Joint Service Caching and Computation Offloading Scheme Based on Deep Reinforcement Learning in Vehicular Edge Computing Systems

Zheng Xue, Chang Liu, Canliang Liao, Guojun Han, *Senior Member, IEEE,* and Zhengguo Sheng, *Senior Member, IEEE*

***Abstract*—Vehicular edge computing (VEC) is a new computing paradigm that enhances vehicular performance by introducing both computation offloading and service caching, to resource-constrained vehicles and ubiquitous edge servers. Recent developments of autonomous vehicles enable a variety of applications that demand high computing resources and low latency, such as automatic driving, auto navigation, etc. However, the highly dynamic topology of vehicular networks and limited caching space at resource-constrained edge servers calls for intelligent design of caching placement and computation offloading. Meanwhile, service caching decisions are highly correlated to the computation offloading decisions, which pose a great challenge to effectively design service caching and computation offloading strategies. In this paper, we investigate a joint optimization problem by integrating service caching and computation offloading in a general VEC scenario with time-varying task requests. To minimize the average task processing delay, we formulate the problem using long-term mixed integer non-linear programming (MINLP) and propose an algorithm based on deep reinforcement learning to obtain a suboptimal solution with low computation complexity. The simulation results demonstrate that our proposed scheme exhibits an effective performance improvement in task processing delay compared with other representative benchmark methods.**

***Index Terms*—Vehicular edge computing, service caching, computation offloading, deep reinforcement learning.**

I. Introduction

THE rapid development of Internet of Things (IoT) and artificial intelligence (AI) has recently led to the emergence of diverse computation-intensive and latency-sensitive vehicular applications, such as augmented reality (AR) navigation and autonomous driving. However, offloading all the computing tasks to the remote cloud results in a heavy backhaul load and unacceptable latency. Through vehicular edge computing (VEC) [1], vehicular users can offload their tasks via vehicle-to-infrastructure (V2I) communications to edge nodes to reduce response latency, which caters for unprecedentedly exploding data traffic and increasingly stringent requirements of vehicular applications [2].

Task computation requires both input task data from users and program data installed on edges, which are defined as content caching and service caching, respectively. Content caching refers to caching of the input data needed and output data generated (e.g., in computational or infotainment applications) at vehicles and edge nodes [3]–[7]. Since these data dominate mobile data traffic, content caching at edge servers can effectively alleviate mobile traffic on backhaul links and reduce content delivery latency [3]. On the other hand, service caching refers to caching the specific programs for task execution [8]–[14]. As a motivating example, in object detection, the input data consists of videos and radar sensor data, and task execution requires the corresponding object detection service program to be cached in the vehicle or the edge server. The input data of object detection service is typically unique and hardly reusable for other executions. In comparison, service program data in the cache is evidently reusable by future executions of the same type of tasks. Because edge servers have limited caching space, how to selectively cache service program over space (e.g., at multiple edge servers) and time resources for achieving optimum transmission and computing performance is crucial for efficient task computation.

The design of optimal computation offloading and service caching faces many challenges in vehicular networks. First, vehicles and edge servers can only cache a small number of service programs at a time due to limited storage space. Thus, which service programs should be cached needs to be decided judiciously. Second, the computing resources on different edge servers may be unevenly distributed. It is critical to balance the computation load by cooperative offloading. Third, the computation offloading decisions and the service caching decisions are closely correlated. Intuitively, we tend to offload a task if the required program is already cached at

Manuscript received June 2, 2022; revised September 4, 2022; accepted December 9, 2022. This work was supported in part by Guangzhou Science and Technology Plan Project under Grant 202201010239; in part by Graduate Education & Innovation Project of Guangdong Province under Grant 2022XSLT022; in part by Guangdong Introducing Innovative and Entrepreneurial Teams of The Pearl River Talent Recruitment Program (2021ZT09X044); in part by Guangdong Introducing Outstanding Young Scholars of The Pearl River Talent Recruitment Program (2021QN02X546); in part by Guangdong Provincial Key Laboratory of Photonics Information Technology (No. 2020B121201011); in part by the Science and Technology Program of Guangzhou under Grant 202102020869, and the Guangdong Basic and Applied Basic Research Foundation under Grant 2022A1515110602 and Grant 2022A1515010153. ($Corresponding\ author: Chang\ Liu.$)

Zheng Xue, Chang Liu, Canliang Liao and Guojun Han are with School of Information Engineering, Guangdong University of Technology, Guangzhou 510006, China. (e-mail: xuezheng@mail2.gdut.edu.cn; liuchang@gdut.edu.cn; canliang148@gmail.com; gjhan@gdut.edu.cn).

Zhengguo Sheng is with the Department of Engineering and Design, University of Sussex, Brighton, BN1 9RH, U.K. (e-mail: z.sheng@sussex.ac.uk).

the edge.[1] Besides, the network status and available resources of edge servers change dynamically during the movement of vehicles. Therefore, it becomes significant and yet very challenging to design an appropriate service caching and computation offloading strategy in the VEC systems.

In this paper, we investigate a joint optimization of service caching and computation offloading for a VEC system with limited storage and computing capacities, taking account of time-varying task requests and dynamic network topology. In order to make full use of the limited caching and computing resources of each node (i.e. vehicles and edge servers) as well as the cooperative offloading between edge servers, we propose a deep reinforcement learning (DRL)-based service caching and computation offloading scheme to provide low-complexity decision making and adaptive resource management. The main contributions of this paper can be summarized as follows:

1) We design a novel edge service caching and computation offloading framework with cooperation among the cloud, edge servers and vehicles, which not only balances the computation load among edge servers, but also enables integration of caching and computing resources combined with edge intelligence.

2) We minimize the cumulative average task processing delay over a long time-slotted horizon, considering dynamic task requesting, offloading and service caching, as well as dynamic channel conditions between vehicles and edge servers at each time slot. To solve the formulated long-term mixed integer non-linear programming (MINLP) problem, we propose an edge caching and offloading scheme based on deep deterministic policy gradient (DDPG) [15] to efficiently make decisions on task offloading and service caching.

3) We carry out extensive simulations to evaluate the average task processing delay and energy consumption of the proposed scheme in VEC. Numerical results demonstrate that our proposed scheme can achieve better performances compared with the other benchmark approaches.

The rest of the paper is organized as follows. We review the related work in Section II. The system model and the problem formulation are described in Section III. In Section IV, the proposed scheme is presented in details. Section V provides numerical results, and Section VI concludes this paper.

II. Related Work

In the past few years, computation offloading in mobile edge computing (MEC) has been intensively discussed [16]–[19]. Zhang $et\ al.$ [16] minimize the average bandwidth consumption with a novel bidirectional computation task model by joint caching and computing offloading policy optimization. Tout $et\ al.$ [17] find the optimal dissemination of computational tasks within MEC-based infrastructures while satisfying persona needs on a wider range of devices and assuring minimal additional fees imposed by remote execution. Multi-user multi-task computation offloading and resource allocation for mobile edge computing are proposed in [19]. For the task offloading in VEC [1], [2], [20]–[24]. Tan $et\ al.$ [2] propose a joint communication, caching and computing strategy to minimize the system cost. To guarantee the reliability of communication systems, powerful error correction codes, such as low-density parity-check (LDPC) codes, can be applied to enhance the anti-noise and anti-fading capability [25]–[27]. Qiao $et\ al.$ [20] minimize the content access cost for a novel cooperative edge caching framework. Ning $et\ al.$ [21] design a mobility-aware edge computing and caching scheme to maximize mobile network operators' profits. However, all these works are based on the implicit assumption that all the services are available in edge servers, which is impractical due to their limited storage capacities.

Service caching, which refers to the caching of related programs for computational task execution, can significantly affect the performances of MEC systems since service caching strategies and computation offloading strategies are always coupled. There has been considerable research focusing on joint service caching, computation offloading and resource allocation for mobile users in MEC system [8]–[12]. Yan $et\ al.$ [8] propose an MEC service pricing scheme to coordinate with service caching decisions and control wireless devices' task offloading behaviors in a cellular network. Ko $et\ al.$ [9] maximize a sum-utility for multi-mobile service caching enabled MEC. Bi $et\ al.$ [10] formulate a mixed integer nonlinear programming (MINLP) that jointly optimizes service caching placement, computation offloading decisions, and system resource allocation to minimize computation delay and energy consumption of mobile user. Zhang $et\ al.$ [12] investigate joint service caching, computation offloading and resource allocation problem in a general multi-user multi-task scenario and aim to minimize the weighted sum of all users computation cost and delay cost. However, the joint caching and computing strategy optimization in MEC systems cannot be directly applied to VEC systems. The high mobility of vehicles results in complicated and dynamic topology as well as frequent server switching.

There have been emerging efforts on content caching and computation offloading in VEC [20], [21], [28]–[31]. Tian $et\ al.$ [30] propose a collaborative computation offloading and content caching method, by leveraging DRL for a self-driving system. Wu $et\ al.$ [31] propose a multi-agent based reinforcement learning (RL) algorithm to make decisions on task offloading and edge caching to optimize both service latency and energy consumption of vehicles. The important difference between content caching and service caching is that the latter not only concerns storage capacity but also the computing capacity. Thus, the service caching brings more challenges to the VEC system design. To the best of our knowledge, so far only a few works [13], [14] have investigated the problem of joint optimization of service caching and computation offloading in VEC system. Tang $et\ al.$ [13] apply application caching to VEC to optimize the response time for the outsourced applications while satisfying the time slot spanned energy consumption constraint. The Lyapunov optimization technology is adopted to tackle this constraint issue. Finally, two greedy heuristic algorithms are incorporated into the drift-plus-penalty based algorithm to help finding the approximate optimal solution. Lan $et\ al.$ [14] propose a fog-

[1] In this paper, we use the terms "edge", "edge node" and "edge server" interchangeably.

based vehicular architecture featuring computation offloading with service caching and design the offloading strategies based on a DRL algorithm to reduce the task computation delay and energy cost. However, the limitation of storage and computing resources in vehicles or edge servers is not considered in [13], [14]. Based on the existing literature, service caching and computation offloading scenarios involving vehicles and edge servers with limited resources in VEC have not been explored.

In recent years, AI-based algorithms have been successfully applied in numerous related works on emerging vehicular applications and services which are mostly delay-sensitive. This smart driving assistance thus can significantly improve driving safety, reduce energy consumption, and enhance traffic management efficiency [32], [33]. Specifically, RL and DRL have been demonstrated to significantly improve the performances of vehicular task offloading [14], [20], [21], [29]–[31], [34]–[39]. Zhu *et al.* [35] propose a multiagent DRL-based computation offloading scheme to minimize the total task execution delay of the considered system during a certain period. Since current vehicular services always contain multiple tasks with data dependency, a knowledge driven service offloading framework is proposed in [36] by exploring the DRL model to find the long-term optimal service offloading policy. Since DRL agents can react to vehicular environment changes in milliseconds to achieve real-time decision making, they have superiority in complex and highly dynamic vehicular networks with fast varying channels and computation load.

The above studies on VEC are mostly based on the assumption that service programs are completely available in vehicles or edge servers with limited resources, which is not always feasible in practice. To the best of our knowledge, Refs. [13] and [14] are the only existing literature that do not assume full availability of service programs. However, they have not taken the resource limitation into consideration. To fill this gap, we aim at joint computation offloading and service caching, taking full advantage of the limited resources of vehicles and edge nodes as well as edge cooperative offloading to minimize the average task processing time.

III. System Model and Problem Formulation

In this section, we first provide the system overview, communication model, as well as the service caching and task offloading model. Then we analytically derive the computation delay, energy consumption, and provide extreme value analysis of single task processing delay and edge node energy consumption. Last we present our problem formulation, which is essential for service caching and task offloading scheme decision making. The primary notations utilized in the following are summarized in Table I.

TABLE I
Primary Notations

Notation	Definition
V	The set of vehicles $V = \{1, 2, ...v, ...\}$
E	The set of edge nodes $E = \{1, 2, ...e, ...N_E\}$
K	The set of service programs $K = \{1, 2, ...k, ...N_K\}$
B	The total bandwidth of each edge server
$B_{v,e}(t)$	The bandwidth of edge server e allocated to vehicle v in time slot t
d_k	The input data size of task k
θ_k	The required storage space of service program k
$S_v^V(S_e^E)$	The storage at each vehicle (edge)
$\gamma_{v,e}$	The signal-to-noise ratio from edge e to vehicle v
$g_{v,e}(t)$	The up-link gain between edge e and vehicle v in time slot t
$f^V(f^E)$	The fixed CPU frequency of each vehicle (edge)
κ	The computing energy efficiency parameter
λ	The nature of service application
$c_{v,k}^V(c_{e,k}^E)$	The caching decision of vehicle v's (edge e's) service program k
$o_{v,k}^V(o_{e,k}^E)$	The offloading decision of vehicle v's (edge e's) task k
$N_e^{task}(t)$	The total number of tasks received at edge e in time slot t
$R_{edge}(R_{cloud})$	The transmission rate from edge to edge (cloud)
$p_{edge}(p_{cloud})$	The transmission power from edge to edge (cloud)

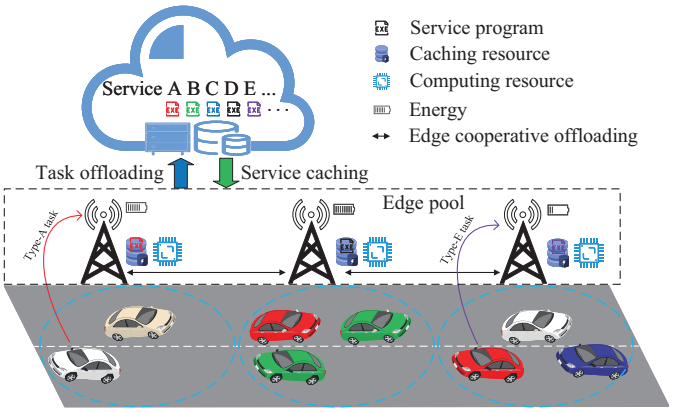

Fig. 1. System illustration.

A. System Overview

As illustrated in Fig. 1, we consider a general vehicular network consisting of an edge pool (the set of edge nodes are denoted as $E = \{1, 2, ...e, ...N_E\}$), the cloud and a number of moving vehicles (the set of vehicles are denoted as $V = \{1, 2, ...v, ...\}$). Suppose that there are N_K service programs (e.g., executable .EXE files) corresponding to service-dependent tasks in the system (i.e., running these tasks requires precaching of their corresponding service programs). If the associated service program of a requested task has been cached at the cloud or the edge pool, vehicles can offload a portion of the computing tasks via wireless communication (e.g., cellular vehicle-to-everything (C-V2X)) to the edge pool or the cloud, depending on the trajectory of the vehicle and the location of the cached service program. The edge pool consists of a set of interconnected edge servers to balance different computing and caching resources. Due to the mobility of vehicles, cooperative edge offloading between multiple edge servers can further improve the caching efficiency. However, unlike the cloud which has abundant computing and storage resources, limited computing and caching resources of edge nodes allow only a small set of services program to be cached at the same time. Therefore, an AI-based management controller (i.e., agent) is typically deployed at the edge pool, which collects information from vehicles and edge servers, and makes decisions on service caching and task offloading. To this end, time is divided into a set of discrete time slot, indexed

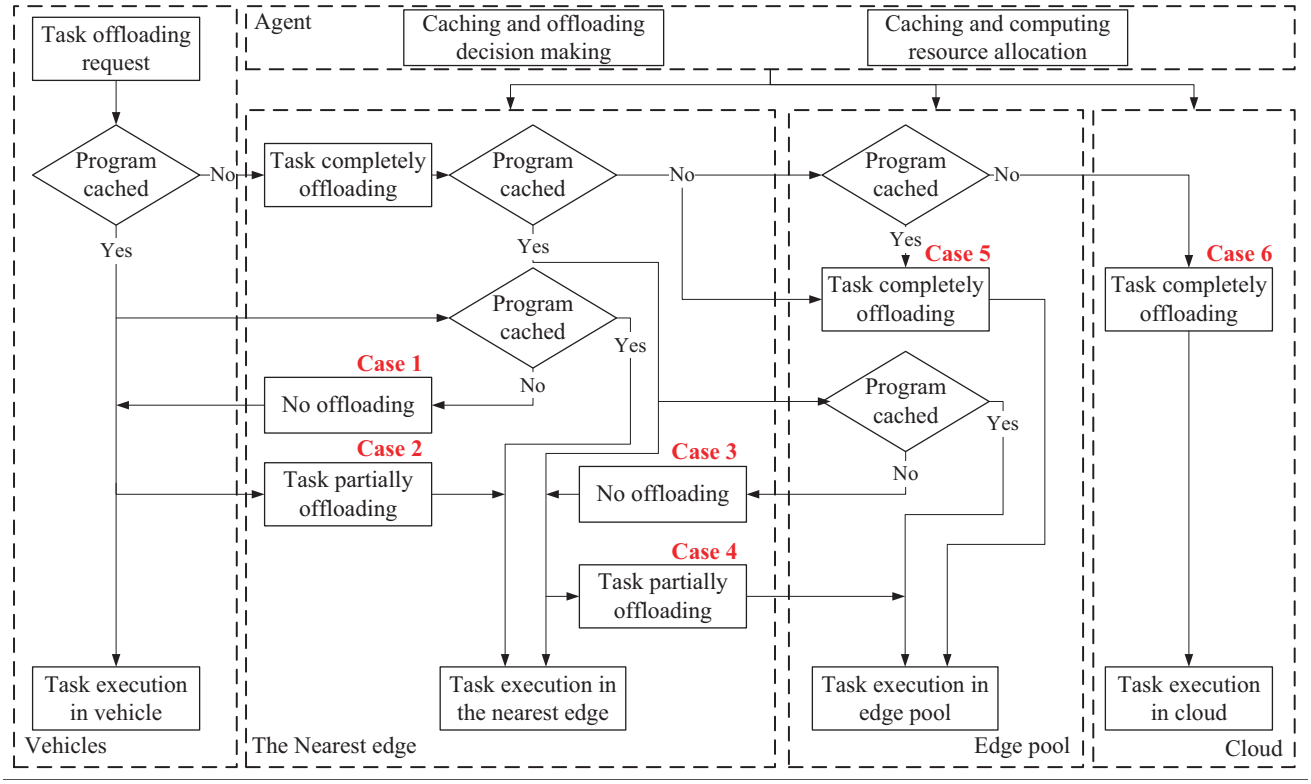

Fig. 2. Flowchart of task offloading.

by $\{1, 2, ...t, ...t_{end}\}$, each of which has an equal duration Δt. The service caching and task offloading strategy can be updated at each time slot.

B. Communication Model

To characterize a practical vehicle moving environment, vehicles with driving speed ranging from V_{min} to V_{max} are considered [4], [5] and edge nodes are assumed to be aware of all the vehicles belonging to its coverage area. We consider a multi-channel uplink model, where each edge has overall bandwidth of B, and each channel has two possible states (i.e., occupied and unoccupied). Note that the allocated spectrum is the same for different edge nodes and the allocated channels are all orthogonal so that the interference is negligible in the coverage area of the same edge node. We assume that each vehicle can only send one task request in time slot t, with $N_e^{task}(t)$ denoting the total number of tasks received at edge node e in time slot t. In this case, the signal-to-interference-plus-noise ratio (SINR) between vehicle v and edge e in time slot t is given by

$$\gamma_{v,e}(t) = \frac{g_{v,e}(t)p_{v,e}(t)}{\sum_{e=1}^{N_E} g_{v,e}(t) \sum_{v=1}^{N_e^{task}(t)} p_{v,e}(t) + \sigma^2}, \quad (1)$$

where σ^2 refers to the variance of additive white Gaussian noise, $g_{v,e}(t)$ denotes the average channel gain, and $p_{v,e}(t)$ represents the average channel transmission power of edge node e.

If vehicle v needs to offload data to the nearest edge e, the wireless transmission rate at time slot t is calculated based on the Shannon's formula:

$$R_{v,e}(t) = B_{v,e}(t) log_2(1 + \gamma_{v,e}(t)), \quad (2)$$

where $B_{v,e}(t)$ is the allocated bandwidth for vehicle v and $B_{v,e}(t) = B/N_e^{task}(t)$.

C. Service Caching and Task Offloading Model

At the beginning of each time slot t, vehicles entering the range of edge nodes update task requests, and complete task offloading and computation in this time slot. Before the end of each time slot, service caching decisions for edge nodes are made by the management controller, while those for vehicles are made by the vehicles themselves based on interests. Let $K = \{1, 2, ...k, ...N_k\}$ denote the set of service programs. Each task can be represented as $\{d_k, \theta_k\}$, $k \in K$ where d_k denotes data size of the input data and θ_k denotes the required storage space of service program k. Then we have the following storage capacity constraints:

$$\sum_{k=1}^{N_K} c_{v,k}^V(t)\theta_k \leq S_v^V, \forall v, t, \quad (3)$$

and

$$\sum_{k=1}^{N_K} c_{e,k}^E(t)\theta_k \leq S_e^E, \forall v, t. \quad (4)$$

where $c_{v,k}^V(t) \in \{0, 1\}, c_{e,k}^E(t) \in \{0, 1\}$ are binary decision variables to denote whether service program k is cached (i.e., $c_{v,k}^V(t) = 1, c_{e,k}^E(t) = 1$) or not (i.e., $c_{v,k}^V(t) = 0, c_{e,k}^E(t) = 0$) on vehicle v and edge node e in time slot t. S_v^V and S_e^E

TABLE II
TASK OFFLOADING RATIO OF DIFFERENT CACHING CASES

Cases for caching		Task offloading ratio			
		Task execution in vehicle	Task execution in the nearest edge	Task execution in edge pool	Task execution in cloud
Case 1	$c^V_{v,k}(t)=1, c^E_{e,k}(t)=0, \sum_{i=1,i\neq e}^{N_E} c^E_{e,k}(t)\geq 0$	$1-o^V_{v,k}(t)=1$	$o^V_{v,k}(t)=0$	0	0
Case 2	$c^V_{v,k}(t)=1, c^E_{e,k}(t)=1, \sum_{i=1,i\neq e}^{N_E} c^E_{e,k}(t)\geq 0$	$1-o^V_{v,k}(t)$	$o^V_{v,k}(t)$	0	0
Case 3	$c^V_{v,k}(t)=0, c^E_{e,k}(t)=1, \sum_{i=1,i\neq e}^{N_E} c^E_{e,k}(t)=0$	$1-o^V_{v,k}(t)=0$	$o^V_{v,k}(t)(1-o^E_{e,k}(t))=1$	$o^V_{v,k}(t)o^E_{e,k}(t)=0$	0
Case 4	$c^V_{v,k}(t)=0, c^E_{e,k}(t)=1, \sum_{i=1,i\neq e}^{N_E} c^E_{e,k}(t)>0$	$1-o^V_{v,k}(t)=0$	$o^V_{v,k}(t)(1-o^E_{e,k}(t))$	$o^V_{v,k}(t)o^E_{e,k}(t)$	0
Case 5	$c^V_{v,k}(t)=0, c^E_{e,k}(t)=0, \sum_{i=1,i\neq e}^{N_E} c^E_{e,k}(t)>0$	$1-o^V_{v,k}(t)=0$	$o^V_{v,k}(t)(1-o^E_{e,k}(t))=0$	$o^V_{v,k}(t)o^E_{e,k}(t)=1$	0
Case 6	$c^V_{v,k}(t)=0, c^E_{e,k}(t)=0, \sum_{i=1,i\neq e}^{N_E} c^E_{e,k}(t)=0$	0	0	0	1

are the storage capacity of each vehicle and each edge server, respectively.

In our system model, a partial offloading strategy is used for vehicles' tasks in each time slot, as illustrated in Fig. 2. For example, when vehicle v within communication range of edge e initiates an offloading request for task k, if the corresponding service program is not locally cached, this task will be completely offloaded to the nearest edge. Then if the edge pool does not have the needed service program for task k precached, the task will be uploaded to the cloud for execution (Case 6 in Fig. 2). If program k is cached in both the vehicle and its nearest edge node, the task can be handled at edge nodes by partial offloading (Case 2 in Fig. 2). If program k is cached in the nearest edge node and the edge pool but not in the vehicle itself, the vehicle completely offloads task k to the nearest edge node, and then task k is partially offloaded from the nearest edge node to the edge pool (Case 4 in Fig. 2). As can be seen from Fig. 2, tasks are offloaded to different computing nodes based on the corresponding service caching strategies. Therefore, caching and computing resources need to be properly allocated by the agent to achieve maximum benefits.

D. Computation Delay and Energy Consumption

Following [10], the time and energy consumption for the computation of task k are calculated as

$$D_k = \frac{\omega_k}{f_k}, \tag{5}$$

and

$$\varepsilon_k = \kappa f_k^{\alpha} D_k = \kappa \omega_k (f_k)^{\alpha-1}, \tag{6}$$

respectively, where f_k denotes the CPU frequency and is constrained by a maximum frequency f_{max} (i.e., $f_k < f_{max}$), κ ($\kappa > 0$) denotes the computing energy efficiency parameter, and α (in this work we assume that $\alpha = 2$) denotes the exponent parameter. ω_k denotes the number of cycles needed for service program k, which is expressed as the number of computation input data d_k multiplied by a factor λ, i.e. $\omega_k = \lambda d_k$. Here λ ($\lambda > 0$) is determined based on the nature of service application, (e.g., computational complexity) [40]. f^V and f^E denote the fixed CPU frequencies of the vehicle and the edge server, respectively.

When vehicle v within communication range of the nearest edge e initiates a offloading request for task k, the agent selects device nodes that task k should be offloaded to according to the cache placement of the service program, and determines the offload ratio for resource allocation. Let $o^V_{v,k}(t) \in [0,1], o^E_{e,k}(t) \in [0,1]$ be a continuous decision variable to denote the ratio of task k offloaded to the nearest edge and edge pool, and (1-$o^V_{v,k}(t)$) and (1-$o^E_{e,k}(t)$) bes the remaining task that is locally executed at vehicle v and the nearest edge e, respectively. Table II lists the mathematical expressions of the task offloading ratio in different caching cases (corresponding to those in Fig. 2). Based on those, the computation time and transmission time of tasks involving different caching cases are derived as follows.

Then the local execution delay of task k in vehicle v at time slot t can be given as

$$T^{local}_{v,e,k}(t) = c^V_{v,k}(t)(1-o^V_{v,k}(t))\frac{\lambda d_k}{f^V}. \tag{7}$$

If task k needs to be offloaded to the nearest edge e, the time consumption for the input data offloading of task k is

$$T^{up}_{v,e,k}(t) = \begin{cases} \varphi(\sum_{e=1}^{N_E} c^E_{e,k}(t)) o^V_{v,k}(t) \frac{d_k}{R_{v,e}(t)}, & R_{v,e}(t) > 0, \\ 0, & R_{v,e}(t) = 0, \end{cases} \tag{8}$$

where

$$\varphi(x) = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases} \tag{9}$$

In (8), $R_{v,e}(t) = 0$ indicates that the caching scheme belongs to Case 1, and tasks do not need to be offloaded to edge nodes.

The execution time on this nearest edge is expressed as

$$T^{edge}_{v,e,k}(t) = c^E_{e,k}(t) o^V_{v,k}(t)(1 - o^E_{e,k}(t))\frac{\lambda d_k}{f^E}. \tag{10}$$

If the nearest edge server e is unavailable and task k needs to be further offloaded to other edge servers in the edge pool, the uploading time is obtained as

$$\begin{aligned} T^{uppool}_{v,e,k}(t) =&(1 - c^V_{v,k}(t))\varphi(\sum_{i=1,i\neq e}^{N_E} c^E_{e,k}(t)) o^V_{v,k}(t) o^E_{e,k}(t) \\ &\times \frac{d_k}{R_{edge}}, \end{aligned} \tag{11}$$

where R_{edge} denotes the transmission rate among edge servers. Then, the delay resulting from the computation at edge pool can be given as

$$\begin{aligned} T^{pool}_{v,e,k}(t) =&(1 - c^V_{v,k}(t))\varphi(\sum_{i=1,i\neq e}^{N_E} c^E_{e,k}(t)) o^V_{v,k}(t) o^E_{e,k}(t) \\ &\times \frac{\lambda d_k}{f^E}. \end{aligned} \tag{12}$$

Since the output data is typically much smaller than the input task size, we ignore the time delay of the output return process [31]. As the computing and storage resources are abundant in the cloud, we ignore the queuing and computing latency incurred by the cloud. However, users need to consider the latency for task offloading to the cloud. Considering the transmission delay and calculation delay under different caching and offloading decisions, the total delay of processing task k in time slot t can be expressed as

$$\begin{aligned} T^{total}_{v,e,k}(t) =& \max\{T^{local}_{v,e,k}(t), T^{up}_{v,e,k}(t) \\ &+ \max\{T^{edge}_{v,e,k}(t), T^{uppool}_{v,e,k}(t) + T^{pool}_{v,e,k}(t)\}\} \\ &+ (1 - c^V_{v,k}(t))(1 - \varphi(\sum_{e=1}^{N_E} c^E_{e,k}(t))) \\ &\times (\frac{d_k}{R_{v,e}(t)} + \frac{d_k}{R_{cloud}}), \end{aligned} \tag{13}$$

which can represent all the cases of service caching and task offloading in Table II. R_{cloud} denotes the transmission rate from an edge server to the cloud.

Additionally, the energy consumption of the computing and offloading of task k on the edge server in time slot t is

$$\begin{aligned} \varepsilon^{total}_{v,e,k}(t) =&\kappa(f^E)^2(T^{edge}_{v,e,k}(t) + T^{pool}_{v,e,k}(t)) + p_{edge} T^{uppool}_{v,e,k}(t) \\ &+ (1 - c^V_{v,k}(t))(1 - \varphi(\sum_{e=1}^{N_E} c^E_{e,k}(t)))\frac{p_{cloud} d_k}{R_{cloud}}, \end{aligned} \tag{14}$$

where p_{edge} denotes the transmission power from an edge server to another edge server and p_{cloud} denotes the transmission power of communication from an edge server to the cloud.

We define the average delay of task processing in time slot t as

$$T_d(t) = \frac{1}{N_E}\sum_{e=1}^{N_E} \frac{1}{N^{task}_e(t)} \sum_{v=1}^{N^{task}_e(t)} T^{total}_{v,e,k}(t). \tag{15}$$

Similarly, the average energy consumption of edge servers in time slot t can be calculated as

$$\varepsilon(t) = \frac{1}{N_E}\sum_{e=1}^{N_E} \frac{1}{N^{task}_e(t)} \sum_{v=1}^{N^{task}_e(t)} \varepsilon^{total}_{v,e,k}(t). \tag{16}$$

E. Problem Formulation

With the emerging of interactive AR/VR services, user experience is crucial for users' viscosity, which is the key to the success of those services. As an important aspect of the user experience, the task processing delay is gradually becoming a crucial wireless network performance metric. The main goal of our work is to design a joint computation offloading and service caching scheme for the purpose of minimizing long-term cumulative average task processing time. Specifically, the optimization problem can be described as:

$$\min_{\{c^E_{e,k}(t), o^V_{v,k}(t), o^E_{e,k}(t)\}} \quad \sum_{t=1}^{t_{end}} \gamma^{t-1}(\frac{T_d(t)}{T_{dmax}}), \tag{17}$$

$$s.t.$$

$$c^V_{v,k}(t) \in \{0,1\}, \forall v \in V, \forall k \in K, \forall t \in \{1,2,...,t_{end}\}, \tag{17a}$$

$$c^E_{e,k}(t) \in \{0,1\}, \forall e \in E, k, t, \tag{17b}$$

$$0 \leq o^V_{v,k}(t) \leq 1, \forall v, k, t, \tag{17c}$$

$$0 \leq o^E_{e,k}(t) \leq 1, \forall e, k, t, \tag{17d}$$

$$0 \leq \gamma \leq 1; \tag{17e}$$

$$\sum_{k=1}^{N_K} c^V_{v,k}(t)\theta_k \leq S^V_v, \forall v, t, \tag{17f}$$

$$\sum_{k=1}^{N_K} c^E_{e,k}(t)\theta_k \leq S^E_e, \forall e, t. \tag{17g}$$

Where γ is the discounted factor. T_{dmax} is the maximum tolerable delay (constant) and $T_{dmax} = \max\{\max\{T_d(t)\}\} = \max\{\max\{T^{total}_{v,e,k}(t)\}\}$. Constraints (17f) and (17g) are the caching capacity limitation of each vehicle and edge server. Note that the problem is a long-term MINLP problem and NP-hard. In order to solve this optimization problem, the key is to make appropriate decisions on task offloading and service caching at each time period. Moreover, constantly changing status of vehicle participation and ephemeral interactions increase the operation complexity of edge management controller. The system state space becomes large with the increasing of vehicles and edges. Thus, we need to find an effective approach to address these issues.

F. The extreme value analysis of $T^{total}_{v,e,k}(t)$ and $\varepsilon^{total}_{v,e,k}(t)$

In order to evaluate the maximum and minimum values of $T^{total}_{v,e,k}(t)$, we need to derive the task processing delay in

time slot t under different caching cases, in which the caching decision variable $c^V_{v,k}(t), c^E_{e,k}(t)$ are not relevant.

1) Case 1:

$$T_d^{Case1}(t) = \frac{\lambda d_k}{f^V}. \tag{18}$$

2) Case 2:

$$\begin{aligned} T_d^{Case2}(t) = &\max\{(1 - o^V_{v,k}(t))\frac{\lambda d_k}{f^V}, \\ &o^V_{v,k}(t)(\frac{d_k}{R_{v,e}(t)} + \frac{\lambda d_k}{f^E})\}. \end{aligned} \tag{19}$$

3) Case 3:

$$T_d^{Case3}(t) = \frac{d_k}{R_{v,e}(t)} + \frac{\lambda d_k}{f^E}. \tag{20}$$

4) Case 4:

$$\begin{aligned} T_d^{Case4}(t) = &\frac{d_k}{R_{v,e}} + \max\{(1 - o^E_{e,k}(t))\frac{\lambda d_k}{f^E}, \\ &o^E_{e,k}(t)(\frac{d_k}{R_{edge}} + \frac{\lambda d_k}{f^E})\}. \end{aligned} \tag{21}$$

5) Case 5:

$$T_d^{Case5}(t) = \frac{d_k}{R_{v,e}(t)} + \frac{d_k}{R_{edge}} + \frac{\lambda d_k}{f^E}. \tag{22}$$

6) Case 6:

$$T_d^{Case6}(t) = \frac{d_k}{R_{v,e}(t)} + \frac{d_k}{R_{cloud}}. \tag{23}$$

From the above analysis, we have

$$\begin{aligned} \max\{T^{total}_{v,e,k}(t)\} &= \max_{\{i=1,\dots 6\}}\{T_d^{Casei}(t)\} \\ &= \max\{T_d^{Case1}(t), T_d^{Case5}(t), T_d^{Case6}(t)\} \\ &= \max\{\frac{\lambda d_k}{f^V}, \frac{d_k}{R_{v,e}(t)} + \frac{d_k}{R_{edge}} + \frac{\lambda d_k}{f^E}, \\ &\quad \frac{d_k}{R_{v,e}(t)} + \frac{d_k}{R_{cloud}}\}. \end{aligned} \tag{24}$$

$$\begin{aligned} \min\{T^{total}_{v,e,k}(t)\} &= \min_{\{i=1,\dots 6\}}\{T_d^{Casei}(t)\} \\ &= \min\{T_d^{Case2}(t), T_d^{Case4}(t), T_d^{Case6}(t)\}. \end{aligned} \tag{25}$$

In order to eliminate the offloading decision variables, it is easy to obtain the minimum delays of Case 2 and Case 4 based on their properties as piecewise linear functions.

$$\min\{T_d^{Case2}(t)\} = \frac{d_k(f^E + \lambda R_{v,e}(t))}{\frac{f^V f^E}{\lambda} + f^V R_{v,e}(t) + f^E R_{v,e}(t)}, \tag{26}$$

where $o_{v,k}(t) = 1/(1 + \frac{f^V}{\lambda}(\frac{1}{R_{v,e}(t)} + \frac{\lambda}{f^E}))$.

$$\min\{T_d^{Case4}(t)\} = \frac{d_k}{R_{v,e}(t)} + \frac{\lambda d_k(\lambda R_{edge} + f^E)}{f^E(2\lambda R_{edge} + f^E)}, \tag{27}$$

where $o_{e,k}(t) = 1/(2 + \frac{f^E}{\lambda R_{edge}})$.

Hence, the minimum value of $T^{total}_{v,e,k}(t)$ can be expressed as

$$\begin{aligned} \min\{T^{total}_{v,e,k}(t)\} = &\min\{\min\{T_d^{Case2}(t)\}, \min\{T_d^{Case4}(t)\}, \\ &T_d^{Case6}(t)\}. \end{aligned} \tag{28}$$

The analysis above indicates that the maximum and minimum task processing delays are related to the communication, computing and caching capabilities of each device. Meanwhile, aiming at minimizing the delay, cache decisions corresponding to the maximum tolerated delay should be avoided as much as possible.

Similarly, in order to evaluate the maximum and minimum values of $\varepsilon^{total}_{v,e,k}(t)$, we need to derive the energy consumption of edge servers in time slot t under different caching cases.

1) Case 1:

$$\varepsilon^{Case1}(t) = 0. \tag{29}$$

2) Case 2:

$$\varepsilon^{Case2}(t) = \kappa(f^E)^2(o_{v,k}(t)\frac{\lambda d_k}{f^E}). \tag{30}$$

3) Case 3:

$$\varepsilon^{Case3}(t) = \kappa(f^E)^2(\frac{\lambda d_k}{f^E}). \tag{31}$$

4) Case 4:

$$\begin{aligned} \varepsilon^{Case4}(t) &= \kappa(f^E)^2((1 - o_{e,k}(t))\frac{\lambda d_k}{f^E} + o_{e,k}(t)\frac{\lambda d_k}{f^E}) + \\ &\quad o_{e,k}(t)p_{edge}\frac{d_k}{R_{edge}} \\ &= \kappa f^E \lambda d_k + o_{e,k}(t)p_{edge}\frac{d_k}{R_{edge}}. \end{aligned} \tag{32}$$

5) Case 5:

$$\varepsilon^{Case5}(t) = \kappa f^E \lambda d_k + p_{edge}\frac{d_k}{R_{edge}}. \tag{33}$$

6) Case 6:

$$\varepsilon^{Case6}(t) = p_{cloud}\frac{d_k}{R_{cloud}}. \tag{34}$$

Based on the above analysis, we have

$$\begin{aligned} \max\{\varepsilon^{total}_{v,e,k}(t)\} &= \max_{\{i=1,\dots 6\}}\{\varepsilon^{Casei}(t)\} \\ &= \max\{\varepsilon^{Case5}(t), \varepsilon^{Case6}(t)\} \\ &= \max\{\kappa f^E \lambda d_k + p_{edge}\frac{d_k}{R_{edge}}, \\ &\quad p_{cloud}\frac{d_k}{R_{cloud}}\}. \end{aligned} \tag{35}$$

From (29) it can be easily observed that $\min\{\varepsilon^{total}_{v,e,k}(t)\} = 0$.

From the analysis of the maximum and minimum values of $T^{total}_{v,e,k}(t)$, it can be seen that in order to reduce the task processing delay, it is necessary to make Case 2 and Case 4 caching decisions as many as possible. That is, tasks should be offloaded to the edge nodes as much as possible. On the other hand, energy consumption is also a critical indicator for edge server operators. Offloading tasks to edge nodes can reduce task processing delays, but it increases energy consumption of edge servers.

IV. Deep Reinforcement Learning for Edge Caching and Offloading

Due to diverse user demands and the constrained computation and caching resources, it is complex to minimize the cumulative system average delay in (17). Most traditional optimization methods (e.g. convex optimization, game theory, etc.) are assumed to have the knowledge of key factors in vehicular networks, such as channel conditions and content popularity. However, these key factors are time-varying and unavailable in reality. These methods can merely achieve optimal or near optimal results for one snapshot, since they ignore how the current decision exerts long-term influence on resource allocation [21]. DRL is viewed as an efficient way to solve the complicated problem in a dynamic environment by optimizing the expected cumulative reward. In DRL, an agent collects the needed information regarding diverse demands of users and available resources in vehicular networks. Then the agent takes an action to manage offloading and caching decisions and optimizes resource allocation. We formulate the joint optimization problem as a discrete-time markov decision process (MDP) and propose a DDPG-based edge caching and offloading scheme for joint service caching and computation offloading strategy design.

A. Problem Formulation Based on DRL

1) State space: At the initial phase of each time slot t, each edge server gathers all the environmental parameters, which contains the following parts:

- $I_{v,k}(t)$: The request indicator for task k by vehicle v within the coverage range of edge node e at time slot t,
- $c^V_{v,k}(t)$: The service caching indicator for vehicle v within the coverage range of edge node e at time slot t.
- $B_{v,e}(t), \gamma_{v,e}(t)$: The bandwidth that edge node e allocates to vehicle v and the received SINR of edge node e at time slot t.
- $c^E_{e,k}(t)$: The service caching indicator of edge node e at time slot t.

The state of edge node e at time slot t is denoted as

$$\begin{aligned}\mathbf{s}_e(t) =&\{[\mathbf{I}(t)]^{\rho_{max}\times N_K}, [\mathbf{c}^V(t)]^{\rho_{max}\times N_K}, [\mathbf{B}(t)]^{\rho_{max}\times N_K},\\ &[\boldsymbol{\gamma}(t)]^{\rho_{max}\times N_K}, [\mathbf{c}^E(t)]^{\rho_{max}\times N_K}\},\end{aligned} \tag{36}$$

[2] where ρ_{max} denotes the maximum vehicle density within each edge node. The system state at time slot t consists of the states of all edge nodes, defined as

$$s_t = \{s_1(t), s_2(t), ..., s_{N_E}(t)\}. \tag{37}$$

Where the states of edge node e is denoted as $s_e(t)$ after dimensionality reduction and normalization.

2) Action space: The agent obtains the states of service caching and communication information between vehicles and edge nodes, and then decides the offloading ratio of all tasks and updates service caching of all edge nodes. Here, the corresponding action contains the following parts:

[2] Bold letters are used to denote matrices.

- $c^E_{e,k}(t)$: The service caching indicator of edge node e after completing all task offloading and calculation in time slot t.
- $o^V_{v,k}(t)$: The proportion of vehicle v offloading task k to edge node e in time slot t.
- $o^E_{e,k}(t)$: The proportion of edge node e offloading task k to edge pool in time slot t.

The action of edge node e in time slot t is denoted as

$$\mathbf{a}_e(t) =\{[\mathbf{c}^E(t)]^{\rho_{max}\times N_K}, [\mathbf{o}^V(t)]^{\rho_{max}\times N_K}, [\mathbf{o}^E(t)]^{\rho_{max}\times N_K}\}. \tag{38}$$

Then the actions of all edge nodes are denoted as $a_e(t)$ after dimensionality reduction and normalization. The system action at time slot t consists of the actions of all edge nodes, defined as

$$a_t = \{a_1(t), a_2(t), ..., a_{N_E}(t)\}. \tag{39}$$

3) Reward: The agent's behavior is reward-based, and the reward should correspond with the objective function. Hence, we set the reward in time slot t as

$$r_t = r(s_t, a_t) = -(\frac{T_d(t)}{T_{dmax}}). \tag{40}$$

B. DDPG-Based Edge Caching and Offloading Scheme

Since the state space consists of a great amount of dynamic environmental information and the action space contains continuous value, we exploit the deep deterministic policy gradient (DDPG) algorithm, a model-free and actor-critic algorithm, to solve the joint computation offloading and service caching problem. The framework of DDPG-based method is illustrated in Fig. 3, consisting of primary networks, target networks and a replay buffer.

Based on the deterministic actor-critic model, we leverage deep neural networks to provide accurate estimation of deterministic policy function $\mu(s_t)$ and action-value function $Q(s_t, a_t)$, which should satisfy the following condition:

$$\begin{aligned}Q(s_t, \mu(s_t|\theta^\mu)|\theta^Q) \approx& r(s_t, a_t)+\\ &\gamma Q(s_{t+1}, \mu(s_{t+1}|\theta^{\mu'})|\theta^{Q'}).\end{aligned} \tag{41}$$

As shown in Fig. 3, The primary networks use the actor network $\mu(s_t|\theta^\mu)$ and the critic network $Q(s_t, a_t|\theta^Q)$ to approximate the policy function and the Q-value function, respectively. In addition, The target networks contain a actor network $\mu(s_t|\theta^{\mu'})$ and a critic network $Q(s_t, a_t|\theta^{Q'})$ with the same structure. The target Q-value can be represented as

$$y_t = r(s_t, a_t) + \gamma Q_\mu(s_{t+1}, \mu(s_{t+1}|\theta^{\mu'})|\theta^{Q'}). \tag{42}$$

We utilize the actor network to explore the policies and the critic network to critic the policies. The actor network architecture is illustrated in Fig. 4, which takes the state s_t as input, and outputs an action a_t. The action variable $c^E_{e,k}(t)$ needs to be discretized (i.e. $\lceil c^E_{e,k}(t))\rceil$).

At the beginning of each time slot t, the agent collects environmental information to get the current system state s_t. In order to solve the exploration problem of deterministic policy, we construct the action space by adding behavior noise n_t to obtain action $a_t = \mu(s_t|\theta^\mu) + n_t$. After vehicles and

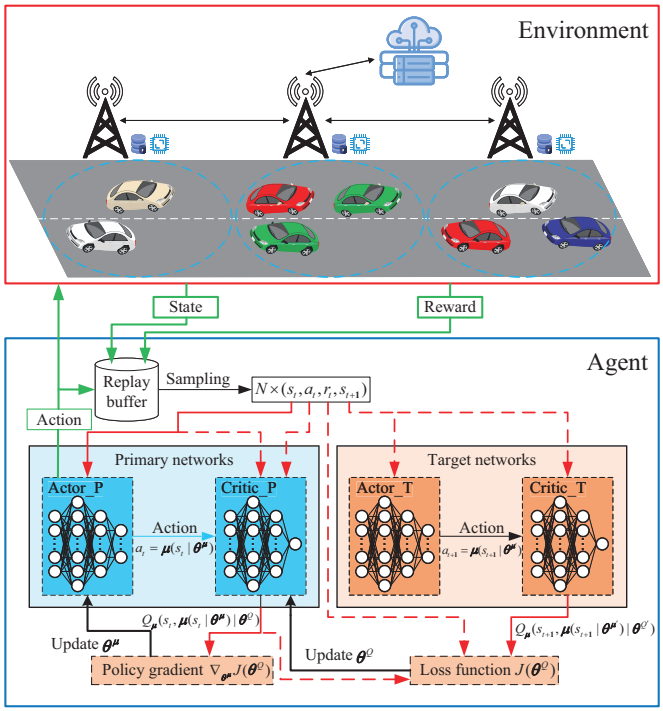

Fig. 3. The framework of the DDPG-based method for vehicular edge caching and offloading.

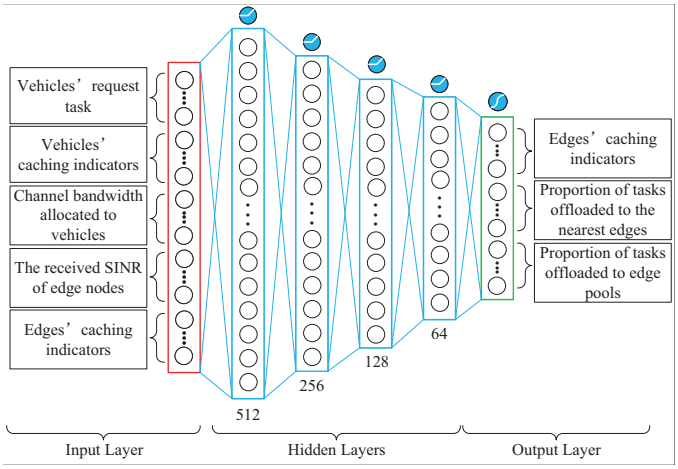

Fig. 4. The actor network architecture.

edges carry out the computing offloading and service caching scheme based on action a_t, the agent can observe the next state s_{t+1} and the immediate reward r_t. Then the transition (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer. This operation can avoid sample-correlation during the training process. After that, the algorithm randomly selects N transitions from the replay buffer to make up a mini-batch sample and inputs it into primary networks and target networks to update network parameters. Then, we update parameter θ^Q in the primary critic network by minimizing the loss function, i.e.,

$$J(\theta^Q) = \frac{1}{N}\sum_{j=1}^{N}(y_j - Q_\mu(s_j, \mu(s_j|\theta^\mu)|\theta^Q))^2. \tag{43}$$

The parameter θ^μ in the primary actor network is updated by the policy gradient, which can be expressed as

$$\begin{aligned}\bigtriangledown_{\theta^\mu} J(\theta^Q) =& \frac{1}{N}\sum_{j=1}^{N}[\bigtriangledown_a Q(s, a|\theta^Q)|_{s=s_j, a=\mu(s_j|\theta^\mu)} \\ &\times \bigtriangledown_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_j}].\end{aligned} \tag{44}$$

Finally, we utilize the soft updating method instead of copying the parameters θ^Q and θ^μ to partially update parameters of the target networks, which can be expressed as

$$\begin{aligned}\theta^{Q'} &\leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}, \\ \theta^{\mu'} &\leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'},\end{aligned} \tag{45}$$

where τ is the updating coefficient. The whole process of the DDPG-based edge caching and offloading scheme is presented in Algorithm 1.

Algorithm 1 DDPG-Based Edge Caching and Offloading Algorithm

Input: The initial parameters: γ, θ^μ, $\theta^{\mu'}$, θ^Q, $\theta^{Q'}$ τ, M, t_{end}, D, N.
Output: Primary optimal actor network parameter θ^μ.

```
Initialize primary networks and target networks.
Empty the experience replay buffer D.
for episode = 1, 2, ...M do
    Initial observation state s_0.
    Add a random Gaussian distributed behavior noise n_t for action exploration.
    for t = 1, 2, ...t_end do
        Agent receives normalized observation state s_t.
        Select action a_t = μ(s_t|θ^μ) + n_t.
        Perform action a_t, calculate immediate reward r_t, and obtain the next normalized state s_{t+1}.
        if the replay buffer is not full then
            Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer D.
        else
            Randomly replace a transition in replay buffer D with (s_t, a_t, r_t, s_{t+1}).
            Randomly sample a mini-batch of N transitions (s_j, a_j, r_j, s_{j+1}), ∀j = 1, 2, ..., N from D.
            Set y_j = r(s_j, a_j) + γQ_μ(s_{j+1}, μ(s_{j+1}|θ^{μ'})|θ^{Q'}).
            Update the θ^Q in critic network by minimizing the loss according to (43).
            Update actor network θ^μ by the gradient of the policy according to (44).
            Update target networks according to (45).
        end if
    end for
end for
```

V. Performance Evaluation

In this section, we evaluate the performances of the proposed scheme through numerical simulations. We first provide experimental settings in Sec. V-A and then present extensive simulation results in Sec. V-B.

TABLE III
SIMULATION PARAMETERS

Parameter	Value
Total number of time slots (t_{end})	40
The duration for each time slot Δt	30 s
Density of vehicles within range of edge (ρ)	[2,5]
Each edge communication range	500 m
Bandwidth of each edge server (B)	20 MHz
The value of SINR	4∼5 dB
Number of edge servers (N_E)	3
Number of service types (N_K)	5
Data size of each task (d_k)	20 Mb
Storage space of each service program (θ_k)	50 Mb
CPU cycles of each vehicle (f^V)	5×10^8 cycles/s
CPU cycles of each edge server (f^E)	1×10^9 cycles/s
Computation intensity for each task (λ)	10^5 cycles/bit
Storage capacity of each vehicle (S_v^V)	50 Mb
Storage capacity of each edge server (S_e^E)	100 Mb
The computing energy efficiency parameter (κ)	1×10^{-26}
Transmission rate between edge servers (R_{edge})	15 Mbps
Transmission rate from edge to the cloud R_{cloud}	10 Mbps
Transmission power between edge servers (p_{edge})	1 W
Transmission power from edge to the cloud (p_{cloud})	2 W
Discount factor (γ)	0.99
Learning rate of actor network (lr_a)	0.001
Learning rate of critic network (lr_c)	0.002
Soft update coefficient (τ)	0.01
Size of mini-batch sample (N)	128
The size of experience replay buffer (D)	10000

A. Experimental Settings

The involved parameters along with their corresponding values are listed in Table III. These previous works [10], [12], [31] can bring rich experience to the parameter settings, including the duration for each time slot Δt, κ, ϵ and so on. For instance, Δt should be set appropriately such that on one hand the agent has enough time to make caching and offloading decisions, and on the other hand tasks offloaded to edge server can be completely accomplished by the end of time slot. We use Python 3.6 to create a simulation environment for the considered vehicular edge caching and task offloading system. In the simulation, each vehicle randomly requests its task of interest at the beginning of each time slot. The duration of each time slot is set appropriately such that the agent has sufficient time to make caching and offloading decisions, while tasks offloaded to other nodes can be accomplished by the end of a time slot. Furthermore, we use TensorFlow 1.14.0 to implement the DDPG-based edge caching and offloading scheme.

We consider the following benchmark methods for performance comparison:

1) *Offloading without edge caching:* Tasks requested by the vehicle are computed locally or offloaded to the cloud.

2) *Offloading based on latency minimization:* The task offloading is performed according to the optimal ratio to achieve the minimum latency under each case based on the DDPG learning algorithm.

3) *Offloading based on energy minimization:* The task offloading is performed according to the optimal ratio to achieve the minimum energy consumption under each case based on the DDPG learning algorithm.

4) *Random edge caching and offloading:* The service caching and task offloading ratios are random in each time slot.

5) *Least recently used (LRU) edge caching and offloading:* The services requested by the edge server in the previous time slot continue to be cached in the next time slot, the services that have not been requested are randomly replaced [41], and the offloading ratio is determined according to the offloading scheme based on latency minimization.

6) *Executing all tasks in the cloud:* All tasks are offloaded to the cloud for execution.

B. Simulation Results

First, we compare the total delay per episode of different schemes based on the DDPG learning algorithm. It can be seen in Fig. 5 that all schemes can approach their stable cumulative average delay as the number of episodes increases. Meanwhile, since the energy consumption is related to the task processing delay, we evaluate the total energy consumption per episode in Fig. 6. As the number of episodes increases, except for the *offloading without edge caching scheme*, the total delay of all the other considered schemes decreases and reaches a stable value, while the total energy consumption increases and reaches a stable value. As analyzed in Sec. III-F, to reduce latency, it is necessary to offload tasks to edge nodes as much as possible, which on the other hand increases the energy consumption of edge nodes. This is verified in Fig. 5 and Fig. 6. Another notable point is that our proposed scheme achieves the lowest task processing latency with the same energy consumption of edge nodes. The *offloading without edge caching scheme* keeps the maximum delay, which is approximately the same as the total delay that the *offloading scheme based on energy minimization* converges to. This is because when aiming to minimize energy consumption, the offloading ratios are determined as the ones that minimizes energy consumption in all the considered cases. However, *offloading without edge caching scheme* consumes the maximum energy. The *offloading without edge caching scheme* remains smooth because agent cannot participate in decision-making, and edge servers cannot provide computing and caching resources. Our proposed scheme can yield the lowest total delay compared with the other benchmark schemes, which demonstrates the efficiency of DDPG-Based edge caching and offloading scheme.

Next, we investigate the effects of vehicle density on the total delay and energy consumption for different schemes in Fig. 7. As the density of vehicle ρ increases, the number of task requests increases, while the bandwidth allocated to each vehicle and the transmission rate of tasks uploaded to edge nodes decrease, resulting in an increase in the total task processing delay. When $\rho = 2, 3$, the total latency of the *executing all tasks in the cloud scheme* is lower than that of the *offloading without edge caching scheme*. This is because as each vehicle occupies more bandwidth resources, tasks can be uploaded faster to the cloud, which has powerful computing resources. As the vehicle density continues to increase, this superiority vanishes. It can be observed that our proposed scheme outperforms the other methods in terms of the total delay in the period t_{end}.

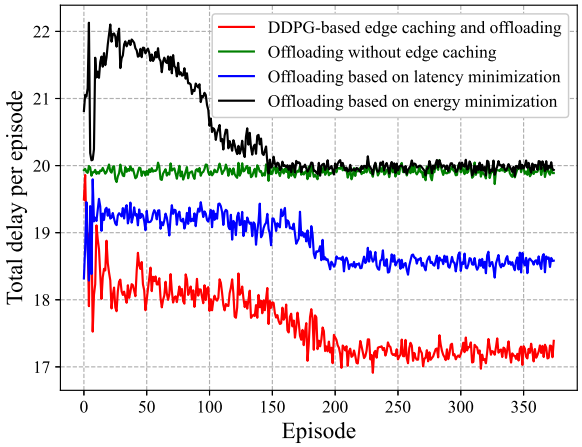

Fig. 5. The performance of total delay per episode.

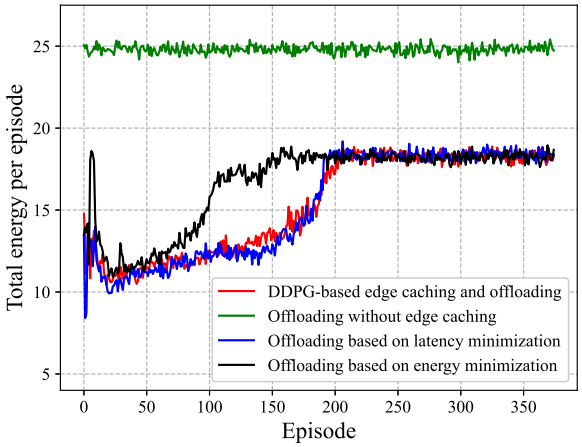

Fig. 6. The performance of total energy per episode.

In order to explore the impacts of different caching decision schemes on latency and energy, we count the average number of edge caching case decisions made by different schemes in the t_{end} period, which is shown in Fig. 8. According to the parameter settings in Table III and the analysis in Sec. III. F, it can be deduced that $\max\{T^{total}_{v,e,k}(t)\} = T^{Case5}_{d}(t), \max\{\varepsilon^{total}_{v,e,k}(t)\} = \varepsilon^{Case6}(t)$. That is, greater number of caching decisions in Case 5 leads to higher task processing delay, and that in Case 6 leads to higher energy consumption at edge nodes. It can be observed that the DDPG-based edge caching and offloading scheme avoids caching decisions in cases with the largest delay as much as possible. This decision can greatly reduce the task processing delay. It is reasonable to observe that *offloading without edge caching scheme* only makes decisions in Case 1 and Case 6, and the *executing all tasks in the cloud scheme* only make Case 6 decisions. For the LRU and $random$ edge caching and offloading schemes, it can be seen that more than 85% of the tasks are executed locally or in the edge pool. This is because the cached service programs at edge nodes increase the task hit ratio, thereby having less energy consumption compared with that when tasks are uploaded to the cloud. Our proposed scheme can jointly optimize caching and offloading decisions, allocate caching and computing resources properly, and improve user experience within a reasonable range of energy consumption.

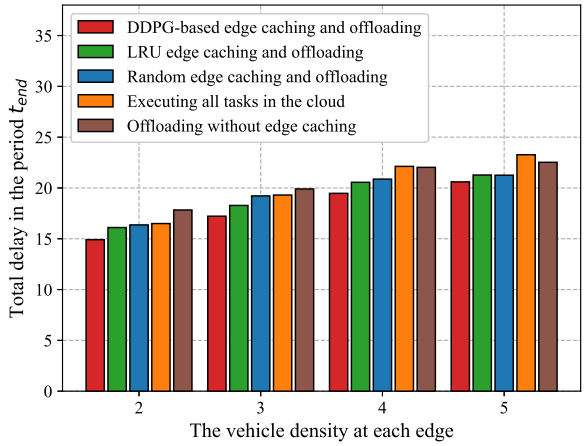

Fig. 7. The effect of the vehicle density ρ on the total task processing delay.

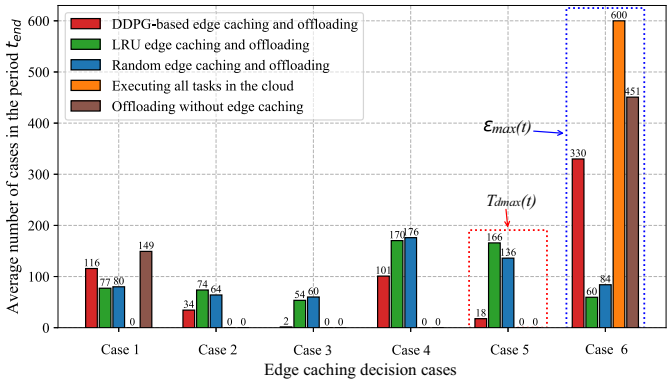

Fig. 8. The average number of edge caching decision cases in the period t_{end} with $\rho = 5$.

Then we investigate the effects of task size on unnormalized total latency and energy consumption using different schemes in Fig. 9 and Fig. 10, respectively. The total delay and energy consumption increases linearly with the task size, which is because that the functions of delay and energy are proportional to the task size d_k, as presented in eqs. (13) and (14). Larger task size increases the transmission delay, computation delay and total energy consumption for all the schemes. However, In order to avoid making Case 5 decisions with the largest task processing delay, more decisions are made in Case 6, which is the case with the largest energy consumption of edge nodes. This explains why our scheme has higher energy consumption than the *LRU* and the *random* edge caching and offloading schemes in Fig. 10.

Fig. 11 shows the impacts of the edge server cycle frequency on the unnormalized total delay using five different schemes. Certainly, higher edge server cycle frequency can reduce the total execution latency except for the *offloading without edge caching* and the *executing all tasks in the cloud schemes*. This is because the tasks of these two schemes do not perform computations at edge nodes, and are not related to the computing power of the edge servers. For the *LRU* and the *random* edge caching and offloading schemes, more tasks are offloaded to the edge pool for execution. As the cycle frequency increases, the total latency is lower than that of

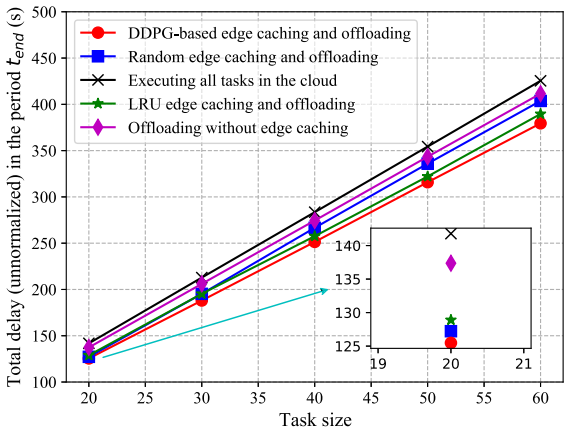

Fig. 9. The total delay (unnormalized) versus task size with $\rho = 5$.

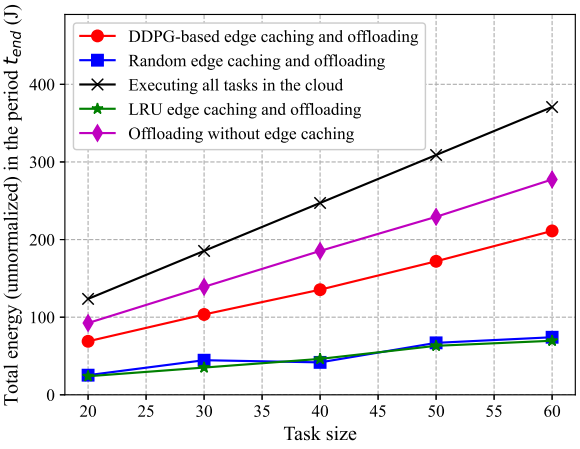

Fig. 10. The total energy (unnormalized) versus task size with $\rho = 5$.

the *executing all tasks in the cloud scheme* and *offloading without edge caching scheme*, indicating that the benefits of task computing on the edge server outweighs offloading to the cloud. In addition, due to randomness, the performances of the *random edge caching and offloading scheme* is quite unstable. Moreover, the result demonstrates that the proposed scheme can significantly reduce the task execution delay.

To conclude, the above evaluation results show that the proposed DDPG-based edge caching and offloading scheme can significantly reduce total delay of service caching and task offloading in dynamic vehicular environments.

VI. Conclusions and Future Work

This paper proposes a novel edge service caching and computation offloading framework in a general VEC system. To minimize the cumulative average task processing delay of task offloading and service caching, we formulate the optimization problem as a long-term MINLP problem, which is challenging to solve since service caching decisions and computation offloading decisions are strongly coupled. Furthermore, we deduce the boundaries of task processing delay and energy consumption in each case in detail. Considering the highly dynamic vehicular environments, we propose a DDPG-based scheme to update offloading decisions and service caching placements. Extensive simulations show that our proposed

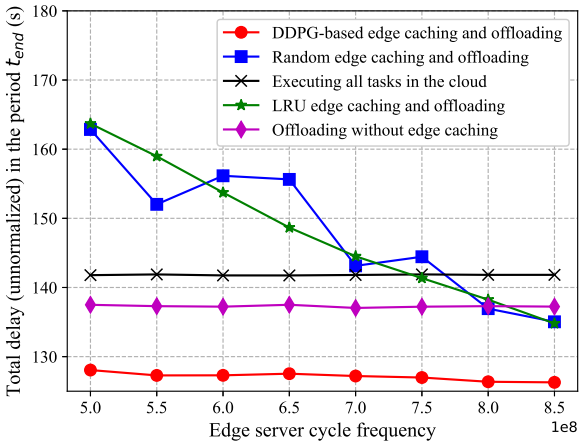

Fig. 11. The total delay (unnormalized) versus edge sever cycle frequency with $\rho = 5$.

scheme can effectively decrease the long-term average task processing delay by utilizing the available caching and computing resources and is easy to implement.

There are several interesting directions in future work. First, the cached service programs from different vehicles may be shared. This can effectively reduce task uploading cost, but on the other hand raises new technical challenges, such as privacy issues in service data sharing. Second, in this work, each vehicle is assumed to only generate a single task. We believe that the edge service caching and computation offloading with multi-tasking vehicles taken into consideration is an interesting step forward. Last but not least, the application of multiagent systems in edge service caching and computation offloading design is expected to become a powerful tool to solve more complex problems and remains much to be explored.

References

[1] X. Jiang, F. R. Yu, T. Song, and V. C. M. Leung, "Resource allocation of video streaming over vehicular networks: A survey, some research issues and challenges," *IEEE Trans. Intell. Transp. Syst.*, pp. 1–21, 2021, doi:10.1109/TITS.2021.3065209.

[2] L. T. Tan, R. Q. Hu, and L. Hanzo, "Twin-timescale artificial intelligence aided mobility-aware edge caching and computing in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3086–3099, Apr. 2019.

[3] Y. Dai, D. Xu, K. Zhang, S. Maharjan, and Y. Zhang, "Deep reinforcement learning and permissioned blockchain for content caching in vehicular edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 4, pp. 4312–4324, Apr. 2020.

[4] J. Chen, H. Wu, P. Yang, F. Lyu, and X. Shen, "Cooperative edge caching with location-based and popular contents for vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 9, pp. 10 291–10 305, Sep. 2020.

[5] Z. Xue, Y. Liu, G. Han, F. Ayaz, Z. Sheng, and Y. Wang, "Two-layer distributed content caching for infotainment applications in VANETs," *IEEE Internet Things J.*, vol. 9, no. 3, pp. 1696–1711, Feb. 2022.

[6] D. Gupta, S. Rani, A. Singh, and J. J. P. C. Rodrigues, "ICN based efficient content caching scheme for vehicular networks," *IEEE Trans. Intell. Transp. Syst.*, pp. 1–9, 2022, doi:10.1109/TITS.2022.3171662.

[7] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. Shen, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Trans. Multimedia*, vol. 21, no. 4, pp. 915–929, Apr. 2019.

[8] J. Yan, S. Bi, L. Duan, and Y.-J. A. Zhang, "Pricing-driven service caching and task offloading in mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 20, no. 7, pp. 4495–4512, Jul. 2021.

[9] S.-W. Ko, S. J. Kim, H. Jung, and S. W. Choi, "Computation offloading and service caching for mobile edge computing under personalized service preference," *IEEE Trans. Wireless Commun.*, pp. 1–16, 2022, doi:10.1109/TWC.2022.3151131.

[10] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4947–4963, Jul. 2020.
[11] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Honolulu, HI, USA, Apr. 2018, pp. 207–215.
[12] G. Zhang, S. Zhang, W. Zhang, Z. Shen, and L. Wang, "Joint service caching, computation offloading and resource allocation in mobile edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 20, no. 8, pp. 5288–5300, Aug. 2021.
[13] C. Tang, C. Zhu, H. Wu, Q. Li, and J. J. P. C. Rodrigues, "Toward response time minimization considering energy consumption in caching-assisted vehicular edge computing," *IEEE Internet Things J.*, vol. 9, no. 7, pp. 5051–5064, Apr. 2022.
[14] D. Lan, A. Taherkordi, F. Eliassen, and L. Liu, "Deep reinforcement learning for computation offloading and caching in fog-based vehicular networks," in *Proc. IEEE Int. Conf. Mob. Ad Hoc Sens. Syst (MASS)*, Delhi, India, Dec. 2020, pp. 622–630.
[15] S. David, L. Guy, H. Nicolas, D. Thomas, W. Daan, and R. Martin, "Deterministic policy gradient algorithms," in *Proc. 31st Int. Conf. Mach. Learn. (ICML)*, Beijing, China, Jun. 2014, pp. 387–395.
[16] L. Zhang, Y. Sun, Z. Chen, and S. Roy, "Communications-caching-computing resource allocation for bidirectional data computation in mobile edge networks," *IEEE Trans. Commun.*, vol. 69, no. 3, pp. 1496–1509, Mar. 2021.
[17] H. Tout, A. Mourad, N. Kara, and C. Talhi, "Multi-persona mobility: Joint cost-effective and resource-aware mobile-edge computation offloading," *IEEE/ACM Trans. Networking*, vol. 29, no. 3, pp. 1408–1421, Jun. 2021.
[18] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.
[19] I. A. Elgendy, W.-Z. Zhang, Y. Zeng, H. He, Y.-C. Tian, and Y. Yang, "Efficient and secure multi-user multi-task computation offloading for mobile-edge computing in mobile IoT networks," *IEEE Trans. Netw. Serv. Manage.*, vol. 17, no. 4, pp. 2410–2422, Dec. 2020.
[20] G. Qiao, S. Leng, S. Maharjan, Y. Zhang, and N. Ansari, "Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 247–257, Jan. 2020.
[21] Z. Ning, K. Zhang, X. Wang, M. S. Obaidat, L. Guo, X. Hu, B. Hu, Y. Guo, B. Sadoun, and R. Y. K. Kwok, "Joint computing and caching in 5G-envisioned internet of vehicles: A deep reinforcement learning-based traffic control system," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 8, pp. 5201–5212, Aug. 2021.
[22] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7944–7956, Aug. 2019.
[23] L.-H. Yen, J.-C. Hu, Y.-D. Lin, and B. Kar, "Decentralized configuration protocols for low-cost offloading from multiple edges to multiple vehicular fogs," *IEEE Trans. Veh. Technol.*, vol. 70, no. 1, pp. 872–885, Jan. 2021.
[24] M. S. Bute, P. Fan, L. Zhang, and F. Abbas, "An efficient distributed task offloading scheme for vehicular edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13 149–13 161, Dec. 2021.
[25] L. Dai, Y. Fang, Z. Yang, P. Chen, and Y. Li, "Protograph LDPC-coded BICM-ID with irregular CSK mapping in visible light communication systems," *IEEE Trans. Veh. Technol.*, vol. 70, no. 10, pp. 11 033–11 038, Oct. 2021.
[26] Y. Fang, Y. Bu, P. Chen, F. C. M. Lau, and S. A. Otaibi, "Irregular-mapped protograph LDPC-coded modulation: A bandwidth-efficient solution for 6G-enabled mobile networks," *IEEE Trans. Intell. Transp. Syst.*, pp. 1–14, 2021, doi:10.1109/TITS.2021.3122994.
[27] Y. Fang, G. Han, G. Cai, F. C. M. Lau, P. Chen, and Y. L. Guan, "Design guidelines of low-density parity-check codes for magnetic recording systems," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 2, pp. 1574–1606, Secondquarter 2018.
[28] Z. Qin, S. Leng, J. Zhou, and S. Mao, "Collaborative edge computing and caching in vehicular networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Seoul, Korea, May 2020, pp. 1–6.
[29] K. Zhang, J. Cao, H. Liu, S. Maharjan, and Y. Zhang, "Deep reinforcement learning for social-aware edge computing and caching in urban informatics," *IEEE Trans. Ind. Inf.*, vol. 16, no. 8, pp. 5467–5477, Aug. 2020.
[30] H. Tian, X. Xu, L. Qi, X. Zhang, W. Dou, S. Yu, and Q. Ni, "CoPace: Edge computation offloading and caching for self-driving with deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13 281–13 293, Dec. 2021.
[31] J. Wu, J. Wang, Q. Chen, Z. Yuan, P. Zhou, X. Wang, and C. Fu, "Resource allocation for delay-sensitive vehicle-to-multi-edges (V2Es) communications in vehicular networks: A multi-agent deep reinforcement learning approach," *IEEE Trans. Network Sci. Eng.*, vol. 8, no. 2, pp. 1873–1886, Apr. 2021.
[32] M. Xu, D. T. Hoang, J. Kang, D. Niyato, Q. Yan, and D. I. Kim, "Secure and reliable transfer learning framework for 6G-enabled internet of vehicles," *IEEE Wireless Commun.*, pp. 1–8, 2022, doi:10.1109/MWC.004.2100542.
[33] J. Kang, Z. Xiong, X. Li, Y. Zhang, D. Niyato, C. Leung, and C. Miao, "Optimizing task assignment for reliable blockchain-empowered federated edge learning," *IEEE Trans. Veh. Technol.*, vol. 70, no. 2, pp. 1910–1923, Feb. 2021.
[34] J. Liu, M. Ahmed, M. A. Mirza, W. U. Khan, D. Xu, J. Li, A. Aziz, and Z. Han, "RL/DRL meets vehicular task offloading using edge and vehicular cloudlet: A survey," *IEEE Internet Things J.*, pp. 1–25, 2022, doi:10.1109/JIOT.2022.3155667.
[35] 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.
[36] Q. Qi, J. Wang, Z. Ma, H. Sun, Y. Cao, L. Zhang, and J. Liao, "Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4192–4203, May 2019.
[37] B. Shang, L. Liu, and Z. Tian, "Deep learning-assisted energy-efficient task offloading in vehicular edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 70, no. 9, pp. 9619–9624, Sep. 2021.
[38] J. Shi, J. Du, J. Wang, and J. Yuan, "Deep reinforcement learning-based V2V partial computation offloading in vehicular fog computing," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Nanjing, China, Mar. 2021, pp. 1–6.
[39] S. M. A. Kazmi, S. Otoum, R. Hussain, and H. T. Mouftah, "A novel deep reinforcement learning-based approach for task-offloading in vehicular networks," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, Madrid, Spain, Dec. 2021, pp. 1–6.
[40] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.
[41] Z. Su, Y. Hui, Q. Xu, T. Yang, J. Liu, and Y. Jia, "An edge caching scheme to distribute content in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 6, pp. 5346–5356, Jun. 2018.

Zheng Xue received the B.E. degree in automotive service engineering from Chongqing Jiaotong University, Chongqing, China, in 2018, and the M.E. degree in electronic and communication engineering from the Guangdong University of Technology, Guangzhou, China, in 2021, where he is currently working toward the Ph.D. degree with the Department of Communication Engineering.

His primary research interests are vehicular networks and cooperative perception.

Chang Liu received the Ph.D. degree from Kansas State University, Manhattan, KS, USA, in 2016.

She is an Associate Professor with the Guangdong University of Technology, Guangzhou, China. Her current research areas include Internet of Vehicles and Internet of Things.

Canliang Liao received the B.E. degree in communication from Wuhan Polytechnic University, Wuhan, China, in 2019. He is currently working toward the M.E. degree with the Department of Communication Engineering, Guangdong University of Technology, Guangzhou, China.

His primary research interest is Internet of Vehicles.

Guojun Han received his Ph.D. from Sun Yatsen University, Guangzhou, China, and the M.E. degree from South China University of Technology, Guangzhou, China.

From March 2011 to August 2013, he was a Research Fellow at the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. From October 2013 to April 2014, he was a Research Associate at the Department of Electrical and Electronic Engineering, Hong Kong University of Science and Technology. He is now a Full Professor and Executive Dean at the School of Information Engineering, Guangdong University of Technology, Guangzhou, China. He has been a Senior Member of IEEE since 2014. His research interests are in the areas of wireless communications, signal processing, coding and information theory. He has more than 14 years experience on research and development of advanced channel coding and signal processing algorithms and techniques for various data storage and communication systems.

Zhengguo Sheng (Senior Member, IEEE) received the B.Sc. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2006, and the M.S. and Ph.D. degrees from Imperial College London, London, U.K., in 2007 and 2011, respectively.

He is currently a Senior Lecturer with the University of Sussex, Brighton, U.K. Previously, he was with UBC, Vancouver, BC, Canada, as a Research Associate and with Orange Labs, Santa Monica, CA, USA, as a Senior Researcher. He has more than 120 publications. His research interests cover IoT, vehicular communications, and cloud/edge computing.