Journal of Network and Computer Applications 229 (2024) 103941

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Federated deep reinforcement learning for task offloading and resource allocation in mobile edge computing-assisted vehicular networks

Xu Zhao [a,b,*], Yichuan Wu [c], Tianhao Zhao [c], Feiyu Wang [c], Maozhen Li [d]

[a] School of Electronic and Information, Xi'an Polytechnic University, Xi'an, 710048, China
[b] Shaanxi Bianyun Collaborative Network Technology Co., Ltd, Xi'an, 712000, China
[c] School of Computer Science, Xi'an Polytechnic University, Xi'an, 710048, China
[d] Department of Electronic and Electrical Engineering, Brunel University London, Uxbridge, UB8 3PH, UK

ARTICLE INFO

Keywords:
Internet of vehicles
Mobile edge computing
Deep reinforcement learning
Federated learning
Task offloading
Resource allocation

ABSTRACT

Mobile edge computing (MEC) enables computation intensive applications in the Internet of Vehicles (IoV) to no longer be limited by device resources. However, the lack of an effective task scheduling strategy will seriously affect users' quality of experience (QoE). In this paper, a task type-based task offloading and resource allocation strategy is proposed to reduce delay and energy consumption during task execution. First, we establish communication, computing, and system cost models based on task offloading schemes, and model the joint optimization problem of task offloading and resource allocation as a Markov decision process. The utility function is obtained based on the task completion rate and the system cost. Second, an algorithm framework based on multi-agent deep deterministic policy gradient (MADDPG) is designed to solve the difficulty that traditional single-agent reinforcement learning algorithms are difficult to converge in a dynamic environment. In distributed scenarios, the proposed framework can also reduce system costs while handling more tasks. Finally, federated learning is introduced in the training process to reduce the impact of non-IID data while protecting privacy. Simulation results show that the proposed algorithm can effectively improve system processing efficiency and reduce device energy consumption compared to the popular reinforcement learning algorithms.

1. Introduction

The rapid development of 5G communication technology has accelerated the application of the Internet of Things (intelligent transportation, intelligent industry, virtual/augmented reality, etc.) (Rao and Prasad, 2018; Bamasag et al., 2020; Attaran, 2023), among which intelligent transportation systems (ITS) have received extensive attention. In the application scenarios of ITS, a large number of different types of communication data are usually generated. The key information such as road conditions and vehicle density contained in these data can effectively improve the service quality of ITS. In autonomous driving, the vehicle-mounted device can generate about 1 GB of data per second (Zhou et al., 2018). If the cloud server cannot respond to data processing requests in a timely manner, the performance of autonomous driving will be severely limited. Therefore, mobile edge computing (MEC) is introduced into the Internet of Vehicles (IoV) to ensure ultra-reliable low-latency communication (URLLC) for users. MEC servers are deployed close to devices, which can reduce data transmission latency and provide flexible task offloading methods. How to efficiently utilize system resources and reasonably offload tasks has become a key issue. In addition, due to the increasing number of vehicle devices in IoV, vehicle to vehicle (V2V) and vehicle to infrastructure (V2I) communication generate a large amount of energy consumption. Reducing energy consumption has also become a major challenge in the implementation of IoV applications

In order to reduce the delay and energy consumption during application usage, many works on task offloading and resource allocation in IoV have been studied. Wu et al. (2018b) proposed a Lyapunov-based algorithm, which uses time delay as a constraint to reduce energy consumption and optimize the task offloading decision in the system. Considering that tasks are randomly generated by mobile users, Yi et al. (2019) proposed a mechanism based on queuing model to achieve non-cooperative game equilibrium among users. However, the above methods simplify the original problem and ignore some limiting factors (bandwidth, computing ability, etc.), which makes it difficult to accurately predict the actual conditions. With the development of Deep

* Corresponding author at: School of Electronic and Information, Xi'an Polytechnic University, Xi'an, 710048, China.
E-mail addresses: zhaoxu@xpu.edu.cn (X. Zhao), 210721091@stu.xpu.edu.cn (Y. Wu), 210721047@stu.xpu.edu.cn (T. Zhao), 210721075@stu.xpu.edu.cn (F. Wang), maozhen.li@brunel.ac.uk (M. Li).

https://doi.org/10.1016/j.jnca.2024.103941
Received 16 July 2023; Received in revised form 17 March 2024; Accepted 7 May 2024
Available online 25 June 2024

Reinforcement Learning (DRL), its excellent perception and decision-making capabilities enable it to perform well in the IoV environment with many interference factors. Some studies choose to offload vehicle data to edge servers to train machine learning models (Wang and Xu, 2020; Chu et al., 2022). There are also studies that solve optimization problems by using a reinforcement learning framework based on Deep Q Network (DQN) and splitting the problem into sub-problems when dealing with joint optimization of task offloading and resource allocation (Zhang et al., 2020; Li et al., 2021c). Although these traditional DRL-based solutions are effective in solving decision-making problems, they still face many problems: (1) The joint optimization of task offloading and resource allocation is a mixed-integer nonlinear programming problem, which is difficult to find the optimal solution by traditional DRL algorithms, and the DQN algorithm is not suitable for dealing with decision problems in the continuous action space; (2) In a complex IoV environment, the policy of each device is changing with training, which makes the environment unstable from the perspective of a single device, and there may be competition for resources such as bandwidth, computing, and storage (Cheng et al., 2020); (3) The localized observation of the environment by each agent may lead to an unbalanced data distribution, which affects the learning effect of the policy; (4) Due to the existence of some malicious nodes (e.g., fake base stations, malicious edge servers), some data generated by vehicles cannot be offloaded to edge or cloud servers for security, and frequent communication between devices will also increase the risk of data leakage (Gao et al., 2022).

In order to address the above problems, a federated deep reinforcement learning algorithm for the joint optimization problem of task offloading and resource allocation is proposed. Compared with DQN-based methods (Yu et al., 2020; Song et al., 2021; Li et al., 2022), multi-agent deep deterministic policy gradient (MADDPG) (Lowe et al., 2017) is used to more effectively address the effects of the unstable and resource-limited environment. The Actor–Critic structure of MADDPG also solves the decision-making problem in the continuous action space. Considering the impact of non-IID data in model training and the risk of privacy leakage, federated learning (FL) is introduced to improve the overall performance of the model while protecting privacy. As a distributed machine learning, FL only allows participants to share the trained model parameters during the training process, which avoids user data from leaving the local area and has good privacy (Yang et al., 2019). The main contributions of this article are summarized as follows:

- We establish communication, computing, and system cost models based on latency and energy consumption according to the network structure and task types in the IoV, and model the joint optimization problem of task offloading and resource allocation as a Markov decision process. Under the constraints of task requirements and resources, the reward function is efficiently designed to accelerate the model training.
- Since the traditional DRL algorithm cannot quickly obtain the optimal task offloading and resource allocation strategy in an unstable environment, MADDPG algorithm is used to solve the joint optimization problem. The MADDPG-based algorithm uses centralized training and distributed execution to obtain the optimal strategy. The actor–critic (AC) structure in MADDPG can help agents achieve better decision performance.
- There is a large amount of data transmission during the model training process. The introduction of federated learning in the training process of MADDPG not only reduces the impact of non-IID data, but also protects data privacy and security. Moreover, the FL based MADDPG algorithm is also suitable for distributed scenarios, which can enable the model to obtain a better convergence.

The rest of the article is organized as follows: Section 2 reviews the related work; Section 3 introduces the task offloading model of IoV; Section 4 presents the problem formulation and the joint optimization algorithm; Section 5 provides the simulation results; Section 6 presents limitations and future works; Section 7 concludes this work.

2. Related work

This section includes two parts: key considerations and existing research. Existing research will be introduced from the aspects of resource allocation, task offloading, and federated learning.

2.1. Key considerations

Task offloading and resource allocation are urgent problems in IoV applications. In order to solve these problems, the following factors should be considered:

(1) User experience: Low latency for transmission and computation is the first consideration to ensure that IoV applications meet users' requirements. Lower latency can provide users with more diverse services, and many delay-driven schemes have been proposed to improve users' quality of experience (QoE).

(2) Efficient scheduling: Artificial intelligence applications have high requirements on the computing capabilities of devices. This type of computation intensive application consumes a large amount of energy from the device and has a great impact on the device endurance, especially for popular electric vehicles. How to fully utilize server resources to process tasks and reduce energy consumption should be carefully considered.

(3) Privacy and security: Many schemes often lack consideration of data security and privacy during model training. The design of the algorithm should not only improve the performance of the model, but also protect privacy.

2.2. Existing research

IoV applications (autonomous driving, in-vehicle entertainment, etc.) have varying degrees of requirements in terms of equipment response time and computing power. Centralized cloud computing technology is gradually difficult to meet the needs of these emerging applications. The distributed deployment mode of mobile edge computing (MEC) can provide lower transmission delay and more flexible task offloading methods for moving vehicle devices, so it is widely used in the Internet of Vehicles (Xu et al., 2021; Singh et al., 2022; Aishwarya and Mathivanan, 2021). Li et al. (2021b) combined software-defined networking and MEC, and proposed a three-layer distributed control framework to provide URLLC services for Internet of Vehicles applications. In order to make application services no longer limited by the vehicle hardware conditions, Kong et al. (2022) provided computing and storage resources to vehicles through MEC servers to meet the needs of computation intensive applications. Since mobile devices usually lack sufficient computing resources to handle artificial intelligence-related tasks, Wu et al. (2018a) proposed an efficient algorithm to find the best offloading solution to reduce task delay and solve the problem of insufficient computing resources on devices. These studies combine MEC technology with the IoV, which greatly improves the system utility and user experience (Zhao et al., 2021b,a).

2.2.1. Resource allocation and task offloading in MEC

Mobile edge computing can provide near-end resources for network edge devices. At present, the allocation of resources including spectrum, computing, caching, and networking has received extensive attention in the research of MEC (Elfatih et al., 2022; Qiu et al., 2022; Zhang et al., 2023). He et al. (2017) proposed an integrated framework for dynamically allocating resources and using DRL to realize automatic decision-making. Liang et al. (2019) studied the allocation of spectrum resources in a high mobility vehicle environment, and improves the transfer efficiency between links by using DQN. This type of work is only considered from the perspective of resource allocation, ignoring the competitive relationship between various tasks. Due to the heterogeneity of edge devices in computing, storage, power, etc., the decision-making of task offloading is usually closely related to

resource allocation. In a multi-user scenario, there will be situations where multiple users offload computing tasks to the same edge. These users not only have to compete for bandwidth resources, but also compete for computing and storage resources with tasks offloaded to the same edge, so the strategy of allocating resources has a huge impact on the task offloading decision. Many studies have addressed resource allocation and task offloading as a joint optimization problem (mixed-integer nonlinear programming problem). There are usually two ways to deal with this kind of problem. One way is to split the joint optimization problem into multiple subproblems. Zhao et al. (2021c) decomposed the optimization problem into three subproblems: unloading rate selection, transmission power optimization, carrier and computational resource allocation, and proposes an iterative algorithm to process them sequentially. Nouri et al. (2019) studied the edge computing application scenario based on NOMA networks and jointly optimizes the device CPU frequency scheduling, transmission power, and computational resources to achieve a balance between latency and energy consumption. However, this type of approach has a high number of iterations and low efficiency. Another way to handle this problem is to use the DRL algorithm to solve the optimization problem. Dai et al. (2020) used DRL-based computing offloading and resource allocation algorithms to minimize system energy consumption in an end-edge–cloud orchestrated network. In Suh et al. (2022), a DQN-based network slicing technique was proposed to find out the resource allocation policy to maximize long-term throughput and met the quality of service requirements in beyond 5G systems.

The above works provide many excellent solutions for resource allocation and task offloading, but the above literature mainly studies the same type of task offloading problems, and how different types of tasks are cooperatively offloaded between different ends has not yet received attention.

2.2.2. Federated learning in MEC

Federated learning is a kind of distributed machine learning, which is mainly used to solve the problems related to distributed training and data security in the application of MEC (Nguyen et al., 2021; Javed et al., 2022; Lim et al., 2020; Salim and Park, 2022). In distributed training, each participant has local datasets with different distributions and devices with different performances. These limitations increase the difficulty of model training. As the basic algorithm of FL, FedAvg can reduce communication overhead and ensure a certain accuracy of the model even when each device has non-IID data when training the model (McMahan et al., 2017). Meanwhile, Wang et al. (2019) analyzed the convergence boundary of FL from a theoretical perspective to verify the feasibility of FL application in MEC systems, and proposed a control algorithm to achieve an optimal trade-off between local updates and global aggregation with limited resources. In order to reduce the impact arising from device heterogeneity, Samarakoon et al. (2019) proposed a novel FL-based distributed approach to estimate the tail distribution of task queue lengths and designed an FL algorithm for asynchronous updates to enhance model training efficiency. In terms of data security, Lu et al. (2020) combined hybrid blockchain architecture and FL to reduce the transmission load and ensure data security. The reliability of the shared data can also be ensured by two-stage verification. Li et al. (2021a) combined DQN and FL to propose a new data sharing scheme to ensure efficient and safe data sharing in the IoV. In Liu et al. (2021), a collaborative intrusion detection mechanism was proposed to ensure data security during distributed training of edge devices, reducing the resource overhead of the central server while ensuring data security and privacy. Therefore, introducing FL into MEC can not only protect private data, but also relieve communication pressure and improve model performance.

Different from the previous work, this study investigates task offloading and resource allocation strategies for end-edge–cloud hierarchical collaboration in IoV. Suitable offloading methods are selected for different tasks, and the MADDPG model is trained in a distributed manner by FL to enhance data privacy and improve model performance.

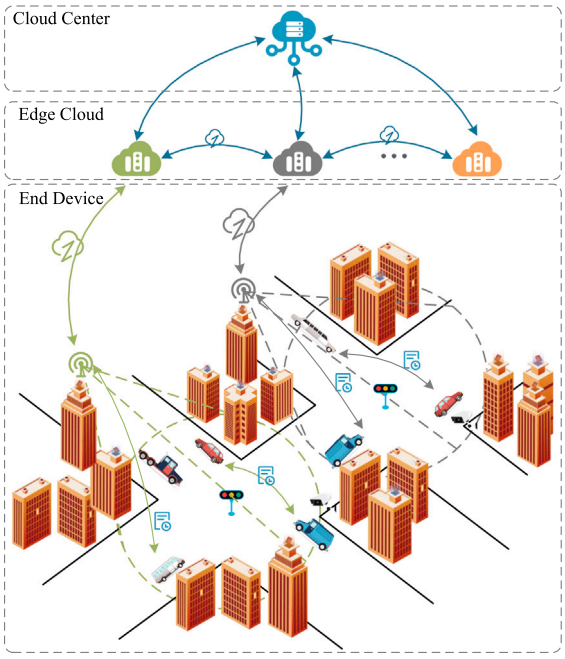

Fig. 1. System model.

3. Task offloading model of IoV

In this section, the modeling process will be discussed in detail, including the communication model, computational model and system cost model.

3.1. Model overview

Due to the development of communication technologies, there may be a large number of computation intensive and latency-sensitive applications in next-generation vehicle networks, which can be processed on local devices, edge servers, or cloud servers. As shown in Fig. 1, we assume a vehicle task offloading application scenario based on end-edge–cloud structure, which includes device layer, edge computing layer, and cloud center layer. The vehicles in the device layer generate various tasks while moving. Due to the limited energy and computing resources of the vehicles, some computing tasks need to be offloaded to the edge or cloud servers via V2I communication to assist in computing. The MEC server in the edge computing layer obtains real-time vehicle status information (task queue, vehicle speed, location, wireless resources, etc.) through the Road Side Unit (RSU), and its faster request response speed provides better task processing services for vehicle devices compared to the cloud center. The cloud center layer contains multi-core cloud servers, although the long distance between it and the MEC server will cause some delay in data transmission, its sufficient computing power can handle the tasks with high computational requirements and time consuming in vehicles.

The edge computing layer is located between the device layer and the cloud center layer, where the MEC servers connect the vehicle devices and the cloud servers. It plays an important role in offloading tasks from the vehicle devices, and a large amount of data is transmitted or stored through it. At the same time, it also decides the resource allocation and task offloading strategy as the decision maker, where the task offloading is executed in the following five ways: (1) local execution; (2) offloading to MEC server; (3) offloading to neighboring MEC servers; (4) offloading to other vehicles; (5) offloading to the cloud center.

The decision process of MEC server is shown in Fig. 2, and the process can be divided into the following steps:

Table 1
Key notations.

Notation	Definition
$r^e_{n,k}$	The transmission rate of uplink sub-channel k allocated to vehicle n
r^e_n	The total transmission rate of uplink channel between vehicle n and MEC server e
$\widetilde{r}^e_{n,k}$	The transmission rate of the downlink sub-channel k assigned to vehicle n
$\widetilde{r}^e_n$	The total transmission rate of downlink channel between vehicle n and MEC server e
$r^{e'}_{e,k}$	The transmission rate of uplink sub-channel k allocated to MEC server e
$r^{e'}_e$	The total transmission rate of uplink channel between MEC server e and nearby MEC server e'
$r^{n'}_{n,k}$	The transmission rate of uplink sub-channel k allocated to vehicle n
$r^{n'}_n$	The total transmission rate of uplink channel between vehicle n and nearby vehicle n'
r^c_e	The transmission rate between MEC server e and cloud server c
$C^{SR}_e, C^{SR}_{e2e}, C^{SR}_{n2n}$	Spectrum resources
K^e, K^{e2e}, K^{n2n}	The total number of channels
$B^e, \widetilde{B}^e, B^{e2e}, B^{n2n}$	Bandwidth of each sub-channel
$h^e_{n,k}, \widetilde{h}^e_{n,k}, h^{e'}_{e,k}, h^{n'}_{n,k}$	Channel gain of sub-channel k
$I^e_{n,k}, \widetilde{I}^e_{n,k}, I^{e'}_{e,k}, I^{n'}_{n,k}$	Channel interference of sub-channel k
$P^e_n, P^{e'}_e, P^{n'}_n, P^c_e$	Power for task offloading
σ^2	Noise power

S1: The vehicle obtains communication resources through RSU and periodically uploads the device status information such as vehicle resources and task queue to the MEC server.

S2: Based on the status information of surrounding devices and the requirements of tasks to be processed in the vehicle, the MEC server establishes a resource model and a system cost model for analyzing resource allocation and task unloading schemes.

S3: To obtain the optimal strategy for resource allocation and task offloading, the MEC server uses the MADDPG-based optimization algorithm for model training and distributes the decisions to the vehicles after the model training.

S4: The vehicle device receives the decision sent by the MEC server through the RSU and performs the corresponding operation. When the task is offloaded to the MEC server or the cloud center, the MEC server will execute the task directly or further offload it to the cloud center, and the resulting calculation result will be fed back to the vehicle device through the RSU. When the task is offloaded to an adjacent vehicle or an adjacent MEC server to assist in its execution, additional communication resources are used to forward the task to the adjacent device.

The key notations used in modeling are listed in Table 1.

3.2. Task type

In the IoV scenario, a large number of different types of application services are available to users (online navigation, autonomous driving, etc.), and the task requirements arising from the applications are varied. Assume that the offloading decision making operates in a time-slotted structure, and the decision making period is discrete time frames $t \in \{0, 1, 2, \ldots, T\}$. $A_{n,t} = \{\varphi^{da}_n, \varphi^{co}_n, \varphi^{de}_n\}$ denotes the tasks generated by the vehicle at time t, where φ^{da}_n, φ^{co}_n and φ^{de}_n denote the size of the task, the required computing resources, and the maximum acceptable delay, respectively. Since the vehicle will generate many different types of tasks while driving, the tasks are divided into three categories according to the delay requirements ($Thre1$ and $Thre2$ denote the delay threshold of the High-priority task and Medium-priority task, respectively):

(1) High-priority tasks $A^h = \{\varphi^{da}_h, \varphi^{co}_h, \varphi^{de}_h\}$: This type of task generally refers to emergency assistance applications. The collision warning and emergency braking functions provided by the vehicle system usually have high requirements for delay, and the maximum acceptable delay φ^{de}_h should be less than $Thre1$. These vehicle safety tasks are suitable for local execution.

(2) Medium-priority tasks $A^m = \{\varphi^{da}_m, \varphi^{co}_m, \varphi^{de}_m\}$: This type of task mainly involves assisted navigation-related applications, such as adaptive cruise system, lane keeping system, map navigation and other assisted driving applications. With the help of these auxiliary applications, the vehicle maintains a relatively stable speed, the task has relatively low requirements for delay, and the maximum acceptable delay φ^{de}_m should be between $Thre1$ and $Thre2$. These tasks are suitable for offloading to MEC server or other vehicles to execute the tasks.

(3) Low-priority tasks $A^l = \{\varphi^{da}_l, \varphi^{co}_l, \varphi^{de}_l\}$: This type of task usually includes applications such as parking navigation and multimedia entertainment. These applications contain a large amount of application data that needs to be processed and have relatively low latency requirements. The maximum acceptable latency φ^{de}_l is usually larger than $Thre2$. These tasks are suitable for offloading to cloud centers for task execution.

Therefore, classification according to different requirements and characteristics of tasks is beneficial to improve the processing efficiency of the system.

3.3. Communication model

The communication resources required for task offloading, policy delivery and result transmission in the IoV are usually provided by the RSU, and the modeling is mainly based on Shannon's formula to calculate the transmission rate of communication.

(1) Vehicle device to MEC server: the RSU is required to provide spectrum resources for the vehicle device and MEC server to communicate. Assume that the task $A_{n,t}$ generated by vehicle n needs to be offloaded to MEC server e. The total amount of uplink spectrum resources is C^{SR}_e, the total number of uplink channels is K^e and the channel set is $\mathbb{K}^e$. Then, $B^e = C^{SR}_e / K^e$ indicates the bandwidth of each subchannel. The transmission rate of uplink channel k allocated to vehicle n can be expressed as:

$$r^e_{n,k} = B^e log_2\left(1 + \frac{P^e_n \cdot h^e_{n,k}}{\sigma^2 + I^e_{n,k}}\right)$$
$$I^e_{n,k} = \sum_{m=1, m\neq k}^{K^e} P^e_n \cdot h^e_{n,m} \tag{1}$$

where P^e_n is the power when vehicle n offloads the task, σ^2 is the noise power, $h^e_{n,k}$ denote the channel gain, which follows Rayleigh distribution with path loss, $I^e_{n,k}$ denote the channel interference. We use $x^e_{n,k}$ to denote whether uplink channel k is allocated to vehicle n. If it is allocated, $x^e_{n,k} = 1$, otherwise, $x^e_{n,k} = 0$. Thus, the total transmission rate between vehicle n and MEC server e can be expressed as:

$$r^e_n = \sum_{k\in\mathbb{K}^e} x^e_{n,k} \cdot r^e_{n,k} \tag{2}$$

Similarly, the transmission rate of the downlink channel k assigned to vehicle n can be expressed as:

$$\widetilde{r}^e_{n,k} = \widetilde{B}^e log_2\left(1 + \frac{P^e_n \cdot \widetilde{h}^e_{n,k}}{\sigma^2 + \widetilde{I}^e_{n,k}}\right) \tag{3}$$

The total transmission rate of downlink channel between vehicle n and MEC server e can be expressed as:

$$\widetilde{r}^e_n = \sum_{k\in\mathbb{K}^e} \widetilde{x}^e_{n,k} \cdot \widetilde{r}^e_{n,k} \tag{4}$$

(2) Between MEC servers: Since MEC servers sometimes cannot meet the computing requirements of tasks, they need to offload tasks to

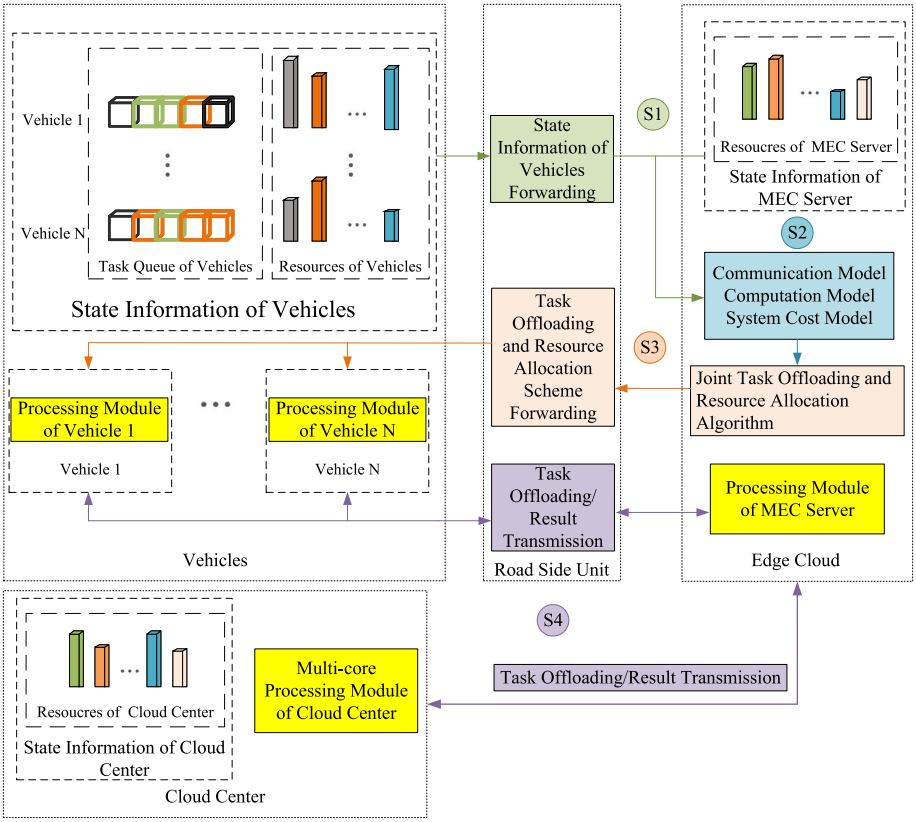

Fig. 2. System framework.

neighboring MEC servers. Assume that MEC server e needs to offload the task to the nearby MEC server $e' \in \mathbb{E}$. The total amount of uplink spectrum resources is C_{e2e}^{SR}, the total number of uplink channels is K^{e2e} and the channel set is $\mathbb{K}^{e2e}$. Then, $B^{e2e} = C_{e2e}^{SR}/K^{e2e}$ indicates the bandwidth of each subchannel. The transmission rate of uplink channel k allocated to MEC server e can be expressed as:

$$\begin{aligned} r_{e,k}^{e'} &= B^{e2e} log_2\left(1 + \frac{P_e^{e'} \cdot h_{e,k}^{e'}}{\sigma^2 + I_{e,k}^{e'}}\right) \\ I_{e,k}^{e'} &= \sum_{m=1, m\neq k}^{K^{e2e}} P_e^{e'} \cdot h_{e,m}^{e'} \end{aligned} \tag{5}$$

where $P_e^{e'}$ is the power when MEC server e offloads the task, σ^2 is the noise power, $h_{e,k}^{e'}$ denote the channel gain, which follows Rayleigh distribution with path loss, $I_{e,k}^{e'}$ denote the channel interference. We use $x_{e,k}^{e'}$ to denote whether uplink channel k is allocated to MEC server e. If it is allocated, $x_{n,k}^{e'} = 1$, otherwise, $x_{e,k}^{e'} = 0$. Thus, the total transmission rate between MEC server e and nearby MEC server e' can be expressed as:

$$r_e^{e'} = \sum_{k \in \mathbb{K}^{e2e}} x_{e,k}^{e'} \cdot r_{e,k}^{e'} \tag{6}$$

(3) Between vehicle devices: In order to reduce the computational burden of the MEC server, when the surrounding vehicle devices have free computational resources that can be utilized, the vehicles can communicate with each other via V2V communication to offload tasks and relieve computing pressure. Assume that the total amount of uplink spectrum resources is C_{n2n}^{SR}, the total number of uplink channels is K^{n2n} and the channel set is $\mathbb{K}^{n2n}$. Then, $B^{n2n} = C_{n2n}^{SR}/K^{n2n}$ indicates the bandwidth of each subchannel. The transmission rate of uplink channel k allocated to vehicle n can be expressed as:

$$\begin{aligned} r_{n,k}^{n'} &= B^{n2n} log_2\left(1 + \frac{P_n^{n'} \cdot h_{n,k}^{n'}}{\sigma^2 + I_{n,k}^{n'}}\right) \\ I_{n,k}^{n'} &= \sum_{m=1, m\neq k}^{K^{n2n}} P_n^{n'} \cdot h_{n,m}^{n'} \end{aligned} \tag{7}$$

where $P_n^{n'}$ is the power when vehicle n offloads the task, σ^2 is the noise power, $h_{n,k}^{n'}$ denote the channel gain, which follows Rayleigh distribution with path loss, $I_{n,k}^{n'}$ denote the channel interference. We use $x_{n,k}^{n'}$ to denote whether uplink channel k is allocated to vehicle n. If it is allocated, $x_{n,k}^{n'} = 1$, otherwise, $x_{n,k}^{n'} = 0$. Thus, the total transmission rate between vehicle n and nearby vehicle n' can be expressed as:

$$r_n^{n'} = \sum_{k \in \mathbb{K}^{n2n}} x_{n,k}^{n'} \cdot r_{n,k}^{n'} \tag{8}$$

(4) MEC server to cloud server: Since the edge server and the cloud center are connected using high-speed fiber, the transmission rate is relatively stable and the speed between them can be expressed as a fixed value r_e^c.

3.4. Computation model

The computational model includes both latency and energy consumption, and the modeling also needs to take into account the impact caused by environmental factors during the transmission and execution of tasks.

(1) Local executing: The vehicle executes tasks with local resources to save the time of task uploading, and this approach meets the requirements of high-priority tasks. When task $A_{n,t}$ is executed locally, vehicle

n assigns computing resources f_n to the task. Thus, the execution time and energy consumption of the task can be expressed as follows:

$$T_n^{loc} = \frac{\varphi_n^{co}}{f_n} \tag{9}$$

$$E_n^{loc} = \eta_n \left(f_n\right)^2 \varphi_n^{co} \tag{10}$$

According to the work in Li et al. (2020) and Jiang et al. (2022), the energy consumption of local executing is proportional to the square of the voltage of CPU, and it is further noticed that the CPU clock frequency f_n is approximately linear proportional to the voltage supply. Therefore, the energy consumption of one CPU cycle can be given as $\eta_n \left(f_n\right)^2$, where the value of η_n depends on the CPU structure of vehicle n.

(2) Offloading to the MEC server: When the computing resources of the vehicle cannot meet the task demand, it can resort to the MEC server to assist the computation. Assuming that vehicles in the same range will offload tasks to the same MEC server. f_e denotes the computing resources of MEC server e, and θ_n^e denotes the proportion of computing resources shared by task $A_{n,t}$. Since the results of medium-priority tasks are usually small, the transmission time of the results can be ignored. Then, the execution time of the tasks can be expressed as:

$$T_{n,e}^{edge} = \frac{\varphi_n^{da}}{r_n^e} + \frac{\varphi_n^{co}}{\theta_n^e f_e} \tag{11}$$

The energy consumption mainly contains both transmission and computation components. Therefore, the total energy consumption required for the completion of task $A_{n,t}$ can be expressed as:

$$E_{n,e}^{edge} = \frac{P_n^e \varphi_n^{da}}{r_n^e} + \varphi_n^{co} C_e \tag{12}$$

where C_e denotes the energy consumption per computing resource of MEC server e.

(3) Offloading to the nearby MEC server: Considering the limitation of computation and storage resources of MEC server, using the resources of nearby MEC server can also relieve the pressure on the server. Assuming that MEC server e offloads tasks to nearby MEC server e', $f_{e'}$ denotes the computing resources of nearby MEC server e', $\theta_n^{e'}$ denotes the proportion of computing resources shared by task $A_{n,t}$, and the execution time of the task can be expressed as:

$$T_{n,e,e'}^{edge} = \frac{\varphi_n^{da}}{r_n^e} + \frac{\varphi_n^{da}}{r_e^{e'}} + \frac{\varphi_n^{co}}{\theta_n^{e'} f_{e'}} \tag{13}$$

The total energy consumption required for the completion of task $A_{n,t}$ can be expressed as:

$$E_{n,e,e'}^{edge} = \frac{P_n^e \varphi_n^{da}}{r_n^e} + \frac{P_e^{e'} \varphi_n^{da}}{r_e^{e'}} + \varphi_n^{co} C_{e'} \tag{14}$$

where $C_{e'}$ denotes the energy consumption per computing resource of MEC server e'.

(4) Offloading to the nearby vehicle: Similar to the offloading to nearby MEC servers, this approach is also suitable for medium-priority tasks. $n' \in \mathbb{N}$ denotes nearby vehicles, $f_{n'}$ denotes computing resources of nearby vehicles, and the execution time of the task can be expressed as:

$$T_{n,n'} = \frac{\varphi_n^{da}}{r_n^{n'}} + \frac{\varphi_n^{co}}{f_{n'}} \tag{15}$$

The total energy consumption required for the completion of task $A_{n,t}$ can be expressed as:

$$E_{n,n'} = \frac{P_n^{n'} \varphi_n^{da}}{r_n^{n'}} + \eta_{n'} \left(f_{n'}\right)^2 \varphi_n^{co} \tag{16}$$

(5) Offloading to the cloud center: Low-priority tasks and few medium-priority tasks that require a lot of computation can be processed by offloading the tasks to the cloud center, and with the powerful computing power of Cloud Center, the execution rate of the tasks will be significantly increased. Among them, since most of these tasks are multimedia applications, the return results usually include a large amount of data, and the delay and energy consumption of data downlink should also be taken into account. Meanwhile, considering that the cloud center server has sufficient computing and storage resources, the execution delay and energy consumption of the task in the cloud center can be ignored. *result* denotes the returned result, then the execution time of the task can be expressed as:

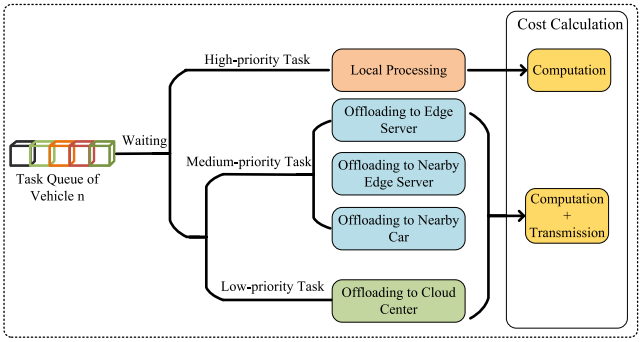

Fig. 3. Cost calculation of different processing methods.

$$T_{n,c}^{cloud} = \begin{cases} \dfrac{\varphi_n^{da}}{r_n^e} + \dfrac{\varphi_n^{da}}{r_e^c}, A_{n,t} \in A^m \\ \dfrac{\varphi_n^{da}}{r_n^e} + \dfrac{\varphi_n^{da}}{r_e^c} + \dfrac{result}{r_e^c} + \dfrac{result}{\widetilde{r}_n^e}, A_{n,t} \in A^l \end{cases} \tag{17}$$

The total energy consumption required for the completion of task $A_{n,t}$ can be expressed as:

$$E_{n,c}^{cloud} = \begin{cases} \dfrac{P_n^e \varphi_n^{da}}{r_n^e} + \dfrac{P_e^c \varphi_n^{da}}{r_e^c}, A_{n,t} \in A^m \\ \dfrac{P_n^e \varphi_n^{da}}{r_n^e} + \dfrac{P_e^c \varphi_n^{da}}{r_e^c} + \dfrac{P_e^c \cdot result}{r_e^c} + \dfrac{P_n^e \cdot result}{\widetilde{r}_n^e}, A_{n,t} \in A^l \end{cases} \tag{18}$$

where P_e^c denotes the transmission power between the MEC server and the cloud center.

3.5. System cost model

The system cost is mainly composed of both energy consumption and delay. As shown in Fig. 3, there are multiple tasks waiting to be processed in the queue at time t. If the tasks cannot be processed immediately, they can wait in the queue for resources to be released. After waiting, the corresponding offloading operation is executed according to the task type. When executing high-priority tasks locally, the system cost only needs to consider the local computation time consumption and energy consumption. When medium-priority tasks are processed, the tasks are forwarded to other devices, and the time and energy consumption during task transfer also need to be considered. When low-priority tasks are offloaded to the cloud center, since most of these tasks are generated by entertainment applications and the processing results are accompanied by large amounts of data, the transfer of the results is additionally included in the system cost.

Based on the different task offloading models mentioned above, the execution time for the task generated by vehicle n at time t can be expressed as:

$$\begin{aligned} T_{n,t} = {} & \phi_l T_n^{loc} + \phi_e T_{n,e}^{edge} + \phi_{e'} T_{n,e,e'}^{edge} + \phi_{n'} T_{n,n'} \\ & + \phi_c T_{n,c}^{cloud} + T_{n,t}^W \end{aligned} \tag{19}$$

where $\phi_l, \phi_e, \phi_{e'}, \phi_{n'}, \phi_c \in \{0, 1\}$ denote five task processing methods, $T_{n,t}$ denotes the waiting time of the task in the queue. Similarly, the

energy consumption for the task can be expressed as:

$$E_{n,t} = \phi_l E_n^{loc} + \phi_e E_{n,e}^{edge} + \phi_{e'} E_{n,e,e'}^{edge} + \phi_{n'} E_{n,n'} + \phi_c E_{n,c}^{cloud} \tag{20}$$

Finally, the total system cost of task execution can be expressed as:

$$U_{n,t} = \alpha_n T_{n,t} + (1 - \alpha_n) E_{n,t} \tag{21}$$

where α_n and $1 - \alpha_n$ denote the weight coefficient of delay and energy consumption, respectively. α_n ranges between 0 and 1.

4. Problem formulation and joint optimization algorithm

4.1. Problem formulation

With the development of MEC, various types of tasks generated by applications can be executed in multiple ways. Due to the limited resources in IoV environment, the MEC server, as a scheduler, needs to give appropriate task offloading and resource allocation strategies to fully utilize the computing resources at both the MEC server and cloud server to reduce energy consumption and delays during task execution. In order to improve the vehicle user experience, the number of completed tasks and the system cost consisting of delay and energy consumption are used to reflect the performance of the task offloading decision, which can be expressed as:

$$\lambda_t = \sum_{n \in N} \frac{D_{n,t}}{U_{n,t}} \tag{22}$$

where N and $D_{n,t}$ denote the total number of vehicles and the total number of tasks completed by vehicle n, respectively. To maximize system processing efficiency under limited environmental resources and different task requirements, the optimization problem is as follows:

$$\begin{aligned}
&max \quad \lambda_t \\
&s.t. \\
&(c1)\ \phi_l, \phi_e, \phi_{e'}, \phi_{n'}, \phi_c \in \{0, 1\} \\
&(c2)\ \left(\phi_l + \phi_e + \phi_{e'} + \phi_{n'} + \phi_c\right) \leq 1 \\
&(c3)\ \theta \in [0, 1] \\
&(c4)\ \sum_{k \in \mathbb{K}^e} x_{n,k}^e \leq 1 \\
&(c5)\ \sum_{k \in \mathbb{K}^{e2e}} x_{n,k}^{e'} \leq 1 \\
&(c6)\ \sum_{k \in \mathbb{K}^{n2n}} x_{n,k}^{n'} \leq 1 \\
&(c7)\ T_{n,t} \leq \varphi_n^{de}
\end{aligned} \tag{23}$$

For the above constraints, constraint $(c1)$ indicates that there are five types of task execution. Constraint $(c2)$ indicates that the vehicle can only choose one way to process the task. Constraint $(c3)$ indicates that the allocation of computing resources to the task cannot exceed the total resources. Constraints $(c4)$, $(c5)$ and $(c6)$ indicate that each uplink and downlink channel can be allocated to only one device in each scheduling period when communication is performed between devices. Constraint $(c7)$ indicates that the task execution time needs to meet the maximum acceptable delay for the task.

4.2. DRL-based solution

The above optimization problem is proposed to accomplish as many tasks as possible while reducing system costs. When more devices are connected to the environment, the generated application tasks will occupy a large amount of system resources. How to allocate communication, computing and other resources in the system for various tasks becomes a complicated problem. First, the optimization problem needs to be converted into a Markovian decision process to describe the decision process of task offloading and resource allocation, then MADDPG is used to train the task offloading and resource allocation strategies for the end-edge–cloud network structure and the multi-intelligence interaction environment. During the training process, the MEC server interacts with the environment as an agent, and each agent has to update the model considering the possible actions of other agents to obtain the most appropriate task offloading and resource allocation policies. Next, the elements of the Markov decision process (state space, action space, and reward function) are introduced in detail.

(1) State Space: At the beginning of each time slot, the MEC server e will collect the state and available resources of the vehicles within the signal range to establish state information. At time t, the state information observed by the MEC server includes the following elements:

(a) Let $S_{e,t}^{task} = (A_{1,t}, A_{2,t}, \ldots, A_{N,t})$ denote the task state of the vehicle within the communication range of the MEC server. If the vehicle n is out of communication range, then $A_{n,t} = 0$.
(b) At time t, the computing resources available to the MEC server e can be expressed as $S_{e,t}^{co} = f_{e,t}$.
(c) At time t, the spectrum resources available to the MEC server e can be expressed as $S_{e,t}^{sr} = C_{e,t}^{SR}$.

Therefore, the state space of the MEC server e at time t can be expressed as $S_{e,t} = \left(S_{e,t}^{task}, S_{e,t}^{co}, S_{e,t}^{sr}\right)$, and the system state space can be defined as $S_t = \left(S_{1,t}, S_{2,t}, \ldots, S_{E,t}\right)$.

(2) Action Space: The tasks in the queue contain a total of five execution methods according to different types. At time t, the MEC server decides the task execution method and allocates computing and communication resources to the vehicle. $\phi_l, \phi_e, \phi_{e'}, \phi_{n'}, \phi_c \in \{0, 1\}$ represent five task offloading decisions. Since tasks can only be offloaded in one way, constraint $\left(\phi_l + \phi_e + \phi_{e'} + \phi_{n'} + \phi_c\right) \leq 1$ needs to be satisfied. The analysis of the five execution actions is as follows:

(a) When task $A_{n,t}$ is executed locally, $\phi_l = 1$, the vehicle also needs to allocate computational resources θ to the task, and the remaining values are 0.
(b) When task $A_{n,t}$ is offloaded to the MEC server, $\phi_e = 1$, the communication resources required for communication between the vehicle, and the MEC server are $x_{n,k}^e$, the computing resources obtained are θ, and the remaining values are 0.
(c) When task $A_{n,t}$ is offloaded to the nearby MEC server, $\phi_{e'} = 1$, the communication resources required for communication between the vehicle and the MEC server are $x_{n,k}^e$, and the communication resources required for communication between neighboring MEC servers are $x_{n,k}^{e'}$, the computing resources obtained are θ, and the remaining values are 0.
(d) When task $A_{n,t}$ is offloaded to other vehicles, $\phi_{n'} = 1$, the required communication resources between the vehicle and the MEC server are $x_{n,k}^{n'}$, the computing resources obtained are θ, and remaining the values are 0.
(e) When task $A_{n,t}$ is offloaded to the cloud center, $\phi_c = 1$, the communication resources required for the task to be forwarded through the MEC server are $x_{n,k}^e$, and the remaining values are 0.

Therefore, the action space of the MEC server at the moment of time can be expressed as $a_{e,t} = (\phi_l, \phi_e, \phi_{e'}, \phi_{n'}, \phi_c, \theta, x_{n,k}^e, x_{n,k}^{e'}, x_{n,k}^{n'})$, and the system action space can be defined as $a_t = (a_{1,t}, a_{2,t} \cdots a_{E,t})$.

(3) Reward Function: In order to maximize the energy efficiency of system processing and achieve a balance between task completion rate and system cost, different reward functions are set based on the constraints and optimization objective. The first step is to ensure the resources constraints. After the action $a_{e,t}$ is taken, if constraints $(c1)$ –

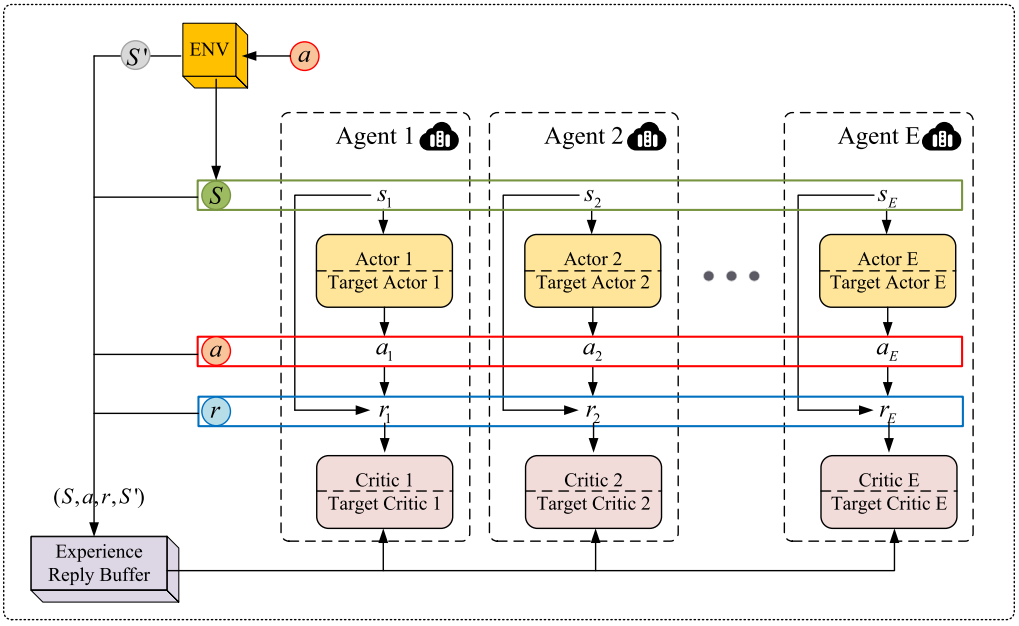

Fig. 4. Algorithm framework.

$(c6)$ are not satisfied, the reward function will be defined as:

$$\begin{aligned} r_{e,t} = {} & l_1 + \beta_1 \cdot (\theta - 1) \cdot \wedge(c3) \\ & + \beta_2 \cdot (\phi_l + \phi_e + \phi_{e'} + \phi_{n'} + \phi_c - 1) \cdot \wedge(c1, c2) \\ & + \beta_3 \cdot (\sum_{k \in \mathbb{K}^e} x^e_{n,k} - 1) \cdot \wedge(c4) \\ & + \beta_4 \cdot (\sum_{k \in \mathbb{K}^{e2e}} x^{e'}_{n,k} - 1) \cdot \wedge(c5) \\ & + \beta_5 \cdot (\sum_{k \in \mathbb{K}^{n2n}} x^{n'}_{n,k} - 1) \cdot \wedge(c6) \end{aligned} \tag{24}$$

where $\wedge(*)$ indicates that if the constraint $(*)$ is not satisfied, the value is -1; otherwise, the value is 0. $l_1, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5$ are the experimental parameters. The second step is to ensure the delay constraint. If only constraints $(c1) - (c6)$ are satisfied, the reward function will be defined as:

$$r_{e,t} = l_2 + \exp(T_{n,t} - \varphi^{de}_n) \tag{25}$$

where l_2 is the experimental parameter. The final step is to maximize the optimization objective. If all constraints $(c1) - (c7)$ are satisfied, the reward function will be defined as:

$$r_{e,t} = l_3 + \beta_6 \cdot \exp(-\lambda_{e,t}) \tag{26}$$

where l_2, β_6 are experimental parameters.

4.3. MADDPG-based joint optimization method

In the IoT environment, each device's strategy is changing with training and the environment becomes less stable. Also, with limited resources in the environment, the competition for resources among devices is intense. This leads to the fact that the traditional single-agent reinforcement learning algorithm is no longer applicable to the current environment. Therefore, a framework based on MADDPG is designed.

As shown in Fig. 4, MADDPG based on actor–critic combines a dual neural network (current network and target network) and an experience replay mechanism (Wang et al., 2020). The actor network computes a suitable action based on the state of the agent, and the critic network evaluates the goodness of the action, which can select the suitable action in the continuous action space. The experience replay buffer is used to save a certain amount of data about the previous state changes of the agents in the environment, and when the network model needs to be updated, the current network can randomly take out data from the experience replay buffer for training, which can break the correlation between data and also make the training process more stable. The training process of MADDPG includes two parts: centralized training and distributed execution. In the centralized training, the training samples are first randomly selected from the experience replay buffer D in the cloud center. The participating agent e only observes its surroundings and input the state information to the actor network to obtain the optimal action a_e. The critic network needs to evaluate the action value based on the actions a and state information of all agents S in the environment, and then update the parameters of the two network models according to the loss function. During the distributed execution, the agents only use the actor network to interact with the environment and collect training samples. The sample data of all the agents (S, a, r, S') are then stored centrally in the experience replay buffer D for future model training. By interacting with the environment and training with MADDPG, each agent can eventually build an optimal policy for task offloading and resource allocation.

The pseudocode of the proposed MADDPG algorithm for the joint optimization problem is illustrated in Algorithm 1. Assuming that the number of MEC servers participating in the centralized training is E. $\mu = \{\mu_{\theta_1}, \ldots, \mu_{\theta_E}\}$ (abbreviated as μ_i) and $\theta = \{\theta_1, \ldots, \theta_E\}$ represent the deterministic policies and network parameters of the participants respectively. Therefore, the gradient of the deterministic policy for MEC server e can be expressed as:

$$\nabla_{\theta_e} J(\mu_e) = \mathbb{E}_{S,a\sim D}\left[\nabla_{\theta_e}\mu_e(a_e|s_e)\nabla_{a_e}Q^{\mu}_e(S, a_1, \ldots, a_E)|_{a_e=\mu_e(s_e)}\right] \tag{27}$$

where $Q^{\mu}_e(S, a_1, \ldots, a_E)$ is the Q-value function. The target Q value is calculated by the target Q-value function, which can be expressed as:

$$y = r_e + \gamma Q^{\mu'}_e(S', a'_1, \ldots, a'_E)|_{a'_j=\mu'_j(S'_j)} \tag{28}$$

where γ is the discount factor. The loss function can be expressed as:

$$L(\theta_e) = \mathbb{E}_{S,a,r,S'}\left[(Q^{\mu}_e(S, a_1, \ldots, a_E) - y)^2\right] \tag{29}$$

The action network is updated through gradient descent, which can be expressed as:

$$\nabla_{\theta_e} J \approx \frac{1}{X}\sum_i \nabla_{\theta_e}\mu_e(s^i_e)\nabla_{a_e}Q^{\mu}_e(s^i, a^i{}_1, \ldots, a^i{}_E)|_{a_e=\mu_e(S'_e)} \tag{30}$$

where X denotes the size of mini-batch, and i represents the index of samples.

Algorithm 1 MADDPG Training Process

Initialize:
Initialize the weights of Actor and Critic networks.
Initialize the replay buffer D.
for $episode = 1 : M$ **do**
  Initialize a random process N for action exploration;
  Receive initial state S;
  for $t = 1 : max\text{-}episode\text{-}length$ **do**
    Each agent e choose action $a_{e,t} = \mu_e(S_{e,t}) + N_t$ by the current policy;
    Execute actions $a_t = (a_{1,t}, a_{2,t}, \cdots, a_{E,t})$;
    Obtain reward r_t and new state S'_t;
    Store (S, a_t, r_t, S') into the reply buffer D;
    $S \leftarrow S'$;
    for agent $e = 1 : E$ **do**
      Sample a random minibatch of X samples (S^i, a^i, r^i, S'^i) from D;
      Set $y^i = {r_e}^i + \gamma Q_e^{\mu'}(S'^i, a'_1, \cdots, a'_E)|_{a'_j = \mu'_j(S'_j)}$;
      Update the critic network by minimizing the loss function: $L(\theta_e) = \frac{1}{X}\sum_i \left(y^i - Q_e^{\mu'}(S^i, a^i_1, \cdots, a^i_E)\right)^2$;
      Update the actor network using the sampled policy gradient: $\nabla_{\theta_e} J \approx \frac{1}{X}\sum_i \nabla_{\theta_e}\mu_e(s^i_e)\nabla_{a_e} Q_e^{\mu}(s^i, {a^i}_1, \cdots, {a^i}_E)|_{a_e=\mu_e(S'_e)}$;
    end for
    Update target network parameters for each agent e: $\theta_e = K\theta_e + (1-K)\theta'_e$;
    if $episodet == t_{agg}$ **then**
      Execute model aggregation;
    end if
  end for
end for

4.4. Computational complexity analysis

Floating point operations (FLOPs) can be used to measure the computational complexity of the proposed algorithm. Considering the bias added in the fully connected layer, the FLOPs of a fully connected layer is $2 \times I \times O$, where I denotes the input dimensionality and O denotes the output dimensionality.

From the above section, the proposed algorithm framework consists of four neural networks including two different structures. The critic network is composed of three fully connected layers, and the actor network is composed of four fully connected layers. Therefore, the computational complexity of the algorithm mainly depends on the structures of neural networks, which can be expressed as:

$$2 \times \sum_{n=1}^{3} I_{actor,n} O_{actor,n} + 2 \times \sum_{n=1}^{4} I_{critic,n} O_{critic,n} \tag{31}$$

where I_n denotes the number of input neurons in the nth layer and O_n denotes the number of output neurons in the nth layer. The computational capacity of a single-core computer is about 2 billion FLOPs per second, which is more than sufficient to meet the computational requirements of critic networks and actor networks. The proposed algorithm framework can be implemented in a real-world environment according to the parameter settings in Section 5.1.

4.5. MADDPG model optimization method based on FL

The goal of the joint optimization algorithm is to handle as many tasks as possible while reducing the system cost. In the training process of the model, a large amount of user data needs to be shared to enhance the model performance, and the sending of data will occupy too much communication resources, which increases the difficulty of resource allocation. In addition, there is also the risk of information leakage during the communication process, and information security issues should be considered. In order to overcome these difficulties, federated learning is introduced in the training framework of MADDPG.

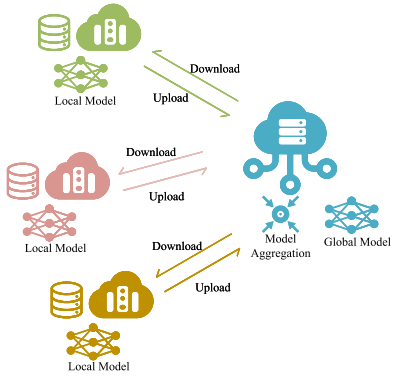

Fig. 5. FL in MADDPG training.

Unlike traditional distributed learning, FL realizes joint modeling and model performance optimization on the premise of ensuring data privacy security. As shown in Fig. 5, the FL model in the edge computing scenario is established. In the model initialization phase, each MEC server receives the global MADDPG model $W(t)$ from the cloud server and initializes the model parameters. In the model training phase, each MEC server only uses local data to train the model $W_1(t), W_2(t), \ldots, W_N(t)$. When the local training of MEC servers is completed, the updated model parameters are uploaded to the cloud server. In the model aggregation phase, the cloud server aggregates the model parameters and updates the global model $W(t+1)$ based on the percentage of local data in the total data during the training of each MEC server, and then sends the model parameters to each MEC server to continue the next round of training. When a new model is received by the MEC server, the model parameters are updated using soft update to reduce the impact of undesirable parameters in the new model. The formulae for local update and cloud aggregation of the model during training are as follows:

$$W_i(t) = \sigma W_i(t) + (1-\sigma) W(t) \tag{32}$$

$$W(t+1) = \sum_{i=1}^{E} \frac{D_i}{D} W_i(t) \tag{33}$$

where σ represents the weight of soft update, D represents the total number of training samples, and D_i represents the number of local training samples in the MEC server i. Algorithm 2 is the pseudocode of the FL-based MADDPG model aggregation algorithm.

5. Simulation results

5.1. Parameter setting

In this section, PyTorch is used to implement the MADDPG-based task offloading and resource allocation optimization algorithm. We first set up the simulation environment for the experiment. Considering that there is an environment of 1 cloud server and 5–9 MEC servers, and 40–100 devices within the communication range of each MEC server. Then, the structure of the neural network in the optimization algorithm is designed. The Actor network contains two fully connected hidden layers and the Critic network contains three connected hidden layers. Both the actor and critic networks use the relu function as the activation function, while the output layer of the actor network uses the tanh function to limit the output results. Based on 3GPP R15, the specific simulation and model training parameters are shown in Table 2. In

Algorithm 2 FL-Based MADDPG Training Model

Initialize:
Server side: At the beginning of the learning phase $t = 0$, initialize the weight values of the MADDPG model $W(t)$;
Agent side: Initialize the weight values of the local MADDPG model $W_i(0), (i = 1, 2, \cdots, N)$;
for $t = 1 : max\text{-}episode\text{-}length$ **do**
  Agent side:
  for $i = 1 : E$ **do**
    Download the parameters $W(t)$ from the Server;
    Update the local parameters: $W_i(t) = \sigma W_i(t) + (1 - \sigma)W(t)$;
    Execute locally training based on local data;
    Update the trained weight values $W_i(t + 1)$ and transmit corresponding values to Server;
  end for
  Server side:
  Aggregate the updated weight values $W_i(t)$ of end devices under its coverage:
  $W(t + 1) = \sum_{i=1}^{E} \frac{D_i}{D} W_i(t)$;
  Broadcast the averaged weight value $W(t + 1)$;
end for

Table 2
Parameter settings.

Parameter	Value
Layer of critic network	5
Layer type of critic network	Fully connected
Neurons of hidden layers for critic network	[1024,512,256]
Learning rate of critic network	0.0001
Layer of actor network	4
Layer type of actor network	Fully connected
Neurons of hidden layers for critic network	[256,128]
Learning rate of actor network	0.0001
Buffer size	6000
Learning rate of FL	0.4
Optimizer	Adam
Activation function	relu
Mini-batch	128
Number of MEC servers	5,7,9
Number of devices	200–800
Noise power	−120 dB
Transmission power of vehicle	0.5 W
Bandwidth of vehicle	1 GHz
Bandwidth of RSU	100 MHz
Computing resources of vehicle	2 GHz
Computing resources of MEC server	16 GHz
Path loss model	128.1 + 37.6lg(D)dB

order to evaluate the performance of the proposed algorithm, the following algorithms are compared:

(1) Random Offloading (RD): Three types of tasks are randomly assigned and executed. The resources allocated to the tasks sometimes do not meet the execution requirements.

(2) DQN: This algorithm combines Q-learning and neural networks to obtain the best offloading action. But in the current scene, continuous actions need to be discretized (Zhao et al., 2022; Yang et al., 2022).

(3) Deep Deterministic Policy Gradient (DDPG): This algorithm adds Actor–Critic structure to DQN to obtain the best unloading policy and uses Actor network to obtain deterministic actions (He et al., 2020; Hazarika et al., 2022; Zhao et al., 2023).

5.2. Performance analysis

The agents are trained using our proposed algorithm in different simulation environments, and the performance of the algorithm is evaluated in terms of convergence, task completion rate, and energy consumption.

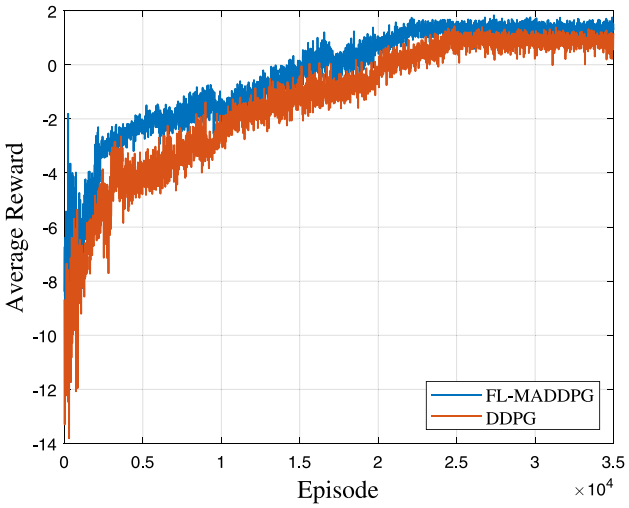

Fig. 6. Convergence of different algorithms.

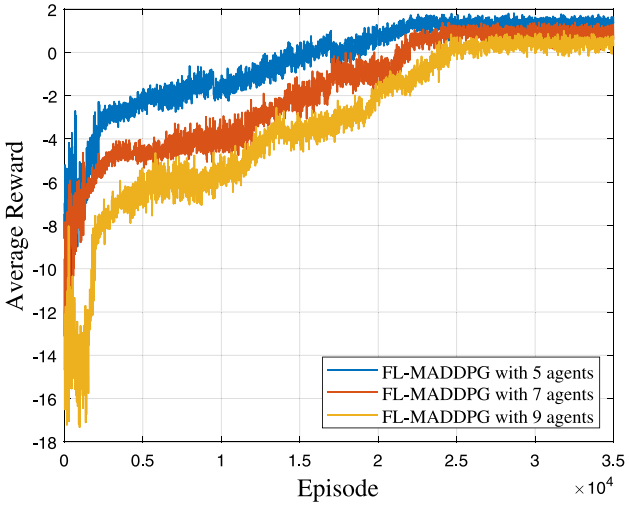

Fig. 7. Convergence of different number of agents.

(1) Convergence Analysis

In the simulation experiments, the agents are trained for a total of 35,000 episodes. The comparison of the convergence performance of our proposed FL-MADDPG algorithm and the DDPG algorithm is shown in Fig. 6.

In the initial 0–3000 episodes, the average rewards of two algorithms fluctuate significantly because the agents in the environment are still in the exploration stage and there is not enough data in the experience replay area for training. As the number of training episodes increases, the average reward of the agents starts to gradually rise. FL-MADDPG performs model aggregation after a certain number of training episodes and is more likely to move away from the local optimal solution, thus the average reward rises more steadily. Finally, their average reward stays in a stable positive reward, but FL-MADDPG has a better convergence performance than MADDPG.

In addition, the convergence performance of the proposed FL-MADDPG algorithm is compared for different numbers of agents. As shown in Fig. 7, the average reward of FL-MADDPG starts to decrease when the number of agents increases, and the model also requires more training episodes to reach convergence. This is because the increase in the number of agents leads to an increase in the dimension

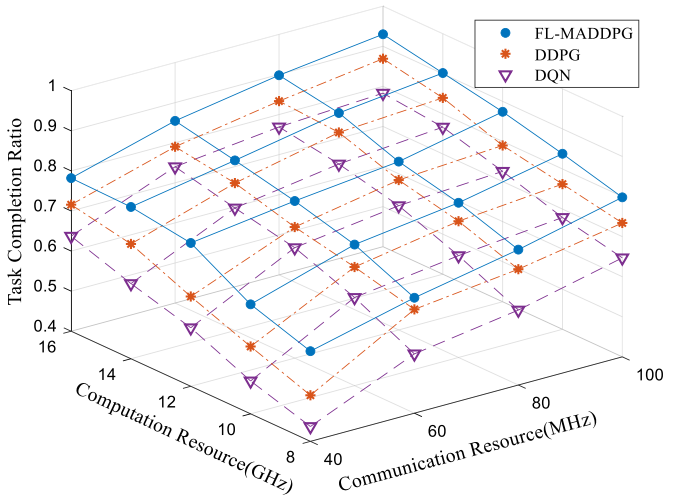

Fig. 8. Task completion rates of different algorithms with different communication and computing resources.

of the action space and state space, which increases the difficulty of training. When the number of agents is increased to 9, it has a greater impact on the model. More episodes are needed to explore the environment in the initial stage of training, and the average reward of the model after convergence decreases significantly. In the comparison experiments, although the increase in the number of agents slow down the convergence of the model, the FL-MADDPG model is still able to maintain its average reward in positive reward after convergence.

(2) Task Delay and Completion Rate Analysis

To evaluate the execution efficiency of FL-MADDPG, the task completion rate of different algorithms under different system resource conditions is analyzed. As shown in Fig. 8, the task completion rate starts to decrease linearly when the communication and computing resources in the system are reduced. In particular, when the communication resources decreased from 60 MHz to 40 MHz, the lack of communication resources caused some tasks could not be offloaded to the MEC server, and the task completion rate was greatly affected. However, in the experiment FL-MADDPG still maintains a high task completion rate compared with DDPG and DQN.

Meanwhile, the task execution efficiency of different algorithms with fixed system resources and increasing number of devices is compared. Fig. 9(a) and (b) show the average completion delay and completion rate of the tasks, respectively. As the increase of devices in the environment leads to the generation of more tasks with different demands and the limited system resources need to be allocated to more devices, the task latency starts to increase and the completion rate starts to decrease. Compared with other algorithms, FL-MADDPG has good performance and relatively flat changes in both task latency and completion rate. This is because FL-MADDPG allocates communication and computing resources to the devices according to the task types and adopts a more appropriate way to process the tasks, which effectively improves the system execution efficiency and reduces resource competition.

In addition, we change the distribution of tasks generated by vehicles under different MEC servers. The task distribution ranges from 0 to 1, indicating the same distribution to different distributions. Fig. 10(a) and (b) show the average completion delay and completion rate of the tasks under different task distribution, respectively. Compared with other algorithms, the introduction of Fl enables FL-MADDPG to maintain high task execution efficiency under different task distributions.

(3) Energy Consumption Analysis

In terms of system energy consumption, we compare the proposed FL-MADDPG algorithm with other existing algorithms. Fig. 11 shows

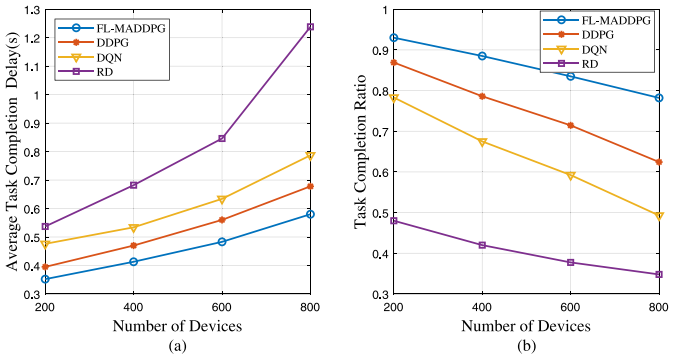

Fig. 9. Execution efficiency of different algorithms.

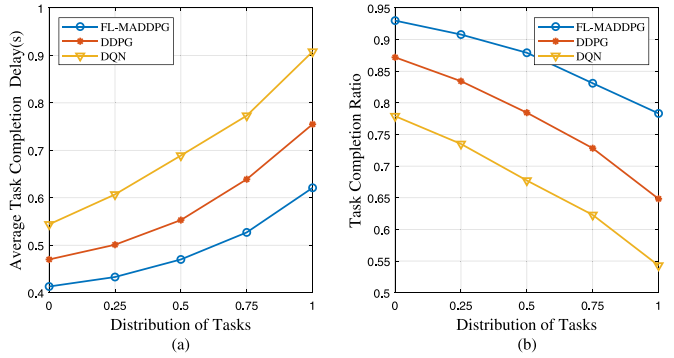

Fig. 10. Execution efficiency of different task distribution.

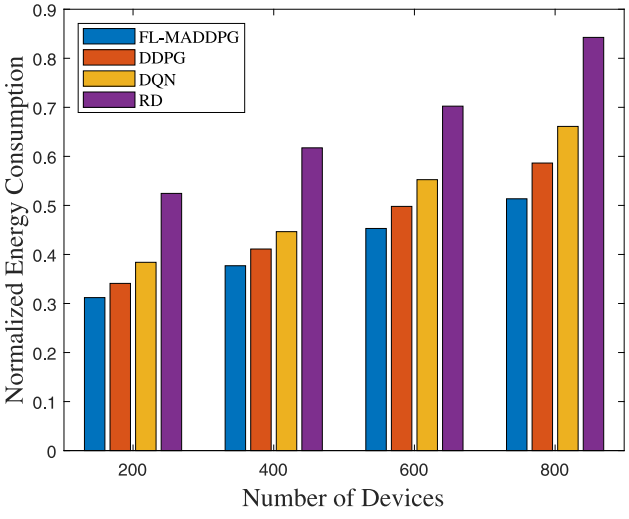

Fig. 11. Normalized energy consumption of different algorithms.

the normalized energy consumption of different algorithms when the number of devices increases. The figure shows that the energy consumption of the task increases as the number of devices increases, while the energy consumption of FL-MADDPG is obviously lower than other algorithms. This is because the reasonable resource allocation method of FL-MADDPG allows the devices to make fuller use of the server-side resources and reduces the unnecessary energy consumption overhead.

6. Limitations and future works

In this section, potential limitations and future improvement directions of this research will be discussed.

6.1. Limitations

Note that our proposed FL-MADDPG still faces some challenges. In terms of privacy, although FL protects data privacy by preventing data from leaving the local device, attackers can still reversely deduce user data through intercepted model gradients (Yang et al., 2023; Yin et al., 2021). In terms of security, the existence of malicious nodes is unavoidable as the number of devices in IoV increases, and some active attack methods (Byzantine attack, Poisoning attack, etc.) will seriously affect the performance of the model (Xia et al., 2021; Makkar et al., 2022; Madni et al., 2023). In addition, the number of agents participating in FL in reality may be more than that set in the experiment, and the difference in computing resources and communication resources is also larger. The efficiency and flexibility of synchronized FL in IoV will be reduced when agents have not only MEC servers but also other devices.

6.2. Future works

For future works, the FL-MADDPG optimization algorithm in IoV is still to be investigated due to the limitations of algorithms. The following are the main directions for future improvement:

(1) Blockchain: The decentralized nature of blockchain matches the application scenarios of MEC. At the same time, blockchain can replace expensive key management of multiple communication protocols and ensure the security of information. The deployment of blockchain in FL-MADDPG will be used to further ensure data security.

(2) Weight mechanism: During the model training process, the weighting mechanism can increase the update weight of agents that are beneficial to the global model, as well as decrease the update weight of malicious agents or even disallow them to participate in model training. How the mechanism is designed will be a concern in our future work.

(3) Asynchronous training: In order to ensure the speed of model convergence, it is necessary to change the weight of the local model at the time of aggregation. Therefore, a time-weighted aggregation method will be further investigated.

7. Conclusion

In this paper, a task offloading and resource allocation strategy based on task types is proposed to reduce the latency and energy consumption during task execution in the IoV under resource-limited conditions. First, we design a hierarchical offloading mechanism for different types of tasks. The task offloading process is analyzed and the communication, computing and system cost models are established. Second, the optimization objective is established based on the above model, and the joint optimization problem of task offloading and resource allocation is modeled as a Markov decision process. An algorithm based on the MADDPG framework is used to maximize the processing energy efficiency of the system. Finally, we combine FL and the MADDPG framework to mitigate the effects caused by non-IID data while protecting users' data privacy. Experiments show that our proposed algorithm performs well in terms of model convergence, energy consumption, and task completion rate.

CRediT authorship contribution statement

Xu Zhao: Writing – original draft. **Yichuan Wu:** Writing – original draft. **Tianhao Zhao:** Writing – review & editing. **Feiyu Wang:** Validation. **Maozhen Li:** Supervision.

Declaration of competing interest

We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of, the manuscript.

Data availability

The authors do not have permission to share data.

Acknowledgments

The work was supported by National Natural Science Foundation of China (71874134), Department of Science and Technology of Shaanxi Province, China(2023QCY-LL-34, 2023QYPY-14), Xi'an Science and Technology Bureau, China (21XJZZ0024, 2023JH-QCYCK-0030). Xian Yang Science and Technology Bureau, China (L2022-ZDYF-GY- 015), Beilin District Science and Technology Bureau (GX2211).

References

Aishwarya, M.R., Mathivanan, G., 2021. AI strategy for stake cloud computing and edge computing: A state of the art survey. In: 2021 5th International Conference on Electronics, Communication and Aerospace Technology. ICECA, IEEE, pp. 920–927.

Attaran, M., 2023. The impact of 5G on the evolution of intelligent automation and industry digitization. J. Ambient Intell. Hum. Comput. 14 (5), 5977–5993.

Bamasag, O., Alghazzawi, D., Alshehri, S., Jamjoom, A., Asghar, M.Z., 2020. Efficient classification of hyperspectral data using deep neural network model. Hum. Cent. Comput. Inf. Sci. 12 (35).

Cheng, Z., Min, M., Gao, Z., Huang, L., 2020. Joint task offloading and resource allocation for mobile edge computing in ultra-dense network. In: GLOBECOM 2020-2020 IEEE Global Communications Conference. IEEE, pp. 1–6.

Chu, H.C., Chang, C.-Y., Chao, H.-C., 2022. Mapping deep learning technologies for mobile networks with the internet of things. Hum. Cent. Comput. Inf. Sci. 12, 59.

Dai, Y., Zhang, K., Maharjan, S., Zhang, Y., 2020. Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond. IEEE Trans. Veh. Technol. 69 (10), 12175–12186.

Elfatih, N.M., Hasan, M.K., Kamal, Z., Gupta, D., Saeed, R.A., Ali, E.S., Hosain, M.S., 2022. Internet of vehicle's resource management in 5G networks using AI technologies: Current status and trends. IET Commun. 16 (5), 400–420.

Gao, H., Huang, W., Liu, T., Yin, Y., Li, Y., 2022. Ppo2: location privacy-oriented task offloading to edge computing using reinforcement learning for intelligent autonomous transport systems. IEEE Trans. Intell. Transp. Syst..

Hazarika, B., Singh, K., Biswas, S., Li, C.-P., 2022. DRL-based resource allocation for computation offloading in IoV networks. IEEE Trans. Ind. Inform. 18 (11), 8027–8038.

He, X., Lu, H., Du, M., Mao, Y., Wang, K., 2020. QoE-based task offloading with deep reinforcement learning in edge-enabled internet of vehicles. IEEE Trans. Intell. Transp. Syst. 22 (4), 2252–2261.

He, Y., Zhao, N., Yin, H., 2017. Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach. IEEE Trans. Veh. Technol. 67 (1), 44–55.

Javed, A.R., Hassan, M.A., Shahzad, F., Ahmed, W., Singh, S., Baker, T., Gadekallu, T.R., 2022. Integration of blockchain technology and federated learning in vehicular (iot) networks: A comprehensive survey. Sensors 22 (12), 4394.

Jiang, H., Dai, X., Xiao, Z., Iyengar, A.K., 2022. Joint task offloading and resource allocation for energy-constrained mobile edge computing. IEEE Trans. Mob. Comput..

Kong, X., Duan, G., Hou, M., Shen, G., Wang, H., Yan, X., Collotta, M., 2022. Deep reinforcement learning-based energy-efficient edge computing for internet of vehicles. IEEE Trans. Ind. Inform. 18 (9), 6308–6316.

Li, X., Cheng, L., Sun, C., Lam, K.-Y., Wang, X., Li, F., 2021a. Federated-learning-empowered collaborative data sharing for vehicular edge networks. IEEE Netw. 35 (3), 116–124.

Li, B., Deng, X., Deng, Y., 2021b. Mobile-edge computing-based delay minimization controller placement in SDN-IoV. Comput. Netw. 193, 108049.

Li, S., Sun, W., Sun, Y., Huo, Y., 2020. Energy-efficient task offloading using dynamic voltage scaling in mobile edge computing. IEEE Trans. Netw. Sci. Eng. 8 (1), 588–598.

Li, C., Zhang, Y., Luo, Y., 2022. A federated learning-based edge caching approach for mobile edge computing-enabled intelligent connected vehicles. IEEE Trans. Intell. Transp. Syst. 24 (3), 3360–3369.

Li, J., Zhao, J., Sun, X., 2021c. Deep reinforcement learning based wireless resource allocation for V2X communications. In: 2021 13th International Conference on Wireless Communications and Signal Processing. WCSP, IEEE, pp. 1–5.

Liang, L., Ye, H., Li, G.Y., 2019. Spectrum sharing in vehicular networks based on multi-agent reinforcement learning. IEEE J. Sel. Areas Commun. 37 (10), 2282–2292.

Lim, W.Y.B., Luong, N.C., Hoang, D.T., Jiao, Y., Liang, Y.-C., Yang, Q., Niyato, D., Miao, C., 2020. Federated learning in mobile edge networks: A comprehensive survey. IEEE Commun. Surv. Tutor. 22 (3), 2031–2063.

Liu, H., Zhang, S., Zhang, P., Zhou, X., Shao, X., Pu, G., Zhang, Y., 2021. Blockchain and federated learning for collaborative intrusion detection in vehicular edge computing. IEEE Trans. Veh. Technol. 70 (6), 6073–6084.

Lowe, R., Wu, Y.I., Tamar, A., Harb, J., Pieter Abbeel, O., Mordatch, I., 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. Adv. Neural Inf. Process. Syst. 30.

Lu, Y., Huang, X., Zhang, K., Maharjan, S., Zhang, Y., 2020. Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles. IEEE Trans. Veh. Technol. 69 (4), 4298–4311.

Madni, H.A., Umer, R.M., Foresti, G.L., 2023. Blockchain-based swarm learning for the mitigation of gradient leakage in federated learning. IEEE Access 11, 16549–16556.

Makkar, A., Kim, T.W., Singh, A.K., Kang, J., Park, J.H., 2022. Secureiiot environment: Federated learning empowered approach for securing iiot from data breach. IEEE Trans. Ind. Inform. 18 (9), 6406–6414.

McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A., 2017. Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics. PMLR, pp. 1273–1282.

Nguyen, D.C., Ding, M., Pham, Q.-V., Pathirana, P.N., Le, L.B., Seneviratne, A., Li, J., Niyato, D., Poor, H.V., 2021. Federated learning meets blockchain in edge computing: Opportunities and challenges. IEEE Internet Things J. 8 (16), 12806–12825.

Nouri, N., Abouei, J., Jaseemuddin, M., Anpalagan, A., 2019. Joint access and resource allocation in ultradense mmWave NOMA networks with mobile edge computing. IEEE Internet Things J. 7 (2), 1531–1547.

Qiu, H., Zhu, K., Luong, N.C., Yi, C., Niyato, D., Kim, D.I., 2022. Applications of auction and mechanism design in edge computing: A survey. IEEE Trans. Cognit. Commun. Netw. 8 (2), 1034–1058.

Rao, S.K., Prasad, R., 2018. Impact of 5G technologies on industry 4.0. Wireless Pers. Commun. 100, 145–159.

Salim, M.M., Park, J.H., 2022. Federated learning-based secure electronic health record sharing scheme in medical informatics. IEEE J. Biomed. Health Inf. 27 (2), 617–624.

Samarakoon, S., Bennis, M., Saad, W., Debbah, M., 2019. Distributed federated learning for ultra-reliable low-latency vehicular communications. IEEE Trans. Commun. 68 (2), 1146–1159.

Singh, A., Satapathy, S.C., Roy, A., Gutub, A., 2022. Ai-based mobile edge computing for iot: Applications, challenges, and future scope. Arab. J. Sci. Eng. 1–31.

Song, Q., Lei, S., Sun, W., Zhang, Y., 2021. Adaptive federated learning for digital twin driven industrial internet of things. In: 2021 IEEE Wireless Communications and Networking Conference. WCNC, IEEE, pp. 1–6.

Suh, K., Kim, S., Ahn, Y., Kim, S., Ju, H., Shim, B., 2022. Deep reinforcement learning-based network slicing for beyond 5G. IEEE Access 10, 7384–7395.

Wang, S., Tuor, T., Salonidis, T., Leung, K.K., Makaya, C., He, T., Chan, K., 2019. Adaptive federated learning in resource constrained edge computing systems. IEEE J. Sel. Areas Commun. 37 (6), 1205–1221.

Wang, L., Wang, K., Pan, C., Xu, W., Aslam, N., Hanzo, L., 2020. Multi-agent deep reinforcement learning-based trajectory planning for multi-UAV assisted mobile edge computing. IEEE Trans. Cognit. Commun. Netw. 7 (1), 73–84.

Wang, G., Xu, F., 2020. Regional intelligent resource allocation in mobile edge computing based vehicular network. IEEE Access 8, 7173–7182.

Wu, Y., Ni, K., Zhang, C., Qian, L.P., Tsang, D.H., 2018a. NOMA-assisted multi-access mobile edge computing: A joint optimization of computation offloading and time allocation. IEEE Trans. Veh. Technol. 67 (12), 12244–12258.

Wu, H., Sun, Y., Wolter, K., 2018b. Energy-efficient decision making for mobile cloud offloading. IEEE Trans. Cloud Comput. 8 (2), 570–584.

Xia, Q., Ye, W., Tao, Z., Wu, J., Li, Q., 2021. A survey of federated learning for edge computing: Research problems and solutions. High Confid. Comput. 1 (1), 100008.

Xu, X., Li, H., Xu, W., Liu, Z., Yao, L., Dai, F., 2021. Artificial intelligence for edge service optimization in internet of vehicles: A survey. Tsinghua Sci. Technol. 27 (2), 270–287.

Yang, H., Ge, M., Xue, D., Xiang, K., Li, H., Lu, R., 2023. Gradient leakage attacks in federated learning: Research frontiers, taxonomy and future directions. IEEE Netw..

Yang, Q., Liu, Y., Chen, T., Tong, Y., 2019. Federated machine learning: Concept and applications. ACM Trans. Intell. Syst. Technol. 10 (2), 1–19.

Yang, C., Xu, X., Zhou, X., Qi, L., 2022. Deep Q network–driven task offloading for efficient multimedia data analysis in edge computing–assisted IoV. ACM Trans. Multimed. Comput. Commun. Appl. (TOMM) 18 (2s), 1–24.

Yi, C., Cai, J., Su, Z., 2019. A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications. IEEE Trans. Mob. Comput. 19 (1), 29–43.

Yin, H., Mallya, A., Vahdat, A., Alvarez, J.M., Kautz, J., Molchanov, P., 2021. See through gradients: Image batch recovery via gradinversion. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16337–16346.

Yu, S., Chen, X., Zhou, Z., Gong, X., Wu, D., 2020. When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5G ultradense network. IEEE Internet Things J. 8 (4), 2238–2251.

Zhang, Q., Gui, L., Hou, F., Chen, J., Zhu, S., Tian, F., 2020. Dynamic task offloading and resource allocation for mobile-edge computing in dense cloud RAN. IEEE Internet Things J. 7 (4), 3282–3299.

Zhang, Q., Wang, Y., Li, H., Hou, S., Song, Z., 2023. Resource allocation for energy efficient STAR-RIS aided MEC systems. IEEE Wireless Commun. Lett. 12 (4), 610–614.

Zhao, X., Huang, G., Gao, L., Li, M., Gao, Q., 2021a. Low load DIDS task scheduling based on Q-learning in edge computing environment. J. Netw. Comput. Appl. 188, 103095.

Zhao, X., Huang, G., Jiang, J., Gao, L., Li, M., 2021b. Research on lightweight anomaly detection of multimedia traffic in edge computing. Comput. Secur. 111, 102463.

Zhao, X., Huang, G., Jiang, J., Gao, L., Li, M., 2022. Task offloading of cooperative intrusion detection system based on deep Q network in mobile edge computing. Expert Syst. Appl. 206, 117860.

Zhao, X., Liu, M., Li, M., 2023. Task offloading strategy and scheduling optimization for internet of vehicles based on deep reinforcement learning. Ad Hoc Netw. 147, 103193.

Zhao, M., Yu, J.-J., Li, W.-T., Liu, D., Yao, S., Feng, W., She, C., Quek, T.Q., 2021c. Energy-aware task offloading and resource allocation for time-sensitive services in mobile edge computing systems. IEEE Trans. Veh. Technol. 70 (10), 10925–10940.

Zhou, Z., Yu, H., Xu, C., Chang, Z., Mumtaz, S., Rodriguez, J., 2018. BEGIN: Big data enabled energy-efficient vehicular edge computing. IEEE Commun. Mag. 56 (12), 82–89.

Xu Zhao is a professor in the School of Electronic and Information, Xi'an Polytechnic University, and Shaanxi, China. In addition, he is also an academic visitor to a Brunel University London. He received the Ph.D. degree from Xi'an University of Architecture and Technology, Xi'an City, and Shaanxi Province, China. His research interest is Edge Computing and Cyber Security.

Yichuan Wu received his Bachelor's degree from the Changshu Institute of Technology in 2021. He is pursuing his Master's degree in the School of Computer Science, Xi'an Polytechnic University. His research interests include internet of vehicles and edge computing.

Tianhao Zhao is a graduate student and is currently working towards a master's degree. His research interests are drone and edge computing

Feiyu Wang is a graduate student and is currently working towards a master's degree. Her research interests are edge computing and task offloading

Maozhen Li is currently a professor of College of Engineering, Design and Physical Sciences Software Engineering at Brunel University London. He received the Ph.D. degree from Chinese Academy of Sciences in 1997. As a research assistant, he completed the postdoctoral research in Cardiff University in 2002. Prof. Li is the fellow of the British Computer Society and the Member of IET, IEEE. His research includes High Performance Computing, Knowledge and Data Engineering.