Wireless Networks
https://doi.org/10.1007/s11276-021-02750-8

Learning-aided fine grained offloading for real-time applications in edge-cloud computing

Qihe Huang[1] · Xiaolong Xu[1,2,3,4] · Jinhui Chen[1]

Accepted: 2 August 2021

Abstract
Edge-cloud computing has been widely adopted to provision abundant resources for latency-sensitive and computation-intensive vehicular applications in Internet of vehicles (IoV), bringing more entertainment, security, and efficiency on the road. Generally, the applications in real world are composed of massive subtasks with dependent relationships which are commonly modelled as directed acyclic graphs (DAGs), and thus fine grained offloading and parallel computing are imperative during offloading to promote the quality of service. However, due to the diversity of DAG-based applications and the complexity of dynamic edge-cloud environment, the vehicle intelligent management system is incapable of scheduling offloading with effect, resulting in additional transmission latency and energy expenditure on wireless channels and backhaul links. To reduce application response time and meanwhile save the energy consumption, a markov decision process is formulated based on the fine grained offloading with the intention of obtaining an optimal policy. Besides, to make offloading more adaptive to various application scales, a learning-aided fine grained offloading for real-time applications, named LFGO, is designed with deep q-learning in edge-cloud empowered IoV. Eventually, experiments are conducted with generated DAGs based on real-world applications, covering a wide range of subtask numbers, transmission rate and computing capability, to verify the efficiency of LFGO.

Keywords Deep reinforcement learning · Fine grained offloading · Edge-cloud computing

1 Introduction

Internet of vehicles (IoV) as an intelligent transportation system (ITS) has developed rapidly by integrating big data with advanced vehicular applications [23, 31]. In the IoV, massive data (e.g., accident notice, road information and weather condition) exchange via different communication patterns such as vehicle-to-infrastructure (V2I), vehicle-to-pedestrian (V2P) and vehicle-to-vehicle (V2V), providing enough information to the vehicles in touch. With analyses on traffic data, mobile vehicular applications (i.e., object detection, traffic forecast and etc.) are adopted in intelligent vehicles, contributing more efficiency, safety and entertainment to driving [9, 33].

Generally, the majority of such vehicular applications are computation-intensive and must be accomplished within seconds [1, 27]. Due to the limited computing resources of vehicle computing module, the timeliness of real-time applications could not be guaranteed, decreasing the quality of services (QoS) of applications [28, 32]. To

✉ Xiaolong Xu
xlxu@nuist.edu.cn

Qihe Huang
qhuang@nuist.edu.cn

Jinhui Chen
000528@nuist.edu.cn

1 School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, China

2 Jiangsu Engineering Center of Network Monitoring, Nanjing University of Information Science and Technology, Nanjing, China

3 Engineering Research Center of Digital Forensics, Ministry of Education, Nanjing, China

4 Jiangsu Collaborative Innovation Center of Atmospheric Environment and Equipment Technology (CICAEET), Nanjing University of Information Science and Technology, Nanjing, China

Published online: 15 August 2021

address the limited resources problem, vehicles tend to offload their latency-sensitive applications to other processing nodes. Mobile edge computing (MEC), deploying the edge servers in close proximity to the roadside, emerges as a novel computing paradigm to supply relatively abundant computing resources and storage spaces for the vehicular applications in IoV [7, 8]. Besides, to make full use of computing resources and realize parallel computing, the MEC is usually combined with cloud computing to be adaptive to the requirements of applications [12, 13]. In this collaborative computing paradigm, MEC is potential to alleviate the high transmission congestion on the backhaul links while the cloud computing is capable of supplying extraordinarily rich resources, making offloading more flexible [10, 11].

In real-world scenarios, the majority of vehicular applications are composed of massive subtasks with dependent relationships [3]. For instance, there are 10-30 subtask components in a small size machine vision application for vehicles [22]. Generally, to realize fine-grained offloading and enable parallel processing, the subtasks with the relationships in vehicular application could be modelled as directed acyclic graphs (DAGs) [20]. Particularly, if two interdependent subtasks are processed at different computing nodes, additional latency for data transmission and growth of energy consumption of vehicles may be produced [2]. In contrast, if two subtasks with dependency are processed at the same computing node, no extra data transmission would be generated. Therefore, the offloading service utility of DAG-based applications is substantially influenced by the scheduling policy.

As to the vehicular applications, the QoS is mainly determined by the application response time which depends on the completion time of all the internal subtasks [4]. Once the latency-sensitive application is not responded within the highly limited time, the intelligent services provided for drivers would fail to take effect, even threatening the safety of driving in severe cases [21]. Besides, the increased energy expenditure in processors and transmission channels resulted from improper processing scheduling is a great problem for the vehicles that consume available and limited energy resources to support energy-hungry applications [5]. Therefore, to avoid unnecessary data transmission and redundant energy consumption of the vehicles, the subtask dependency must be fully utilized. However, except for a few extremely simple cases, the DAG-based application offloading problem is inherently NP-hard due to the dynamic states of edge-cloud computing and the complexity of DAG [22].

To realize fine grained offloading efficiently, deep reinforcement learning (DRL) which combined the perceptual ability of deep learning (DL) with the decision-making ability of reinforcement learning (RL), is adopted to improve the service utility [29]. In DRL environment, the edge-cloud empowered IoV system including vehicular applications, computing nodes, wireless channels and vehicle intelligent management system (VIMS) is formulated as a markov decision process (MDP), and thus an optimal offloading strategy could be learned by trial and error. In this paper, a learning-aided fine grained offloading for real-time applications, named LFGO, is proposed based on deep q-learning, reducing the application response time, saving energy consumption, meanwhile making offloading more adaptive to various application scales.

The main contributions of this paper are:

- Analyze the response time of processing vehicular applications containing interdependent subtasks in the edge-cloud empowered IoV.
- Model the fine-grained offloading process as a markov decision process (MDP) and the definition of optimal scheduling scheme is put forward.
- Adopt DRL in LFGO with the intention of reducing application response time, decreasing energy consumption and making it more adaptive to various application scales.
- Conduct experiments with generated DAGs based on real-world applications, covering a wide range of subtask numbers, transmission rate and computing capability.

The rest of the paper is organized as follows. The related work of this paper is reviewed in Sect. 2. The system model of offloading vehicular applications and problem formulation is presented in Sect. 3. Section 4 described the design of LFGO in detail. Section V analyzed the experiment results obtained by simulation evaluation. Finally, this paper is concluded and the important future works are presented in Sect. 6.

2 Related work

Various mobile applications have emerged with the intention of providing more comfort, intelligence and safety for vehicles in the rapid developing trend of IoV. For instance, assisted driving and emergency failure management make it possible for drivers to achieve flexible driving on the road. To realize such service, the requirement of application response time is very strict for most of the mobile applications in IoV are latency-sensitive and computationally intensive. If all of the complete applications are run locally in vehicles, the vehicle computing module would be overloaded and unable to ensure the service quality. To improve the efficiency and service quality of vehicular applications, edge-cloud computing as a novel computing paradigm with function of computation

offloading is proposed to lay the resource foundation for IoV.

The edge-cloud empowered IoV is promising to provide available computing resources for vehicles with the assistance of servers respectively deployed along the road and in remote data center. Existing computation offloading studies in edge-cloud empowered IoV generally focus on optimizing resource allocation and reducing latency. Hou et al. [9] introduced reliable task allocation mechanism and partial computation offloading strategy to better using edge computing resources in a complex IoV environment. Xu et al. [26] aimed at designing a collaborative strategy for the placement and quantification of edge servers near the road side unit, providing vehicle media services with high reliability and low response time. Pei et al. [17] proposed a power allocation method with non-orthogonal multiple access, reducing the system latency in IoV. Ning et al. [16] pushed artificial intelligence inspired edge resources to the proximity of vehicles and formulated a mix integer non-linear programming problem to optimize network delay in edge-enabled IoV. Xu et al. [25] focused on reducing the suspect of service trustworthiness, and they presented a trust-aware task offloading strategy in IoV with edge computing. Zhou et al. [30] contributed to designing a decentralized reinforcement learning to better control traffic light, promoting the utilization of the collected data from the IoV. Qu et al. [19] focused on the fine grained offloading and they proposed a deep meta reinforcement learning-based offloading strategy with parallel DNNs. Altogether, the work in edge-cloud empowered IoV is moslty to promote the allocation of computing resources, e.g., task offloading and edge server deployment, while the algorithms and framework are in constant progress. For example, [17] only focus on reduce latency while [25] integrate it with privacy reservation.

For the complexity of edge-cloud computing environment, it is frequently difficult to decide whether or where offload the task, and thus using artificial intelligence (AI) algorithm to make offloading decision is practicable. DRL owns powerful representation ability which could fully fit the optimal strategy and adapt to complex environment, and is widely considered as an effective method to solve decision-making problems in complex environment. Recently, to make offloading strategies more adaptive in dynamic edge environment, studies on computation offloading using DRL have emerged. Dai et al. [6] exploited the DRL algorithm to achieve an optimized content caching strategy empowered by blockchain for vehicular edge computing. Peng et al. [18] utilized RL to optimize the multi-dimensional resource allocation in two typical multi-access edge computing architecture for vehicle network. Zhan et al. [29] addressed the problem of privacy exposure in terms of preference and bandwidth in edge environment, and they used DRL-based approach to make the offloading decision without such information. Luo et al. [15] investigated on the vehicular edge computing and solved the data scheduling of minimizing data processing cost with certain latency constraints using deep q-network (DQN). Liu et al. [14] studied on the mobile edge computing mounted in unmanned aerial vehicles, planning the trajectory of unmanned aerial vehicles to improve the quality of service of offloading computational tasks for terminal users. In general, DRL is potential to be adopted to design an adaptive offloading method for DAG-based applications in edge-cloud empowered IoV.

3 System model

In this section, an edge-cloud computing framework in IoV for vehicular applications is proposed. After that, the vehicular application offloading with improving service utility problem is formulated. Key terms frequently used in the model is listed in Table 1.

3.1 Framework of fine grained offloading for IoV applications in edge-cloud computing

A framework of fine grained offloading for vehicular applications in edge-cloud empowered IoV is shown in Fig. 1. In the IoV, the vehicle need process its mobile application, i.e., vehicle detection, speed prediction, etc, to better its driving. Generally, a mobile application is composed of multiple subtasks with dependencies (e.g., machine vision application contains 10-30 subtasks). To realize fine-grained offloading, a vehicular application can be modelled as a DAG denoted by $AP = (ST, E)$, where ST and E are the vertex set of subtasks and the edge set of the dependency relationships, respectively. Each vertex st_i $(st_i \in ST)$ represents a subtask in the application, and each

Table 1 Key terms and descriptions

Terms	Descriptions
ST	The set of subtasks, $ST = \{st_1, st_2, \ldots, st_n\}$
F^{lo}	Local execution power of vehicle computing module
K^{lo}	Local CPU clock speed of vehicle computing module
K^{ed}	CPU clock speed of edge server
U_{ul}^{ed}, U_{dl}^{ed}	Edge transmission rate of up-link and down-link
K^{cd}	CPU clock speed of cloud server
U_{ul}^{cd}, U_{dl}^{cd}	Cloud transmission rate of up-link and down-link
c_i	Required CPU cycles of the subtask st_i
d_i^{in}, d_i^{out}	Input and output data size of subtask st_i

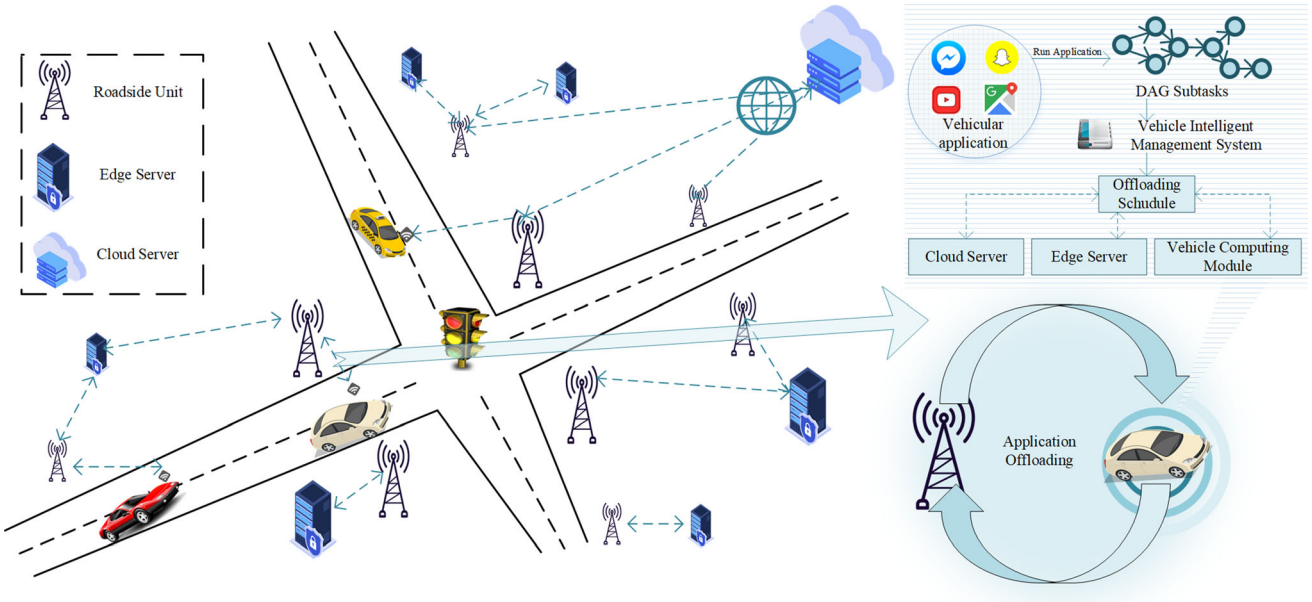


Fig. 1 A framework of fine grained offloading for vehicular applications in edge-cloud empowered IoV

directed edge $e(st_i, st_j)$ $(e(st_i, st_j) \in E)$ means that the execution of task st_j depends on task st_i. To be specific, for a dependency relationship $e(st_i, st_j)$, st_i is called the predecessor of st_j, while st_j is the successor of st_i. A subtask can only be executed after all its predecessor subtasks have been completed. In DAG, the subtask without any predecessor is called the entry subtask, and the subtask without any successor is called the exit subtask. An application can have multiple entry subtasks and exit subtasks at the same time. For a subtask st_i $(i \leq n)$, it is defined by a tuple $(d_i^{in}, d_i^{out}, c_i)$, where d_i^{in},d_i^{out} and c_i are the input data size, output data size and required CPU cycles of the subtask st_i, respectively.

These subtasks could be executed locally by the vehicle computing module or offloaded to the cloud server or edge server through wireless channels.

The cloud server with sufficient computation and storage resources is deployed in a remote cloud data centre. As a substitute for the cloud server, edge server is deployed near the road to improve feedback speed. The vehicle intelligent management system (VIMS) deals with execution sequence and the offloading of the subtasks of mobile applications. According to the different implementation requirements of subtasks, the offloading is decided.

3.2 Application response time model of fine-grained offloading

The quality of offloading service largely depends on the completion time, and thus the application response time model is proposed to evaluate the service quality. Let a logical variable ω_i denote whether st_i is offloaded, where $\omega_i = 0$ means st_i is run locally and $\omega_i = 0$ means it is offloaded. Then a logical variable λ_i (1 for yes, 0 for no) is used to represent if st_i is offloaded to the edge server. The total time to complete a subtask is composed of the execution time and transmission time. The execution time of st_i is given by

$$T_i^{exe} = \omega_i \cdot \frac{c_i}{K^{lo}} + (1-\omega_i) \cdot \left(\lambda_i \cdot \frac{c_i}{K^{ed}} + (1-\lambda_i) \cdot \frac{c_i}{K^{cd}}\right), \tag{1}$$

where K^{lo}, K^{ed} and K^{cd} are the CPU clock speed of the vehicle computing module, the edge serve and the cloud server, respectively. If a subtask is not run locally, apart from execution time, the transmission time accounts for a large proportion of response time.

The transmission time of st_i is expressed as

$$\begin{aligned} T_i^{tra} =& (1-\omega_i) \cdot \left(\lambda_i \cdot \left(\frac{d_i^{in}}{U_{ul}^{ed}} + \frac{d_i^{out}}{U_{dl}^{ed}}\right)\right. \\ &\left. + (1-\lambda_i) \cdot \left(\frac{d_i^{in}}{U_{ul}^{cd}} + \frac{d_i^{out}}{U_{dl}^{cd}}\right)\right), \end{aligned} \tag{2}$$

where U_{ul}^{ed}, U_{dl}^{ed}, U_{ul}^{cd} and U_{dl}^{cd} are edge up-link transmission rate, edge down-link transmission rate, cloud up-link transmission rate and cloud down-link transmission rate, respectively. In this work, the output data size is considered to be much smaller than the input data size and the up-link transmission rate is set as the same with down-link transmission rate [22]. After calculating the transmission time and the execution time, the completion time of st_i is

$$T_i^{fin} = T_i^{exe} + T_i^{tra}. \tag{3}$$

Before scheduling st_i, it must be ensured that all of its predecessors have been scheduled. The ready time of scheduling st_i is given by

$$RT_i = \max_{st_j \in \text{pred}(st_i)} \{RT_j + T_j^{fin}, 0\}, \tag{4}$$

where $pred(st_i)$ is the predecessor subtask set of st_i. If the subtask is the entry subtask, its ready time would be zero. The finish time of st_i is expressed as

$$FT_i = RT_i + T_i^{fin}. \tag{5}$$

The total response time of the entire application is the time when all of its exit subtasks have been completed (the data return has been completed), and it is given by

$$T^{total}(AP) = \max_{st_i \in \text{exit}(AP)} \{RT_i + T_i^{fin}\} \tag{6}$$

where $exit$ (AP) is the set of all exit subtasks in this application.

3.3 Energy-aware model in offloading application

During offloading process, there is an inevitable loss of energy. The energy consumption mainly consists of transmission consumption and execution consumption. The transmission energy consumption is given by

$$E_i^{exe} = \omega_i \cdot c_i \cdot F^{lo} \tag{7}$$

where F^{lo} is the execution power of the vehicle computing module.

The transmission power is derived by Shannon formula, and it is calculated as

$$p_i = \frac{\delta(2^{\frac{U}{B}} - 1)}{H^2}, \tag{8}$$

where H is channel gain (fading degree of radio channel normalized to transmission distance), δ is variance of complex Gaussian white noise and B is channel bandwidth. The transmission energy consumption is calculated as

$$E_i^{tra} = \left(\lambda_i \cdot \left(\frac{d_i^{in}}{U_{ul}^{ed}} + \frac{d_i^{out}}{U_{dl}^{ed}}\right) + (1 - \lambda_i) \cdot \left(\frac{d_i^{in}}{U_{ul}^{cd}} + \frac{d_i^{out}}{U_{dl}^{cd}}\right)\right) \cdot p_i. \tag{9}$$

The total energy consumption of the application is expressed by

$$E^{total}(AP) = \sum_{i=1}^{n} E_i^{exe} + E_i^{tra}. \tag{10}$$

3.4 Problem formulation

For mobile applications in IoV scenarios are usually delay sensitive, to maximize the user experience, we need to find the optimal task scheduling scheme for different APs to minimize the $T^{total}(AP)$. Besides, to improve vehicle endurance and reduce energy waste, minimizing the local energy consumption is also a necessity. The optimized objective is expressed as

$$\begin{aligned} &\min \sum_{st_i \in AP} E_i^{exe} + E_i^{tra}, \\ &\min \max_{st_i \in \text{exit}(AP)} \{FT_i\}, \end{aligned} \tag{11}$$

In the process of offloading scheduling, it is necessary to determine the execution mode (local execution or offloading execution) and scheduling order (when there are multiple ready tasks at the same time) of each task, The scheduling of tasks on service instances (the scheduling order of ready tasks) will also affect the results of global scheduling. In this paper, it is stipulated that the scheduling order of tasks on service instances is consistent with that determined by the task scheduling module in vehicle. The problem in Eq. (11) is NP-hard due to the highly dynamic DAG topologies and edge-cloud environment states, so finding the optimal offloading strategy can be extremely challenging [20]. In the next section, we present the details of LFGO for handling this problem.

4 LFGO for IoV in edge-cloud computing using deep reinforcement learning

In this section, LFGO is designed for the fine-grained offloading with edge-cloud computing. First, the markov decision process (MDP) is introduced in the fine-grained offloading. Based on the MDP, deep Q-Learning is applied to optimize decision quality of the offloading for multiple mobile applications of vehicles.

4.1 Decide scheduling priority of subtasks

For any mobile application $AP = (ST, E)$, the scheduling cost of each subtask $st_i (st_i \in ST)$ is defined as

$$sc_i = \min\{T_i^{exe} + T_i^{tra}\}, \forall \omega_i \in \{0, 1\}, \forall \lambda_i \in \{0, 1\}. \tag{12}$$

That is the shortest time to execute the subtask by offloading. Based on sc_i, the scheduling priority $PO(st_i)$ of task st_i is determined recursively as

$$PO(st_i) = \max_{st_j \in suc(st_i)} \{PO(st_j) + sc_i\}, st_i \notin \text{exit}(AP), \tag{13}$$

$$PO(st_i) = sc_i, st_i \in \text{exit}(AP), \tag{14}$$

where $suc(st_i)$ denotes all the successor subtasks of st_i. In particular, if the st_i is the exit subtask, its scheduling priority equals sc_i. By traversing the whole DAG from the exit subtask, the scheduling priority of all subtasks can be calculated recursively. By arranging all the tasks on AP in descending order based on the scheduling priority, the scheduling subtask sequence is obtained, which is expressed as

$$\mu = \{st'_1, st'_2, \ldots st'_{|ST|}\}. \tag{15}$$

The scheduling of subtasks in AP starts from the first element st'_1, and scheduling decisions would be made in turn until all tasks are executed. It is worth noting that $mu$is also a topological sort of AP. Scheduling tasks in this order ensures the original dependency between subtasks. The while process of determine scheduling priority of AP is shown in Algorithm 1.

Algorithm 1: Determine scheduling priority of AP

```
Data: AP = (ST, E)
Result: μ
for each st_i in exit (AP) do
    PO(st_i) ← min{T_i^exe + T_i^tra}, ∀ω_i ∈ {0,1}, ∀λ_i ∈ {0,1}
end
Intialize the list of PO-updated subtasks μ with the exit(AP)
while |μ| ≠ |ST| do
    for each st_j in μ do
        for each st_i in pre(st_j) do
            if st_i ∉ μ and suc(st_i) ⊆ μ then
                PO(st_i) ← max{PO(suc(st_i)) + sc_i}
                Add subtask st_i to μ
            end
        end
    end
end
Sort μ in descending order
return μ
```

4.2 Fine-grained offloading as a Markov decision process

For any $AP = (ST, E)$, its corresponding sorted subtask sequence is μ. The process of offloading tasks one by one in μ can be modelled as an MDP, expressed as $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. The elements from left to right represent the state space, action space, state transition matrix, reward function and discount factor of the problem, respectively.

4.2.1 Action space

The action space is defined as $\mathcal{A} = \{0, 1, 2\}$. A is a history sequence actions of μ, which describes the offloading of all the subtasks of AP. A is defined as $A = (a_1, a_2, ..., a_{|\mu|})$, where a_i is the action of offloading subtask st'_i. The action $a_i = 0$ means to execute the current subtask locally, the action $a_i = 1$ means to offload the current subtask to an edge server, the action $a_i = 2$ means to offload the current subtask to the cloud server. In particularly, if the subtask st'_i has not been scheduled, the a_i will be filled with -1. So when the subtask st'_i is scheduled, a_i could be expressed as

$$a_i = \omega_i \cdot (\gamma_i + 1), \tag{16}$$

whereω_i and γ_i are the decision variables mentioned in the Sect. 3 of offloading st'_i.

4.2.2 State space

The state is defined based on the the topological sequence μ being scheduled. The state space can be expressed as

$$\mathcal{S} = \{s | s = (d(st'_i), c(st'_i), A, i), 1 \le i \le |\mu|\} \tag{17}$$

where A and m are the history action sequence of μ and the number of scheduled subtasks, respectively.

4.2.3 Reward function

For any state $s_i = (d(st_i'), c(st_i'), A, i)$ is acknowledged, the scheduled subtask sequence in μ is represented as $\mu_{1:i-1}$. Since μ is a topological sequence of AP, its scheduled subtask sequences in μ could be constructed as a subgraph, expressed as $AP_{1:i}$. It's worth mentioning that $AP_{1:|\mu|}$ is the complete AP and $AP_{1:0}$ means there is no scheduled subtask. The difference between the running time and energy of the scheduled subgraph according to action a_i in state s_i are give as

$$e_i = E^{total}(AP_{1:i}) - E^{total}(AP_{1:i-1}), \tag{18}$$

$$t_i = T^{total}(AP_{1:i}) - T^{total}(AP_{1:i-1}). \tag{19}$$

To minimize the application response time and optimize the energy consumption of the vehicles, the reward function is defined as

$$r_i = -\left(\alpha \cdot \frac{e_i - e_{min}}{e_{max} - e_{min}} + \beta \cdot \frac{t_i - t_{min}}{t_{max} - t_{min}}\right), \tag{20}$$

where e_{min} and e_{max} are the minimal and maximal subtask energy consumption among ST, t_{min} and t_{max} are the minimal and maximal subtask latency among ST, α and β are the weight coefficients satisfied with $\alpha + \beta = 1$. Normalization is to reduce the difference between different subtask size. The reward is negative for the time and energy consumption are both supposed to be small.

4.2.4 Markov decision process

The offloading strategy of the VIMS is defined as a conditional probability function expressed as $\pi(a_i|s_i)$. Based on the current state s_i, the strategy function π gives the probability of selecting different actions a_i. Starting from the initial state s_1, the decision process will enter a new state and get a reward every time VIMS performs an action according to $\pi(a_i|s_i)$. When the last subtask $st'_{|ST|}$ in μ is completed, the mobile application is finished. The whole subtask scheduling process can be expressed as

$$(s_1, a_1, r_1, ..., s_{|ST|}, a_{|ST|}, r_{|ST|}, s_{end}), \tag{21}$$

where s_{end} is the ending state, indicating that all subtask scheduling has been completed. The cumulative reward with discount for this process is given by

$$R = \sum_{i=1}^{|V|} \gamma^{i-1} \left(\alpha \cdot \frac{e_i - e_{min}}{e_{max} - e_{min}} + \beta \cdot \frac{t_i - t_{min}}{t_{max} - t_{min}}\right). \tag{22}$$

If the state transition matrix of the MDP is known, the value function of the MDP can be expressed recursively by Bellman equation. Then, the optimal scheduling strategy $\pi(a_i|s_i)$ can be obtained by value iteration or policy iteration. Due to the diversity of DAGs being scheduled and the state space of the problem is infinite, it is impossible to obtain the state transition matrix in advance. Therefore LFGO is used to find the optimal strategy for the unloading scheduling module.

4.3 Deep Q-learning in LFGO

The deep neural network is used to fit the strategy function $\pi(a_i|s_i)$ in the modelled MDP. The offloading strategy fitted by neural network is called main network, which is expressed as $\pi_\theta(a_i|s_i)$, where θ is the parameter of neural network. The input of neural network is any state $s_i = (d(st_i'), c(st_i'), A, i)$. To obtain the optimal subtask scheduling strategy π_θ, the training objective of LFGO can be given as

$$\max_\theta L(\theta) = \max_\theta E\left[\pi_\theta^*(\mu \mid AP) \sum_{i=1}^{|\mu|} \gamma^{i-1} \mathcal{R}(s_i, a_i)\right], \tag{23}$$

where $\pi_\theta^*(\mu \mid \boldsymbol{AP}) = \prod_{i=1}^{|\mu|} \pi_\theta(a_i \mid s_i)$ meaning the probability distribution of different scheduling decision sequences based on subtask DAG of AP. The ultimate goal of training is to maximize the expectation of reward under different scheduling decision sequences. The algorithm continuously collects the scheduled DAGs in the system to construct the training DAG set, and trains the scheduling strategy on the training DAG set. Based on the excellent generalization ability of deep neural network, the trained task scheduling strategy can also schedule DAGs outside the training set. It can be considered that deep neural network can learn the general mode of DAG scheduling well through training, which can be verified in subsequent experiments. In order to help the training converge better, the single step excitation value is scaled in the training process. By dividing the single step reward value by the maximum reward value in the scheduling process, the single step reward is kept in [0,1].

Algorithm 2: LFGO using deep q-learning

Intialize Q with random parameters θ
Intialize Q^{target} with parameters $\theta^* = \theta$
$step \leftarrow 0$
for *episode* ***from*** *1* ***to*** M **do**
  Intialize sequence $\mathcal{S}$ of a random μ
  for i ***from*** *1* ***to*** N **do**
    $a_i \leftarrow argmax_a Q(s_i, a; \theta)$
    Execute a_i to offload a subtask
    Achieve s_{i+1}, r_{i+1} after execution
    Store transition $(s_i, a_i, r_{i+1}, s_{i+1})$ in D
    Sample a random batch $(s_t, a_t, r_{t+1}, s_{t+1})$ from D
    if s_{t+1} *is the end state* **then**
      $y_t \leftarrow r_{t+1}$
    end
    else
      $y_t \leftarrow r_{t+1} + \gamma \cdot max_{a'} Q^{target}(s_{t+1}, a'; \theta^*)$
    end
    Perform a gradient descent step on $(y_t - Q(s_t, a; \theta))^2$ with respect to θ of Q
  end
  $step \leftarrow step + 1$
  if *step % X is 0* **then**
    $Q^{target} \leftarrow Q$
  end
end
return θ

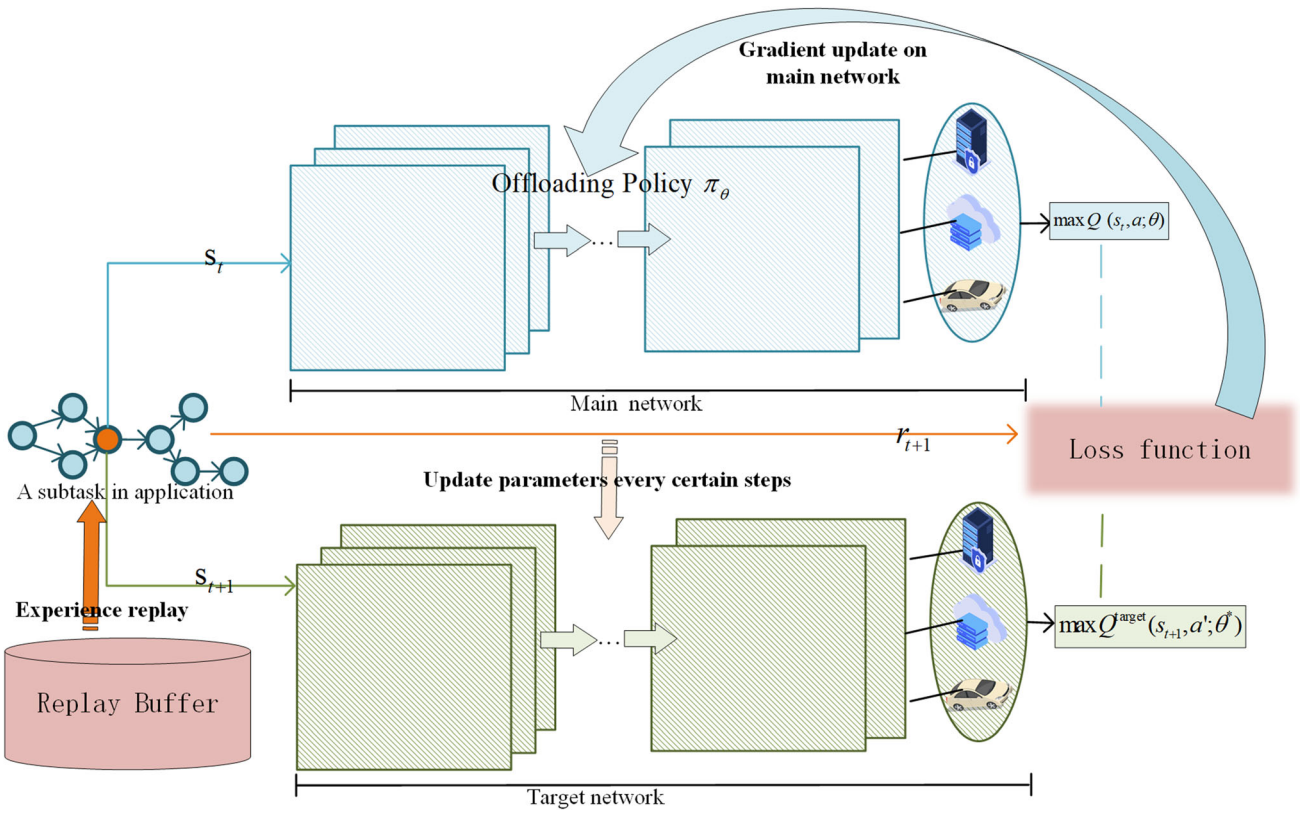


Fig. 2 An example of training process in LFGO with experience replay and target network

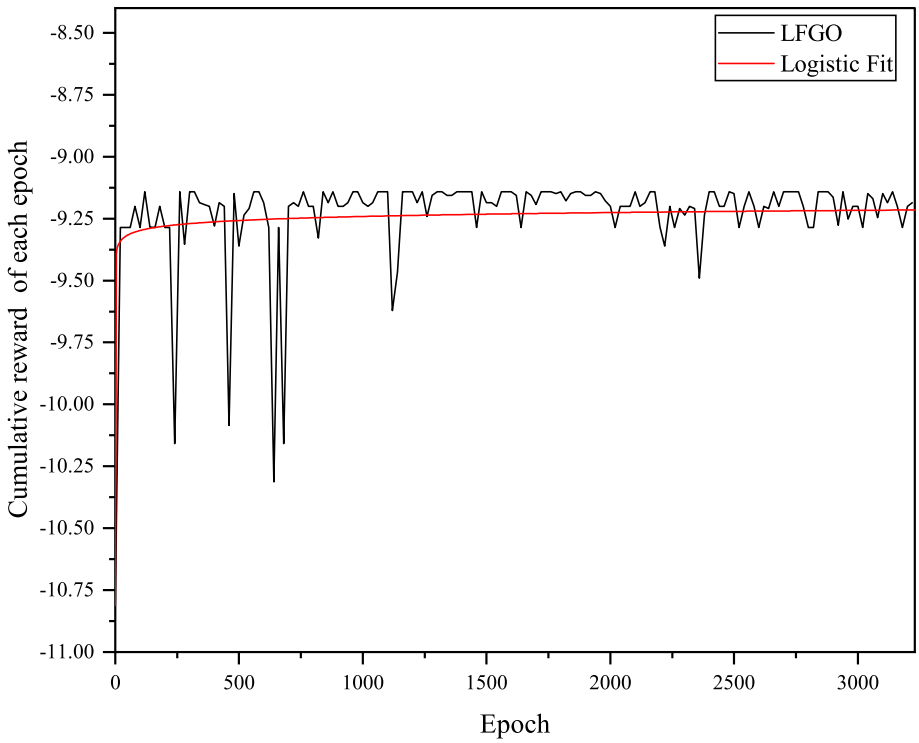


Fig. 3 Training curve of LFGO with increasing epochs

As to training process, time deference is introduced to train the network parameters, meaning the network would perform a gradient descent every offloading step after accumulating a certain amount of offloading data. Besides, for the update of parameters θ of Q is prone to shock and presents unstable learning behavior, the target network is proposed. Firstly, the parameters θ^* of target network is initialized by the same parameters of the main network. In the target network, θ^* will be updated independently of the main network at every X step. The basis of the update for main network is

$$loss = (r_{t+1} + \gamma \cdot max_{a'} Q^{target}(s_{t+1}, a'; \theta^*) - Q(s_t, a; \theta))^2. \tag{24}$$

This allows the parameters of main network to be fixed temporarily during the training process, making the learning process more stable. Futhermore, experience replay is proposed to learn different strategies. It stores the data obtained from the offloading process, and then randomly samples the data to update the θ.

Figure 2 describes an example of training process in LFGO with experience replay and target network. Firstly, a subtask of DAG-based application which is represented as $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay buffer is taken out. Then the maximum Q value of s_t is calculated by main network while the maximum Q value of s_{t+1} is given by target network. Next, the loss is calculated by Eq. (24) and the parameters of main network would be updated by gradient descent based of the loss. Last, the parameters in target network would be replaced with θ of main network every certain steps. The training process of achieving offloading strategy of LFGO using deep q-learning is presented in Algorithm 2.

5 Experiment and performance

In this section, extensive simulations have been completed to evaluate the performance of LFGO. Firstly, the simulation parameters of experiments are given while three comparison methods are introduced. Then the convergence ability of LFGO is verified. Next, the efficiency in terms of application response time and energy consumption are compared with the other three offloading methods. Finally, the adaptability of offloading different applications is confirmed.

5.1 Experiment setup

The input data size of each subtask in DAG ranges from 20 KB to 100 KB while the output data size is set to be much little than the input. The computation size of each subtask is sampled from 1×10^7 CPU cycles to 2×10^8 CPU cycles. The edge transmission rate of up-link and down-link is both 10 Mbps while the remote cloud transmission rate of up-link and down-link is both set as 3 Mbps. The CPU clock speed of vehicle computation module, edge server and cloud server are respectively set as

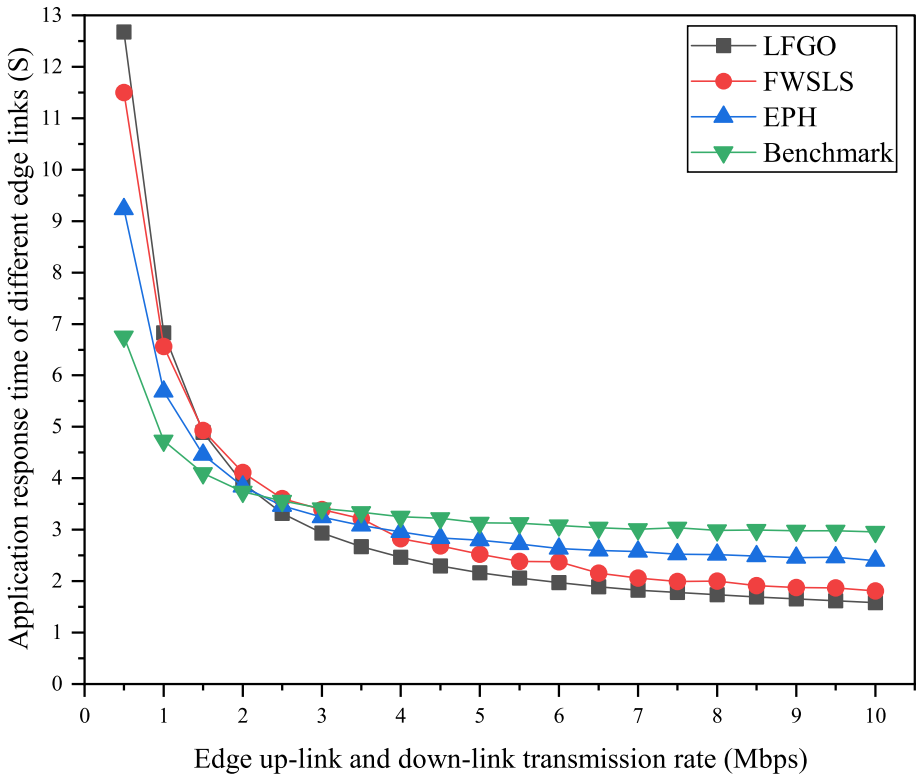


Fig. 4 Comparison of application response time with different edge transmission rate when $U_{ul}^{cd} = U_{dl}^{cd} = 3$ Mbps

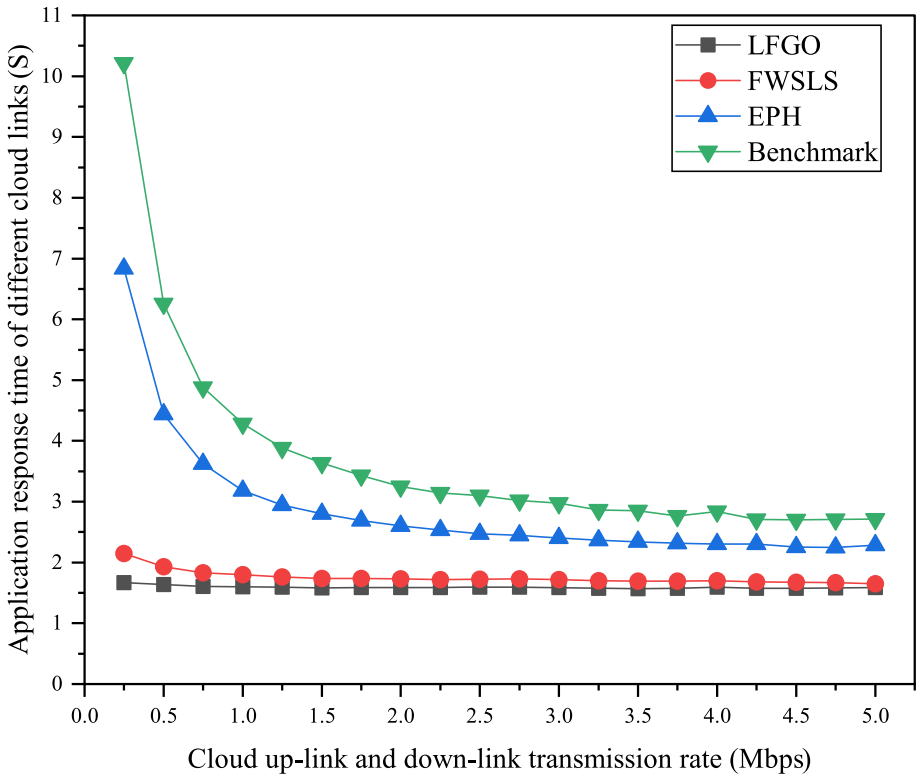


Fig. 5 Comparison of application response time with different cloud transmission rate when $U_{ul}^{ed} = U_{dl}^{ed} = 10$ Mbps

Table 2 Application response time (s) with different edge computing capability

CPU clock speed	LFGO	FWSLS	EPH	Benchmark
5 GHZ	2.58018	2.761253	3.020203	3.29328
10 GHZ	1.581934	1.708054	2.422973	2.928039
15 GHZ	1.259312	1.373971	2.202718	2.788667
20 GHZ	1.087813	1.202973	2.12275	2.751379
25 GHZ	0.991592	1.118138	2.041429	2.753782
30 GHZ	0.927882	1.036563	2.022181	2.709256
35 GHZ	0.880623	1.012567	1.957241	2.67706
40 GHZ	0.843798	0.962118	1.970494	2.709005
45 GHZ	0.814242	0.932444	1.909209	2.669747
50 GHZ	0.793687	0.918876	1.922134	2.678683

2×10^9,1×10^{10} and 5×10^{10} CPU cycles per second. The transmission power is 0.1 J/s and the execution power of vehicle computation module is 0.2J/10^9 CPU cycles. The reward discount factor γ in LFGO is 0.9, and α and β are both 0.5.

5.2 Comparative offloading methods

- Fine-Tuned Win Stay Lose Shift (FWSLS) [24]: FWSLS is a behavioural model based on simple learning method for the VIMS. At the beginning, the action probability of the subtask is the same. Then, with the continuous migration, the probability of each action will change. If the reward of offloading the previous subtask is higher than a certain threshold, the action probability of migrating the previous task will increase; correspondingly, if the reward of offloading the previous task is lower than a certain threshold, the action probability of migrating the previous task will decrease.
- Edge Priority Highest (EPH): The subtasks are arranged into a topological sequence, and then they are offloaded to the cloud, edge and end circularly according to a certain probability. The probability of offloading the subtask to the edge is pretty higher than other choices.
- Benchmark: Benchmark is a random-based algorithm, in which each subtask would be run locally or offloaded with random.

5.3 Analysis on the convergence ability of LFGO

Firstly, the convergence ability of the algorithm is verified. The proposed LFGO is trained based on 1000 training applications, and the training results are recorded during the training process. In this work, the batch size of testing data is 100 and the applications used for training and testing are all composed of 100 subtasks. At the end of each epoch, a batch of applications used for testing are input into the current training policy network to obtain the offloading scheduling decisions. Then, the subtasks are scheduled in the simulation environment based on the decision, and the average cumulative discounted reward is recorded. As shown in Fig. 3, the cumulative discounted reward increased sharply in the early stage of training and began to stabilize after 1700 epochs. Besides, the cumulative average reward occasionally fluctuates greatly in the early stage of training due to the stochastic policy. But this large fluctuation of rewards becomes smaller and smaller with the deepening of training, especially after 1700 epochs. By fitting the training curve, we can find that the cumulative discounted reward would be stable at -9.25 with small fluctuations, meaning a fine training effect has been achieved. The experiment verifies the significant convergence ability of LFGO.

5.4 Analysis on the service quality of offloading scheme

As to real-time mobile applications of vehicles, the service quality , i.e., application response time, is extremely important in improving the efficiency of intelligent driving. Thus, the application response time is supposed to be short enough to satisfy the effectiveness for a given period during driving, always within seconds.

5.4.1 Evaluation with diversity in transmission rate

To evaluate the performance of LFGO with diversity in transmission rate, the edge and cloud link are adjusted in the following experiment. Firstly, as shown in Fig. 4, we set edge transmission rate grows from 1 Mbps to 10Mbps with a step size of 0.5 Mbps and let $U_{ul}^{cd} = U_{dl}^{cd} = 3$ Mbps. In the beginning, when edge transmission rate is extremely slower than the transmission speed of cloud link ($U_{ul}^{ed} = U_{dl}^{ed} \leq 2$ Mbps), the application response time of LFGO is higher than the other three offloading methods. For LFGO is trained in the environment where the edge transmission speed is faster than the cloud, unable to cope with such extreme conditions. Then the edge transmission rate enters the normal (faster than the cloud), the LFGO performs better in terms of reducing the application response time, reaching up to 1.58s when edge uplink and downlink transmission rate are both 10 Mbps. In average, LFGO shortens response time 13.2% more than FWSLS, 25.3% more than EPH and 35.2% more than Benchmark. Next, the impact of changes in cloud transmission rate on offloading is evaluated. As shown in Fig. 5, the edge transmission rate is set as 10 Mbps and cloud transmission

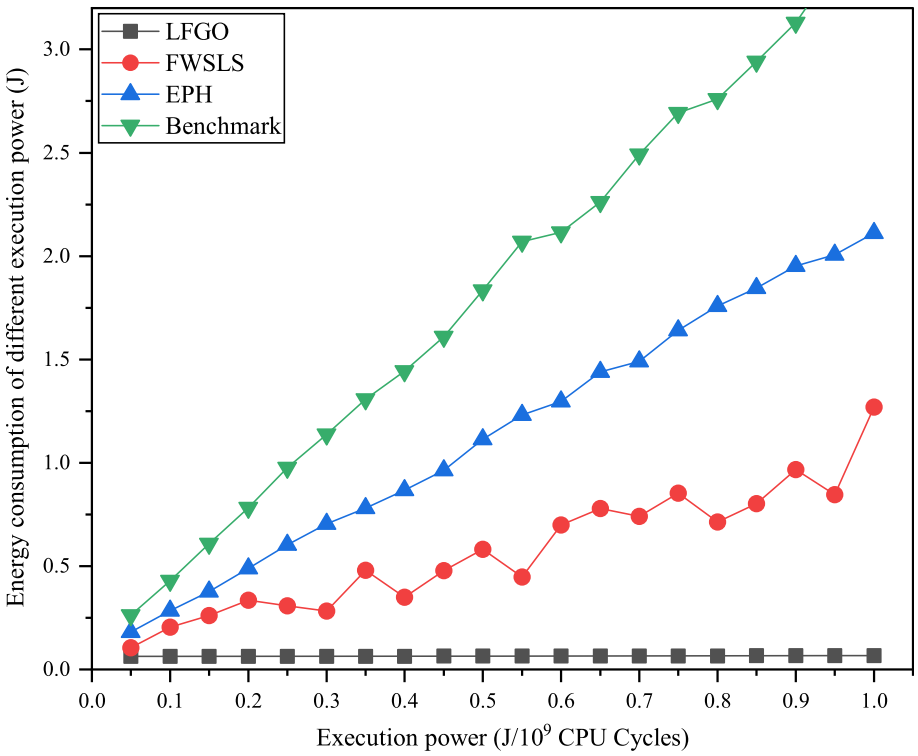


Fig. 6 Comparison of energy consumption with different local execution power when transmission power is 0.1 J/s

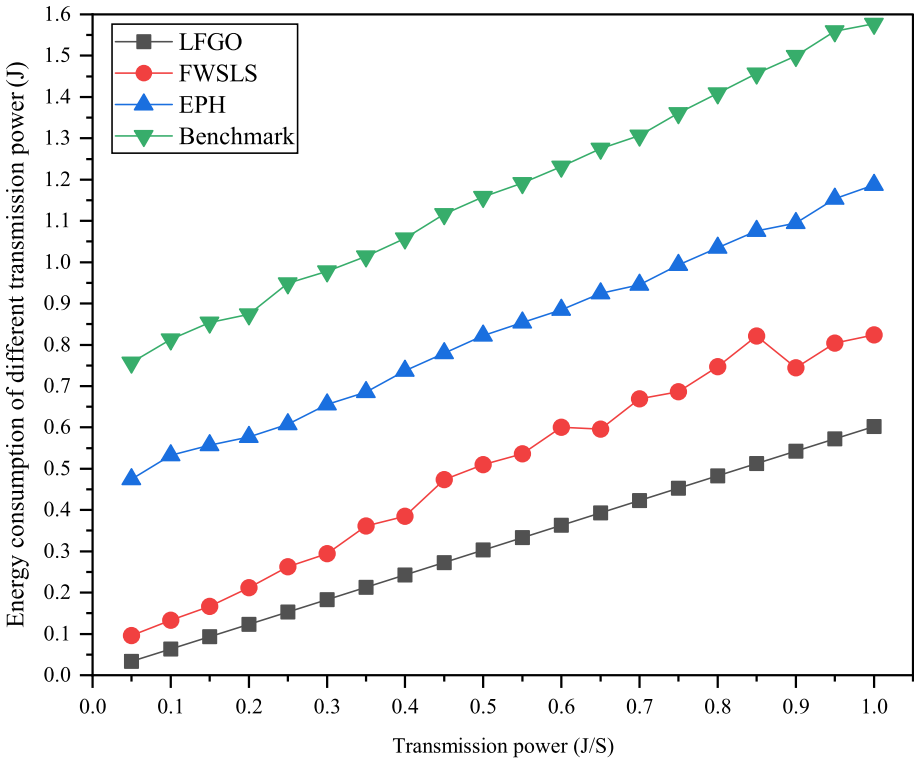
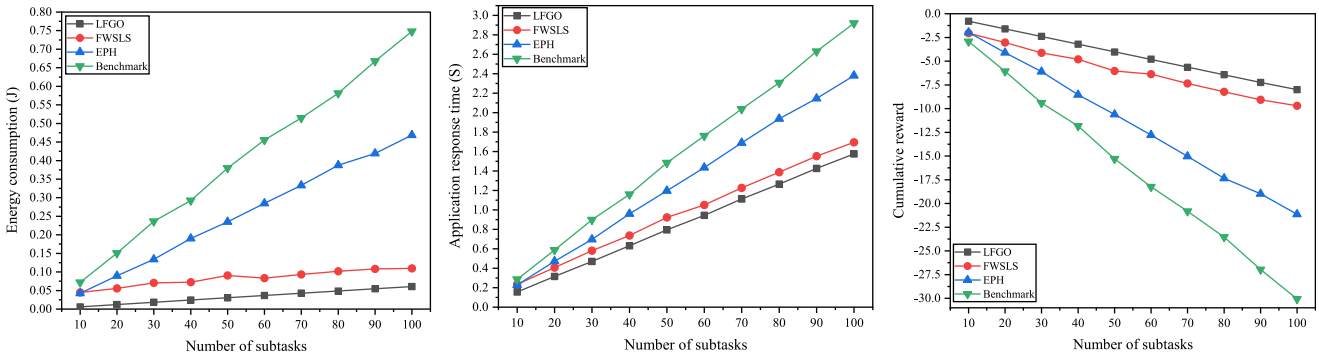


Fig. 7 Comparison of energy consumption with different transmission power when local execution power is 0.2 J/10^9 CPU cycles



(a) Energy consumption **(b)** Application response time **(c)** Cumulative reward

Fig. 8 Adaptability evaluation with various scales of applications

rate grows from 0.25 Mbps to 5 Mbps with a step size of 0.25 Mbps. The experiment shows LFGO is very little affected by changes of the cloud link, and the application response time is pretty stable. LFGO maintains 1.58 s with the fluctuation of no more than 0.02 s during the increasing of cloud transmission rate. In average, LFGO decreases application latency 8.7% more than FWSLS, 40.3% more than EPH and 52.2% more than Benchmark in the changing cloud links. Thus, LFGO is able to obtain low latency of responding the applications with various transmission rate.

5.4.2 Evaluation with diversity in computing capability

Table 2 exhibits the application response time of four offloading methods with diversity in edge computing capability. With CPU clock speed of edge rising from 5 GHZ to 50 GHZ, all the offloading methods reduce their application response time. When the CPU clock speed reaches 25 GHZ, LFGO reduces the response time to less than 1s, which only takes 0.991 s. From the experimental results, LFGO is better than EPH and Benchmark especially when the edge CPU clock speed is high, and outperforms FWSLS consistently (9.8% shorter than FWSLS averagely). The result reveals LFGO is available to reduce response latency in different computing environments.

5.5 Analysis on the energy consumption of vehicles with various types in IoV

The energy consumption of vehicles in the process of offloading is mainly composed of two parts, one is the consumption of local computing, the other is the transmission consumption caused by offloading. For the execution power and transmission power vary greatly with the type of vehicle, to evaluate the performance in terms of energy consumption influenced by vehicle types, the execution and transmission power are adjusted in the following experiment. Firstly, the transmission power is set as 0.1 J/s and the execution power is selected from 0.05 J/10^9 CPU cycles to 1×10^9 CPU cycles with a step size of 0.05 J/10^9 CPU cycles. Figure 6 shows that LFGO keeps energy consumption low (0.065 J in average) in the execution power growing period while the other offloading methods have a substantial increase of energy expenditure. The result implicates LFGO is well adaptive to different execution power, and thus suits vehicles with different computing ability. Next, the execution power is set as 0.1 J/10^9 CPU cycles and the transmission power is selected from 0.05 J/s to 1 J/s with a step size of 0.05 J/s. Figure 7 shows that the energy consumption of LFGO increases with the growth of transmission power while the trend of increasing is lower than the other offloading methods, and this trend gap is getting smaller. In average, LFGO saves 39.1% less energy than FWSLS, 50.4% less energy than EPH and 74.9% less energy than Benchmark. Thus, LFGO is potential to decrease the energy consumption of vehicles with various types in IoV.

5.6 Analysis on the adaptability to different applications

To assess the adaptability of offloading methods, the performance in terms of application response time, energy consumption and cumulative reward is evaluated with different scales of applications. The subtask size of application increases from 10 to 100 with a step size of 10. Figure 8a shows that LFGO constantly keeps the energy consumption low of different applications while the expenditure of EPH and Benchmark are both badly affected by the changing application scale. In performance of energy consumption with different application scales, LFGO decreases energy expenditure averagely 62.8% less than FWSLS, 86.9% less than EPH and 91.8% less than Benchmark. Figure 8b illustrate that LFGO does well in reducing application response time and thus promoting service quality, and LFGO controls the latency within 1 s

when the number of subtasks is less than 70. The application response time of LFGO and FWSLS both increase linearly with the growth of subtask number while LFGO keeps outperforming FWSLS with approximately 14.7% ahead. As to cumulative discounted reward which is the cumulative discounted negative values of weighted and normalized energy expenditure and response time, LFGO also maintains highest value compared to the other offloading methods. Figure 8c shows LFGO increases cumulative rewards 32.4% more than FWSLS, 61.8% more than EPH and 73.3% more than Benchmark. These results indicate that LFGO is capable of adapting to new applications with better performance than the other offloading methods.

6 Conclusion and future work

In this paper, an edge-cloud computing paradigm is proposed to empower the vehicular applications in IoV. For the applications in real world scenario are composed of massive interdependent subtasks, the subtasks in applications are modeled as directed acyclic graphs (DAGs) with the intention of realizing fine grained offloading and implementing parallel computing. Besides, markov decision process (MDP) based offloading process is proposed to describe and analyze the system state during offloading subtasks in edge-cloud empowered IoV. To tackle with the additional transmission latency and energy expenditure resulting from the complex edge-cloud environment and various application scales, a learning-aided fine grained offloading method, named LFGO, is proposed. In LFGO, deep q-learning, a deep reinforcement learning (DRL) algorithm integrating reinforcement learning (RL) with deep learning (DL), is adopted to fit the optimal strategy, reducing application response time and saving energy consumption of vehicles, meanwhile making offloading more adaptive to various application scales. In the last, experiments are conducted with generated DAGs based on real-world applications, covering a wide range of subtask numbers, transmission rate and computing capability, to verify the efficiency of LFGO. In our future work, we will contribute to applying LFGO to real scene, taking more reliable details of vehicular applications and IoV environment into account.

Acknowledgements This research is supported by the Financial and Science Technology Plan Project of Xinjiang Production and Construction Corps under Grant No. 2020DB005, the National Natural Science Foundation of China under Grant No.61702277. This research is also supported by the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD) fund and NUIST Students' Platform for Innovation and Entrepreneurship Training Program under Grant No. 202010300024Z.

References

1. Chen, C., Xiao, T., Qiu, T., Lv, N., & Pei, Q. (2020). Smart-contract-based economical platooning in blockchain-enabled urban internet of vehicles. *IEEE Transactions on Industrial Informatics, 16*(6), 4122–4133. https://doi.org/10.1109/TII.2019.2954213
2. Dai, H., Wu, X., Xu, L., Chen, G., & Lin, S. (2013). Using minimum mobile chargers to keep large-scale wireless rechargeable sensor networks running forever. In: *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, (pp 1–7).
3. Dai, H., Wu, X., Chen, G., Xu, L., & Lin, S. (2014). Minimizing the number of mobile chargers for large-scale wireless rechargeable sensor networks. *Computer Communications, 46*, 54–65.
4. Dai, H., Chen, G., Wang, C., Wang, S., Wu, X., & Wu, F. (2015). Quality of energy provisioning for wireless power transfer. *IEEE Transactions on Parallel and Distributed Systems, 26*(2), 527–537.
5. Dai, H., Ma, H., Liu, A. X., & Chen, G. (2018). Radiation constrained scheduling of wireless charging tasks. *IEEE/ACM Transactions on Networking, 26*(1), 314–327.
6. Dai, Y., Xu, D., Zhang, K., Maharjan, S., & Zhang, Y. (2020). Deep reinforcement learning and permissioned blockchain for content caching in vehicular edge computing and networks. *IEEE Transactions on Vehicular Technology, 69*(4), 4312–4324. https://doi.org/10.1109/TVT.2020.2973705
7. Fu, X., Yu, F. R., Wang, J., Qi, Q., & Liao, J. (2020). Performance optimization for blockchain-enabled distributed network function virtualization management and orchestration. *IEEE Transactions on Vehicular Technology, 69*(6), 6670–6679. https://doi.org/10.1109/TVT.2020.2985581
8. He, Q., Cui, G., Zhang, X., Chen, F., Deng, S., Jin, H., Li, Y., & Yang, Y. (2020). A game-theoretical approach for user allocation in edge computing environment. *IEEE Transactions on Parallel and Distributed Systems, 31*(3), 515–529. https://doi.org/10.1109/TPDS.2019.2938944
9. Hou, X., Ren, Z., Wang, J., Cheng, W., Ren, Y., Chen, K. C., & Zhang, H. (2020). Reliable computation offloading for edge-computing-enabled software-defined iov. *IEEE Internet of Things Journal, 7*(8), 7097–7111. https://doi.org/10.1109/JIOT.2020.2982292
10. Hu, P., Chen, W., He, C., Li, Y., & Ning, H. (2020). Software-defined edge computing (sdec): Principle, open iot system architecture, applications, and challenges. *IEEE Internet of Things Journal, 7*(7), 5934–5945. https://doi.org/10.1109/JIOT.2019.2954528
11. Li, B., He, Q., Chen, F., Jin, H., Xiang, Y., & Yang, Y. (2021). Auditing cache data integrity in the edge computing environment. *IEEE Transactions on Parallel and Distributed Systems, 32*(5), 1210–1223. https://doi.org/10.1109/TPDS.2020.3043755
12. Li, E., Zeng, L., Zhou, Z., & Chen, X. (2020a). Edge ai: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications, 19*(1), 447–457. https://doi.org/10.1109/TWC.2019.2946140
13. Li, Y., Wang, X., Gan, X., Jin, H., Fu, L., & Wang, X. (2020b). Learning-aided computation offloading for trusted collaborative mobile edge computing. *IEEE Transactions on Mobile Computing, 19*(12), 2833–2849. https://doi.org/10.1109/TMC.2019.2934103
14. Liu, Q., Shi, L., Sun, L., Li, J., Ding, M., & Shu, F. (2020). Path planning for uav-mounted mobile edge computing with deep reinforcement learning. *IEEE Transactions on Vehicular*

Technology, 69(5), 5723–5728. https://doi.org/10.1109/TVT.2020.2982508
15. Luo, Q., Li, C., Luan, T. H., & Shi, W. (2020). Collaborative data scheduling for vehicular edge computing via deep reinforcement learning. *IEEE Internet of Things Journal, 7*(10), 9637–9650. https://doi.org/10.1109/JIOT.2020.2983660
16. Ning, Z., Zhang, K., Wang, X., Guo, L., Hu, X., Huang, J., et al. (2020). Intelligent edge computing in internet of vehicles: A joint computation offloading and caching solution. *IEEE Transactions on Intelligent Transportation Systems*. https://doi.org/10.1109/TITS.2020.2997832.
17. Pei, X., Yu, H., Wang, X., Chen, Y., Wen, M., & Wu, Y. (2020). Noma-based pervasive edge computing: Secure power allocation for iov. *IEEE Transactions on Industrial Informatics*. https://doi.org/10.1109/TII.2020.3001955.
18. Peng, H., & Shen, X. (2020). Deep reinforcement learning based resource management for multi-access edge computing in vehicular networks. *IEEE Transactions on Network Science and Engineering, 7*(4), 2416–2428. https://doi.org/10.1109/TNSE.2020.2978856
19. Qu, G., Wu, H., Li, R., & Jiao, P. (2021). Dmro: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing. *IEEE Transactions on Network and Service Management*. https://doi.org/10.1109/TNSM.2021.3087258.
20. Sahni, Y., Cao, J., Yang, L., & Ji, Y. (2020). Multi-hop offloading of multiple dag tasks in collaborative edge computing. *IEEE Internet of Things Journal*. https://doi.org/10.1109/JIOT.2020.3030926.
21. Tang, M., & Wong, V. W. S. (2020). Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing*. https://doi.org/10.1109/TMC.2020.3036871.
22. Wang, J., Hu, J., Min, G., Zomaya, A. Y., & Georgalas, N. (2020). Fast adaptive task offloading in edge computing based on meta reinforcement learning. *IEEE Transactions on Parallel and Distributed Systems, 32*(1), 242–253.
23. Wang, T., Cao, Z., Wang, S., Wang, J., Qi, L., Liu, A., Xie, M., & Li, X. (2020). Privacy-enhanced data collection based on deep learning for internet of vehicles. *IEEE Transactions on Industrial Informatics, 16*(10), 6663–6672. https://doi.org/10.1109/TII.2019.2962844
24. Xu, X., Shen, B., Ding, S., Srivastava, G., Bilal, M., Khosravi, M. R., Menon, V. G., Jan, M. A., & Maoli, W. (2020). Service offloading with deep q-network for digital twinning empowered internet of vehicles in edge computing. *IEEE Transactions on Industrial Informatics*
25. Xu, X., Wu, Q., Qi, L., Dou, W., Tsai, S., & Bhuiyan, M. Z. A. (2020). Trust-aware service offloading for video surveillance in edge computing enabled internet of vehicles. *IEEE Transactions on Intelligent Transportation Systems*. https://doi.org/10.1109/TITS.2020.2995622.
26. Xu, X., Shen, B., Yin, X., Khosravi, M. R., Wu, H., Qi, L., & Wan, S. (2021). Edge server quantification and placement for offloading social media services in industrial cognitive iov. *IEEE Transactions on Industrial Informatics, 17*(4), 2910–2918. https://doi.org/10.1109/TII.2020.2987994
27. Xue, X., Wang, S., Zhang, L., Feng, Z., & Guo, Y. (2018). Social learning evolution (sle): Computational experiment-based modeling framework of social manufacturing. *IEEE Transactions on Industrial Informatics, 15*(6), 3343–3355.
28. Xue, X., Chen, Z., Wang, S., Feng, Z., Duan, Y., & Zhou, Z. (2020). Value entropy: A systematic evaluation model of service ecosystem evolution. IEEE Transactions on Services Computing
29. Zhan, Y., Guo, S., Li, P., & Zhang, J. (2020). A deep reinforcement learning based offloading game in edge computing. *IEEE Transactions on Computers, 69*(6), 883–893. https://doi.org/10.1109/TC.2020.2969148
30. Zhou, P., Chen, X., Liu, Z., Braud, T., Hui, P., & Kangasharju, J. (2020). Drle: Decentralized reinforcement learning at the edge for traffic light control in the iov. *IEEE Transactions on Intelligent Transportation Systems*. https://doi.org/10.1109/TITS.2020.3035841.
31. Zhou, X., Liang, W., Shimizu, S., Ma, J., & Jin, Q. (2020). Siamese neural network based few-shot learning for anomaly detection in industrial cyber-physical systems. *IEEE Transactions on Industrial Informatics, 17*(8), 5790–5798.
32. Zhou, X., Liang, W., She, J., Yan, Z., & Wang, K. (2021a). Two-layer federated learning with heterogeneous model aggregation for 6g supported internet of vehicles. *IEEE Transactions on Vehicular Technology*
33. Zhou, X., Xu, X., Liang, W., Zeng, Z., Shimizu, S., Yang, L. T., & Jin, Q. (2021b). Intelligent small object detection based on digital twinning for smart manufacturing in industrial cps. *IEEE Transactions on Industrial Informatics*

Qihe Huang is currently pursuing the B.S. degree in software engineering with the School of Computer and Software, Nanjing University of Information Science and Technology, China. His research interests include edge computing and deep learning.

Xiaolong Xu received the Ph.D. degree in computer science and technology from Nanjing University, China, in 2016. He was a Research Scholar with Michigan State University, USA, from April 2017 to May 2018. He is currently a Professor with the School of Computer and Software, Nanjing University of Information Science and Technology. He has published more than 80 peer-review articles in international journals and conferences. He received the Best Paper Award from the IEEE CBD 2016, IEEE CPCSCom 2020 and SPDE 2020. His research interests include edge computing, the Internet of Things (IoT), cloud computing, and big data. He is a fellow of EAI (European Alliance for Innovation).

JinHui Chen is currently an Associate Professor with the School of Computer and Software, Nanjing University of Information Science and Technology, China. Her research interests include services computing and sensor networks.