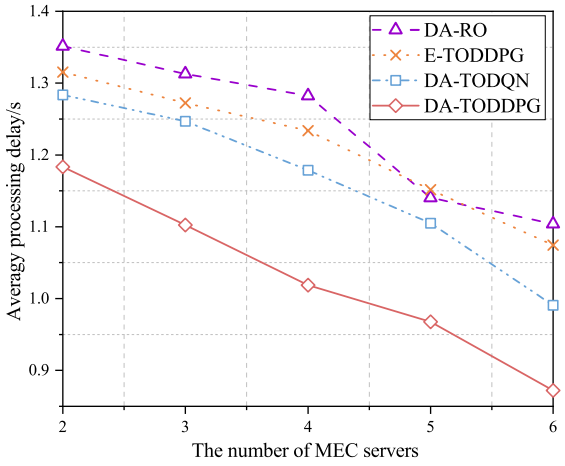

Figure 9 Compare of average processing delay with variety in MEC server number

3) Analysis on the Variety of Average size of Input Data: In Figure 10, the average processing delay with diversity in the average size of input data is analyzed. Experiments are conducted with the average size of raw data ranging from 25 to 60 MB, while the other variables remain unchanged. It can be seen that the average processing delay increases when the average size of input data, which is because the raw data of offloaded tasks should be transmitted to a MEC server or the cloud server, which results in increase of the time cost of transmission. Peculiarly, As the average size of input data increases from 25 to 60 MB, the DA-TODDPG outperforms the DA-RO, DA-TODQN, and E-TODDPG by the improvements of 20.2%, 11.2%, and 20.9% on average. In addition, When the average size of input data increases to 60MB, the performance of E-TODDPG drops sharply. This is reason that the E-TODDPG is a coarse-grained task offloading approach that does not consider the parallelism of subtasks on the vehicle, MEC servers and the cloud server, uploading all data to the MEC server and the cloud server which increases the transmission delay, Overall, we can see that the DA-TODDPG always gains the optimal performance of the average processing delay under the different average size of input data.

Conclusion

In this paper, we proposed an efficient dependency-aware task offloading scheme for reducing the average processing delay of tasks with end-edge-cloud collaborative computing. In this scheme, we first used the

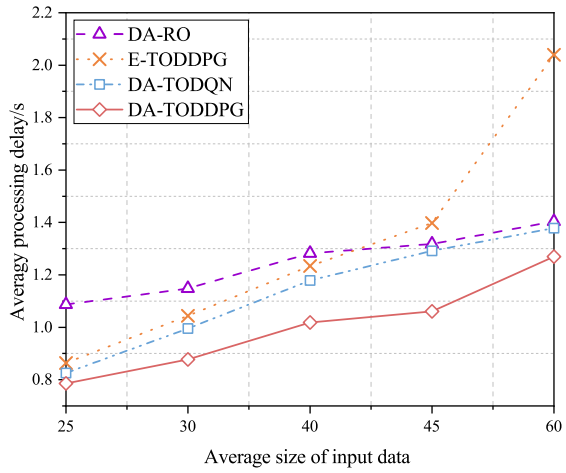

Figure 10 Compare of average processing delay with variety in average size of input data

directed acyclic graph technique to model the inter-subtask dependency. Then, we designed a dependency-aware task offloading algorithm based on DDPG to select the optimal offloading strategy, in which the continuous control problems and allocation of edge server resources were considered. Simulation results show that our proposed dependency-aware offloading scheme can effectively reduce the average processing delay of tasks.

For future work, for one thing, our proposed task offloading scheme lacks the verification of real data. We plan to collect a large number of historical data and verify the performance of our proposed task offloading scheme using real historical data. For another, we will consider both the mobility of vehicles and data migration between edge servers.

Acknowledgements
The authors would like to thank the staff and postgraduate students at the School of Big Data and Intelligent Engineering of Southwest Forestry University for their assistance and valuable advice.

Funding
This work has been supported by the Project of Key Science Foundation of Yunnan Province under Grant No. 202101AS070007, Dou Wanchun Expert Workstation of Yunnan Province No.202105AF150013, Science and Technology Youth lift talents of Yunnan Provincethe, the Project of National Natural Science Foundation of China under Grant No. 61862065 and 12163004, the Major Project of Science and Technology of Yunnan Province under Grant No. 202002AD080002, the Project of Science and Technology of Yunnan Province under Grant No. (202001AT070135), and the Project of Scientific Research Fund Project of Yunnan Education Department under Grant No. 2022Y561.

Declarations

Ethics approval and consent to participate
The work is a novel work and has not been published elsewhere nor is it currently under review for publication elsewhere.

Consent for publication
Informed consent was obtained from all individual participants included in the study.

Availability of data and materials
Not applicable.

Authors' contributions
Guozhi Liu: Writing-Original draft preparation, Conceptualization, Methodology, Software, Funding acquisition, Visualization, and Data Curation. Fei Dai: Conceptualization, Methodology, Writing-Reviewing and Editing, Funding acquisition, and Validation. Bi Huang: Writing-Reviewing and Editing, Resources, and Formal analysis. Zhenping Qiang: Supervision, and Validation. Shuai Wang: resource allocation, and supervision. Lecheng Li: Writing-Reviewing and Editing, and Investigation.

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

Author details
[1]School of Big Data and Intelligent Engineering, Southwest Forestry University, Kunming, China. [2]School of Big Data and Intelligent Engineering, Southwest Forestry University, Kunming, China.

References

1. Yang, F., Wang, S., Li, J., Liu, Z., Sun, Q.: An overview of internet of vehicles. China communications **11**(10), 1–15 (2014)
2. Xu, X., Shen, B., Ding, S., Srivastava, G., Bilal, M., Khosravi, M.R., Menon, V.G., Jan, M.A., Wang, M.: Service offloading with deep q-network for digital twinning-empowered internet of vehicles in edge computing. IEEE Transactions on Industrial Informatics **18**(2), 1414–1423 (2020)
3. Liu, Y., Li, Y., Niu, Y., Jin, D.: Joint optimization of path planning and resource allocation in mobile edge computing. IEEE Transactions on Mobile Computing **19**(9), 2129–2144 (2019)
4. Zhang, J., Guo, H., Liu, J., Zhang, Y.: Task offloading in vehicular edge computing networks: A load-balancing solution. IEEE Transactions on Vehicular Technology **69**(2), 2092–2104 (2019)
5. Nguyen, D., Ding, M., Pathirana, P., Seneviratne, A., Li, J., Poor, V.: Cooperative task offloading and block mining in blockchain-based edge computing with multi-agent deep reinforcement learning. IEEE Transactions on Mobile Computing (2021)
6. Li, Y., Qi, F., Wang, Z., Yu, X., Shao, S.: Distributed edge computing offloading algorithm based on deep reinforcement learning. IEEE Access **8**, 85204–85215 (2020)
7. Li, E., Zeng, L., Zhou, Z., Chen, X.: Edge ai: On-demand accelerating deep neural network inference via edge computing. IEEE Transactions on Wireless Communications **19**(1), 447–457 (2019)
8. Xu, X., Jiang, Q., Zhang, P., Cao, X., Khosravi, M.R., Alex, L.T., Qi, L., Dou, W.: Game theory for distributed iov task offloading with fuzzy neural network in edge computing. IEEE Transactions on Fuzzy Systems (2022)
9. Dai, F., Liu, G., Mo, Q., Xu, W., Huang, B.: Task offloading for vehicular edge computing with edge-cloud cooperation. World Wide Web, 1–19 (2022)
10. Liu, Y., Chen, C.S., Sung, C.W., Singh, C.: A game theoretic distributed algorithm for feicic optimization in lte-a hetnets. IEEE/ACM Transactions on Networking **25**(6), 3500–3513 (2017)
11. Guo, H., Liu, J.: Collaborative computation offloading for multiaccess edge computing over fiber–wireless networks. IEEE Transactions on Vehicular Technology **67**(5), 4514–4526 (2018)
12. Chen, L., Wu, J., Zhang, J., Dai, H.-N., Long, X., Yao, M.: Dependency-aware computation offloading for mobile edge computing with edge-cloud cooperation. IEEE Transactions on Cloud Computing (2020)
13. Aceto, L., Morichetta, A., Tiezzi, F.: Decision support for mobile cloud computing applications via model checking. In: 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, pp. 199–204 (2015). IEEE
14. Shu, C., Zhao, Z., Han, Y., Min, G., Duan, H.: Multi-user offloading for edge computing networks: A dependency-aware and latency-optimal approach. IEEE Internet of Things Journal **7**(3), 1678–1689 (2019)
15. Ning, Z., Dong, P., Kong, X., Xia, F.: A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things. IEEE Internet of Things Journal **6**(3), 4804–4814 (2018)
16. Chen, M., Hao, Y.: Task offloading for mobile edge computing in software defined ultra-dense network. IEEE Journal on Selected Areas in Communications **36**(3), 587–597 (2018)
17. Zhu, D., Bilal, M., Xu, X.: Edge task migration with 6g-enabled network in box for cybertwin-based internet of vehicles. IEEE Transactions on Industrial Informatics **18**(7), 4893–4901 (2021)
18. Yang, L., Zhang, H., Li, X., Ji, H., Leung, V.C.: A distributed computation offloading strategy in small-cell networks integrated with mobile edge computing. IEEE/ACM Transactions on Networking **26**(6), 2762–2773 (2018)
19. Xie, J., Wang, S., Yin, C.: Machine learning based task scheduling for wireless powered mobile edge computing iot networks. In: 2019 11th International Conference on Wireless Communications and Signal Processing (WCSP), pp. 1–6 (2019). IEEE
20. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
21. Liu, H., Zhao, H., Geng, L., Feng, W.: A policy gradient based offloading scheme with dependency guarantees for vehicular networks. In: 2020 IEEE Globecom Workshops (GC Wkshps, pp. 1–6 (2020). IEEE
22. Huang, X., He, L., Chen, X., Wang, L., Li, F.: Revenue and energy efficiency-driven delay constrained computing task offloading and resource allocation in a vehicular edge computing network: a deep reinforcement learning approach. IEEE Internet of Things Journal (2021)
23. Liao, H., Li, X., Guo, D., Kang, W., Li, J.: Dependency-aware application assigning and scheduling in edge computing. IEEE Internet of Things Journal **9**(6), 4451–4463 (2021)
24. Qiu, X., Zhang, W., Chen, W., Zheng, Z.: Distributed and collective deep reinforcement learning for computation offloading: A practical perspective. IEEE Transactions on Parallel and Distributed Systems **32**(5), 1085–1101 (2020)
25. He, X., Lu, H., Du, M., Mao, Y., Wang, K.: Qoe-based task offloading with deep reinforcement learning in edge-enabled internet of vehicles. IEEE Transactions on Intelligent Transportation Systems **22**(4), 2252–2261 (2020)
26. Yao, L., Xu, X., Bilal, M., Wang, H.: Dynamic edge computation offloading for internet of vehicles with deep reinforcement learning. IEEE Transactions on Intelligent Transportation Systems (2022)
27. Liu, Y., Wang, S., Zhao, Q., Du, S., Zhou, A., Ma, X., Yang, F.: Dependency-aware task scheduling in vehicular edge computing. IEEE Internet of Things Journal **7**(6), 4961–4971 (2020)
28. Awada, U., Zhang, J., Chen, S., Li, S.: Airedge: A dependency-aware multi-task orchestration in federated aerial computing. IEEE Transactions on Vehicular Technology **71**(1), 805–819 (2021)
29. Wang, J., Hu, J., Min, G., Zhan, W., Zomaya, A., Georgalas, N.: Dependent task offloading for edge computing based on deep reinforcement learning. IEEE Transactions on Computers (2021)
30. Wang, Y., Fang, W., Ding, Y., Xiong, N.: Computation offloading optimization for uav-assisted mobile edge computing: a deep deterministic policy gradient approach. Wireless Networks **27**(4), 2991–3006 (2021)
31. Ren, Y., Yu, X., Chen, X., Guo, S., Xue-Song, Q.: Vehicular network edge intelligent management: A deep deterministic policy gradient approach for service offloading decision. In: 2020 International Wireless Communications and Mobile Computing (IWCMC), pp. 905–910 (2020). IEEE
32. Li, M., Gao, J., Zhao, L., Shen, X.: Deep reinforcement learning for collaborative edge computing in vehicular networks. IEEE Transactions on Cognitive Communications and Networking **6**(4), 1122–1135 (2020)
33. You, C., Huang, K., Chae, H., Kim, B.-H.: Energy-efficient resource allocation for mobile-edge computation offloading. IEEE Transactions on Wireless Communications **16**(3), 1397–1411 (2016)
34. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K.,

Ostrovski, G., *et al.*: Human-level control through deep reinforcement learning. nature **518**(7540), 529–533 (2015)
35. Chen, X., Zhang, H., Wu, C., Mao, S., Ji, Y., Bennis, M.: Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. IEEE Internet of Things Journal **6**(3), 4005–4018 (2018)
36. Sun, Y., Zhou, S., Xu, J.: Emm: Energy-aware mobility management for mobile edge computing in ultra dense networks. IEEE Journal on Selected Areas in Communications **35**(11), 2637–2646 (2017)
37. Song, F., Xing, H., Wang, X., Luo, S., Dai, P., Li, K.: Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach. Future Generation Computer Systems **128**, 333–348 (2022)
38. Xu, Y.-H., Yang, C.-C., Hua, M., Zhou, W.: Deep deterministic policy gradient (ddpg)-based resource allocation scheme for noma vehicular communications. IEEE Access **8**, 18797–18807 (2020). doi:10.1109/ACCESS.2020.2968595