DOI:10.1587/transinf.2024IIP0005

Publicized:2024/09/24

PAPER *Special Section on Intelligent Information Processing Technology to be Integrated into Society*

A Multi-Agent Deep Reinforcement Learning Algorithm for Task offloading in future 6G V2X Network*

Jiakun LI[†], Jiajian LI[†], Yanjun SHI[†a)], Hui LIAN[††], Haifan WU[†], ***Nonmember***

SUMMARY In future 6G Vehicle-to-Everything (V2X) Network, task offloading of mobile edge computing (MEC) systems will face complex challenges in high mobility, dynamic environment. We herein propose a Multi-Agent Deep Reinforcement Learning algorithm (MADRL) with cloud-edge-vehicle collaborations to address these challenges. Firstly, we build the model of the task offloading problem in the cloud-edge-vehicle system, which meets low-latency, low-energy computing requirements by coordinating the computational resources of connected vehicles and MEC servers. Then, we reformulate this problem as a Markov Decision Process and propose a digital twin-assisted MADRL algorithm to tackle it. This algorithm tackles the problem by treating each connected vehicle as a agent, where the observations of agents are defined as the current local environmental state and global digital twin information. The action space of agents comprises discrete task offloading targets and continuous resource allocation. The objective of this algorithm is to improve overall system performance, taking into account collaborative learning among the agents. Experimental results show that the MADRL algorithm performed well in computational efficiency and energy consumption compared with other strategies.

key words: *multi-agent deep reinforcement learning, mobile edge computing, task offloading, cloud-edge-vehicle system.*

1. Introduction

The sixth generation (6G) Vehicle-to-Everything (V2X) networks [1][2], as an integral part of the next-generation communication technology, are poised to bring unprecedented transformations to autonomous driving. With the rapid development of the digital society, there is an increasingly urgent demand for future communication networks to handle large-scale data, support ultra-low latency applications, and achieve comprehensive intelligent connectivity. However, high mobility and dynamic environment pose more complex challenges for task offloading in mobile edge computing (MEC) systems.

In recent years, driven by MEC, artificial-intelligent assisted methods have achieved significant success in the field of wireless communications [2] [3]. Specifically, Deep Reinforcement Learning (DRL), widely applied in network optimization to make optimal decisions, has garnered considerable attention in the edge computing domain. For instance, Dudu et al. [5] proposed a reinforcement learning algorithm based on Deep Deterministic Policy Gradient (DDPG) to achieve optimal computation offloading. Li et al. [6] modeled the offloading problem as a Markov decision process and hence employed the Actor-Critic algorithm to minimize service costs.

Recent evidences suggest that digital twin (DT), an approach to providing real-time replicas of physical objects, continuously synchronizing with physical objects, and optimizing the operation of physical systems, will play a crucial role in 6G networks [9]. Some studies integrate to DT with wireless communications to enhance system performance. Lu et al.[11] introduced DT into wireless edge networks, proposing the DT edge network model to assist MEC servers in making optimal edge computing decisions by perceiving dynamic network states. The results of these studies demonstrate the effectiveness of DT in reducing latency and optimizing the edge computing performance.

However, due to the high mobility and dynamic environment of mobile networks, MEC encounters such thorny challenges. Therefore, MEC needs to flexibly schedule computing resources to meet the high demands of 6G V2X networks.

To address these issues, we build the model of the task offloading problem in the cloud-edge-vehicle system. Furthermore, we develop a Multi-Agent Deep Reinforcement Learning (MADRL) algorithm to find optimal task offloading decisions and resource allocation strategies. The main contributions of this paper are summarized as follows:

- We propose a task offloading and resource allocation problem in the cloud-edge-vehicle system. The DT is integrated with the model to synchronize real-time network status and assist to make optimal decisions.
- We reformulate task offloading and resource allocation as a partially observable Markov Decision Process, where the goal of each CV agent is to select the best processing device and request appropriate resources to handle incoming tasks, thereby minimizing the system cost of task execution (i.e., energy consumption and latency).

†The author is with the School of Mechanical Engineering, Dalian University of Technology, Dalian, China.

††The author is with Tebian Electric Apparatus Co., Ltd., Xinjiang, China.

*This work was supported by Science and Technology development program for Innovation-driven Development Pilot Zone of Silk Road Economic Belt and Wuchangshi National Independent Innovation Demonstration Zone under Grant 2023LQ02004.

a)E-mail: syj@ieee.org (Corresponding author)

- We propose a Digital Twin-assisted MADDPG (DT-MADDPG) algorithm, where CVs act as agents make independent actions and train critic networks. In this algorithm, we tackle the problem of mixed discrete and continuous action spaces while making sure to keep the coupling constraints between different variables.

The structure is as follows. In Section 2, we present a review of related work. In Section 3, we outline the cloud-edge-vehicle system. In Section 4, we formulate the task migration problem to minimize latency and energy consumption. In Section 5, we present an MDP formulation and propose a DT-MADDPG algorithm. In Section 6, the proposed approach is validated through experimental data. Finally, Section 7 summarizes the paper.

2. Related work

To meet the evolving demands of future networks, a range of technologies has emerged, enabling MEC systems to achieve higher performance. Among these emerging technologies, DT and artificial intelligence (AI) have found extensive applications in MEC systems.

2.1 Application of DT in MEC

In recent years, DT has garnered increasing attention in various application scenarios such as mobile networks and the Internet of Things (IoT). In industrial settings, Tao et al. [10] provided a comprehensive overview of the current development status and major applications of DT in achieving intelligent manufacturing.

Recently, there has been growing interest in integrating DT with mobile edge networks. The paradigm of DT edge network [13] has emerged, utilizing DT to enhance the efficiency and quality of MEC applications. Lin et al. [14] investigated the potential of applying DT to wireless networks, namely DT. The authors further delineated typical application scenarios of DT, including manufacturing, healthcare, intelligent transportation systems, and smart cities. Tang et al. [15] proposed a DT-Assisted Resource Allocation framework for personalized Industrial Internet of Things (IIoT) services in Industry 4.0, addressing challenges in network slicing and resource allocation through distributed DRL. Sun et al. [16] proposed the use of dynamic DT and federated learning for air-ground network applications. When designing dynamic incentive schemes for federated learning, they also considered changing DT biases and network dynamics.

As part of integrating DT into network virtualization, Shen et al. [12] proposed a new virtualization architecture called holistic network virtualization, which aims to enhance the capability of managing networks and providing services to end-users. Lu et al. [7] designed a novel DT wireless network model, introducing DT into wireless networks to alleviate unreliable and long-distance communication between users and base stations, where user data is synchronized to the base stations to construct corresponding DT. Inspired by the aforementioned works, we constructed a virtual network to capture the real-time dynamics of computing network resources through a DT, thereby addressing the issue of real-time task resource allocation caused by resource fluctuations in the computing network.

2.2 Application of AI in MEC

To enhance resource utilization, AI has emerged as a key enabler for resource allocation in MEC applications. DRL has been widely adopted to optimize resource allocation for efficient edge computing. In order to reduce long-term total latency and energy costs, Zhang et al. [17] investigated the cloud-edge-end cooperative partial task offloading and resource allocation problem in the IIoT and proposed an improved decentralized multiagent DRL algorithm to optimize task offloading and resource allocation decisions. Liu et al. [18] addressed the problem of multi-user computation offloading and wireless caching resource allocation in vehicular edge computing systems, focusing on linearly related requests, and employed a DDPG algorithm to optimize execution latency. Zhao et al. [19] utilized an optimization-driven hierarchical DDPG framework to achieve performance optimization through inner and outer loop learning methods and multiagent extensions. Lin et al. [20] proposed an algorithm that combines DRL and linear programming to address the mixed-integer nonlinear programming problem, optimizing the energy efficiency of unmanned aerial vehicles and the fairness of offloading in MEC scenarios.

Inspired by the considerations mentioned above, we propose a novel approach that utilizes DT to assist the entire process of task offloading, including optimization of task offloading targets and resource allocation. We assume that each user generates only one task at any given moment, and each edge server (ES) has limited computing resources and wireless transmission bandwidth, without restrictions on the number or size of tasks it can receive. During the task offloading process, several additional considerations need to be taken into account. During the task offloading process, several factors merit consideration. Primarily, we should determine whether tasks should be locally computed or offloaded to ESs. Additionally, if CVs opt to offload tasks to ESs the wireless transmission power becomes a consideration. Furthermore, when an ES is tasked with serving multiple CVs, optimizing the allocation of limited wireless transmission bandwidth and CPU frequency becomes paramount. Given the complexity of these issues and the highly dynamic nature of the environment, addressing these challenges inherently presents significant difficulties.

In the following sections, we propose a DT-MADDPG algorithm to tackle the problem of adaptive real-time solutions and intend to undertake long-term systematic performance optimization. As our decision variables involve

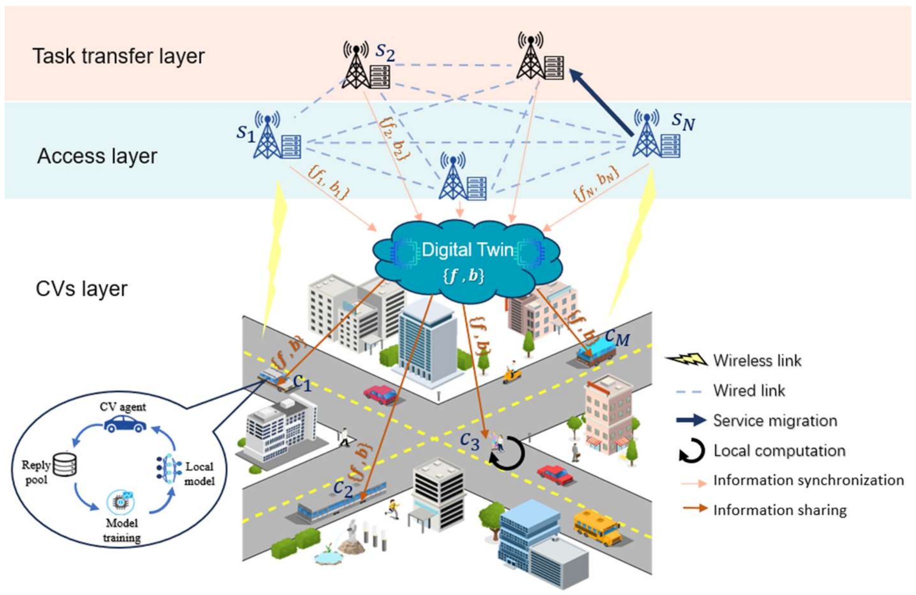


Fig. 1 Architecture of the cloud-edge-vehicle system.

both discrete and continuous variables, although MADDPG can only address continuous variables, we plan to improve the algorithm to accommodate our problem.

3. System Model

In comparison to the existing 5G V2X networks, future 6G V2X networks offer significant advancements in flexibility, resource management, and intelligence. The current 5G V2X networks rely on a combination of centralized cloud computing and relatively static Multi-access Edge Computing platforms, with pre-allocated resources and predefined rules for task offloading. In contrast, 6G V2X networks employ a cloud-edge-vehicle system architecture, with dynamic deployment of edge nodes and integration of lightweight DT technology. This allows real-time monitoring and updating of edge server status, enabling intelligent and adaptive resource management. Additionally, 6G V2X leverages machine learning for complex decision-making, ensuring ultra-low latency, high bandwidth, and efficient resource utilization. These enhancements make 6G V2X networks better suited for supporting complex and diverse V2X applications.[23]

3.1 Network Model

Fig. 1 shows the diagram of the designed cloud-edge-vehicle system architecture, the set of CV and ES is represented by $\mathcal{M} = \{1,....,M\}$ and $\mathcal{N} = \{1,...,N\}$, where $i \in \mathcal{M}$, $j \in \mathcal{N}$. we use c_i and s_j respectively represents the *i*-th CV and *j*-th ES. The designed cloud-edge-vehicle system model comprises the CVs layer, access layer, and task transfer layer:

- *CVs Layer:* This layer consists of CVs, allowing random movement in the horizontal plane. In each time slot, each CV generates only one computationally intensive task, which is considered indivisible as a whole. The tasks are initiated by CVs and can also be computed locally within the CVs layer.
- *Access Layer*: In the access layer, CVs connect wirelessly to nearby ESs. Tasks can be computed by these ESs or accessed by them to enter the MEC network.
- *Task Transfer Layer:* In the task transmission layer, ESs connect to the MEC network, and tasks can be transmitted within the MEC network via wired connections. The target ESs provide computational services to fulfill the computational requests from CVs. To perceive the dynamic distribution of computational capabilities, DT runs in ESs to replicate their real-time states, especially available CPU frequency and available bandwidth.

We define a lightweight DT to reflect the temporal variability of computing resource, and assume that DT runs on the edge cloud. The definition of the computing capability DT for ES s_j at the time slot *t* is as follows:

$$\mathbf{DT_j}(t) = \Theta\left(f_j(t), b_j(t)\right) \tag{1}$$

where $f_j(t)$ represents the available CPU frequency of s_j at the time slot *t*, $b_j(t)$ is the available bandwidth of s_j at the time slot *t*. The connected DT in the ESs form a DT. We define the DT as follows:

$$\boldsymbol{DT} = \{\mathbf{DT_1},...,\mathbf{DT_N}\} \tag{2}$$

The DT is an innovative technological approach designed to facilitate the collection, sharing, and updating of real-time status information to support decision-making processes. Through the DT, CVs can make offloading decisions based on real-time status information, thereby effectively managing resources and optimizing system performance.

Upon CV decision-making, the DT immediately updates to reflect the latest system status. This instantaneous updating mechanism ensures that the next CV can access the updated global status information. This information includes all ESs' available CPU frequency and available bandwidth, enabling decisions that are most suitable for the current state.

In order to ensure that each CV receives the latest status information, at a time slot, the DT updates M times simultaneously. The efficient sharing and updating mechanism of information ensures robust decision support for CVs, contributing to the optimization of system performance.

In terms of cost, DT's lightweight design incorporates only a minimal amount of critical state information, focusing primarily on the CPU frequency and bandwidth of each ES, which results in very limited resource usage. During a time slot, DT's data transmission is less than 1 kbit, whereas the computational task data for CVs typically exceeds 1000 kbit. Therefore, we consider the impact of DT on system storage, computational resources, and bandwidth to be negligible in the cloud-edge-vehicle system architecture.

As for accuracy, our primary objective is to enhance task offloading efficiency through DT. To simplify the analysis and clearly demonstrate the potential of DT in improving system efficiency, we assume that the DT model in the system is completely accurate.

3.2 Task Computation and Transmission Model

We assume M CVs and N ESs in the network. In each time slot t, each CV randomly generates a task, denoted as $D_i(t)$ (in kbit). Tasks can be computed in three ways: locally, nearby ES computation, or distant ES computation.

1) Local Computation

In the local computation mode, tasks are executed locally on the CVs. The available CPU frequency of c_i is denoted as f_i, and the number of CPU cycles required to execute one task is denoted as ζ. Thus, the computation delay of executing task $D_i(t)$ locally can be written as:

$$T_i^{\text{cmp}} = \frac{\zeta \cdot D_i(t)}{f_i(t)} \tag{3}$$

2) Nearby ES Computation

In the nearby computation mode, tasks are offloaded from the CV to the ES at the access layer via wireless communication. Thus, the computation delay of executing task $D_i(t)$ through nearby ES computation can be written as:

$$T_i^{\text{cmp}} = \frac{\zeta \cdot D_i(t)}{f_j(t)} \tag{4}$$

The transmission data rate of wireless communication is determined by available spectrum, interference, and total bandwidth. This paper adopts Orthogonal Frequency Division Multiple Access (OFDMA) [20] as the access mode for CVs. OFDMA divides the system bandwidth into multiple parallel orthogonal sub-channels. Due to the orthogonality between sub-carriers, there is no mutual interference between them. Therefore, OFDMA can achieve interference-free and parallel data transmission between multiple devices. The transmission data rate between c_i and the nearby ES s_j can be expressed as:

$$r_{ij}(t) = b_{ij}(t)\log\left(1+\frac{p_i(t)h_{ij}(t)d_{ij}(t)^{-\alpha}}{\sigma^2}\right) \tag{5}$$

where $b_{ij}(t)$ is the bandwidth allocated to c_i by s_j, $p_i(t)$ is the transmission power of c_i, and $h_{ij}(t)$ is the current channel gain between c_i and s_j at time slot t. $d_{ij}(t)^{-\alpha}$ is calculated based on the positions of c_i and s_j:

$$d_{ij}(t)^{-\alpha} = \| l_i(t) - l_j(t) \|^{-\alpha} \tag{6}$$

where $-\alpha$ is the path loss exponent between the CV and the ES, σ^2 is the background noise power.

The wireless transmission delay can be expressed as:

$$T_{ij}^{\text{com}} = \frac{D_i(t)}{r_{ij}(t)} \tag{7}$$

The task processing delay includes both the wireless transmission delay of the task and the computation delay of the ES, and can be expressed as:

$$T_i = T_j^{cmp} + T_{ij}^{com} \tag{8}$$

3)Remote ES Computation

When the tasks to be executed exceed the maximum computational capacity of nearby ESs under the required latency constraints, the tasks need to be offloaded to distant ESs for collaborative computing. If a task needs to be offloaded from the access layer to the task transfer layer for computation, it is defined as Remote ES Computation. The task is first offloaded via wireless communication to the ES s_j in the access layer, which then transfers it via wired communication to the target ES $s_{j'}$ for computation.

The wired transmission delay of $D_i(t)$ can be expressed as:

$$T_{jj'}^{com} = \frac{D_i(t)}{r_{j'j}(t)} \tag{9}$$

The wired transmission rate between s_j and $s_{j'}$ is represented as:

$$r_{j'j}(t) = \frac{\phi}{d_{j'j}} \tag{10}$$

where ϕ is the distance factor.

The total task processing delay includes the wireless transmission delay of the task, the wired transmission delay, and the computation delay of $s_{j'}$, and can be expressed as:

$$T_i = T_{j'}^{cmp} + T_{ij}^{com} + T_{jj'}^{com} \tag{11}$$

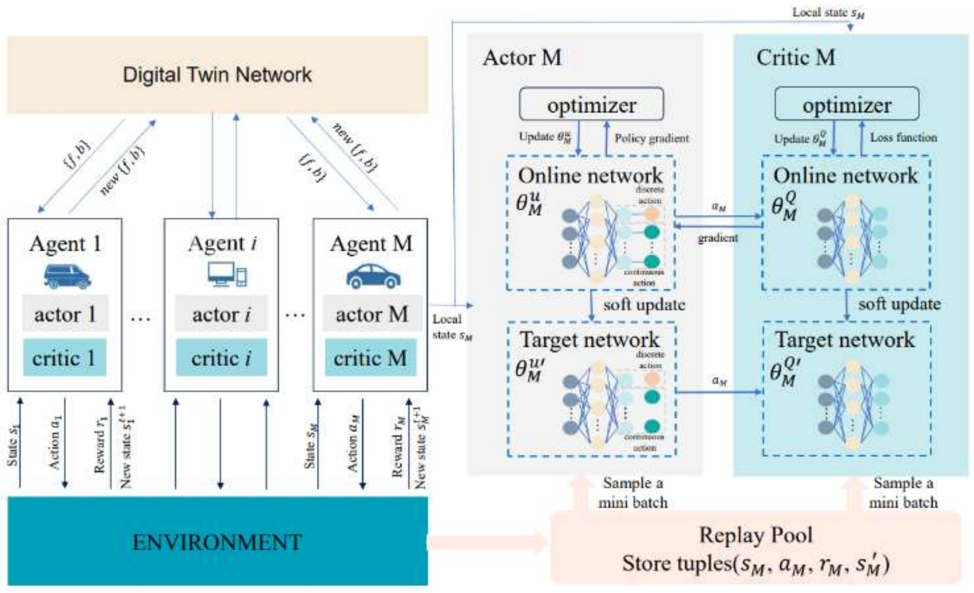


Fig. 2 DT-MADDPG learning framework.

3.2 Energy Consumption Model

Energy consumption consists of task computation energy consumption and task transmission energy consumption. The energy consumption for executing task $D_i(t)$ locally is given by:

$$E_i^{cmp}(t) = D_i(t)\varphi_i \zeta f_i^2 \tag{12}$$

where φ_i is the effective capacitance constant of the CPU chip. Similarly, if computation is performed on an ES, the energy consumption for $D_i(t)$ is:

$$E_j^{cmp}(t) = D_i(t)\varphi_j \zeta f_j^2 \tag{13}$$

During the task transmission phase, energy consumption mainly occurs during uplink task transmission and downlink computation result transmission. Since the size of computation results is much smaller than that of computation tasks, we only consider the energy consumption of uplink task transmission. The energy consumption for c_i to transmit tasks to s_j via wireless communication can be written as:

$$E_{ij}^{\text{com}} = \frac{D_i(t)p_i(t)}{r_{ij}(t)} \tag{14}$$

where $p_i(t)$ is the transmission power of c_i. During local computation, when tasks are computed locally without considering transmission energy consumption, the energy consumption for c_i to process tasks is:

$$E_i(t) = E_i^{cmp}(t) \tag{15}$$

For nearby ES computation, when tasks are executed within the wireless transmission range of ESs, the total energy consumption equals the sum of wireless transmission energy consumption and the energy consumption of the ES for computing tasks:

$$E_i(t) = E_{ij}^{\text{com}}(t) + E_j^{cmp}(t) \tag{16}$$

For remote ES computation, when tasks are executed outside the wireless transmission range of ESs, the energy consumption equals the sum of uplink wireless transmission energy consumption from the CV to the nearest base station, the wired transmission energy consumption from the base station to the target ES, and the energy consumption of the target ES for computing tasks. Here, since the wired transmission energy consumption from the base station to the target ES is negligible, the total energy consumption for task offloading is:

$$E_i(t) = E_{ij}^{\text{com}}(t) + E_{j'}^{cmp}(t) \tag{17}$$

4. Problem Formulation in the Model

In this section, we propose a new task offloading problem in the cloud-edge-vehicle system, aiming to improve energy efficiency and latency performance in task execution by optimizing task offloading strategies. We assume the system has M CVs and N ESs, the fundamental problem in the cloud-edge-vehicle system is to allocate offloading targets and resources for each task.

As mentioned earlier, tasks can be computed locally, on nearby ES, or on remote ESs. Introducing a weighting factor w_i constructs the objective function, adjusting the weights of energy consumption and latency according to the task preferences of CV. The total cost over a procedure cycle consists of execution time cost and energy cost, represented as follows:

$$\psi = \sum_{t=1}^{T}\sum_{i=1}^{N}\left(w_i T_i + (1-w_i)E_i\right) \tag{18}$$

where *t* is the current time slot in the iterative calculation process, The task offloading problem can be formulated as follows:

$$\min_{\mathbf{x},\mathbf{p},\mathbf{f},\mathbf{b}} \psi \tag{19}$$

$$C1: x_i \in \{0,1,2,...,j,\dots,N\}, \forall i \in \mathcal{M}$$

$$C2: \sum_{i\in\mathcal{M}} b_{ij} \le B_j^{max}, \forall j \in \mathcal{N}$$

$$C3: \sum_{i\in\mathcal{M}} f_{ij} \le F_j^{max}, \forall j \in \mathcal{N}$$

$$C4: 0 \le p_i \le p_{max}, \forall i \in \mathcal{M}$$

Constraint 1 represents the association policy between tasks from CV and target devices, where $x_{ij} = 0$ denotes local computation and $x_{ij} = j$ denotes computation on s_j. Constraints 2 and 3 ensure that the allocated computing resources for CVs do not exceed the maximum bandwidth and CPU frequency of ESs. Constraint 4 ensures that the CV transmission power is less than the maximum transmission power and non-negative.

This problem is a combinatorial problem with highly complex coupled variables. Due to the challenges of task offloading in the cloud-edge-vehicle system, solving the task offloading problem through existing optimization algorithms is highly challenging and cannot effectively handle dynamic system conditions. We propose the use of MADRL to find the optimal solution to the problem.

5. DT-MADDPG Task Offloading Algorithm

5.1 DT Empowered Multiagent Network Model

Since task offloading is a sequential decision-making process, we model the task offloading problem as a Markov Decision Process (MDP). Therefore, the task offloading problem can be solved using DRL based methods.

In the cloud-edge-vehicle system model, each CV is treated as a DRL agent. These agents are responsible for issuing task requests and selecting task offloading strategies based on task requirements and system environment. In each time slot, the agents sequentially observe the state environment according to ID order. To ensure that the agents make optimal selections and maintain the coupling relationship of ESs resources, combined with the aforementioned DT, the observation of the agents is defined as the current local environmental state and the global DT information.

Agent *i* takes action a_i in its local environment, and all agents take joint actions $a(t) = \{a_1(t), a_2(t), \dots, a_m(t)\}$. The immediate reward for the agents is obtained based on the joint action $a(t)$. The system state $s(t)$ evolves to the next state $s(t+1)$ under the action $a(t)$ with transition probability $P(s' \mid s, a)$. Then, agent *i* observes the new state from its current local environmental state and the DT. The state space, action space, and reward function of the established MDP model are as follows:

- *State Space:* The state of cloud-edge-vehicle system consists of the task size $D_i(t)$, the distance between agent and server $\mathrm{d_i}(t) = \{d_{i1}(t), d_{i2}(t), \dots, d_{iN}(t)\}$, the channel gain of the link between agent and server $\boldsymbol{h_i}(t) = \{h_{i1}(t), h_{i2}(t), \dots, h_{iN}(t)\}$, the available computing capabilities of all servers $\boldsymbol{f}(t) = \{f_1(t), f_2(t), \dots, f_N(t)\}$,and the available bandwidth of all servers $\mathbf{b}(t) = \{b_1(t), b_2(t), \dots, b_N(t)\}$. In this context, $\boldsymbol{f}(t)$ and $\boldsymbol{b}(t)$ are obtained from the DT. The local state observed by agent i at time slot t is represented as follows:
$$\boldsymbol{s_i}(t) = \{D_i(t), \boldsymbol{d}_i(t), \boldsymbol{h}_i(t), \boldsymbol{f}(t), \boldsymbol{b}(t)\} \tag{20}$$
The system state at time slot t is represented as follows:
$$\boldsymbol{s}(t) = \{\mathbf{s_1}(t), \mathbf{s_2}(t), \dots, \mathbf{s_M}(t)\} \tag{21}$$
- *Action Space:* At time slot *t,* each agent *i* executes its action based on its observation $s_i(t)$ and its action policy $a_i(t)$. The action determines the task offloading strategy $x_i(t)$, determining the target devices for task computation, transmission power $p_i(t)$, the allocation of CPU frequency $f_i(t)$, and channel allocation strategy $b_i(t)$. The action set can be formulated as follows:
$$a_i(t) = \{x_i(t), p_i(t), f_i(t), b_i(t)\} \tag{22}$$
- *Reward Function:* Since we consider a coordinated learning scheme for agents, the reward function aims to improve the overall system performance. The reward function for the agents is defined as follows:
$$\mathcal{R}_i(t) = -(w_i T_i + (1-w_i)E_i) \tag{23}$$
During the multi-agent collaboration process, the overall reward function is designed based on global optimization objectives, defined as follows:
$$R(t) = \sum_{i=1}^{M} \lambda_i \mathcal{R}_i(t) \tag{24}$$
where λ_i represents the weighting factor in the cumulative term.

5.2 DT-MADDPG learning framework

We employ a MADRL model for task offloading decisions in computational capability networks. We propose a DT-MADDPG learning framework, as shown in Fig. 2. In the proposed framework, each agent interacts with its local environment to obtain state observations and takes actions for task offloading and resource allocation. During this process, agents need to exchange their local information to obtain global state and action observations for training Actor and Critic networks. However, this exchange significantly increases transmission overhead and latency performance. To deal with these issues, we propose using DT to update and acquire the global state of agents. DT perfectly reflect the global system state at each moment, reducing the interaction cost among agents during the training process.

In each time step loop, based on the observed local state $s_i(t)$, agent i takes action $a_i(t)$, and the DT updates synchronously according to $a_i(t)$ .According to agent's ID, the next agent i' in sequence, generates actions $a_{i'}(t)$ based on the updated local system state $s_{i'}(t)$. After M iterations, based on the actions $a_t(t)=\{a_1(t),a_2(t),\ldots,a_m(t)\}$, the system state estimates rewards $R(t)$ and get next state $s(t+1)$.

In this framework, our output actions are divided into two categories: discrete actions and continuous actions. Innovatively, in the actor network, we handle the two types of actions output by the neural network separately:

1) *Continuous-to-discrete action processing:* The neural network's output actions represent task transition decisions, determining the target computation node for executing these tasks, where $x_i(t)=0$ indicates local execution and $x_i(t)=j$ indicates transmission to the j-th ES for execution, including 1+n discrete variables. After the neural network output is processed by the tanh activation function, the data is transformed into a continuous function $x\in(-1,\ 1)$. Continuous actions are then projected onto the interval (0, n) through a linear transformation (LT) and subsequently rounded down (floor), resulting in {0, 1, ..., n} discrete actions. The specific process is illustrated in the Fig. 3.

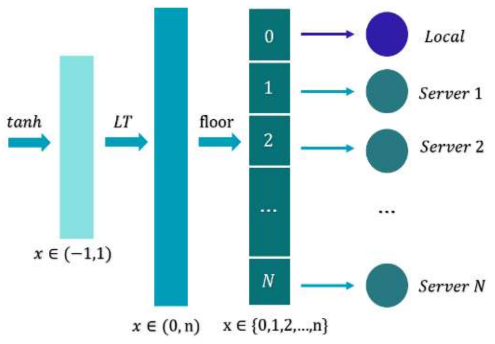


Fig.3. Continuous-to-discrete action processing

2) *Continuous Action Normalization:* To ensure the coupling relationships between the output actions of different agents, a tanh activation layer is added to the output actions of all continuous variables, including upline power, bandwidth, and CPU frequency. This is followed by linear transformation (LT) to constrain the action variables between 0 and 1. Let $a_i=\{p_i(t),f_i(t),b_i(t)\}$ represent the actor network's output action, bounded between 0 and 1, $a_{max}=\{p_{max}(t),f_{max}(t),b_{max}(t)\}$ represent the maximum attainable value, and $\widetilde{a_i}$ represent the actual value. Then, the reward calculation can be recovered from the following equation:

$$\widetilde{a_i}=a_i*a_{max} \tag{25}$$

Because at the same time step t, if agent i offloads to server j, after the agent i executes the action, the remaining transmission bandwidth and computation resources on server j will change accordingly, allowing the maximum transmission bandwidth $b_{\max}(t)$ and maximum computation resources $f_{\max}(t)$ to be offloaded by the next agent i'. The DT synchronizes the state, updating to s'. Then, after the next agent i' receives observation from the DT, it can recover from the following equation:

$$\widetilde{a_{i'}}=a_{i'}*a_{max}' \tag{26}$$

where $\widetilde{a_{i'}}$ represent the actual value of next agent i', $a_{i'}$ represent the actor network's output action of next agent i', a_{max}' represent the maximum attainable value of next agent i'.

Algorithm 1 DT-MADDPG Learning Algorithm

```
1   Initialize critic network with weights $\theta_i^Q$ and actor
    network with weights $\theta_i^\mu$; Initialize replay pool.
2   for episode = 1 to N do
3     Initialize the local network environment to state;
4     for each time slot t = 1, 2,... do
5       for each agent i= 1, 2,...,m do
6         Observe current system state $s_t$ from DT;
7         Select and take action $a_i$ towards $s_t$, which
          is generated by actor network;
8         Obtain reward $r_i$ according to Eq.22
9         Update the new state to the DT;
10        Get a new state observation $s_{t+1}$;
11        Store $(s_i,a_i,r_i,s_{i+1})$ to replay pool;
12      end for
13          for each agent i= 1, 2,...,m do
14        Take a batch of samples from the replay
15        Update the Critic network $\theta_i^Q$;
16        Update the Actor network $\theta_i^\mu$;
17        Update Actor target network $\theta_i^{\mu'}$ and Critic
          target network $\theta_i^{Q'}$;
18      end for
19    end for
20  end for
```

We adopt a MADRL approach based on DT-MADDPG to find the optimal solution to the constructed problem. Each CV is modeled as a DDPG agent, which consists of an actor network and a critic network, both of which are deep neural networks. The neural network adopts a Multi-Layer Perceptron structure, consisting of three fully connected layers, with each hidden layer having a dimension of 64. The input layer first receives input data and undergoes batch normalization for normalization. Following are two hidden layers, each utilizing Rectified Linear Units as the activation function to introduce non-linearity. Finally, the output layer produces network output, processed through the hyperbolic tangent function (tanh) to accommodate both discrete and continuous action spaces. Specifically, the Actor network takes the local state observed by agent *i* as input and outputs its selected action. On the other hand, the Critic network takes the global state and actions as input and outputs an estimate of the current state. For agent *i*, the parameters of its Actor and Critic networks are denoted as θ_i^{μ} and θ_i^{Q}.

During the training process, the parameters of the Actor network are updated using policy gradient methods, denoted as:

$$\theta_i^{\mu} \leftarrow \theta_i^{\mu} + \alpha_{\pi} \nabla_{\theta_i^{\mu}} \log \pi\left(s_t, a_t^m\right) Q\left(s, a_1, a_2, \ldots, a_m, w\right) \tag{27}$$

where α_{π} is the learning rate of the actor network, and $Q\left(s, a_1, a_2, \ldots, a_m, w\right)$ is the action-value function. Similarly, the update for the Critic network is as follows:

$$\theta_i^{Q} \leftarrow \theta_i^{Q} - \alpha_{Q_i} \nabla_{\theta_i^{Q}} \log \pi\left(s_t, a_t^m\right) Q\left(s, a_1, a_2, \ldots, a_m\right) \tag{28}$$

In MADRL, the system state $s(t)=\{s_1(t), s_2(t), \ldots, s_m(t)\}$ is the shared global state among all agents, and the joint action $a(t)=\{a_1(t), a_2(t), \ldots, a_m(t)\}$ consists of actions from each agent. The reward $r(t)=\{r_1(t), r_2(t), \ldots, r_m(t)\}$ for each agent is calculated accordingly. The actor and critic networks are updated by sampling a small batch of data from the replay memory at each time slot.

On the other hand, the parameters of the target networks are slowly updated by the main network every cycle. The parameters of the Actor target and Critic target networks, $\theta_i^{\mu'}$ and $\theta_i^{Q'}$, respectively, are updated as follows:

$$\begin{aligned} \theta_i^{\mu'} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta_i^{\mu'} \\ \theta_i^{Q'} &\leftarrow \tau\theta^{Q} + (1-\tau)\theta_i^{Q'} \end{aligned} \tag{29}$$

where τ is the temperature factor, which is utilized to regulate the degree of softness or hardness in this update. A larger value of τ leads to a greater extent of parameter synchronization."

The overall processes of the DT-MADDPG learning algorithm for task transfer are shown in Algorithm 1.

6. Numerical Results

In this section, the effectiveness of the DT-MADDPG task transfer algorithm is evaluated. To closely mimic real-world conditions, we carefully selected parameters based on existing studies and practical considerations. The experimental scene is defined within a 1000m by 1000m coordinate range, where the positions of the CVs are randomly selected. The number of servers is set to 3, the number of vehicles is varied at {10, 20, 40, 60}, and vehicle speeds are set to {5, 10, 15, 20, 30} m/s. The task size per time slot of CVs is set to {20, 50, 100, 150, 200} kbit [24].

For communication channel characteristics, we referred to [25], which reflects common signal propagation conditions. Specifically, the transmission bandwidth is set to 20 MHz, the maximum transmission power is set to 0.2 W, the noise power is set to -113 dBm, and the wireless communication radius of ES is set to 500 meters. For computational resources, referring to literature [7], we set the CPU frequency of CVs to 2 GHz and the CPU frequency of ESs to [5, 25] GHz. The network bandwidth is set to 20 MHz, and the transmission power is set to 0.2 W.

The transmission channel bandwidth is set to 20MHz, and the maximum transmission power of CV is set to 0.2W. The noise power is set to -113 dBm/Hz. The number of CVs in the simulation is set to 10, 20, 40, and 60, respectively. In the MADRL algorithm, the number of training iterations is set to 50, the discount factor γ is set to 0.7, and the learning rate is set to 0.001. Detailed parameter settings are shown in Table 1.

Table 1 Simulation Parameters

	Meaning	value
M	Number of CVs	{10，20，40，60}
N	Number of ESs	3
D_{ui}	Task size per time slot of CVs	{20,50,100,150,200}kbit
R	Wireless communication radius	500m
σ	White noise power	-113dBm
f_i	CPU frequency of CVs	2GHz
f_j	CPU frequency of ESs	{5,10,15,20,25}GHz
φ	Computation complexity of the	500cycle/bit
B_{ij}	bandwidth	20MHz
P_{ij}^{max}	transmission power	0.2W
V_{ui}	velocity of CVs	{5,10,15,20,30}m/s

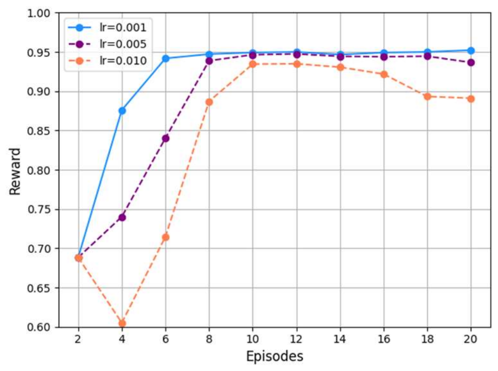


Fig.4. Convergence of different learning rates

To analyze the training efficiency differences of the DT-MADDPG algorithm under different training parameter settings, we selected learning rates of 0.001, 0.005, and 0.01 for comparison in Figure 4, observing their impact on algorithm performance. As shown in Figure 4, the reward curves exhibit an overall upward trend, demonstrating the effectiveness of the algorithm. Additionally, lower learning rates lead to faster convergence of the CV's reward value. Specifically, we observed that when the learning rate was

0.001, the algorithm converged at the 8th iteration and achieved the highest final convergence reward of 0.9501 within the same training epochs. This indicates that smaller learning rates facilitate the algorithm to converge more stably to higher reward levels during training. When the learning rate increased to 0.005, the convergence speed slowed down, with the algorithm converging at the 10th iteration, and the final convergence reward slightly decreased to 0.9447. Further increasing the learning rate to 0.01 resulted in the reward initially decreasing and then increasing, converging around the 18th iteration, with a final convergence reward of 0.8893. This suggests that larger learning rates introduce greater instability during the update process, making the algorithm more susceptible to the influence of local extreme values and challenging to achieve higher final convergence rewards. These results indicate that lower learning rates enable the DT-MADDPG algorithm to converge stably while reaching relatively optimal convergence levels.

To demonstrate the superiority of the proposed algorithm, we also compare its performance with four benchmark strategy:

1) All MEC Computing Strategy (AMCS): In this strategy, all tasks are offloaded to the MEC server for execution. To ensure experimental fairness, agents also utilize information collected and shared by the DT. The choice of AMCS as a benchmark strategy aims to evaluate the performance of offloading all tasks to the ES, providing insight into the system's performance under ideal conditions and serving as a reference for comparing other strategies.

2) Random Offloading Computing Strategy (ROCS): In this strategy, tasks are randomly executed either at the MEC server or locally at the CV. ROCS is chosen to evaluate the system's performance under conditions of randomness, providing a baseline to measure the effectiveness of other optimization strategies.

3) All Local Computing Strategy (ALCS): In this strategy, all tasks are computed locally on the CV without any offloading. The offloading cost consists of the local computing delay and the energy consumption incurred by the CPU. This strategy reflects the extreme case of relying entirely on local computation. By comparing with other offloading strategies, ALCS provides a reference to evaluate the advantages of task offloading in terms of performance.

4)DDPG[22]: DDPG is a widely used deep reinforcement learning algorithm, suitable for solving task offloading and resource allocation problems in edge computing environments. Selecting DDPG as a benchmark aligns with common practices in the literature[26]. To ensure fairness in information access, we test the DDPG algorithm within our designed cloud-edge-vehicle system framework, with the DT collecting and sharing ES resource information with the agent. The agent updates its actor and critic networks based on observations (including DT information and CVs' states) and actions to maximize its expected return. By comparing the performance of the DT-MADDPG algorithm, we highlight the advantages of multi-agent learning in task offloading and resource allocation, demonstrating the potential application value of multi-agent systems in complex edge computing environments.

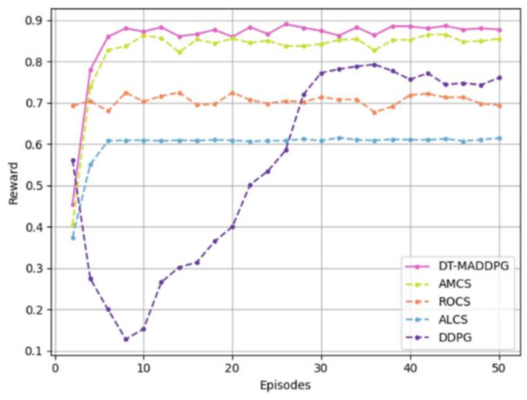


Fig.5. Reward vs. episodes

In the model training process depicted in Figure 5, we evaluated the convergence of the proposed task offloading-based strategy on DT-MADDPG. The simulation settings included 50 training episodes, each comprising 100 steps. In Figure 5, the x-axis represents the number of training iterations, while the y-axis represents the cumulative reward during the training process, i.e., the total cost incurred by each agent in terms of system delay and energy consumption. It can be observed that when the training set is less than 10, the agents fail to effectively respond to the environment. The agents accumulate training data over the initial 10 episodes and then continuously learn and improve their neural networks, thereby enhancing the model's accuracy. After 10 iterations, the cumulative reward begins to stabilize. Consequently, the training of the DT-MADDPG model gradually converges, indicating that the model has completed training and demonstrating its effectiveness. After convergence, the average system rewards obtained by DT-MADDPG, AMCS, DDPG, ROCS, and ALCS decrease sequentially. Additionally, ALCS exhibits more stable rewards post-training, attributed to minimal influence from CV mobility during local computations, resulting in smaller state changes and stable rewards post-convergence. The convergence behavior of the DDPG algorithm, characterized by initial reward reduction followed by subsequent increase, may be attributed to the exploratory nature of agents in the early stages of training. Agents typically engage in more exploration, attempting various actions to understand the environment. This may lead to greater reward volatility as agents have yet to discover the optimal strategy. As training progresses, agents gradually shift towards exploiting learned knowledge.

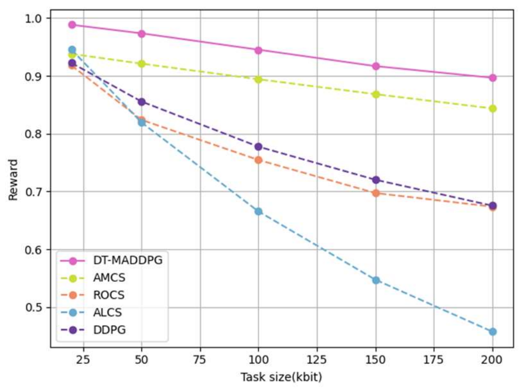


Fig.6. Reward vs. Task Size

In Figure 6, we compare the impact of different input data sizes of tasks on the average system reward of the agents. As shown, the average system reward of all offloading strategies decreases with increasing task input data size. This is because larger task input data sizes lead to greater computation latency, transmission latency, and energy consumption. From the comparison in this figure, it is evident that the average system reward of the task offloading strategy based on DT-MADDPG outperforms other offloading strategies. When the task input data size is small, the agents can execute tasks locally, which is less costly than offloading tasks to ESs. However, when the task input data size is large, most tasks are offloaded to ESs for execution, as the abundant computing resources of ESs can handle these computationally intensive tasks. As the task input data size increases from 100 kbit to 150 kbit, the average system reward of the proposed DT-MADDPG strategy increases by 3.72%, while the average system costs of the AMCS, ROCS, DDPG and ALCS strategies increase by at least 9.48%, 19.14%, 20.26% and 23.60%, respectively. This indicates that the average system rewards of all offloading strategies increase as the network load increases, while the average system costs of DT-MADDPG remain at a lower level.

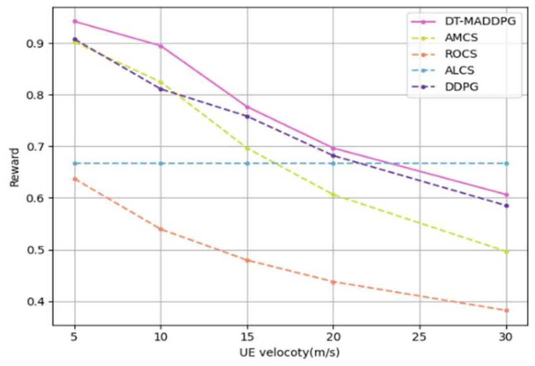
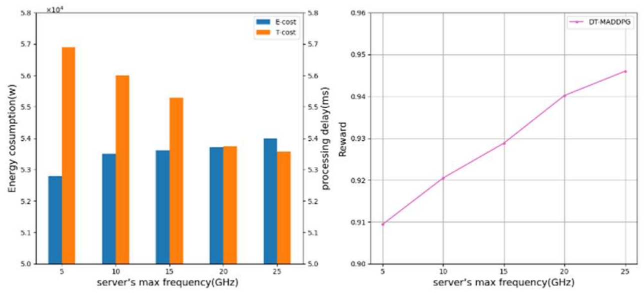


Fig 7. Reward vs. CV velocity

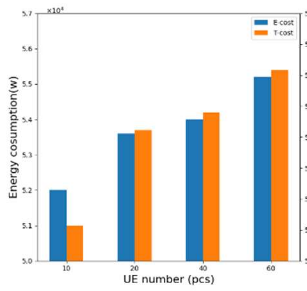
In Figure 7, we compare the impact of different mobility velocity of CV on the average system reward. As depicted, the average system reward of all offloading strategies decreases with increasing task input data size. When the mobility speed is slow, such as 5 m/s and 10 m/s, the average system rewards of various offloading strategies exhibit relatively stable performance. However, as the mobility speed increases to 15, 20, and 30 m/s, significant differences in average system rewards. Specifically, with increasing mobility speed, the average system reward of the DT-MADDPG strategy remains relatively stable at a higher level, indicating its effectiveness in improving the average system reward under different mobility velocity. In contrast, the performance of the AMCS, ROCS and DDPG strategies deteriorates significantly at higher mobility velocity, with a noticeable decrease in average system reward. This may be attributed to these strategies' inability to effectively coordinate task allocation and execution in high-speed mobility environments, resulting in performance degradation. The ALCS strategy, which executes tasks locally and is unaffected by mobility, maintains a stable average system reward regardless of speed changes. As shown in the figure, when the CV speed is between 20 and 25 m/s, the reward obtained by ALCS will be greater than the other four strategies. Therefore, it can be concluded that our strategy is more suitable for use when the CV speed is below 20 m/s.



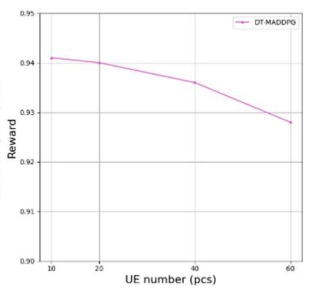
(a)E-cost and T-cost (b) Reward

Fig.8. The impact of varying ES's frequency on (a) E-cost and T-cost, (b) Reward

In Figure 8(a), the blue bars represent the average energy consumption of the agents, denoted as E-cost, while the pink bars denote the average time delay of the agents, denoted as T-cost. In Figure 8(b), the curve illustrates the variation of rewards of the DT-MADDPG with changes in the maximum CPU frequency of the ESs, while keeping other conditions constant. With the increase in the computing capability of ES, the latency of task offloading significantly decreases, while energy consumption slightly increases. This is because the enhanced computing capability of ES accelerates the computation speed of tasks offloaded to the ESs. However, computational energy consumption is directly proportional to the computation frequency. As the computation frequency increases, the energy consumption also increases proportionally. Rewards are obtained by weighting energy consumption and latency. From the aforementioned figures, it can be observed that as the maximum computation frequency of ESs increases, the reward also increases. Therefore, this proves that the algorithm is adaptable to changes in CPU frequency.



(a)E-cost and T-cost



(b) Reward

Fig.9. The impact of varying CV numbers on (a) E-cost and T-cost, (b) Reward

In Figures 9, comparing the average system costs of the four offloading strategies for different numbers of CVs, it's evident that with an increase in CVs quantity, the average system rewards for each offloading strategy decrease. As the number of CVs increase, the system's average reward decline, while both latency and energy consumption noticeably increase. This is because when a large number of tasks are offloaded to the same ES, the allocated computational resources for each CV decrease, leading to increased computational time delays and energy consumption. Consequently, the system's average rewards relatively decrease. Therefore, this proves that the algorithm is adaptable to changes in CVs numbers.

7. Conclusion

In this study, we investigated the joint optimization problem of task offloading and resource allocation in future 6G networks. To solve this problem, we built the cloud-edge-vehicle system model and proposed a task offloading strategy based on DT-empowered DRL. This algorithm minimized the total system cost in terms of system latency and energy consumption. Through simulation experiments, we tested that the proposed strategy achieved better reward performance under different sizes of task input data, CVs' mobility velocity, ES computing capacities, and CVs' quantities. Additionally, the algorithm showed good adaptability in handling latency and energy consumption issues for varying ES computing capacities and CVs' quantities.

In future research, we will consider strategies of task partitioning, and allow tasks from an intelligent agent at a certain moment to be distributed across different ESs for computation to improve computational speed. Furthermore, we will investigate accurate energy consumption by DT's synchronization information for better performance. Moreover, since the assumption that DT information is completely accurate may not hold true in the actual scene, future work should explore the performance of the proposed method under different accuracy levels of DT information to better reflect the real world situation.

Acknowledgments

This work was supported by Science and Technology development program for Innovation-driven Development Pilot Zone of Silk Road Economic Belt and Wuchangshi National Independent Innovation Demonstration Zone under Grant 2023LQ02004.

References

[1] M Noor-A-Rahim, Z Liu, H Lee, MO Khyam, J He, D Pesch, K Moessner, W Saad and HV Poor. "6G for Vehicle-to-Everything (V2X) Communications: Enabling Technologies, Challenges, and Opportunities," in Proceedings of the IEEE, vol. 110, no. 6, pp. 712-734, June 2022.

[2] S. B. Prathiba, G. Raja, S. Anbalagan, K. Dev, S. Gurumoorthy and A. P. Sankaran, "Federated Learning Empowered Computation Offloading and Resource Management in 6G-V2X," in IEEE Transactions on Network Science and Engineering, vol. 9, no. 5, pp. 3234-3243, 1 Sept.-Oct. 2022

[3] X. Zhu, F. Ma, F. Ding, Z. Guo, J. Yang and K. Yu, "A Low-Latency Edge Computation Offloading Scheme for Trust Evaluation in Finance-Level Artificial Intelligence of Things," in IEEE Internet of Things Journal, vol. 11, no. 1, pp. 114-124, 1 Jan.1, 2024.

[4] J. Lin, S. Huang, H. Zhang, X. Yang and P. Zhao, "A Deep-Reinforcement-Learning-Based Computation Offloading With Mobile Vehicles in Vehicular Edge Computing," in IEEE Internet of Things Journal, vol. 10, no. 17, pp. 15501-15514, 1 Sept.1, 2023.

[5] J. Du, Z. Kong, A. Sun, J. Kang, D. Niyato, X. Chu, FR. Yu. "MADDPG-Based Joint Service Placement and Task Offloading in MEC Empowered Air–Ground Integrated Networks," in IEEE Internet of Things Journal, vol. 11, no. 6, pp. 10600-10615, 15 March15, 2024.

[6] Q. Li, Y. Cui, T. Song and L. Zheng, "Federated Multiagent Actor–Critic Learning Task Offloading in Intelligent Logistics," in IEEE Internet of Things Journal, vol. 10, no. 13, pp. 11696-11707, 1 July1, 2023.

[7] Y. Lu, B. Ai, Z. Zhong and Y. Zhang, "Energy-Efficient Task Transfer in Wireless Computing Power Networks," in IEEE Internet of Things Journal, vol. 10, no. 11, pp. 9353-9365, 1 June1, 2023.

[8] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Blockchain empowered asynchronous federated learning for secure data sharing in Internet of Vehicles" IEEE Trans. Veh. Technol., vol. 69, no. 4, pp. 4298–4311, Apr. 2020.

[9] Y. Dai and Y. Zhang, "Adaptive DT for Vehicular Edge Computing and Networks," in Journal of Communications and Information Networks, vol. 7, no. 1, pp. 48-59, March 2022.

[10]T. H.-J. Uhlemann, C. Lehmann, and R. Steinhilper, "The DT: Realizing the cyber-physical production system for industry 4.0," Procedia Cirp, vol. 61, pp. 335–340, Dec. 2017.

[11]Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Low-latency federated learning and blockchain for edge association in DT empowered 6G networks," IEEE Trans. Ind. Informat., vol. 17, no. 7, pp. 5098–5107, Jul. 2021.

[12]X. Shen, J. Gao, W. Wu, M. Li, C. Zhou, and W. Zhuang, "Holistic network virtualization and pervasive network intelligence for 6G," IEEE Commun. Surveys Tuts., vol. 24, no. 1, pp. 1–30, 1st Quart., 2022.

[13]Y. Zhao, L. Li, Y. Liu, Y. Fan, K. Lin. "Communication-efficient federated learning for DT systems of industrial Internet of Things." IFAC-PapersOnLine. Vol.55, no. 2, pp. 433-438, Apr.2022.

[14]X. Lin, J. Wu, J. Li, W. Yang and M. Guizani, "Stochastic Digital-Twin Service Demand With Edge Response: An Incentive-Based Congestion Control Approach," in IEEE Transactions on Mobile Computing, vol. 22, no. 4, pp. 2402-2416, Apr.2023.

[15]L. Tang, Y. Du, Q. Liu, J. Li, S. Li and Q. Chen, "Digital-Twin-Assisted Resource Allocation for Network Slicing in Industry 4.0 and Beyond Using Distributed Deep Reinforcement Learning," in IEEE Internet of Things Journal, vol. 10, no. 19, pp. 16989-17006, 1 Oct.1, 2023

[16]W. Sun, N. Xu, L. Wang, H. Zhang, and Y. Zhang, "Dynamic DT and federated learning with incentives for air-ground networks," IEEE Trans. Netw. Sci. Eng., vol. 9, no. 1, pp. 321–333, Jan./Feb. 2022.

[17]F. Zhang, G. Han, L. Liu, Y. Zhang, Y. Peng and C. Li, "Cooperative Partial Task Offloading and Resource Allocation for IIoT Based on Decentralized Multiagent Deep Reinforcement Learning," in IEEE Internet of Things Journal, vol. 11, no. 3, pp. 5526-5544, 1 Feb.1, 2024

[18]L. Liu and Z. Chen, "Joint Optimization of Multiuser Computation Offloading and Wireless-Caching Resource Allocation With Linearly Related Requests in Vehicular Edge Computing System," in IEEE Internet of Things Journal, vol. 11, no. 1, pp. 1534-1547, 1 Jan.1, 2024.

[19] S. Zhao, Y. Liu, S. Gong, B. Gu, R. Fan and B. Lyu, "Computation Offloading and Beamforming Optimization for Energy Minimization in Wireless-Powered IRS-Assisted MEC," in IEEE Internet of Things Journal, vol. 10, no. 22, pp. 19466-19478, 15 Nov.15, 2023.

[20]N. Lin, H. Tang, L. Zhao, S. Wan, A. Hawbani and M. Guizani, "A PDDQNLP Algorithm for Energy Efficient Computation Offloading in UAV-Assisted MEC," in IEEE Transactions on Wireless Communications, vol. 22, no. 12, pp. 8876-8890, Dec. 2023

[21]S. Wan, J. Lu, P. Fan, Y. Shao, C. Peng, and K. B. Letaief, "Convergence analysis and system design for federated learning over wireless networks," IEEE J. Sel. Areas Commun., vol. 39, no. 12, pp. 3622–3639, Dec. 2021.

[22]J. Ren and S. Xu, "DDPG Based Computation Offloading and Resource Allocation for MEC Systems with Energy Harvesting," 2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring), Helsinki, Finland, pp. 1-5, Apr.2021

[23]A.Dogra, R.K.Jha, and S.Jain. "A survey on beyond 5G network with the advent of 6G: Architecture and emerging technologies." IEEE access 9, vol. 9, pp. 67512-67547, 2020

[24]S. R. Jeremiah, L. T. Yang, J. H. Park, "Digital twin-assisted resource allocation framework based on edge collaboration for vehicular edge computing," Future Generation Computer Systems, vol. 150, pp. 243-254, Jan 2024.

[25]Y. S. Nasir and D. Guo, "Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks," IEEE J. Sel. Areas Commun., vol. 37, no. 10, pp. 2239–2250, Oct. 2019.

[26]M.Khani, MM.Sadr, S.Jamali. "Deep reinforcement learning‐based resource allocation in multi‐access edge computing." Concurrency and Computation: Practice and Experience, vol. 36, no. 15, pp. e7995, Jul 10. 2024.

Jiakun LI received the Bachelor's degree in Mechanical Engineering from Qingdao University of Technology in 2022 and is currently pursuing a Master's degree at Dalian University of Technology. Her research interests include edge computing and reinforcement learning.

Jiajian LI received the B.S. degree in mechanical engineering from Dalian University of Technology, Dalian, China, in 2020.
He continues to pursue the Ph.D degree in Dalian University of Technology, Dalian, China. His research interests include Internet of Vehicles, intersection coordination, and intelligent traffic.

Yanjun SHI received the B.S. degree in mechanical engineering from Dalian Ocean University, China, in 1996, the M.S. degree in mechatronic engineering from Beihang University, China, in 1999, and the Ph.D. degree in computer engineering from Dalian University of Technology, China, in 2005.

He is currently an Professor with the School of Mechanical Engineering, Dalian University of Technology, Vice Chair of IEEE SMC Dalian Chapter. He is an Associate Editor of the IET Collaborative Intelligent Manufacturing, and has published over 80 papers in scientific journals and international conferences. His research interests include collaborative planning and scheduling, multi-access edge computing, 5G applications, autonomous vehicles.

Hui LIAN received the B.S. degree from Harbin University of Science and Technology, Heilongjiang, China, in 2005.
He currently works at TBEA Co., Ltd., serving as the Deputy Director of the Cable Research Institute.

Haifan WU received the B.S. degree in mechanical engineering from Dalian University of Technology, China, in 1990, the M.S. degree in mechanical engineering from Dalian University of Technology, in 1996, and the Ph.D. degree in Control systems from Tokyo Institute of Technology, Japan, in 2006.
He is currently an associate professor with the school of Mechanical Engineering, Dalian University of Technology. His research fields are Rehabilitation Robot and Warehouse Automation.