Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

A collaborative computation and dependency-aware task offloading method for vehicular edge computing: a reinforcement learning approach

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

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 enables performing tasks with low latency by deploying computing and storage resources close to vehicles. However, the traditional task offloading schemes only focus on one-shot offloading, taking less into consideration task dependency. Furthermore, the continuous action space problem during task offloading should be considered. In this paper, an efficient dependency-aware task offloading scheme for VEC with vehicle-edge-cloud collaborative computation is proposed, where subtasks can be processed locally or can be offloaded to an edge server, or a cloud server for execution. Specifically, first, the directed acyclic graph (DAG) is utilized to model the dependency of subtasks. Second, a task offloading algorithm based on Deep Deterministic Policy Gradient (DDPG) was proposed to obtain the optimal offloading strategy in a vehicle-edge-cloud environment, which efficiently solves the continuous control problem and helps reach fast convergence. Finally, extensive simulation experiments have been conducted, and the experimental results show that the proposed scheme can improve performance by about 13.62% on average against three baselines.

Keywords: Task offloading, Task dependency, Vehicular edge computing, Vehicle-edge-cloud collaborative computing, Deep deterministic policy gradient, Deep reinforcement learning

Introduction

The Internet of Vehicles (IoV) is a new paradigm that combines traditional vehicle ad hoc network and vehicle remote information processing, which can effectively improve vehicular services and augment the capabilities of vehicles [1, 2]. In IoV, an intelligent vehicle is capable of running various applications [3, 4], such as collision warning [5], automatic driving [6], and auto navigation [7]. Unfortunately, these applications not only require significant computation resources and storage resources but also have stringent delay requirements [2, 8]. As a result, it is challenging to execute them on vehicles with low latency that have limited resources.

Vehicular edge computing (VEC) is proposed as a promising solution to solve the above problem, which integrates mobile edge computing (MEC) into IoV [9]. VEC can improve vehicle service quality [10–12] by deploying MEC servers' computation resources and storage resources close to vehicles. Specifically, computation-intensive and delay-sensitive tasks can be offloaded to MEC servers for execution via wireless networks [2, 13]. Compared to resource-constrained vehicles, MEC servers with more computation resources can efficiently reduce the execution latency of these tasks.

*Correspondence: daifei@swfu.edu.cn

School of Big Data and Intelligent Engineering, Southwest Forestry University, Kunming, China

Although VEC can reduce execution latency of tasks, MEC servers cannot ensure load balancing due to their limited computing and storage ability of [14, 15]. To better improve resource utilization, some tasks can be executed locally or offloaded to the cloud server for execution. Thus, this type of task offloading with vehicle-edge-cloud collaborative computing can obtain a low latency for various vehicular tasks. Such approaches have been extensively studied [9, 16]. For example, Dai et al. propose an offloading method for VEC, which offloads tasks based on vehicle-edge-cloud collaborative computing [9]. Xu et al. [16] presented a game theory-based service offloading approach to minimize task processing latency of users, where both predictions of traffic flow and the allocation of resources are considered. In these works, tasks were considered as a whole during task offloading. Their assumption is that offloaded tasks are atomic. However, a typical application task consists of a series of subtasks [17, 18], which are originally designed to enable multithread processing [18]. As shown in Fig. 1(a), we divide a task into five subtasks and use a directed acyclic graph (DAG) to describe the inter-subtask dependency. Figure 1(b) illustrates the fine-grained task offloading approach, where subtask 2, subtask 3, and subtask 4 are executed in parallel through vehicle-edge-cloud collaborative computing. Therefore, it is necessary to take task dependency into account during task offloading.

Deep reinforcement learning (DRL) is a branch of artificial intelligence (AI), which utilizes the perceptual ability of deep learning and the decision-making ability of reinforcement learning [19]. DRL can obtain the optimal offloading strategies by directly interacting with the dynamic vehicular network. For example, Dai et al. proposed an efficient task offloading approach based on the deep Q-network (DQN) to minimize the processing delay of tasks. He et al. [20] first introduce a novel Quality of Experience (QoE), and then proposed a task offloading algorithm based on DRL to improve QoE for IoV. In these works, these value-based reinforcement learning methods mainly focused on discrete actions. However, the problem of task offloading in a vehicle-edge-cloud computing environment is a continuous action space (i.e., continuously allocating computation resources and bandwidth resources of edge servers). Therefore, it is necessary to take continuous action space into account during task offloading.

To tackle the above challenges, an efficient dependency-aware task offloading scheme based on a deep deterministic policy gradient (DDPG) named DA-TOD-DPG, is proposed. Compared with the existing task offloading methods, this approach considers both task dependencies and the continuous control problem during task offloading. Specifically, the main contributions of this article are summarized as follows:

- Modeling the system model of task offloading in a vehicle-edge-cloud collaborative computing environment, where DAG is utilized to model the dependencies of subtasks in the task model.
- Proposing an efficient dependency-aware task offloading scheme based on DDPG to achieve the opti-

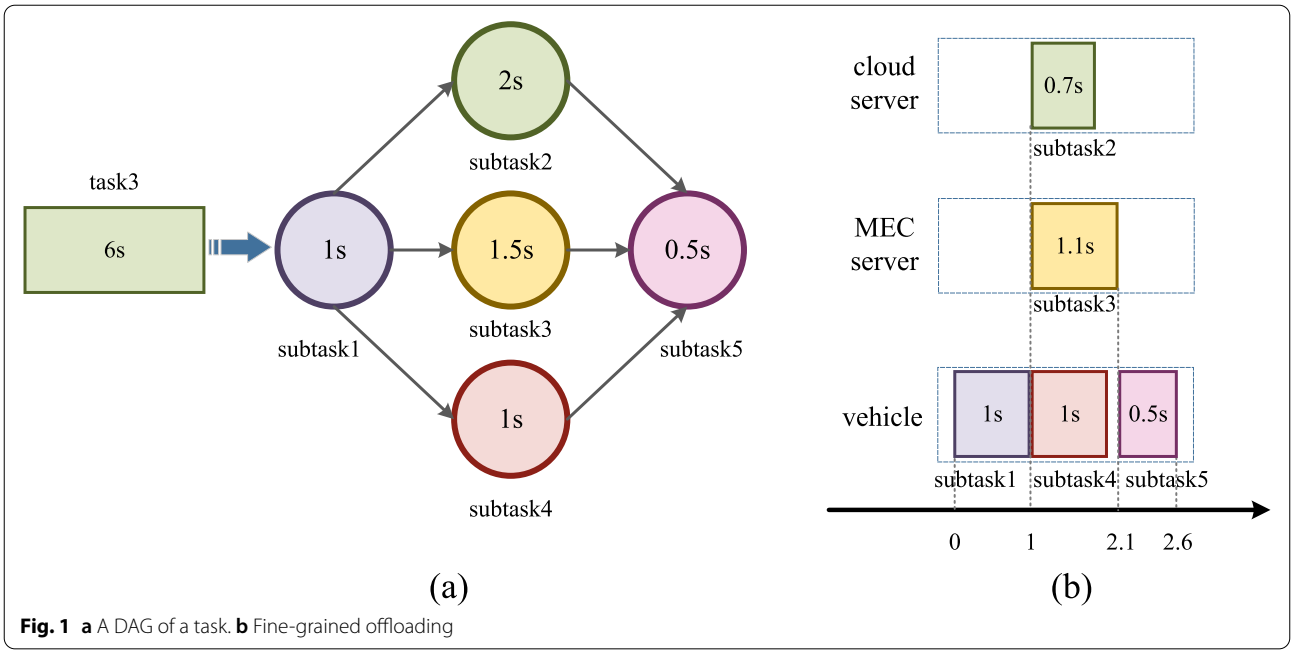

Fig. 1 **a** A DAG of a task. **b** Fine-grained offloading

mal fine-grained offloading strategy, which considers both task dependencies and continuous action spaces.
- Conducting 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 works. Section 3 presents the system model and problem definition. Section 4 proposes the design of the dependency-aware task offloading scheme based on DDPG. Section 5 evaluates the performance of our proposed offloading scheme. Section 6 concludes this paper and outlines future work.

Related works

There are some studies work on task offloading for vehicular edge computing, which consists of mobility-aware task offloading [21], dependency-aware task offloading [22], learning-based task offloading [13], and traffic flow prediction task offloading [16]. Most work has studied the task offloading between vehicles and MEC servers. Yao et al. [19] considered both energy consumption and processing delay and proposed a twin delayed deep deterministic policy gradient algorithm to achieve the optimal offloading strategy. Zhang et al. [23] proposed an offloading method, where both the heterogeneous requirements of the computation tasks and the mobility of the vehicles are considered. Ren et al. [24] studied task offloading problems with multiple constraints and proposed a DDPG-based task offloading algorithm. Zhan et al. [25] studied the offloading problem without information sharing and proposed a computation offloading game formulation. However, these works only considered the resources of vehicles and MEC servers and did not fully utilize the rich resources of the cloud.

Recently, some works have begun to consider the problem of task offloading in the vehicle-edge-cloud computing environment. Xu et al. [16] presented a game theory-based service offloading approach to minimize task processing latency of users, where both predictions of traffic flow and the allocation of resources are considered. Dai et al. [9] proposed an efficient task offloading approach to minimize the processing delay of tasks, which jointly considered the edge-cloud opportunities and the convergence of deep reinforcement learning. Zhang et al. [26] studied the problem of resource allocation for edge services under the vehicle-edge-cloud computing environment and proposed two algorithms to maximize social welfare and profit. However, these works did not consider the subtask dependencies within a task.

With respect to dependency-aware task offloading, Chen et al. [27] studied the problem of dependency-aware task offloading, where considered both the collaboration between MEC servers and the cloud servers and the collaboration among MEC servers. Fan et al. [28] proposed a heuristic algorithm with the aim of reducing the consumed energy of vehicles. Pan et al. [29] considered both energy consumption and task dependency in a vehicle-edge-cloud computing environment. A Q-learning-based framework was proposed to select optimal strategies. Chen et al. [30] studied multiple dependent tasks offloading problems under an end-edge-cloud collaborative computing environment. A DRL-based algorithm was proposed to reduce the average energy-time cost. However, these works assume that each MEC server has enough resources for task offloading.

Besides, some studies have attempted to solve the task offloading problem via deep reinforcement learning. Qu et al. [31] proposed a deep meta-reinforcement learning-based offloading (DMRO) algorithm for reducing latency and energy consumption. Binh et al. [32] proposed a based on Deep Q-Network (DQN) algorithm to enhance the average quality of experience. Huang et al. [33] considered both task offloading and resource allocation and proposed a Deep-Q Network based algorithm to minimize the overall offloading cost in terms of energy cost, computation cost, and delay cost. However, the research works simply assume that the action space of task offloading is discrete due to using a value-based function.

Unlike these above works, we investigate dependency-aware task offloading for IoV in the vehicle-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.

In addition, based on the above discussion in the field of task offloading for IoV, a side-by-side comparison is presented in Table 1 in terms of strategy, applied metrics, advantages, and weaknesses of each technical study.

System model and problem formulation

In this section, the system model of dependency-aware task offloading is designed, and then the problem of dependency-aware task offloading is formulated as an optimization problem. The key terms and the related descriptions in the system are listed in Table 2.

Network model

As illustrated in Fig. 2, the network model of dependency-aware task offloading for VEC is presented, which

Table 1 A side-by-side comparison of offloading strategies

Reference	Strategy	Applied metrics	Collaborative way	Advantages	Weakness
[17]	deep deterministic policy gradient (TD3)	Delay, Energy	Vehicle-Edge	Appropriate computation model	Not considering the resources of cloud services
[21]	Machine Learning	Delay, Cost	Vehicle-Edge	Considering the mobility of the vehicles	Not considering the resources of cloud services
[22]	Deep Deterministic Policy Gradient (DDPG)	Delay	Vehicle-Edge	Considering V2V and V2I	High complexity
[23]	A policy gradient deep reinforcement learning	Delay, Energy	Vehicle-Edge	Acceptable system model	Not considering the resources of cloud services
[14]	Takagi-Sugeno fuzzy neural network and game theory	Delay, Energy	Vehicle-Edge-Cloud	Traffic flow prediction combined with task offloading	High complexity
[7]	Deep Q-Network (DQN)	Delay	Vehicle-Edge-Cloud	Acceptable complexity	Not considering task dependencies
[24]	Machine Learning	Profit	Vehicle-Edge-Cloud	Edge and cloud share resources in the form of wholesale and buyback	High dimensional state space
[25]	A Greedy Based	Delay	Vehicle-Edge-Cloud	Considering task dependencies	Single offloading
[26]	Heuristic Algorithm	Energy, Cost	Vehicle-Edge-Cloud	Considering task dependencies	High complexity
[27]	Q-learning	Delay	Vehicle-Edge-Cloud	Acceptable system model	High dimensional state space
[28]	Actor-Critic Mechanism	Delay, Energy	Vehicle-Edge-Cloud	Considering task dependencies	Not considering continuous action space
[29]	Deep Meta Reinforcement Learning-based Offloading (DMRO)	Cost	Vehicle-Edge-Cloud	Fine-grained offloading	Not considering continuous action space
[30]	Deep Q-Network (DQN)	QoE	Vehicle-Edge	Solving for high dimensional state space	Model limitations
[31]	Deep Q-Network (DQN)	Cost	Vehicle-Edge	Appropriate mathematical proof	Not considering continuous action space

Table 2 Notations and descriptions

Symbol	Description
$Q_{m,i}^k$	The ith subtask belonging to task Q_m^k
$d_{m,i}^k$	The input data size of the ith subtask
$c_{m,i}^k$	The amount of computation resource required to complete subtask $Q_{m,i}^k$
$\lambda_{k,e}$	The data transmission rate between vehicle k and RSU r
p_k	The transmission power of vehicle k
$\delta_{k,e}$	The channel gain between vehicle k and RSU r
w	The bandwidth of each sub-channel between vehicle k and RSU r
σ^{-2}	The surrounding noise power
$t_{r,c}$	The round-trip time of transmission data between RSU r and the cloud server
f_k	The computation capability of vehicle k

consists of three layers: user layer, edge layer, and cloud layer. The user layer is a collection of all vehicles, where each vehicle can process some subtasks of a task. The edge layer consists of RSUs and MEC servers, where each RSU is equipped with a MEC server. Each RSU can collect vehicle service requests within its coverage range and each MEC server can execute subtasks through offloading. The cloud layer is the cloud server, which has enough computing and storage resources.

We assume that all tasks generated by vehicles can be broken down into smaller subtasks. Each subtask can be processed locally or offloaded to a MEC server or the remote cloud server for execution.

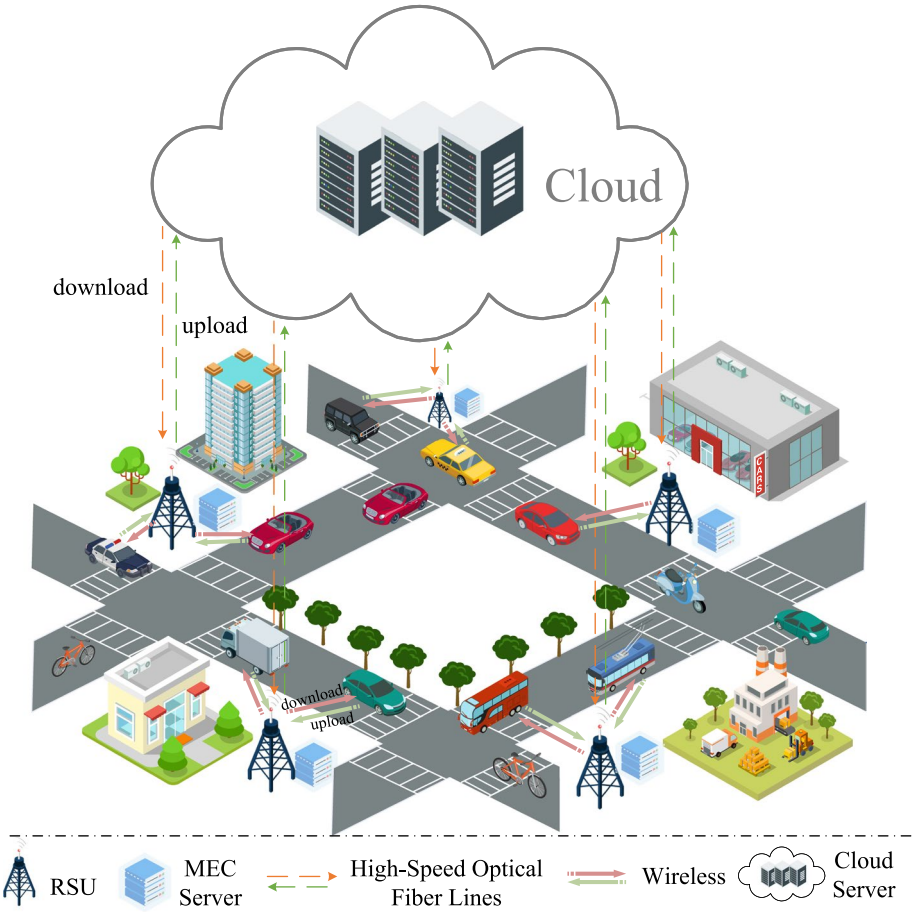

Fig. 2 The architecture of VEC in a vehicle-edge-cloud computing environment

The set of RSUs is denoted as $R = \{1, 2, ..., r, ..., R\}$, and the set of vehicles is denoted as $K = \{1, 2, ..., k, ..., K\}$. In addition, we assume that each vehicle has M computation-intensive and delay-sensitive tasks to be executed within a stringent completion time constraint. The variable Q_m^k is used to represent the nth task of vehicle k, represents one vehicle. For ease of reference, we show the key notations used in this article in Table 2.

Task model

In this section, the task model of dependency-aware task offloading is introduced. Since tasks are not atomic, their subtask may be interdependent, i.e., the output of some subtask is the input of another subtask [2]. To describe subtask dependencies with a task, each subtask can either be processed on the vehicle or offloaded to the edge server or cloud server for computation. Each task can be modeled as a directed acyclic graph (DAG), i.e., $G = (\mathcal{I}, \mathcal{E})$, where $\mathcal{I}$ is the set of subtasks, and $\mathcal{E}$ is the set of directed edges. Let $I = |\mathcal{I}|$ denote the total number subtasks of task Q_m^k. In the task graph, node $Q_{m,i}^k$ means the ith subtask belonging to task Q_m^k, and a directed edge $(Q_{m,i}^k, Q_{m,j}^k)$ denotes the subtask dependency that subtask $Q_{m,j}^k$ cannot be performed until subtask $Q_{m,i}^k$ has been completed, $i, j \in I$.

To accurately illustrate the task dependencies, we show an example in Fig. 3. 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.

Each subtask $Q_{m,i}^k$ can be described in two terms as $Q_{m,i}^k = \left\langle d_{m,i}^k, c_{m,i}^k \right\rangle$, $d_{m,i}^k$ denotes the input data size of

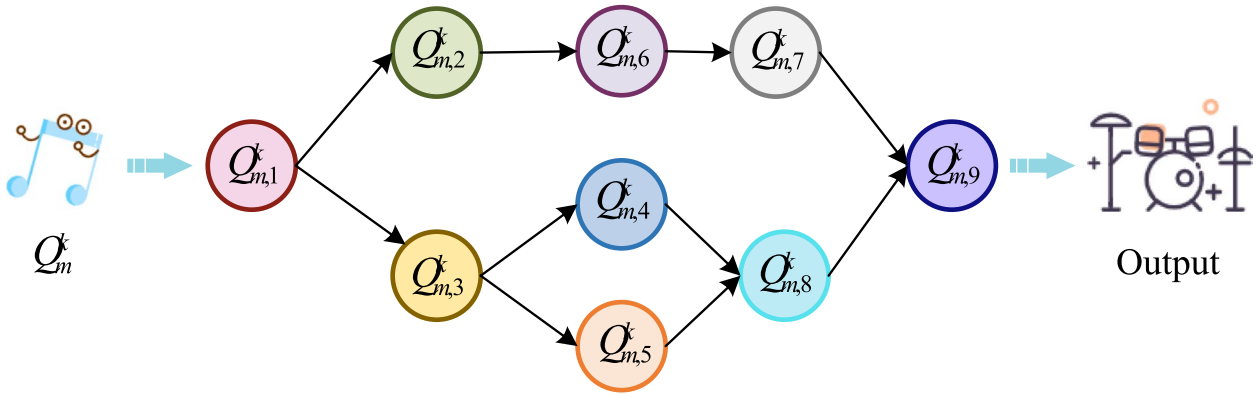

Fig. 3 An example of a directed task graph

the ith subtask and $c^k_{m,i}$ denotes the amount of computation resource required to complete subtask $Q^k_{m,i}$.

Communication model

In this section, the communication model of the task offloading is introduced, which consists of the communication model of vehicle to RSUs and the communication model of RSUs to the cloud server.

1) Vehicles to RSUs: we consider that the wireless communication (i.e., 4G, 5G, and WiFi) between vehicles and RSUs is based on the orthogonal frequency-division multiple access [2]. Specifically, we let $\lambda_{k,e}$ denote the data transmission rate between vehicle k and RSU r, which can be calculated as [2, 34]

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

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

2) RSUs to cloud server: we consider that the wireline communication (i.e., high-speed optical fiber lines) between RSUs and the cloud server. In the case MEC servers cannot provide computation services for vehicles, vehicles offload the subtasks to the remote cloud server via RSUs. Especially, it should be noted that the transmission latency of the MEC server transmitting data to the cloud server is equal to 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 [2, 34]. We let $t_{r,c}$ denote the round-trip time of transmission data between RSU r and the cloud server, which can be calculated as [2, 34]

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

where t^{cloud}_{off} denotes the transmission latency of RSU r transmitting data to the cloud server.

Computation model

Subtasks with a task can either be performed on the vehicle or be offloaded to the MEC server or cloud server for execution. Thus, in this section, the processing time of subtasks is discussed in the vehicle, the MEC server, and the cloud server, respectively.

1) Local computing: when the subtask is assigned to be executed on the vehicle, we assume that each vehicle processes one subtask at a time. In this case, if there is a subtask computed in a vehicle, other subtasks executed locally need to wait in a task queue based on the first-in-first-out principle until the computation resources are available [9, 35]. Thus, the total local completion delays of subtask $Q^k_{m,i}$ consist of the local execution delay and the local waiting delay, which is given by:

$$t^{local,completion}_{k,m,i} = \frac{c^k_{m,i}}{f_k} + t^{local,wait}_{k,m,i}. \tag{3}$$

where $\frac{c^k_{m,i}}{f_k}$ denotes the local execution delay, f_k denotes the computation capability of vehicle k, $c^k_{m,i}$ denotes the amount of computation resource required to complete subtask $Q^k_{m,i}$, and $t^{local,wait}_{k,m,i}$ is the local waiting delay of the subtask $Q^k_{m,i}$, denoted by the difference between the execution starting time and requested time as $t^{local,wait}_{k,m,i} = t^{local,start}_{k,m,i} - t^{local,request}_{k,m,i}$.

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

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

where f_e^{mec} represents the computation capability of the MEC server e.

Thirdly, the edge waiting time is similar to the local waiting time.

$$t_{k,m,i}^{mec,wait} = t_{k,m,i}^{mec,start} - t_{k,m,i}^{mec,request}. \tag{6}$$

Where $t_{k,m,i}^{edge,start}$ denotes the start execution time of the subtask, and $t_{k,m,i}^{edge,request}$ denotes the requested time of subtask.

According to equations (4), (Eq5), and (Eq6), the total completion delays for offloading subtask to the MEC server e can be defined as

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

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 RSU r, and then is transmitted from RSU r 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 [2, 16]. Therefore, the total completion delays of offloading subtask $Q_{m,i}^k$ to the cloud server can be broken down into two parts: the transmission delay between vehicle k and RSU r, and the transmission delay between RSU r and the cloud server, which is defined as

$$\begin{aligned} t_{k,m,i}^{cloud,completion} &= t_{k,m,i}^{mec,trans} + t_{e,c} \\ &= \frac{d_{m,i}^k}{\lambda_{k,e}} + t_{e,c}. \end{aligned} \tag{8}$$

where $\frac{d_{m,i}^k}{\lambda_{k,e}}$ denotes the transmission delay between vehicle k and RSU r, 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

$$\min_{(p_{m,i}^k, r_{m,i}^k)} \sum_{k \in V, m \in \mathcal{Q}} t_{m,i}^k \tag{9}$$

$$\text{s.t.} \quad p_{k,m,i}^{local} + p_{k,m,i}^{edge} + p_{k,m,i}^{cloud} = 1 \tag{10a}$$

$$\sum_{k \in V} \beta_{k,e} \le B \tag{10b}$$

$$\sum_{k \in V} x_{k,e} \le C \tag{10c}$$

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-TOD-DPG, is proposed. Compared with the value-based reinforcement learning (RL) approach (e.g., DQN) [24, 36], 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 [37], thereby efficiently solving continuous control problems and helping reach the fast convergence. Figure 4 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 (9). 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.

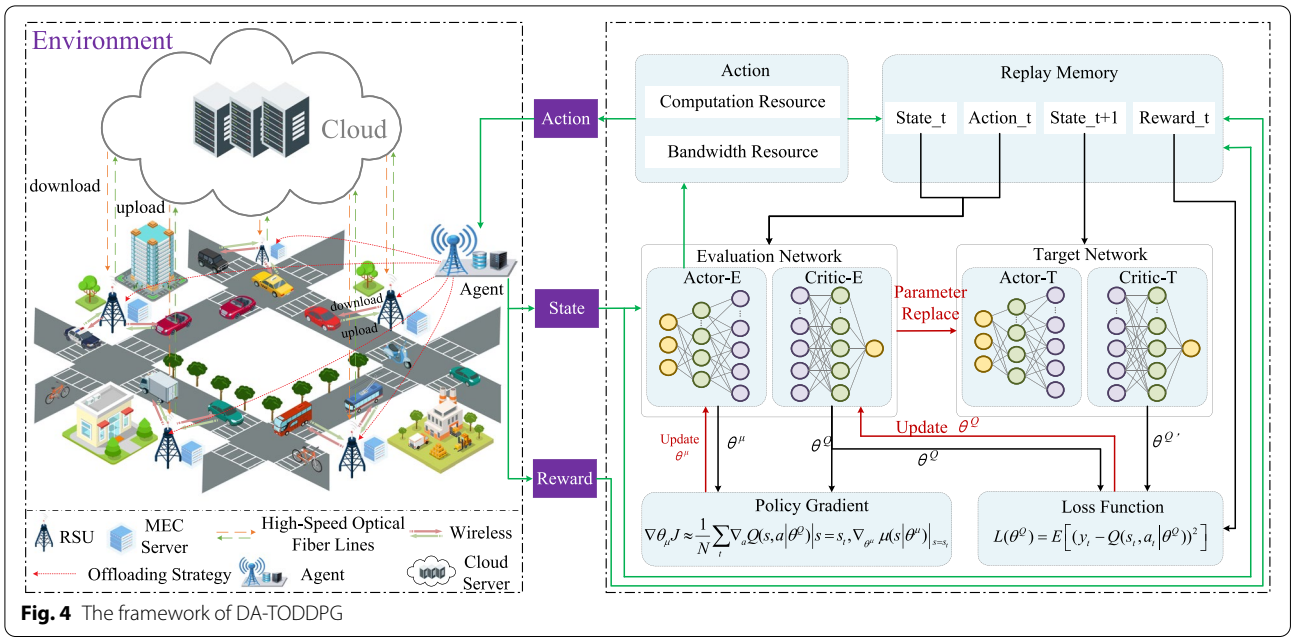

Fig. 4 The framework of DA-TODDPG

Similar to [38], we assume that there is one centralized MEC server as the agent in DA-TODDPG. The agent identifies the optimal fine-grained offloading strategy by interacting with the environment through a sequence of observations, actions, and rewards [39]. 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

$$env = \{V, \mathcal{Q}, p, \delta, B, C\}. \tag{11}$$

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

$$s_t = \{B_a, C_a\}. \tag{12}$$

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

$$a^t_{k,m,i} = \{b_{k,m,i}, c_{k,m,i}\}. \tag{13}$$

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

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

$$r^t_{k,m,i} = \begin{cases} 0, \text{ if subtask } i \text{ is executed locally} \\ t^{lcoal,completion}_{k,m,i} - t^{mec,completion}_{k,m,i}, \text{if subtask} \\ i \text{ is offloaded to the MEC server} \\ t^{local,completion}_{k,m,i} - t^{cloud,completion}_{k,m,i}, \text{if subtask} \\ i \text{ is offloaded to the cloud server} \end{cases} \tag{14}$$

where $r^t_{k,m,i}$ denotes the benefits of the action $a^t_{k,m,i}$ of subtask i of task $Q^k_{m,i}$ at the t-th time step.

Task offloading algorithm

The illustration of DA-TODDPG is shown in Fig. 4, which combines the characteristics of DQN and the actor-critic (AC) [37]. Therefore, it consists of three components, namely 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 is used to store 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 described in Algorithm 1, which mainly includes three main parts: selection (Line 7), reward evaluation (Line 9), and network update (Line 14).

Input: The size of replay memory N;
The current state s_t //Eq. (12)
Output: An action a_t
1 Initialize the 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 the target actor network with weights $\theta^{\mu'} \leftarrow \theta^{\mu}$ and the 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 $\theta^{Q}, \theta^{\mu}, \theta^{Q'}, \theta^{\mu'} = NetworkUpdate(D)$
14 **end**
15 **end**
16 **end**
17 Output a_t

Algorithm 1 DA-TODDPG Algorithm.

Reward evaluation algorithm

Evaluating the reward of each offloading decision not only enables us to obtain an optimal offloading strategy but also accelerates the convergence of the DA-TODDPG algorithm. The goal of the DA-TODDPG algorithm is to maximize the reward of performing actions. 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).

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 $Q^k_{m,i}$ // Eq. (3)
2 Execute on the MEC server completion time $t^{mec,completion}_{k,m,i}$ of subtask $Q^k_{m,i}$ // Eq. (6)
3 Execute on the cloud completion time $t^{cloud,completion}_{k,m,i}$ of subtask $Q^k_{m,i}$ // Eq. (7)
4 **if** $b_{m,i} = 0$ **then**
5 $r_t = t^{local,completion}_{k,m,i} - t^{local,completion}_{k,m,i}$ //Eq. (14)
6 **end**
7 **if** $b_{m,i} > 0$ *and* $c_{m,i} > 0$ **then**
8 $r_t = t^{local,completion}_{k,m,i} - t^{mec,completion}_{k,m,i}$ //Eq. (14)
9 **end**
10 **if** $b_{m,i} = 0$ *and* $c_{m,i} > 0$ **then**
11 $r_t = t^{local,completion}_{k,m,i} - t^{cloud,completion}_{k,m,i}$ //Eq. (14)
12 **end**
13 Output r_t

Algorithm 2 Reward Evaluation Algorithm.

Network upload algorithm

The network update is outlined in Algorithm 3. Specifically, in each training step, a minibatch of experience 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).

Input: the replay memory D
Output: θ^{Q}, θ^{μ}, $\theta^{Q'}$, $\theta^{\mu'}$
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**
  Calculate the target value: $y_t = r_t + \gamma Q(s_{t+1}, \mu(s_t \mid \theta^{\mu'}) \mid \theta^{Q'})$
  Update the θ^{Q} in evaluation critic network by minimizing the loss: $L(\theta^{Q}) = E\left[(y_t - Q(s_t, a_t \mid \theta^{Q}))^2\right]$
  Update the weights $\theta^{Q'}$ of target actor network every C step: $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
  Update the weights $\theta^{\mu'}$ of target critic network every C step: $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
  output θ^{Q}, θ^{μ}, $\theta^{Q'}$, $\theta^{\mu'}$
end
Output θ^{Q}, θ^{μ}, $\theta^{Q'}$, $\theta^{\mu'}$

Algorithm 3 Network Update Algorithm.

Experimental performance

In this section, extensive experiments are carried out to evaluate the performance of our proposed scheme. The experimental setting details are 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 the DA-TODDPG scheme.

Experimental setting

A VEC system is simulated, 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, the other parameters in the experiments are set in Table 3.

To demonstrates the effectiveness of DA-TODDPG, three baselines are selected to compare the DA-TODDPG as follows.

- **Dependency-aware random offloading (DA-RO).** The DA-RO is a traditional offloading approach without utilizing optimization algorithms, where the edge server randomly assigns sub-channels 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 sub-channels and computational resources. It is an implementation of MORL-ODT [42].
- **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 [43].

Table 3 Parameter values

Parameter	Value
the transmission power p	2 ± 0.2 Watt [40]
the channel gain δ	144 ± 14.4 dB [41]
noise power σ^{-2}	1.5×10^{-8} Watt [41]
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

Evaluation of train performance

As a DDPG-based algorithm, the train performance of the model should be guaranteed. To prove the train performance of DA-TODDPG, we first compare the convergence under different learning rates, and we second compare its convergence with other baselines.

1) Convergence with Different Learning Rates: through the simulations with different learning rates, the most appropriate value for learning rate is 0.01, which has an average reward of 2024, as shown in Fig. 5. The worst value for this parameter is 0.08, and its average reward is 2003. Thus, the learning rate is set to 0.01 in the following system simulations and evaluations.

2) Convergence with Different Baselines: Figure 6 shows the train performance of different baselines (i.e., E-TODDPG, DA-TODQN), which is the average result of multiple experiments. 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

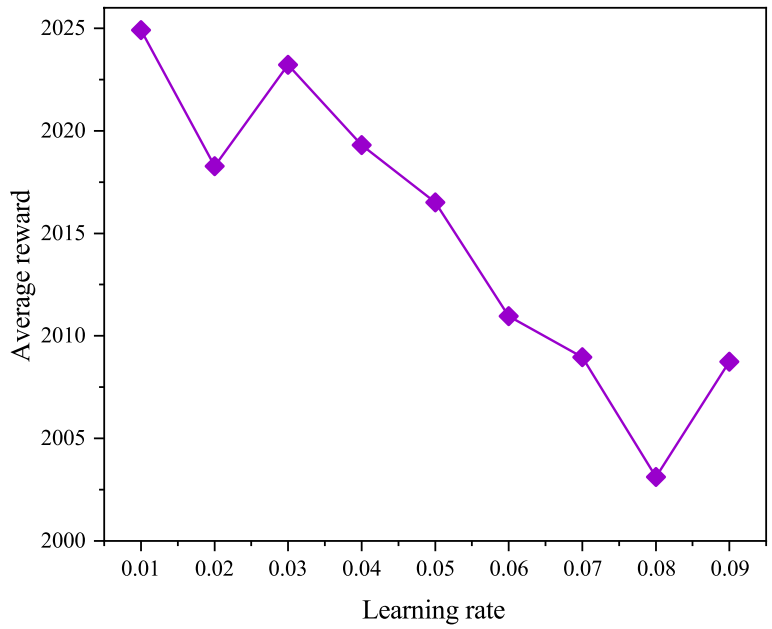

Fig. 5 The convergence performance of average reward under different learning rates

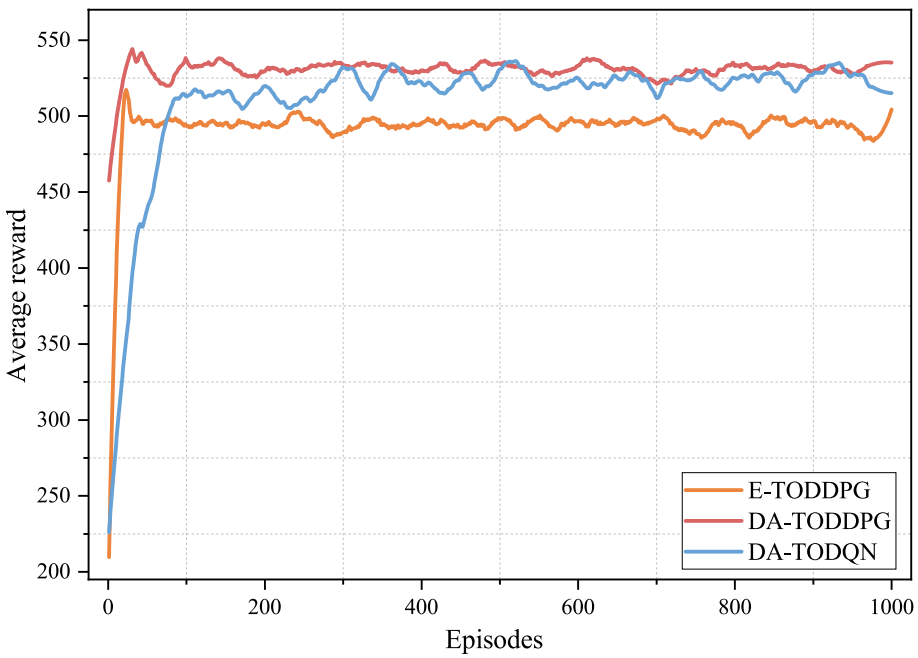

Fig. 6 The convergence performance of average reward under different baselines

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.

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 3. In each set of experiments, the value of one variable fluctuated around the control value and the other variables remained constant (Table 4).

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 Fig. 7. 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.

Table 4 Controlled Variables setting

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

2) Analysis on the Variety of MEC Server Number: In Fig. 8 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 dependency, and utilizing DDPG to solve the problem of continued action space.

3) Analysis on the Variety of Average size of Input Data: In Fig. 9, the average processing delay with diversity in the average size of input data is analyzed.

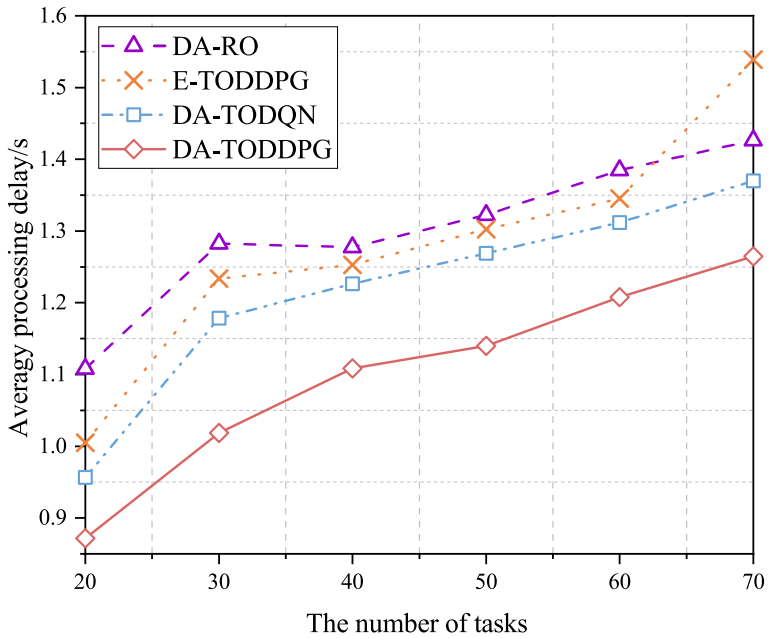

Fig. 7 Comparison of average processing delay with variety in task number

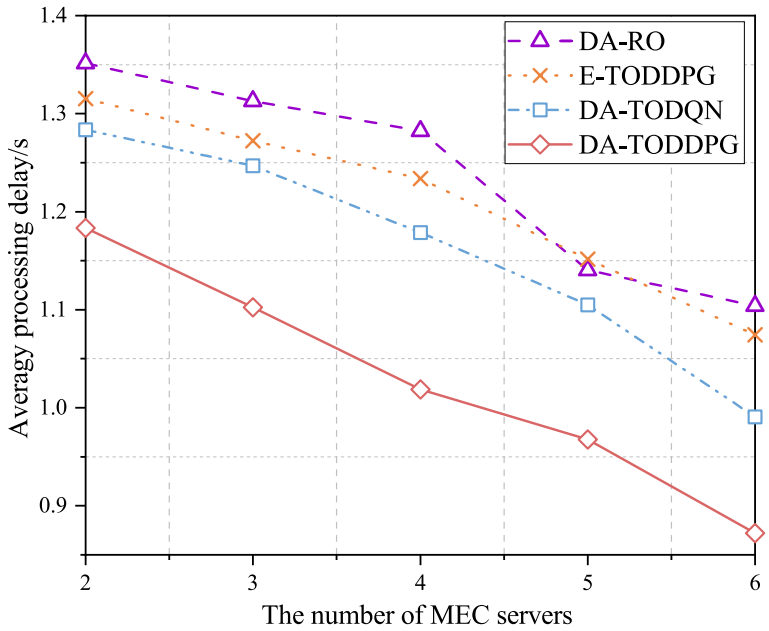

Fig. 8 Comparison of average processing delay with variety in MEC server number

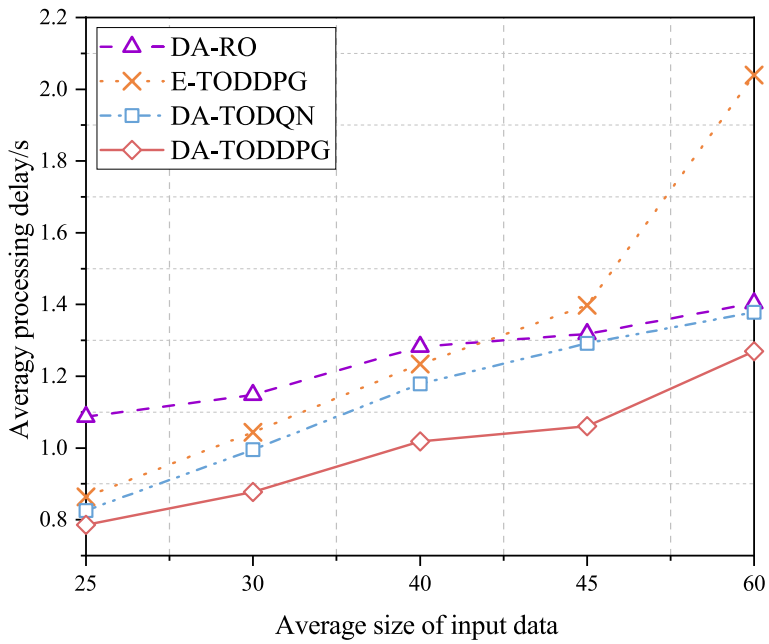

Fig. 9 Comparison of average processing delay with variety in the average size of input data

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 sub-tasks 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 DATODDPG always gains the optimal performance of the average processing delay under the different average size of input data.

Conclusion

In this paper, an efficient dependency-aware task offloading scheme is proposed for reducing the average processing delay of tasks with vehicle-edge-cloud collaborative computing. In this scheme, the directed acyclic graph technique is utilized to model the inter-subtask dependency. Then, a dependency-aware task offloading algorithm based on DDPG is designed 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, we will consider both the mobility of vehicles and data migration between edge servers during task offloading.

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.

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. The author(s) read and approved the final manuscript.

Funding
This work has been supported by the Project of National Natural Science Foundation of China under Grant No. 62262063, the Project of Key Science Foundation of Yunnan Province under Grant No. 202101AS070007, Dou Wanchun Expert Workstation of Yunnan Province No.202205AF150013, Science and Technology Youth lift talents of Yunnan Province, and the Project of Scientific Research Fund Project of Yunnan Education Department under Grant No. 2022Y561.

Availability of data and materials
Not applicable.

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.

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

Received: 28 July 2022 Accepted: 1 October 2022
Published online: 20 October 2022

References

1. Ji H, Alfarraj O, Tolba A (2020) Artificial intelligence-empowered edge of vehicles: architecture, enabling technologies, and applications. IEEE Access 8:61020–61034
2. Liu Y, Wang S, Zhao Q, Du S, Zhou A, Ma X, Yang F (2020) Dependency-aware task scheduling in vehicular edge computing. IEEE Internet Things J 7(6):4961–4971
3. Xu X, Shen B, Ding S, Srivastava G, Bilal M, Khosravi MR, Menon VG, Jan MA, Wang M (2020) Service offloading with deep q-network for digital twinning-empowered internet of vehicles in edge computing. IEEE Trans Ind Inform 18(2):1414–1423
4. Chen Y, Zhang N, Zhang Y, Chen X, Wu W, Shen XS (2019) Toffee: Task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing. IEEE Trans Cloud Comput 9(4):1634–1644
5. Liu Y, Li Y, Niu Y, Jin D (2019) Joint optimization of path planning and resource allocation in mobile edge computing. IEEE Trans Mob Comput 19(9):2129–2144
6. Zhang J, Guo H, Liu J, Zhang Y (2019) Task offloading in vehicular edge computing networks: A load-balancing solution. IEEE Trans Veh Technol 69(2):2092–2104
7. Chen Y, Zhao F, Chen X, Wu Y (2021) Efficient multi-vehicle task offloading for mobile edge computing in 6g networks. IEEE Trans Veh Technol
8. Nguyen D, Ding M, Pathirana P, Seneviratne A, Li J, Poor V (2021) Cooperative task offloading and block mining in blockchain-based edge computing with multi-agent deep reinforcement learning. IEEE Trans Mob Comput
9. Dai F, Liu G, Mo Q, Xu W, Huang B (2022) Task offloading for vehicular edge computing with edge-cloud cooperation. World Wide Web:1–19
10. Shakarami A, Ghobaei-Arani M, Shahidinejad A (2020a) A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. Comput Netw 182:107496
11. Shakarami A, Ghobaei-Arani M, Masdari M, Hosseinzadeh M (2020b) A survey on the computation offloading approaches in mobile edge/cloud computing environment: a stochastic-based perspective. J Grid Comput 18(4):639–671
12. Shakarami A, Shahidinejad A, Ghobaei-Arani M (2020c) A review on the computation offloading approaches in mobile edge computing: A g ame-theoretic perspective. Softw Pract Experience 50(9):1719–1759
13. Shakarami A, Shahidinejad A, Ghobaei-Arani M (2021) An autonomous computation offloading strategy in mobile edge computing: A deep learning-based hybrid approach. J Netw Comput Appl 178:102974
14. Liu Y, Chen CS, Sung CW, Singh C (2017) A game theoretic distributed algorithm for feicic optimization in lte-a hetnets. IEEE/ACM Trans Netw 25(6):3500–3513
15. Guo H, Liu J (2018) Collaborative computation offloading for multiaccess edge computing over fiber-wireless networks. IEEE Trans Veh Technol 67(5):4514–4526
16. Xu X, Jiang Q, Zhang P, Cao X, Khosravi MR, Alex LT, Qi L, Dou W (2022) Game theory for distributed iov task offloading with fuzzy neural network in edge computing. IEEE Trans Fuzzy Syst
17. Aceto L, Morichetta A, Tiezzi F (2015) Decision support for mobile cloud computing applications via model checking. In: 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering. IEEE, pp 199–204
18. Shu C, Zhao Z, Han Y, Min G, Duan H (2019) Multi-user offloading for edge computing networks: A dependency-aware and latency-optimal approach. IEEE Internet Things J 7(3):1678–1689

19. Yao L, Xu X, Bilal M, Wang H (2022) Dynamic edge computation offloading for internet of vehicles with deep reinforcement learning. IEEE Trans Intell Transp Syst
20. He X, Lu H, Du M, Mao Y, Wang K (2020) Qoe-based task offloading with deep reinforcement learning in edge-enabled internet of vehicles. IEEE Trans Intell Transp Syst 22(4):2252–2261
21. Yang C, Liu Y, Chen X, Zhong W, Xie S (2019) Efficient mobility-aware task offloading for vehicular edge computing networks. IEEE Access 7:26652–26664
22. Wang J, Hu J, Min G, Zhan W, Zomaya A, Georgalas N (2021) Dependent task offloading for edge computing based on deep reinforcement learning. IEEE Trans Comput
23. Zhang K, Mao Y, Leng S, He Y, Zhang Y (2017) Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading. IEEE Veh Technol Mag 12(2):36–44
24. Ren Y, Yu X, Chen X, Guo S, Xue-Song Q (2020) Vehicular network edge intelligent management: A deep deterministic policy gradient approach for service offloading decision. In: 2020 International Wireless Communications and Mobile Computing (IWCMC). IEEE, pp 905–910
25. Zhan Y, Guo S, Li P, Zhang J (2020) A deep reinforcement learning based offloading game in edge computing. IEEE Trans Comput 69(6):883–893
26. Zhang Y, Lan X, Ren J, Cai L (2020) Efficient computing resource sharing for mobile edge-cloud computing networks. IEEE/ACM Trans Networking 28(3):1227–1240
27. Chen L, Wu J, Zhang J, Dai HN, Long X, Yao M (2020) Dependency-aware computation offloading for mobile edge computing with edge-cloud cooperation. IEEE Trans Cloud Comput
28. Fan Y, Zhai L, Wang H (2019) Cost-efficient dependent task offloading for multiusers. IEEE Access 7:115843–115856
29. Pan S, Zhang Z, Zhang Z, Zeng D (2019) Dependency-aware computation offloading in mobile edge computing: A reinforcement learning approach. IEEE Access 7:134742–134753
30. Chen J, Yang Y, Wang C, Zhang H, Qiu C, Wang X (2021) Multi-task offloading strategy optimization based on directed acyclic graphs for edge computing. IEEE Internet Things J
31. Qu G, Wu H, Li R, Jiao P (2021) Dmro: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing. IEEE Trans Netw Serv Manag 18(3):3448–3459
32. Binh TH, Vo HK, Nguyen BM, Binh HTT, Yu S et al (2022) Value-based reinforcement learning approaches for task offloading in delay constrained vehicular edge computing. Eng Appl Artif Intell 113:104898
33. Huang L, Feng X, Zhang C, Qian L, Wu Y (2019) Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. Digit Commun Netw 5(1):10–17
34. Xu X, Fang Z, Qi L, Dou W, He Q, Duan Y (2021) A deep reinforcement learning-based distributed service off loading method for edge computing empowered internet of vehicles. Chin J Comput 44(12):2382–2405
35. Chen X, Liu Z, Chen Y, Li Z (2019) Mobile edge computing based task offloading and resource allocation in 5g ultra-dense networks. IEEE Access 7:184172–184182
36. Wang Y, Fang W, Ding Y, Xiong N (2021) Computation offloading optimization for uav-assisted mobile edge computing: a deep deterministic policy gradient approach. Wirel Netw 27(4):2991–3006
37. Li M, Gao J, Zhao L, Shen X (2020) Deep reinforcement learning for collaborative edge computing in vehicular networks. IEEE Trans Cogn Commun Netw 6(4):1122–1135
38. You C, Huang K, Chae H, Kim BH (2016) Energy-efficient resource allocation for mobile-edge computation offloading. IEEE Trans Wirel Commun 16(3):1397–1411
39. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533
40. Chen X, Zhang H, Wu C, Mao S, Ji Y, Bennis M (2018) Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. IEEE Internet Things J 6(3):4005–4018
41. Sun Y, Zhou S, Xu J (2017) Emm: Energy-aware mobility management for mobile edge computing in ultra dense networks. IEEE J Sel Areas Commun 35(11):2637–2646
42. Song F, Xing H, Wang X, Luo S, Dai P, Li K (2022) Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach. Futur Gener Comput Syst 128:333–348
43. Xu YH, Yang CC, Hua M, Zhou W (2020) Deep deterministic policy gradient (ddpg)-based resource allocation scheme for noma vehicular communications. IEEE Access 8:18797–18807

Publisher's Note