Efficiency and Fairness Oriented Dynamic Task Offloading in Internet of Vehicles

Chen Chen, *Senior Member, IEEE*, Haofei Li, Huan Li, Rufei Fu, Yangyang Liu, and Shaohua Wan, *Senior Member, IEEE*

***Abstract*—The rapid development of the Internet of Vehicles (IoV) leads to various on-board applications including delay-sensitive and compute-intensive applications. However, vehicles with limited computing resources cannot meet the requirements of the delay for these applications. The application of edge computing on the IoV can deal with the above problems. However, most existing edge computing offloading work only considers static scenarios. These schemes are difficult to adapt to dynamic processes under continuous time. To solve this problem, in this article, we propose an end-edge-cloud architecture of vehicles for task computation offloading, where considers three task computing methods. For the dynamically changing environment in the IoV, we utilize an Asynchronous Advantage Actor-Critic (A3C) based computation offloading algorithm to solve the problem and seek optimal offloading decisions. While considering efficiency and fairness, our solution enables vehicle users to obtain computing services from edge servers in real-time and conveniently. The reward function in this paper contains a relative efficiency factor and a relative fairness factor, which are indicators to measure the efficiency of task completion as well as the relative fairness between vehicles. These indicators can be adjusted to obtain more targeted offloading decisions in different scenarios. The experimental results show that the scheme proposed in this paper can achieve good convergence performance and adapt to the dynamic offloading scene. Compared with the other scheme, our scheme can achieve better performance in terms of efficiency of task completion and fairness among tasks. The numerical results show that compared to the Deep Q Network-based scheme, the local computing scheme, the edge computing scheme, and the random scheme, our algorithm can save 7.2%, 18.5%, 41.8%, and 33.5% of the mean for average delay, respectively.**

***Index Terms*—Internet of Vehicles, mobile edge computing, reinforcement learning.**

Manuscript received 6 October 2021; revised 16 February 2022 and 28 March 2022; accepted 6 April 2022. Date of publication 15 April 2022; date of current version 19 August 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFE0126000; in part by the National Natural Science Foundation of China under Grant 62072360, Grant 62001357, Grant 62172438, and Grant 61901367; in part by the Key Research and Development Plan of Shaanxi Province under Grant 2021ZDLGY02-09 and Grant 2020JQ-844; in part by the Natural Science Foundation of Guangdong Province of China under Grant 2022A1515010988; in part by the Xi'an Science and Technology Plan under Grant 20RGZN0005; and in part by the Xi'an Key Laboratory of Mobile Edge Computing and Security under Grant 201805052-ZD3CG36. *(Corresponding author: Shaohua Wan.)*

Chen Chen, Haofei Li, Huan Li, and Rufei Fu are with the State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an 710071, China (e-mail: cc2000@mail.xidian.edu.cn; lihaofei@stu.xidian.edu.cn; huanli_Yoly@163.com; fstickman@163.com).

Yangyang Liu is with the Department of Intelligent Connected Vehicles, China Automotive Technology & Research Center Company Ltd., Tianjin 300300, China (e-mail: liuyangyang2017@catarc.ac.cn).

Shaohua Wan is with the Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China, Shenzhen 518110, China (e-mail: shaohua.wan@ieee.org).

Digital Object Identifier 10.1109/TGCN.2022.3167643

I. Introduction

With the continuous development of Internet of Vehicles(IoV) devices and technologies, the functions of intelligent cars in the future will be more complete and diversified. Advanced communication technology and massive data captured by sensors provide drivers with services such as autonomous driving, real-time road condition sharing, video analysis, automatic navigation, and parking information acquisition in corresponding road sections [1]. The current 5G communication technology and traditional car networking architecture are no longer sufficient to meet the needs of future intelligent transportation services. The wide variety of services and the influx of data processing demands put enormous pressure on local devices. Although today's communication networks can provide a maximum data rate of 20GB per second, this is not sufficient for the ultra-low latency requirements of less than one millisecond for IoV [2]. At the same time, the task calculation consumes the energy of cars, such as gasoline or electricity. Therefore, green-oriented communication needs to find a more efficient way to complete the assignment and calculation of tasks.

With the rapid growth of the number of cars and the increase in the proportion of electric vehicles globally, the IoV has ushered in a new research direction. Mobile cloud computing (MCC) was recently seen as a viable technology to reduce the computational pressure on local devices and extend the useful life of local devices. This approach offloads the computationally intensive tasks of the vehicle to a cloud center for computation. However, due to the limitation of distance, IoV deployments that rely on cloud computing to complete tasks cannot provide rapid response for many time-sensitive applications, such as emergency response and traffic monitoring [3], [4]. When a large number of tasks flood in, the long-distance and load pressure will lead to transmission delay and the lag of information, which can not be ignored [5].

Given above bottlenecks, as an emerging computing paradigm, Mobile Edge Computing (MEC) can meet the low-latency communication requirements of vehicles to a certain extent [6], [7], and assist in expanding the computing capacity provided by the existing IoV. This aligns with the vision of high bandwidth, real-time interaction, and massive connectivity for next-generation networks. Therefore, Roadside Units (RSUs) equipped with MEC servers can contribute to a more effective solution to the computing offloading problem of the IoV in the future [8]. However, facing complex vehicle networks, the computation offloading problem is difficult to solve. Deep Learning (DL) algorithms have brought opportunities to many traditional fields and guided researchers in new directions. Intelligent Connected Vehicles (ICVs), wise information technology of medical [9] and smart water conservancy [10] all show great promise when integrated with DL. Some Deep Reinforcement Learning (DRL) algorithms have been used to demonstrate their efficiency in solving decision problems with high-dimensional state and action spaces. Agents can interact with the environment in real-time to provide dynamic resource management solutions, especially for the issue of continuous state and action spaces [11], [12].

The emergence of advanced technology has brought more possibilities for developing the IoV, and it is necessary to propose new measures to the traditional resource management scheme. The reasonable allocation of computing resources and the low latency characteristics make the computing offloading design of the IoV popular in the vehicle network. At the same time, most of the existing research on the end-edge-cloud architecture focuses more on the design of the architecture and optimization goals, and few studies consider the evaluation indicators of dynamic task arrival environment and computing offloading [13]. For the above situation, we study the fairness and efficiency of and computation offloading in the dynamic environment.

In this work, we propose an end-edge-cloud architecture to assist vehicles for task computation offloading, where task requests are generated by vehicle users located within the coverage of the RSU. The calculation of vehicle tasks includes three methods. For the dynamically changing environment in the IoV, we utilize a DRL method based on the A3C algorithm to seek optimal offloading decisions. The contribution and novelty of our work are mainly reflected in the following aspects:

- *System Structure:* We propose an end-edge-cloud three-layer architecture to assist vehicles in task offloading computation. Offloading requests are generated by vehicle users located within the coverage area of the RSU. The calculation of vehicle tasks includes three methods, namely local computing, edge computing, and cloud computing.
- *Evaluation System:* The dynamic arrival characteristics and performance evaluation method of tasks in the IoV are considered in the model. A state matrix of idle resources is designed considering the dynamic arrival of tasks and dynamic changes of resources. The relative efficiency factor and the relative fairness factor are proposed to measure the efficiency and fairness of task offloading computation.
- *Algorithm Design:* A joint optimization objective of computation offloading and resource allocation is formulated. It can ensure that the resource allocation scheme complies with constraints such as time delay. A DRL method based on the A3C algorithm is proposed to seek the optimal offloading decision and resource allocation scheme.
- *Numerical Evaluation:* Numerical results are provided to verify the performance of the A3C-based algorithm proposed in this paper. The results show that our proposed algorithm achieves certain advantages in terms of average delay compared with other comparison schemes. At the same time, we also verified the effectiveness of our proposed relative efficiency factor and relative fairness factor in the IoV environment.

The rest of this paper is organized as follows. Section II introduces the research progress of the end-edge-cloud architecture, and then discusses the work of computing offloading using DRL. Section III presents the system model and hypothesis of this paper. In Section IV, we analyze the completion delay for different offloading methods respectively, and give the definition of relative efficiency factor and relative fairness factor. Section V introduces the specific application method of the A3C model, including the design of state, action, reward. Then, we describe the network structure of A3C. Section VI describes experiments and analysis of results. Section VII summarizes the research work and prospects of this paper.

II. Related Work

In this section, we first introduce the research progress of the end-edge-cloud architecture, and then discuss the work of computation offloading using DRL.

A. End-Edge-Cloud Architecture

The upcoming 6G technology makes the end-edge-cloud three-layer architecture a new research trend [13]–[15]. In IoV, the end-edge-cloud three-layer architecture has become a potential research direction. Reference [16] proposed a cache-based dense IoV with terahertz frequency links. It addresses the resource allocation problem of power optimization and subband allocation to maximize energy efficiency. In [1], a two-layer federated learning model is proposed to guarante data privacy and reduce communication overhead for typical distributed end-edge-cloud architecture. A collaborative learning-based routing scheme for multi-access vehicular edge computing environments is proposed in [17], which employs a reinforcement learning algorithm based on end-edge-cloud collaboration to find routes proactively with low communication overhead. Reference [18] proposed an end-edge-cloud computing framework and implemented two-stage reinforcement learning to obtain the optimal policy for vehicle control. Many researchers also studied the optimization problem of computing offloading strategy under the end-edge-cloud architecture. Reference [5] designs an optimal computing offloading and resource allocation strategy for the end-edge-cloud architecture. It formulates the joint computing offloading and resource allocation problem as a Markov Decision Process (MDP), and proposes a new DRL algorithm to minimize the system energy consumption. Reference [19] proposed a low-complexity hierarchical heuristic approach for server selection, and a closed-form approach based on the Cauchy-Schwards inequality to determine resource allocation efficiently. Reference [20] constructed a potential game for computation offloading in end-edge-cloud environments, where each client selfishly minimizes its payoff.

B. Computation Offloading

The reasonable allocation of computing resources and the characteristics of low latency make the computation offloading design of the IoV a popular research direction. So far, researchers have put a lot of effort into the problem of vehicle computing offloading. Initially, the offloading of vehicle tasks were mainly solved by traditional algorithms or mathematical methods, such as game theory, genetic algorithm, and matching theory. In [21], it proposed a game-based algorithm for computing the offloading of vehicle edge networks to reduce vehicles' computational cost significantly. In [22], a multi-level offloading scheme was proposed based on Stackelberg game theory to maximize the net benefit by utilizing both the vehicle and the computing server to satisfy the delay constraint of the computing task. Reference [23] have designed a bi-directional dynamic joint task offloading and resource allocation algorithm based on the vehicle network. It jointly optimize the wireless resource allocation sum of the edge servers through game theory to make an offloading decision for vehicle computing tasks. In [24], the concept of Vehicle Cloud (VC) is proposed, which offloads tasks to VC. It then uses integer coding Genetic Algorithm (GA) to schedule VC resources. This method significantly improves the utilization of computing resources and ensures low latency of the system.

However, as the number of devices and services increases, the problem becomes complicated. Making decisions in dynamic environments is a big challenge for traditional computation offloading algorithms. The development of reinforcement learning and the emergence of intelligent edge devices bring more possibilities for developing the IoV. It can make decisions for complex dynamic problems with an appropriate reward mechanism based on the state of the environment.

In [25], a multi-agent DRL-based computation offloading scheme is proposed. In this scheme, the uncertainty of the multi-vehicle environment is considered so that vehicles can make offloading decisions to achieve optimal long-term rewards and minimize the total task processing delay in the long term. In [26], a DRL method is proposed for latency-aware and energy-efficient offloading in dynamic MEC networks with multiple users and multiple MEC servers. Unlike previous work, the model solve the offloading problem in an end-to-end manner using DRL. An optimization problem is formulated in [27] to joint minimize total latency and energy consumption, which is solved by a DRL-based algorithm. This work differs from other work in that it utilizes the spare resources of mobile devices such as desktop computers as one of the available computing resources. Reference [28] studies the problem of stochastic computation offloading in virtual edge computing systems, in which a algorithm based on dual deep Q-networks (Duel DQN) are proposed to maximize long-term utility. In [29], a A3C-based algorithm is proposed to maximize the weighted sum of computation rate and throughput in blockchain-enabled MEC systems. Reference [30] proposed an adaptive computation offloading paradigm in MEC-enabled vehicle systems, where a deep deterministic policy gradient-based algorithm is used to minimize transmission delay, energy consumption, and cost of bandwidth resources. An algorithm named LyDROO that combines the advantages of Lyapunov optimization and DRL (DRL) is proposed in [31]. It first applies Lyapunov optimization to decouple multi-stage stochastic MINLP into smaller-sized MINLP subproblems, then solve it by DRL.

In addition, similar to our work, some work focus on exploring other technical details to promote further the development of the IoV based on edge computing. Reference [32] focuses more on the optimization of the number and location of edge servers, and has developed a dynamic ES placement (DEP) method, which uses non-dominated sorting genetic algorithm III (NSGA-III) to achieve a layout scheme with better performance. Reference [33] combines the edge computing technology of the IoV with digital twins, and proposes a service offloading (SOL) method with deep reinforcement learning. Utilizing deep Q-networks (DQNs), this SOL algorithms can adapt to different environments and make efficient offloading decisions.

For the research on the network structure of computation offloading, most of the existing research focuses on finding solutions with the least energy consumption or the lowest processing delay as the goal. To the best of the authors' knowledge, few studies focus on designing indicators to evaluate the effectiveness and fairness of computation resource allocation.

Aiming at the above deficiencies in existing research, we design a dynamic task offloading scheme for vehicle network. In this scheme, efficiency and fairness were considered, and the offloading selection was optimized by deep reinforcement learning.

III. System Model

In this section, we first introduce our end-edge-cloud computation offloading model. Then we explain the dynamic resource state we proposed. The key notations are listed in Table I.

In this paper, the task information generated by vehicles should be transmitted to the MEC or cloud server. The decisions of computation offloading and allocates resources are made by the MEC server. If the MEC server agrees to process a task, the task will be offloaded to the MEC server and assigned computation resources. The MEC server does not assign resources to a task when it is determined that the task can be handled by local computation.

A. System Model Design

As shown in Figure 1, the scenario is a section of road with a number of RSUs configured at specified interval distances. The system model is a three-layer network model. The bottom layer is the vehicle and road layer, which consists of different types of vehicles and road environments. The vehicle randomly generates tasks, and requests resources from the MEC server within the communication range. The second layer is

TABLE I
KEY NOTATIONS

Notations	Description
D_j	Size of jth task
C_j	Computing power required by jth task
$T_j^{\max}$	Maximum delay limit for completing jth task
τ	Slot size
k	Index of k slot
S_t	Dynamic computing resource state
R_{V2I}^j	Data transfer rate of task j
z_j	Allocated bandwidth proportion for the task j
B_{V2I}	System bandwidth for the communication
N_0	White Gaussian noise of the channel
h_j	Channel gain to transfer the jth task
P_j	Transmitted power for the vehicle to send the task j
$punish$	Penalty for the cloud computing

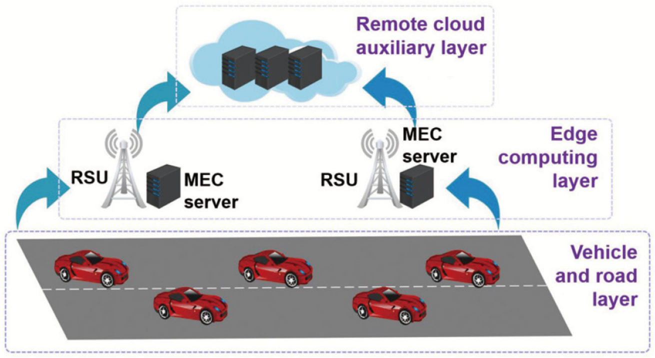


Fig. 1. Three layer system model.

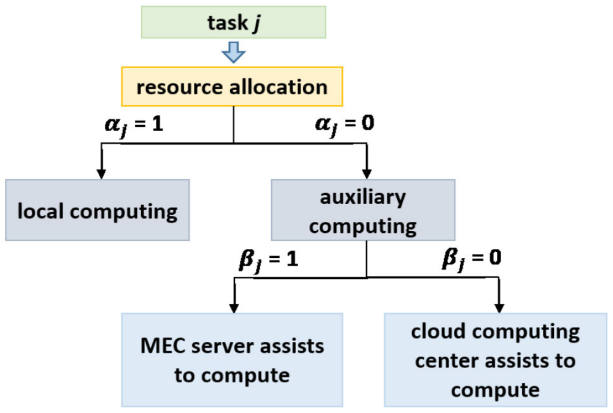


Fig. 2. The process of task offloading.

the edge computing layer. It consists of RSUs with computing, caching, and communication functions [34]. Each RSU will be equipped with a MEC server. The tasks generated by the vehicle can be transmitted to the edge computing layer via a wireless channel. The edge computing layer makes computation offloading decisions and allocates resources. The top layer is the remote cloud auxiliary layer, which is a cluster of multiple physical hosts. As a resource sharing warehouse, it provides a global task scheduling scheme. When the resources of the edge computing layer are insufficient, it serves as a standby resource to complete the tasks of the newly arrived edge computing layer, etc.

To describe the optimization model more accurately, the following assumptions are first stated:

(1) Each vehicle generates tasks independently, and each vehicle can only complete one task at a time. When the current task generated by the vehicle needs to be completed locally, the next task needs to wait until the current task has been completed. Queuing problems for locally computation when a vehicle generate multiple tasks are not considered in this article.

(2) It assumes cn MEC servers and C CPUs per server. They can do tasks independently and in parallel. Each CPU has the same computing power. When a task is offloaded to a MEC server for computation, the entire CPU resources are fully occupied until this task is completed, and a task cannot occupy computing resources of multiple CPUs at the same time.

(3) This paper considers indivisible tasks. The whole task can only choose one of the local CPU, edge CPU, or cloud CPU to compute.

(4) When multiple tasks reach the edge computing layer, the task that arrives first has higher priority and can use CPU resources first.

We assumes that M tasks arrive within τ moments. These tasks are generated by separate vehicles. Each task is represented by a triple $\{D_j, C_j, T_j^{\max}\}$. The size of each task is D_j. The computing power required is C_j cycles. The maximum delay limit for completing a task is $T_j^{\max}$, which depends on how long the vehicle and RSU can communicate. As shown in Figure 2, the completion of tasks mainly includes two ways: local computing and auxiliary computing. The auxiliary computing includes 2 cases. One is the computation under the MEC server's assistance, and the other is computation at the remote cloud server. The decisions of resource allocation and computation offloading are made by the edge computing layer. If the edge computing layer does not allocate any resource to the task, it can only be computed locally. If the edge computing layer hopes to provide resources to the task but does not meet the corresponding conditions, the task will be completed in cloud computing center. When the edge computing layer can provide resources to the task and meet the delay limit, the task completes the computation on the MEC server.

It assumes that the local computing power of the vehicle generating the task is f_j cycles/s. The local calculation delay is:

$$local\_delay_j = \frac{C_j}{f_j}. \quad (1)$$

When a task is assigned the resources of a MEC server, it can be offloaded to the edge computing layer.

B. Dynamic Computing Resource State

Suppose that the allocated resource is one of the K time slots, $k \in \{0, 1, 2, \ldots, K-1\}$, which is starting from the current time *t*. The slot size is τ. A period of $[t + k\tau, t + (k+1)\tau]$ represents the *k*th time slot. The dynamic computing resource state of a CPU in a time slot starting at *t* can be represented by S_t:

$$S_t = \begin{bmatrix} cpu_{0,0} & cpu_{0,1} & \cdots & cpu_{0,C-1} \\ cpu_{1,0} & cpu_{1,1} & \cdots & cpu_{1,C-1} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ cpu_{K-1,0} & cpu_{K-1,1} & \cdots & cpu_{K-1,C-1} \end{bmatrix}, \quad (2)$$

where $cpu_{k,i}$ represents the remaining available time for the *i*th CPU in the *k*th slot. It has a value range of $[0, \tau]$.

C. Task Delay

The calculation delay mainly consists of the following parts:

Transmission delay: If a task is determined to be offloaded to the MEC server, the task data needs to be transferred to the RSU first. For the multiple tasks which need to be transmitted, the Frequency Division Multiple Access (FDMA) access modes is adopted in this paper. It means that each task is allocated a fixed portion of bandwidth resources. This part of bandwidth can be monopolized by the task until its transmission is completed [35]. The data transfer rate of task *j* is expressed as:

$$R^j_{V2I} = z_j B_{V2I} \log_2\left(1 + \frac{P_j |h_j|^2}{z_j B_{V2I} N_0}\right), \quad (3)$$

where z_j represents the allocated bandwidth proportion for the task *j* and z_j in [0, 1], $\sum_{j=0}^{M-1} z_j = 1$. B_{V2I} is the system bandwidth for the communication between the vehicle and the mobile edge. N_0 is the white Gaussian noise of the channel. h_j is the channel gain. P_j is the transmitted power for the vehicle to send the task *j*. Therefore, the transmission delay of each task is:

$$transmit\_delay_j = \frac{D_j}{R^j_{V2I}}. \quad (4)$$

Waiting delay: We assumes that the task is allocated computing resources of the *k*th slot that starting from the current slot. This task needs to wait until the *k*th time slot. The waiting delay is given by:

$$slot\_wait^k_j = k\tau. \quad (5)$$

Queuing delay: Each newly arrived task will be allocated computing resources in the corresponding time slot. We assume that each MEC server is a parallel computing system with multiple CPUs. When there is a CPU in the system and its remaining available time is greater than the maximum completion delay limit of the task, the task can be offloaded to the MEC server for calculation. Since different tasks require different computing time, and the arrival time of tasks is random, the resource occupancy of each CPU is different. Each task is assigned to the CPU with the longest remaining available time. When the remaining available time of multiple CPUs is the same, the MEC server makes a random assignment selection for the task. Therefore, the queuing delay can be expressed as:

$$slot\_innerwait^k_j = \tau - \max\{cpu_{k,0}, cpu_{k,1}, \ldots, cpu_{k,c-1}\}$$
$$s.t.\ max\{cpu_{k,0}, cpu_{k,1}, \ldots, cpu_{k,c-1}\} \geq T^{\max}_j. \quad (6)$$

Therefore, when the task can be calculated and offloaded in the edge calculation layer to the *k*th slot from the current time slot, the total delay required is:

$$\begin{aligned} edge\_delay^k_j &= transmit\_delay^k_j + slot\_wait^k_j \\ &\quad + slot\_innerwait^k_j. \end{aligned} \quad (7)$$

If the task is assigned to one CPU in *k*th time slot , whose remaining available time less than the maximum completion delay limit of the task, this task can only be transferred to the cloud for computing. If the original elements of the task are directly offloaded to the cloud computing center, the communication bandwidth is wasted while the potentially sensitive information of the vehicle is also exposed. Therefore, we believe that security considerations are necessary and compliant with the General Data Protection Regulation (GDPR) in when cloud-assisted computing is used. So we add an additional limitation to the cloud computing case. We replace the sum of task transmission and computation time in cloud computing with a penalty term *punish*. Therefore, the delay of completing the task on the cloud auxiliary layer is:

$$\begin{aligned} &cloud\_delay^k_j = transmit\_delay_j + punish \\ &s.t.\ max\{cpu_{k,0}, cpu_{k,1}, \ldots, cpu_{k,c-1}\} < T^{\max}_j. \end{aligned} \quad (8)$$

IV. Problem Formulation

In this section, the problem is formulated, and the indicators proposed in this paper that comprehensively consider offloading efficiency and fairness are introduced.

The actual delay in completing the task *j* is given by:

$$\begin{aligned} &actual\_delay_j = \alpha_j local\_delay_j + (1 - \alpha_j) \\ &\quad \times \left[\beta_j edge\_delay^k_j + (1 - \beta_j) cloud\_delay^k_j\right] \\ &s.t.\ C1\colon \forall \alpha_j \in \{0, 1\} \\ &\quad C2\colon \beta_j = \begin{cases} 1, \max\{cpu_{k,0}, \ldots, cpu_{k,c-1}\} \geq T^{\max}_j \\ 0, \max\{cpu_{k,0}, \ldots, cpu_{k,c-1}\} < T^{\max}_j \end{cases} \\ &\quad C3\colon \forall j \in \{0, 1, 2, \ldots, M-1\} \\ &\quad C4\colon k \in \{0, 1, 2, \ldots, K-1\}. \end{aligned} \quad (9)$$

Here the value of α_j is 1 indicates the task is locally computed. The value of α_j is 0 indicates that the task is completed computed by auxiliary computing method. Based on the CPU state and the maximum delay limit for task completion, the value of β_j is 1 indicates that the task is completed at the MEC server. The value of β_j is 0 indicates that the task cannot be completed by edge computing within the maximum completion delay limit. It requires the cloud server to calculate.

After getting the actual delay for completing each task in the current moment τ, the average delay for all arriving tasks can be calculated. The smaller the average delay is, the more reasonable the resource allocation of the edge server is. At the same time, the system can serve more tasks, and the task processing efficiency is higher. The average delay is calculated as follows:

$$delay\_avg_t = \frac{1}{M} \sum_{j=0}^{M-1} actual\_delay_j. \quad (10)$$

This paper considers the real-time dynamic IoV environment, so the task information sent to the MEC server is not same in different time. For convenience of comparison, we use local completion delay as a benchmark. The relative task completion delay $delay\_reduce\_delay_t$ is defined as the difference between the actual completion delay and the local completion delay, which is given by:

$$delay\_reduce\_delay_j = local\_delay_j - actual\_delay_j. \quad (11)$$

The larger the value is, the better task offloading decision will be.

Relative efficiency factor: The proposed relative efficiency factor can be expressed as average delay $delay\_reduce\_avg_t$ of all tasks arrived within the time *t*. It is shown in formula (12), which is the mean value of all tasks relative to task completion delay $delay\_reduce\_delay_t$. Therefore, the relative efficiency factor can be described as:

$$delay\_reduce\_avg_t = \frac{1}{M}\sum_{j=0}^{M-1} delay\_reduce\_delay_j. \quad (12)$$

It should be noted that the larger this value is, the more time will be saved than the local computing case.

Relative fairness factor: The fairness between tasks is measured by the variance of the time delay reduced by the actual completion delay of all tasks and the average delay. It can be described as the variance of $delay\_reduce\_delay_t$ relative to $delay\_reduce\_avg_t$ for all tasks at time t. The variance is the relative fairness factor, which is denoted by $delay\_reduce\_std_t$:

$$delay\_reduce\_std_t = \frac{1}{M}\sum_{j=0}^{M-1}\big(delay\_reduce\_delay_j - delay\_reduce\_avg_t\big)^2. \quad (13)$$

The smaller the relative fairness factor is, the closer the relative task completion delay is to the relative efficiency factor. Tasks can compete more fairly for MEC server resources.

Therefore, the optimization goal is that it ensures the average delay of all tasks arriving within a period is small. It gives a high degree of fairness between tasks. In period *T*, the more significant the task's average reduction in time delay relative to entirely local computation, the larger the relative efficiency factor. It means the larger the reward value is. The smaller the variance of difference of delay, the smaller the relative fairness factor and the larger the reward value.w_1 and w_2 are the weights of relative efficiency factor and relative fairness factor. When w_1 is large, it focuses on ensuring the task performance efficiency of the system. w_2 is used to ensure fairness of system tasks.

$$\max \sum_{t=0}^{T} w_1 delay\_reduce\_avg_t - w_2 delay\_reduce\_std_t$$
$$s.t. \quad C1{:}\ w_1 + w_2 = 1, w_1, w_2 \in [0,1]. \quad (14)$$

V. A Dynamic Computing Offloading Scheme Based on A3C

The optimization model was established in the previous section. In this section, we introduce the process of using DRL method to interact with complex environments of IoV to make computational offloading decisions. The reason for using the A3C model is that the state space in this model is continuous with large search space. Studies in paper [36] show that common RL methods cannot adapt to a highly random environment or obtain accurate state value. Experiential playback method limit the accuracy of newly arrived tasks to calculate offload decisions. It can only sample and update with data generated by the old policy. And these experiences require additional memory resources on the MEC server.

A. Deep Reinforcement Learning

DRL deals with problems by placing learning agents in the environment to achieve goals. The agents execution actions a_t according to the strategy π, which maps states s_t to actions a_t. It receives the next status s_{t+1} and an immediate reward r_t. The immediate reward is a feedback signal that indicates how well the learning agent performs the action and how far it is from the optimal strategy (expressed as π^*). The process that agent interacts with the environment continues until the termination reached. Starting with the time step *t*, the expected total discount reward can be expressed as $R_t = \sum_{k=0}^{\infty}\gamma^k r_{t+k}$, where the discount factor $\gamma \in (0,1]$İn reinforcement learning, there is a concept called action value function that used to predict the future rewards of the action.

B. Markov Decision Model

The Markov decision process (MDP) can be represented as$\langle S, A, P(S_{t+1} \mid S_t, A_t), R, \gamma\rangle$ by a five-tuple. S and A are state set and action set respectively. $P(S_{t+1} \mid S_t, A_t)$ is the state $S_t \in S$ at moment *t* takes action with a certain probability $A_t \in A$ and then the state of moves to $S_{t+1} \in S$ at the next moment $t+1$. $R(S_t, A_t)$ is the instant reward. $\gamma \in [0,1]$ is the discount factor. The diminishing importance of the present reward to the future is reflected. The goal of MDP is to find a strategy. Determine the selected action $A_t = \pi(S_t)$ under state S_t to maximize the value function. The value function can usually be defined as the Bellman equation for accumulating rewards. When $P(S_{t+1} \mid S_t, A_t)$ is known and does not contain any random factors, the Bellman equation can be solved using dynamic programming [37]. But in the scenario of this paper, it is obviously affected by the random parameters of the task and solved by RL.

Time *t* is the start of the current slot. The remaining available time within the *K* time slot of *C* CPU from the current moment can be expressed as the current state of the system. It's noted as S_t. As shown in Equation (2), it's a $K \times C$ matrix. The row parameter $K - 1$ represents a new time slot in which resources are not used, and its value is the size of the time slot τ. Figure 3 shows a schematic diagram of the system state over time.

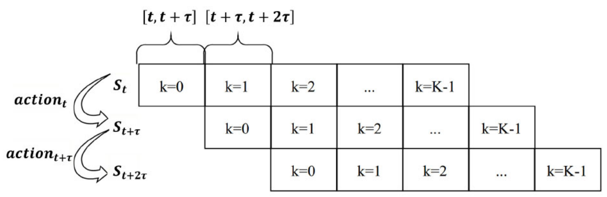


Fig. 3. System state transition diagram.

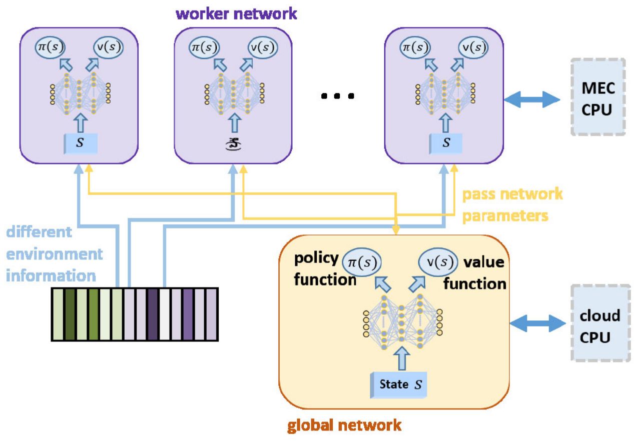


Fig. 4. A asynchronous training framework of A3C.

When in state S_t, the system takes the following actions:

$$action_t = \{A_{\alpha t}, A_{kt}\} \tag{15}$$

where, $A_{\alpha t} = \{a_{0,t}, a_{1,t}, \ldots, a_{j,t}, \ldots, a_{M-1,t}\}$ represents the local or auxiliary offloading of all M tasks arriving at the time t. The value is 1 represents local computing. 0 denotes edge offloading and $\forall a_{j,t} \in \{0, 1\}$. $A_{kt} = \{b_{0,t}, b_{1,t}, \ldots, b_{j,t}, \ldots, b_{M-1,t}\}$ means that M tasks are offloaded to K time slots starting from the current time slot and $\forall b_{j,t} \in \{0, 1, \ldots, k, \ldots, K-1\}$. When the system is in state S_t at t, different actions $action_t = \{A_{\alpha t}, A_{kt}\}$ are taken will result in different values for the next state $S_{t+\tau}$.

According to the optimized objective function proposed in Equation (16), the reward obtained by each state transfer system is set $reward_t$:

$$\begin{aligned} reward_t = w_1 * delay\_reduce\_avg_t \\ - w_2 * delay\_reduce\_std_t. \end{aligned} \tag{16}$$

C. A3C Network Structure

Figure 4 is the model of the A3C network we adopted. It should be noted that the worker and global network have the same structure but different parameters. Figure 5 shows the worker network visualized by Tensorboard during the experiment. We redraw it in a more concise and clear form. It consists of an AC frame. The global network has the same structure as the worker network. The s in the figure represents the input to the system. It is the current MEC server resource occupancy state. Actor and critic networks share a full connection layer $l1$ to extract valid state information. The obtained information is input into three new networks $ap1$, $ap2$, and v respectively. $ap1$ and $ap2$ networks get the action strategy

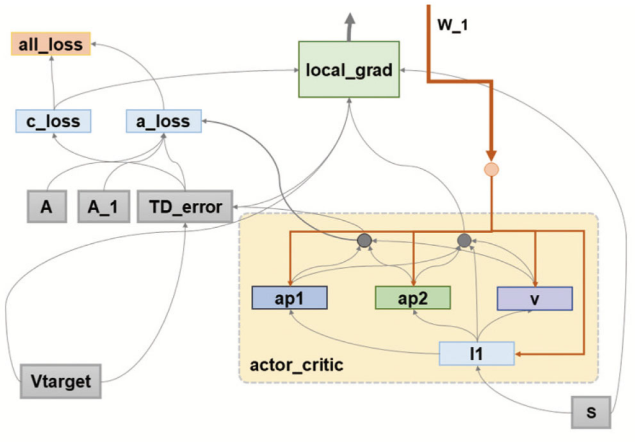


Fig. 5. A visualization of the deployed worker network.

information, respectively. A is the action described above $A_{\alpha t}$. A_1 is action A_{kt}. critic network estimates the value function under state s. According to the rewards of environment interaction, the actual state value V is obtained. It calculates the TD error between the estimated value and the actual value. actor networks and critic networks are optimized with loss functions *a_loss* and *c_loss* to update network parameters.

D. A3C Model Deployment

Figure 6 shows the actual deployment diagram of the A3C model on the IoV scenario. The remote cloud auxiliary layer is deployed as a global network. cn MEC servers are noted as $worker_1$, $worker_2, \ldots, worker_{cn}$, which act as learning agents. In fact, the A3C model can also be deployed on a separate MEC server, where multiple CPUs process interact with the environment as worker networks. However, this approach's coverage of IoV environments will greatly reduce due to the limited by the number of CPUs, which is not enough for agents to learn. Therefore, our research focuses on the former application scenario.

We explain the process of the computation offloading model of the A3C algorithm proposed in this paper as follows. At the moment t of the MEC server, the worker network deployed in the MEC server observes the state S_t in the environment at the current time and selects the corresponding actions $action_t$ according to the policy π_s. It should be noted that t also can be understood as a step. They are distributed to the corresponding connected vehicle environment. According to the information of the arrived task and the resource occupation status of the MEC server, the result of the computing offloading decision is carried out in the computing offloading process. The computing offload decision is the action. Then the CPU utilization status of the MEC server S_{t+1} and the reward $reward_t$ at the next time $t+1$ is obtained. The next moment status with the earned reward is sent to the agent. The learning agent stores the state transfer process $(S_t, action_t, S_{t+1}, reward_t)$ in the cache. After T time which represents the number of steps, the worker uploads its network gradient information asynchronously to the global network. The global network receives and updates all parameters uploaded by the learning agent and

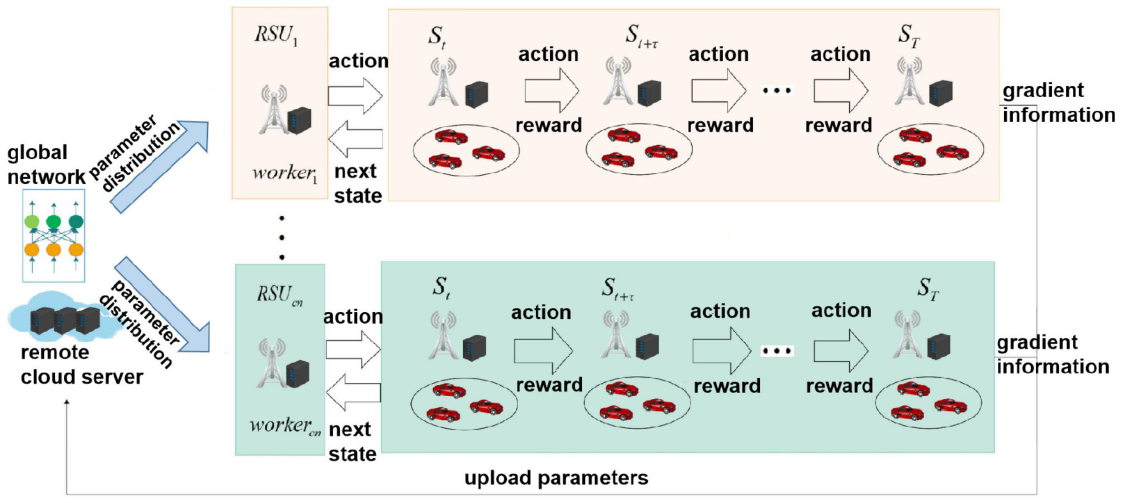


Fig. 6. Actual deployment diagram of A3C network.

TABLE II
SIMULATION PARAMETERS

Parameter description	Value
Number of edge computing servers cn	4
Task local computing power	$[0.5 - 1] \times 10^9$ cycle/s
Computing power per CPU of edge computing server	$[1, 1.5, 2, 2.5, 3, 3.5, 4] \times 10^9$ cycles/s
Computing power required for tasks	$[1 - 2] \times 10^9$ cycle/s
Number of time slots K	3
Number of CPUs C	3
Number of tasks M	[2,3,4,5,6,7,8,9,10]
Task size D_m	[50,100] kbits
Penalty for transmission to the cloud $punish$	5 s
Slot size τ	1 s
Gaussian white noise power	100 dBm

sends the integrated information to the worker network at the right time.

VI. SIMULATION AND ANALYSIS

The experimental parameter Settings are shown in Table II. The experimental conditions used in this paper are Windows10 and 64-bit 8-core CPU operating systems, where each CPU runs one process as an agent. Python 3.5.4 and TensorFlow 1.9.0 were used for simulation experiments. The environment code uses the gym library to simulate, which consists of two main parts. The first part is the *reset* function, and it generates the initial state of the system. The *task step* function is responsible for receiving system's actions. According to the result, the calculated reward will be returned to the worker while the next set of tasks will generate randomly. The parameters settings of the A3C algorithm network are shown in Table III.

As shown in Figure 7, the learning rate *LR_C* of the critic network is 0.1. The learning rate of actor network *LR_A* is the variation of the total reward of the system reacted with different values. With the increase of training time, the total reward of the system increases first and then becomes stable. When *LR_A* is 0.01, the reward value is low. However, when

TABLE III
NETWORK PARAMETERS

Parameter description	Value
Global max execution steps	100
Max execution steps of worker network	20
Upload parameter step interval of worker network	10
Reward loss factor γ	0.9
Learning rate of actor network LR_A	$\{10^{-2}, 10^{-3}, 10^{-4}\}$
Learning rate of critic network LR_C	$\{10^{-1}, 10^{-2}, 10^{-3}\}$

it plateaued, the total reward value was larger. Therefore, in the subsequent experiment, *LR_A* was 0.01. Figure 8 shows the relationship between the learning rate of the actor network and the total rewards of the critic network when the learning rate is 0.01. It can be seen from the numerical results that the convergence speed is faster when *LR_C* is 0.1. Therefore, *LR_C* was set as 0.1 for subsequent experiments.

The comparison schemes we used in our experiments are described as follows:

- *DQN_based*: Based on the settings proposed in this paper, another DRL scheme DQN network is used to

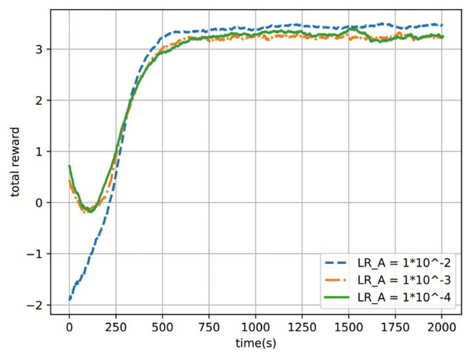


Fig. 7. Comparison of total reward with variety in actor network learning rate.

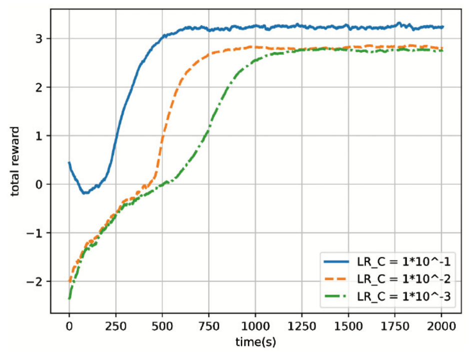


Fig. 8. Comparison of total reward with variety in critic network learning rate.

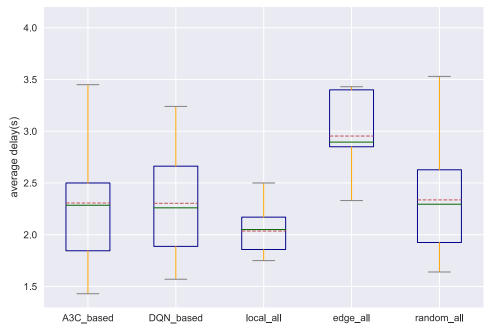


Fig. 9. When A3C model is in learning phase, the relationship between execution time and average delay of completing all tasks.

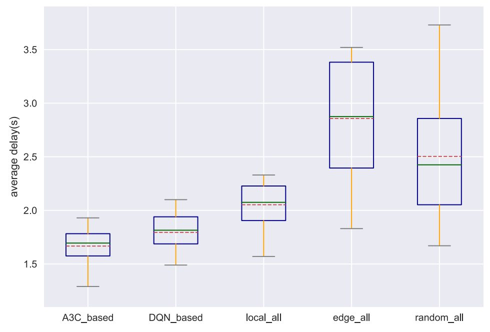


Fig. 10. When A3C model is in stable phase, the relationship between execution time and average delay of completing all tasks.

find an appropriate resource allocation scheme. This is similar to the A3C-based method proposed in this paper, which is a solution that can take into account both system efficiency and task fairness.

- *local_all*: All the tasks generated by vehicle are not offloaded and all executed on the local CPU. It means for all the tasks, none of the extra computing resources except the local CPU can got. In this case, the relative fairness factor is 0. There is a state of absolute fairness in resource usage between tasks.
- *edge_all*: All vehicle tasks are calculated by the edge servers. Whether the edge server has enough available idle resources or whether it meets the delay limit is not considered. In this approach, tasks are offloaded to the cloud server for computing when the edge server is out of resources. Due to the different order in which the task information arrives at the MEC server, there still exists the problem of the priority of using resources between tasks.
- *random_all*: On the basis of our proposed end-edge-cloud architecture, the vehicle randomly selects whether the task is offloaded, and randomly decide the offloading location and slot. This scheme are highly unstable but relatively fair.

Figure 9 and Figure 10 respectively show the comparison of the average delay between the first 20 steps (the first 20 seconds) when the worker starts to learn in the A3C model and the 20 steps after the worker reaches stability by using different computation offloading schemes. They represent the proportional curve between the average delay in completing the task and the execution time. We use boxplots to represent the statistical results. The uppermost and lowermost gray horizontal lines in the boxplot represent the maximum and minimum values of the average delay of tasks, respectively. The bottom and top of the blue boxes represent the statistical data at the 25th and 75th percentile. The green horizontal line is the median of the average delay. The red dotted line represents the average of the average delays. Each time the learning agent in the model interacts with the environment, vehicles in the environment with different performances randomly generate tasks. By comparing Figure 9 and Figure 10, we found that for all the methods, the average delay for completing all tasks is different at the beginning time and the time after the algorithm has stabilized. As can be seen from Figure 7 and Figure 8, the model did not learn practical knowledge or only learned part of knowledge in the first 20 seconds of learning. The total reward of the system is in a state of constant increase. The results in Figure 9 also show that both the *A3C_based* method and *DQN_based* method are still in the exploratory stage at this time. In terms of average delay, the performance of those DRL schemes is not stable, even equivalent to *random_all* scheme. But the performance of *local_all* is relatively stable, because it is not affected by the offloading decision, and task processing is only determined by the computing power of the local vehicle. Figure 10 shows the results when the training of the model reaches a stationary state. At this time, the task arrived at by each time step has great randomness. The average delay of the *A3C_based* method and *DQN_based* method is lower than the other three computation offloading

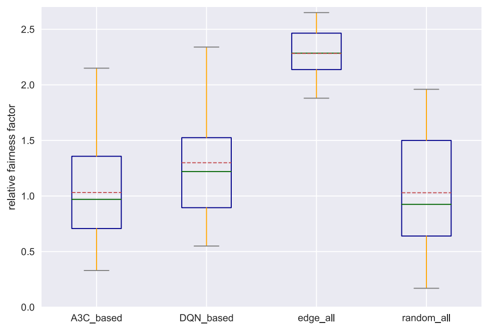


Fig. 11. The relationship between execution time and relative fairness factor when A3C model is in learning stage.

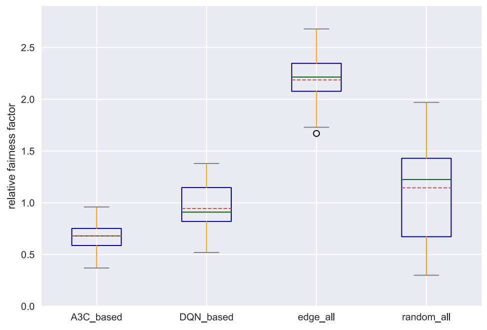


Fig. 12. The relationship between execution time and relative fairness factor when A3C model is in stable stage.

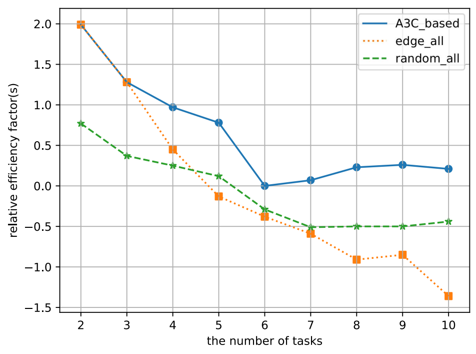


Fig. 13. The relationship between the number of tasks and the relative efficiency factor.

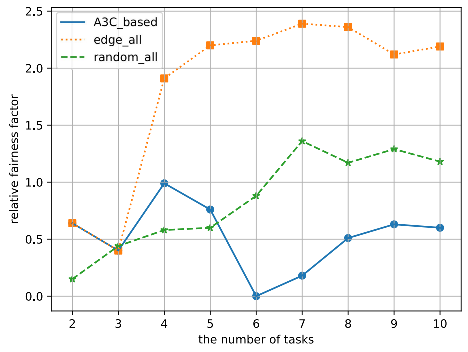


Fig. 14. The relationship between the number of tasks and the relative fairness factor.

schemes. But among the two DRL schemes, *A3C_based* has a lower average latency. From Figure 10, our algorithm can save 7.2%, 18.5%, 41.8%, and 33.5% of the average delay, which is respectively compared to the *DQN_based* scheme, the *local_all* scheme, the *edge_all* scheme and the *random_all* scheme. This shows that the real-time decision made by the A3C model based on MEC server resource occupation and environmental task information is reasonable, which provides a performance boost.

Figure 11 and Figure 12 respectively show the ratio of the relative fairness factor to the execution time of all tasks for the 20 steps at the beginning of learning and the 20 steps after reaching the stability under the different calculation offloading schemes. We do not discuss the local offloading scheme here because it is not helpful to study the variation of the relative fairness factor. Combined with the two result figures, the relative fairness factor of *edge_all* has not changed much but is still influenced by the real-time arrival task, and the generated value fluctuates. When the DRL model has not reached a stable state, the *A3C_based* and *DQN_based* schemes are all closer to the *random_all* scheme in terms of relative fairness. This is because the learning agent is undergoing repeated trials and interactions with the environment. After reaching stability, the value of the relative fairness factor for these DRL methods are relatively small and stable. This is because access to MEC server resources is relatively equitable between tasks. Moreover, this decision is made after comprehensive consideration of efficiency and fairness. But in these two methods, *A3C_based* method is a better choice. Because *A3C_based* scheme is more suitable for dealing with dynamically changing situations.

Figure 13 and Figure 14 respectively show the relationship between the number of tasks, relative efficiency factor, and relative fairness factor when the computing power of each CPU of the MEC server is 2.5×10^9 cycles/s. In the experimental results, the coordinates of the horizontal axis is the number of tasks arriving at each moment. Each data point in the figure is obtained by averaging the results of 10-time steps that the model interacts with the environment after stabilization. Due to the influence of the environment, the performance when the number of tasks is small may not be better than that when the number of tasks is large. As can be seen from Figure 13, with the increase of the number of tasks arriving at each moment, the average reduction delay of the relative efficiency factor of *edge_all* and *random_all* method relative to *local_all* method shows a downward trend. As the number of tasks increases, the MEC server cannot carry all the tasks that arrive at the current moment. Some of the tasks are forced to be transferred to the cloud layer for computing, which leads to an increase latency. When the number of tasks arrived at each time is high enough, the value of negative will occur in Figure 13. That means the performance of the *edge_all* is even worse than *local_all* method. It can be seen from Figure 14 that the relative fairness factor of *edge_all* and *random_all* methods increases with the increase of the number of tasks. As the number of tasks increases, the fairness of obtaining resources between tasks decreases. The *A3C_based* scheme combines the relative efficiency factor and the relative fairness factor as

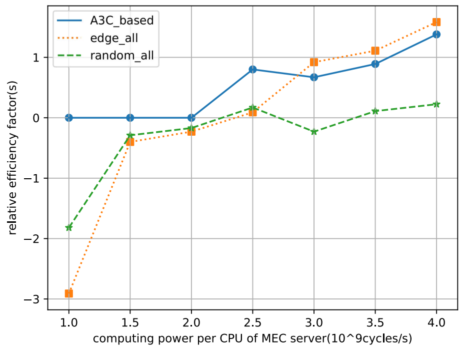


Fig. 15. The relationship between the computing power of each CPU and the relative efficiency factor of MEC servers.

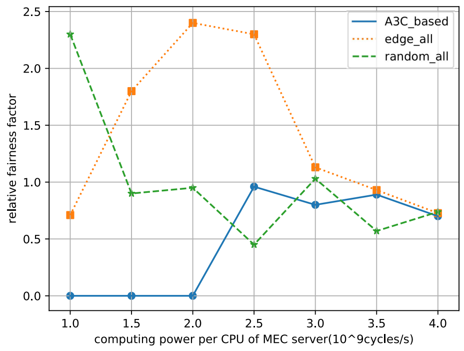


Fig. 16. The relationship between the computing power of each CPU and the relative fairness factor of MEC servers.

the consideration factor. When 2 or 3 tasks are reached per second, the *A3C_based* method may choose the same computing offloading scheme as *edge_all*. That is because the values of the relative efficiency factor and relative fairness factor under 2 or 3 tasks for these 2 methods are precisely same. When the number of tasks is 6, the relative efficiency factor value is 0 and the relative fairness is 0. Then we can deduce that the scheme obtained by the A3C model is all tasks completed locally. When the number of tasks is greater than 6, the relative efficiency factor of the *A3C_based* scheme increases. The performance in terms of average delay has been improved. At the same time, the relative fairness factor increases, which means that the fairness decreases accordingly.

Figure 15 and Figure 16 show that the number of arriving tasks at each moment is 5, and the computing capacity of each CPU of the MEC server is $[1, 1.5, 2, 2.5, 3, 3.5, 4] \times 10^9$ cycles/s, respectively. It represents the relationship between the three computing offloading schemes and the relative efficiency factors and the relative fairness factors. In the two figures, the randomness of the *random_all* scheme is greater, and the result changes are also greater. For the *edge_all* scheme, the relative efficiency factor is negative when the CPU computing power is small. It performs worse than the *local_all* scenario. With the increase of CPU's computing power, the relative efficiency factor gradually increases and reaches a positive value. The average time delay for completing all tasks per moment is gradually reduced compared to the *local_all* method. However, it can be seen from Figure 15 that the relative fairness factor of the *edge_all* method increases firstly and then decreases. Due to the low power of CPU, the MEC server is difficult to complete all the task of the vehicle. Most of the tasks are transferred to the remote cloud auxiliary layer for completion with a large delay. The fairness between tasks is high in the per MEC's CPU. When the computing power of each CPU of the MEC server approaches 2×10^9 cycles/s, the relative fairness factor reaches its maximum value. Tasks compete fiercely for the resources of the MEC server. Tasks completed at the MEC server have reduced latency relative to those completed locally. Some of the tasks transferred to the remote cloud have increased latency relative to local computing. As the computing power of the MEC server continues to increase, the MEC server can handle more and more computing tasks. There is less competition between tasks for resources. The relative fairness factor also decreases gradually. For the *A3C_based* scheme, the scheme learned at the beginning is close to the *local_based* scheme. The value of the relative fairness factor is small, and the average delay of completing all tasks is also small. When the computing capacity of the MEC server increases, the *A3C_based* scheme selects more tasks to be completed by the MEC server. In terms of task completion efficiency, the *A3C_based* scheme is superior to *edge_all* scheme. When the computing power of each CPU of the MEC server continues to increase, the result of the fairness for *A3C_based* scheme is close to the *edge_all* scheme.

VII. Conclusion

This paper mainly considers the complex environment in IoV and designs a dynamic computing offloading scheme based Asynchronous Advantage Actor-Critic (A3C) which considering different resource occupancy states of MEC servers. The system can be used in a highly random environment, while in consideration of task execution efficiency and fairness among tasks. The simulation experiment proves that, compared with the other schemes, our scheme proposed has better performance in both efficiency and fairness. The long-term reward value gradually increases and finally reaches a relatively stable level, which proves the effective of the proposed algorithm in computation offloading. On the basis of the current work done in this paper, some points should be considered for further research. For example, this paper does not include the way of V2V offloading into the task completion method for solving. On the other hand, this paper only takes the computing resource occupancy of the mobile edge server as the state of the system, which can be extended in the further study. In the next step, we will make detailed reasoning about the communication details and the model of vehicle motion and angle. Based on security and privacy considerations, this work may remove the penalty item for cloud computing and combine with federated learning in the future.

References

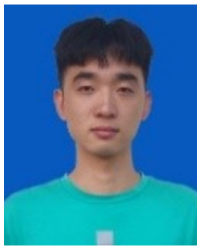
[1] X. Zhou, W. Liang, J. She, Z. Yan, and K. I.-K. Wang, "Two-layer federated learning with heterogeneous model aggregation for 6G supported Internet of Vehicles," *IEEE Trans. Veh. Technol.*, vol. 70, no. 6, pp. 5308–5317, Jun. 2021.

[2] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Distributed federated learning for ultra-reliable low-latency vehicular communications," *IEEE Trans. Commun.*, vol. 68, no. 2, pp. 1146–1159, Feb. 2020.
[3] C. Chen, L. Liu, S. Wan, X. Hui, and Q. Pei, "Data dissemination for industry 4.0 applications in Internet of Vehicles based on short-term traffic prediction," *ACM Trans. Internet Technol.*, vol. 22, no. 1, pp. 1–18, 2021.
[4] T. Qiu, X. Wang, C. Chen, M. Atiquzzaman, and L. Liu, "TMED: A spider-Web-like transmission mechanism for emergency data in vehicular ad hoc networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 9, pp. 8682–8694, Sep. 2018.
[5] X. Zhang, M. Peng, S. Yan, and Y. Sun, "Joint communication and computation resource allocation in fog-based vehicular networks," *IEEE Internet Things J.*, early access, Jan. 6, 2022, doi: 10.1109/JIOT.2022.3140811.
[6] S. Liu, J. Yu, X. Deng, and S. Wan, "FedCPF: An efficient-communication federated learning approach for vehicular edge computing in 6G communication networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 2, pp. 1616–1629, Feb. 2022.
[7] X. Deng, Z. Sun, D. Li, J. Luo, and S. Wan, "User-centric computation offloading for edge computing," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12559–12568, Aug. 2021.
[8] S. Wan, S. Ding, and C. Chen, "Edge computing enabled video segmentation for real-time traffic monitoring in Internet of Vehicles," *Pattern Recognit.*, vol. 121, Jan. 2022, Art. no. 108146.
[9] A. Sujith, G. S. Sajja, V. Mahalakshmi, S. Nuhmani, and B. Prasanalakshmi, "Systematic review of smart health monitoring using deep learning and artificial intelligence," *Neurosci. Inform.*, vol. 2, no. 3, 2022, Art. no. 100028.
[10] C. Chen, J. Jiang, Y. Zhou, N. Lv, X. Liang, and S. Wan, "An edge intelligence empowered flooding process prediction using Internet of Things in smart city," *J. Parallel Distrib. Comput.*, vol. 165, pp. 66–78, Jul. 2022.
[11] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L.-C. Wang, "Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges," *IEEE Veh. Technol. Mag.*, vol. 14, no. 2, pp. 44–52, Jun. 2019.
[12] S. B. Prathiba, G. Raja, K. Dev, N. Kumar, and M. Guizani, "A hybrid deep reinforcement learning for autonomous vehicles smart-platooning," *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13340–13350, Dec. 2021.
[13] Y. Zhang, Y. Zhang, J. Ren, J. Misic, and A. M. Tulino, "Guest editorial introduction to the special section on vehicular networks in the era of 6G: End-edge-cloud orchestrated intelligence," *IEEE Trans. Veh. Technol.*, vol. 70, no. 6, pp. 5192–5196, Jun. 2021.
[14] Z. Chang, S. Liu, X. Xiong, Z. Cai, and G. Tu, "A survey of recent advances in edge-computing-powered artificial intelligence of things," *IEEE Internet Things J.*, vol. 8, no. 18, pp. 13849–13875, Sep. 2021.
[15] C. Roy, R. Saha, S. Misra, and K. Dev, "Micro-safe: Microservices- and deep learning-based safety-as-a-service architecture for 6G-enabled intelligent transportation system," *IEEE Trans. Intell. Transp. Syst.*, early access, Sep. 27, 2021, doi: 10.1109/TITS.2021.3110725.
[16] Y. Zhang, H. Zhang, and K. Long, "Energy efficient resource allocation in cache based terahertz vehicular networks: A mean-field game approach," *IEEE Trans. Veh. Technol.*, vol. 70, no. 6, pp. 5275–5285, Jun. 2021.
[17] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang, "Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 12175–12186, Oct. 2020.
[18] C. Wu, Z. Liu, F. Liu, T. Yoshinaga, Y. Ji, and J. Li, "Collaborative learning of communication routes in edge-enabled multi-access vehicular environment," *IEEE Trans. Cogn. Commun. Netw.*, vol. 6, no. 4, pp. 1155–1165, Dec. 2020.
[19] C. Sun *et al.*, "Task offloading for end-edge-cloud orchestrated computing in mobile networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2020, pp. 1–6.
[20] Y. Ding, K. Li, C. Liu, and K. Li, "A potential game theoretic approach to computation offloading strategy optimization in end-edge-cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 6, pp. 1503–1519, Jun. 2022.
[21] Y. Liu, S. Wang, J. Huang, and F. Yang, "A computation offloading algorithm based on game theory for vehicular edge networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2018, pp. 1–6.
[22] K. Zhang, Y. Mao, S. Leng, S. Maharjan, and Y. Zhang, "Optimal delay constrained offloading for vehicular edge computing networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2017, pp. 1–6.
[23] J. Du, F. R. Yu, X. Chu, J. Feng, and G. Lu, "Computation offloading and resource allocation in vehicular networks based on dual-side cost minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1079–1092, Feb. 2019.
[24] S. Fei *et al.*, "Cooperative task scheduling for computation offloading in vehicular cloud," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11049–11061, Nov. 2018.
[25] X. Zhu, Y. Luo, A. Liu, M. Z. A. Bhuiyan, and S. Zhang, "Multiagent deep reinforcement learning for vehicular computation offloading in IoT," *IEEE Internet Things J.*, vol. 8, no. 12, pp. 9763–9773, Jun. 2021.
[26] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 3, pp. 881–892, Sep. 2021.
[27] J. Zhang, W. Shi, R. Zhang, and S. Liu, "Deep reinforcement learning for offloading and shunting in hybrid edge computing network," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, 2021, pp. 1–6.
[28] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.
[29] J. Feng, F. R. Yu, Q. Pei, X. Chu, J. Du, and L. Zhu, "Cooperative computation offloading and resource allocation for blockchain-enabled mobile-edge computing: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6214–6228, Jul. 2020.
[30] H. Ke, J. Wang, L. Deng, Y. Ge, and H. Wang, "Deep reinforcement learning-based adaptive computation offloading for MEC in heterogeneous vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7916–7929, Jul. 2020.
[31] S. Bi, L. Huang, H. Wang, and Y.-J. A. Zhang, "Stable online computation offloading via Lyapunov-guided deep reinforcement learning," in *Proc. IEEE Int. Confera Multi-Agenence Commun.*, 2021, pp. 1–7.
[32] W. Wei, R. Yang, H. Gu, W. Zhao, C. Chen, and S. Wan, "Multi-objective optimization for resource allocation in vehicular cloud computing networks," *IEEE Trans. Intell. Transp. Syst.*, early access, Aug. 3, 2021, doi: 10.1109/TITS.2021.3091321.
[33] C. Chen, Y. Zhang, Z. Wang, S. Wan, and Q. Pei, "Distributed computation offloading method based on deep reinforcement learning in ICV," *Appl. Soft Comput.*, vol. 103, May 2021, Art. no. 107108.
[34] W. Cong, C. Chen, P. Qingqi, J. Zhiyuan, and X. Shugong, "An information centric in-network caching scheme for 5G-enabled Internet of connected vehicles," *IEEE Trans. Mobile Comput.*, early access, Dec. 21, 2021, doi: 10.1109/TMC.2021.3137219.
[35] H. Q. Le, H. Al-Shatri, and A. Klein, "Efficient resource allocation in mobile-edge computation offloading: Completion time minimization," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2017, pp. 2513–2517.
[36] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic scheduling for stochastic edge-cloud computing environments using A3C learning and residual recurrent neural networks," *IEEE Trans. Mobile Comput.*, vol. 21, no. 3, pp. 940–954, Mar. 2022.
[37] R. Li *et al.*, "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74429–74441, 2018.

Chen Chen (Senior Member, IEEE) received the B.Eng., M.Sc., and Ph.D. degrees in telecommunication from Xidian University, Xi'an, China, in 2000, 2006, and 2008, respectively, where he is currently a Professor with the Department of Telecommunication and a Member of The State Key Laboratory of Integrated Service Networks. He is also the Director of the Xi'an Key Laboratory of Mobile Edge Computing and Security and the Intelligent Transportation Research Laboratory, Xidian University. He was a Visiting Professor with the Department of EECS, University of Tennessee and the Department of CS, University of California. He has authored/coauthored two books, over 130 scientific papers in international journals and conference proceedings. He has contributed to the development of five copyrighted software systems and invented over 100 patents. He serves as the general chair, the PC chair, the workshop chair, or the TPC member of a number of conferences. He is also a Senior Member of China Computer Federation and China Institute of Communications.

Haofei Li received the B.E. degree in communication engineering from Hohai University, Nanjing, China, in 2019. She is currently pursuing the Doctoral degree with Xidian University, Xi'an, China. Her research interests include wireless communications, mobile edge computing, and satellite terrestrial networks.

Huan Li received the B.S. degree in communication engineering from Xidian University, in 2021, where she is currently pursuing the master's degree in transportation information engineering and control. Her research interests include, intelligent transportation, mobile edge computing, and reinforcement learning.



Rufei Fu received the B.Eng. degree in communication engineering from Xidian University, Xi'an, China, in 2020, where he is currently pursuing the master's degree. His research interests include IoV and blockchain.

Yangyang Liu received the Master of Science degree in electrical and computer engineering from the University of Illinois at Chicago, Chicago, USA, in 2016. Since 2017, he has been working with China Automotive Technology and Research Center. His research interests include vehicle wireless communication, vehicle control engineering, and vehicle cyber-security.

Shaohua Wan (Senior Member, IEEE) received the Ph.D. degree from the School of Computer, Wuhan University in 2010. He is currently a Professor with the Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China. He has authored over 150 peer-reviewed research papers and books, including over 40 IEEE/ACM Transactions papers, such as IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, *ACM Transactions on Internet Technology*, IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, IEEE TRANSACTIONS ON MULTIMEDIA, IEEE TRANSACTIONS ON COMPUTATIONAL SOCIAL SYSTEMS, IEEE TRANSACTIONS ON MULTIMEDIA, and IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE, and many top conference papers in the fields of Edge Intelligence.