Caching-Assisted Collaborative Task Offloading for Vehicular Edge Computing: A Deep Reinforcement Learning-Based Approach

Chaogang Tang, *Member, IEEE*, Shucai Wang, Huaming Wu, *Senior Member, IEEE*, and Ruidong Li, *Senior Member, IEEE*

***Abstract*—Collaborative task offloading in vehicular edge computing (VEC) primarily emphasizes the diversity of offloading destinations, such as cloud centers, roadside units (RSUs), and other entities with underutilized resources. However, it often neglects the collaborative potential among vehicles whose tasks are associated with the same service. In this paper, we propose a collaborative task offloading strategy from the perspective of vehicles with offloading requests. Vehicles collectively accomplish task offloading by dividing responsibilities for specific service component offloading. To enhance the performance of the VEC system, we introduce a caching-assisted collaborative task offloading strategy. An optimization problem is formulated to minimize the response latency of tasks in VEC. Due to the complexity of solving this Mixed Integer Nonlinear Programming (MINLP) problem, we decompose it into three subproblems: the Task Offloading and Service Caching (TOSA) problem, the Computing Resource Allocation (RA) problem, and the Service Component Assignment (CA) problem. We address the RA problem using a Lagrangian duality-based approach, solve the CA problem with a heuristic algorithm, and tackle the TOSA problem using a Proximal Policy Optimization (PPO)-based deep reinforcement learning (DRL) algorithm. Extensive simulations are conducted to evaluate the performance of the proposed strategy. The simulation results demonstrate that our solution outperforms existing methods in multiple dimensions, including convergence rate, average response latency, and task success rate.**

***Index Terms*—Vehicular edge computing, collaborative offloading, service caching, resource allocation, deep reinforcement learning.**

Received 5 November 2025; revised 28 December 2025; accepted 30 December 2025. Date of publication 5 January 2026; date of current version 7 May 2026. This work was supported in part by the Emerging Frontiers Cultivation Program of Tianjin University Interdisciplinary Center, Natural Science Foundation of Tianjin under Grant 25JCYBJC01540, in part by the National Natural Science Foundation of China under Grant 62071327, and in part by Tianjin Science and Technology Planning Project under Grant 22ZYYYJC00020. Recommended for acceptance by C. Lin. *(Corresponding author: Huaming Wu.)*

Chaogang Tang and Shucai Wang are with the School of Computer Science and Technology/School of Artificial Intelligence, China University of Mining and Technology, Xuzhou 221116, China, and also with the Mine Digitization Engineering Research Center of the Ministry of Education, Xuzhou 221116, China (e-mail: cgtang@cumt.edu.cn; ts24170120p31@cumt.edu.cn).

Huaming Wu is with the Center for Applied Mathematics, KL-AAGDM, Tianjin University, Tianjin 300072, China (e-mail: whming@tju.edu.cn).

Ruidong Li is with the Institute of Science and Engineering, Kanazawa University, Kanazawa 920-1192, Japan (e-mail: liruidong@ieee.org).

Digital Object Identifier 10.1109/TMC.2025.3650617

I. Introduction

With the rapid development of intelligent transportation systems, smart vehicles are demonstrating superior competitiveness over traditional fuel vehicles in terms of intelligence, interactivity, design aesthetics, and infotainment capabilities [1]. The tasks generated by smart vehicles are also becoming increasingly complex with respect to task diversity, data volume, and requirements for computing resources. Some computationally intensive and delay-sensitive vehicular tasks will pose significant challenges to the limited computing capacity of vehicles. Thus, local task execution may prolong the response latency and contradict the pursuit of the ultra-low latency experience.

Against this backdrop, Vehicular Edge Computing (VEC) has emerged as an effective solution to tackle the aforementioned challenge, enabling the offloading of resource-intensive tasks from vehicles to edge servers, such as Roadside Units (RSUs). Current research predominantly centers on optimizing independent offloading in VEC. For instance, individual vehicles may perform a portion of computations locally while offloading the remaining tasks to the edge node through partial offloading. Meanwhile, vehicles can opt to offload the entire task to the edge node through binary offloading. In addition, collaborative task offloading schemes have been widely investigated in recent years [2], [3], [4], [5]. These studies primarily focus on the diversity of offloading destinations, including cloud centers, network edges, and other entities with underutilized resources. Utilizing the dispersed resources of these entities can indeed enhance the performance of VEC systems. Nevertheless, few studies have examined collaborative task offloading in VEC from the perspective of smart vehicles. Actually, vehicles can collectively accomplish task offloading when their tasks are associated with the same vehicle service. Here, vehicle service refers to software-defined functional modules that enable specific in-vehicle capabilities, e.g., route planning or real-time safety analytics. The rapid proliferation of smart vehicles has led to a significant increase in the number of vehicles associated with the same service. Collaborative task offloading, where each vehicle is responsible for offloading specific service components, can effectively reduce the offloading latency, thereby improving the performance of VEC systems.

On the other hand, service caching has gained considerable attention in recent years [6], [7], [8]. By caching services, such as executable programs and related databases, at the network edge, vehicular tasks can be offloaded to the edge server instead of being executed in the remote cloud center. Caching-assisted task offloading in VEC not only relieves the burden of the front-haul links between RSU and vehicles, but also drastically reduces the response latency of tasks [9]. However, for tasks whose associated services are not cached on the server, the response latency does not significantly decrease. Generally, the performance improvement achieved through service caching may be compromised, particularly when the number of cached services or the caching capacity of the edge server is limited.

To address the limitations of caching-based task offloading in VEC, we propose integrating the strengths of service caching and collaborative task offloading to further enhance the performance of VEC systems. Particularly, a collaborative task offloading strategy is used for tasks whose associated services are not cached at the edge server. By doing so, the offloading latency for these tasks can be reduced, thereby achieving performance comparable to that of tasks whose associated services are cached at the RSU. In this paper, we propose a caching-assisted collaborative task offloading strategy for VEC, which not only caters to the inherent mobility patterns and resource heterogeneity of vehicular networks but also strives to meet the stringent demands of high-real-time VEC scenarios.

The proposed task offloading strategy in this paper involves four different decision variables: the offloading policy that determines which tasks should be offloaded, the caching policy that specifies which services should be cached, the component assignment policy that identifies which component should be offloaded by which vehicles, and the computing resource allocation policy that dictates how the computing resources of edge server should be distributed among these offloaded tasks. The goal of this paper is to minimize the overall response latency for all vehicular tasks in the VEC system, by jointly optimizing the offloading policy, caching policy, component assignment, and computing resource allocation policies. In summary, the major contributions of this paper are outlined as follows.

- We put forward a caching-assisted collaborative task offloading framework for VEC. In this framework, for tasks whose associated services are cached at the edge server, vehicles only need to offload the task-input data. For tasks whose associated services are not cached at the edge server, vehicles requesting the same service collaboratively accomplish task offloading by dividing the workload, where each vehicle is responsible for offloading specific components in parallel.
- We formulate an optimization problem to minimize the overall response latency in the VEC system. This problem is essentially a Mixed Integer Nonlinear Programming (MINLP) problem, which is inherently difficult to solve directly due to its complexity. To address this challenge, we decompose it into three subproblems: the Task Offloading and Service Caching (TOSA) problem, the Computing Resource Allocation (RA) problem, and the Service Component Assignment (CA) problem.
- The RA problem is a convex optimization problem, and we employ a Lagrangian duality-based method to determine the optimal resource allocation policy. The CA problem is an NP-hard problem, and we propose a heuristic algorithm to address it. Finally, we develop a greedy PPO algorithm to solve the TOSA problem, where we use a greedy rule to guide service caching and leverage a PPO-based Deep Reinforcement Learning (DRL) algorithm to determine the task offloading profile.
- We conduct extensive simulations to evaluate the performance of the proposed caching-assisted collaborative task offloading strategy. Through comparative analysis with several baseline strategies, our solution outperforms them in multiple dimensions, e.g., convergence rate, average response latency, and task success rate.

The subsequent sections of this paper are structured as follows: A comprehensive review of the state-of-the-art literature on this topic is provided in Section II. Section III presents the system model, while Section IV formulates the optimization problem. The algorithm design is presented in Section V, followed by the numerical evaluation in Section VI. Finally, the conclusion comes in Section VII.

II. Related Work

Most of the state-of-the-art works on this topic focus on enhancing the performance of edge computing systems, including VEC [10], Mobile Edge Computing (MEC) [11], and Aerial Edge Computing (AEC) [12], by optimizing task offloading, resource allocation, service caching, and deployment, among other aspects.

A. Caching Assisted Task Offloading Approaches

To reduce the latency in service caching, a collaborative caching framework involving multiple edge servers was proposed in [13]. This framework targets multiple objectives, including optimizing task response latency and service deployment latency. By leveraging Lyapunov optimization theory, the authors formulated a long-term optimization problem and transformed it into a per-slot optimization problem.

To address the challenges in deploying latency-sensitive services due to mobile data traffic, Zhang et al. [15] proposed using UAVs as auxiliary edge nodes to provide computation and caching services for terminal devices. Considering the fact that the effectiveness of caching is often limited by a low hit rate and significant network delays, the authors in [16] leveraged vehicle function features and transportation correlations to tackle the caching problem. They formulated a service caching problem and designed two caching strategies to solve it.

To enhance the performance of the VEC system, Cheng et al. [17] utilized idle resources from vehicles and cache services at both RSU and vehicles. They formulated a joint optimization problem for task offloading and service caching in VEC, aiming to minimize task processing delays and related costs. A suboptimal algorithm was designed to tackle this NP-hard problem.

Vehicles can serve as the offloading destinations by caching the required services (i.e., service vehicles). The high mobility

of vehicles makes the task processing timea critical performance metric for evaluating the VEC system. Hence, Gao et al. [18] put forward a task offloading and service caching problem in VEC, aiming to minimize the response latency while adhering to long-term energy consumption constraints. UAV-assisted MEC utilizes UAVs as the resource providers to improve the response latency of the MEC system. Following this rule, Zhao et al. [7] considered UAVs as both caching and computation nodes to satisfy the complex requirements of user equipment. A joint optimization for content cache, service placement, and task offloading was formulated, with the objective of maximizing the Quality of Experience (QoE).

The explosive growth of vehicular applications makes it possible that computation results can be reused. To improve the efficiency of task offloading, a content caching-assisted framework was put forward in [19], where a DRL-based edge caching scheme was proposed and a graph attention convolution kernel was integrated to abstract the features of entities.

B. Collaborative Task Offloading Approaches

Collaborative task offloading in edge computing environments mainly focuses on the diversity of offloading destinations [27], [28]. For instance, Chen et al. [20] underlined the cooperation between UAVs and vehicles, where tasks with multidimensional goals are assigned with optimal resources. To reduce the matching complexity, they proposed a clustering-based approach to address the problem, and the simulation results demonstrate the effectiveness of the strategy.

Li et al. [4] investigated the joint optimization of task offloading and resource allocation problems in small-cell MEC. They tried to optimize energy consumption for all the user devices, and adopted a Multi-Agent Proximal Policy Optimization (MAPPO)-based strategy, allowing collaboration among small-cell base stations to accelerate the optimization process. Similarly, to optimize task failure rates and payments, the authors in [21] jointly optimized task offloading and resource allocation. Specifically, they designed different offloading models and virtual resource units. Experimental results demonstrated the superior performance of their proposed strategy in terms of task failure rate and fairness.

The users' QoE is a key incentive to guide the improvement on the performance of edge computing systems. Hence, Chu et al. [22] proposed to maximize users' QoE by determining service caching, task offloading and resource allocation in the MEC environment. To address this NP-hard problem, they propose a two-stage algorithm that leverages approximation and decomposition theory. Due to the dynamic MEC environment, edge caching performance is constrained, which leads to failures and inconsistencies. Hence, a redundant service caching framework is proposed in [29], where an optimization problem of service caching and task offloading is formulated. A learning-based method is adopted to determine redundant service caching, and the simulation results demonstrate the advantages of their strategy.

The high mobility of vehicles and the limited coverage of RSUs pose significant challenges to the performance of edge caching in VEC. For instance, a service cached at one RSU cannot be accessed by a vehicle that has moved out of its coverage area. To address this issue, Li et al. [23] proposed a collaborative task offloading and service caching replacement strategy. Specifically, adjacent RSUs facilitate the migration of vehicular tasks, while also enabling service caching replacement. Their approach aims to minimize both computational costs and delays through a DRL-based strategy. The comparison of the existing literature is shown in Table I.

The aforementioned works primarily focus on the diversity of offloading destinations when designing collaborative task offloading strategies, yet they tend to neglect the cooperation potential among vehicles with offloading requests. To the best of our knowledge, this is the first work that integrates the advantages of service caching and collaborative task offloading in VEC. By leveraging caching-assisted collaborative task offloading, we aim to achieve fine-grained task offloading, thereby reducing the response latency for vehicular tasks in VEC.

III. System Model

The application scenario considered in this paper is illustrated in Fig. 1, encompassing three distinct entities: vehicle, RSU, and the cloud center. RSU is connected to the cloud center via wired communication links that may traverse the backbone network. Vehicle services can be downloaded from the cloud center and cached at RSUs for efficiently serving the offloading requests from nearby vehicles. The set of services is denoted by $\mathcal{K} = \{1, \ldots, K\}$, where K represents the total number of cacheable services. Each service $k \in \mathcal{K}$ is characterized by a tuple (s_k, w_k), where s_k denotes the data size of service k (e.g., in KB), and w_k indicates the required computing resources for service k (e.g., in CPU cycles). Service typically comprises computational logic or executable components. Therefore, in this paper, we assume that services are divisible, and each service k can be partitioned into M_k independent components, i.e., $k = (k_1, \ldots, k_{M_k})$. Each component k_i has a data size s_{k_i}, satisfying $\sum_{i=1}^{M_k} s_{k_i} = s_k$.

Vehicles utilize Vehicle-to-Infrastructure (V2I) communication technologies to interact with RSUs upon entering their communication range. Thus, vehicles can offload their tasks to the RSU for execution, leveraging its richer computing resources compared to their own. The set of vehicles is denoted by $\mathcal{N} = \{1, \ldots, N\}$ and N is the number of vehicles that can offload their tasks to the RSU R. Suppose each vehicle $n \in \mathcal{N}$ generates a service-related task, which can be represented by a triplet (I_n, k, D_n), where I_n denotes the size of the task-input data, k represents the specific service associated with task n, and D_n indicates the deadline by which the task must be completed.

In the VEC system, vehicles can offload their computationally intensive tasks to RSUs for execution, thereby achieving ultra-low-latency QoE. If the service related to the vehicular task has already been cached at the RSU, vehicle n only needs to offload the task-input data. Otherwise, vehicle n must offload both the task-input data and the associated service to R. Define $\alpha_{n,k}$ ($n \in \mathcal{N}$ and $k \in \mathcal{K}$) as a binary variable to indicate the correspondence relationship between task n and service k. Specifically, $\alpha_{n,k} = 1$ indicates that task n requests service k,

TABLE I
COMPARISON BETWEEN OUR WORK AND THE EXISTING LITERATURE

Reference	Collaborative Offloading	Caching Strategy	Computing Nodes	Optimization Objective
[2]	✗	✗	Edge server and mobile devices	Energy consumption
[3]	✗	✗	UAVs	Processing latency
[14]	✗	✗	RSUs	Processing latency and energy consumption
[13]	✗	✗	Edge server	Processing latency
[15]	✗	✓	Edge server and UAV	Processing latency
[16]	✗	✓	RSUs	Service hit rate/ network delay
[17]	✗	✓	RSUs and vehicles	Processing delay and costs
[18]	✗	✓	Vehicles	Processing delay
[7]	✗	✓	Vehicles	Processing delay
[19]	✗	✓	RSUs	Task computing latency
[20]	✓	✗	UAVs and Vehicles	Value of service
[9]	✗	✓	RSU	Processing latency
[4]	✗	✗	Small-cell base station	Energy consumption
[21]	✗	✗	Fog nodes	Task failure rate, payments and fairness
[22]	✗	✓	Edge node	Users' QoE
[23]	✓	✓	RSUs	Delay and computing costs
[24]	✓	✓	Edge node	Processing delay
[25]	✗	✓	RSU	Offloading efficiency
[26]	✗	✓	Edge server	Energy consumption
Proposed Work	✓	✓	RSU	Processing delay

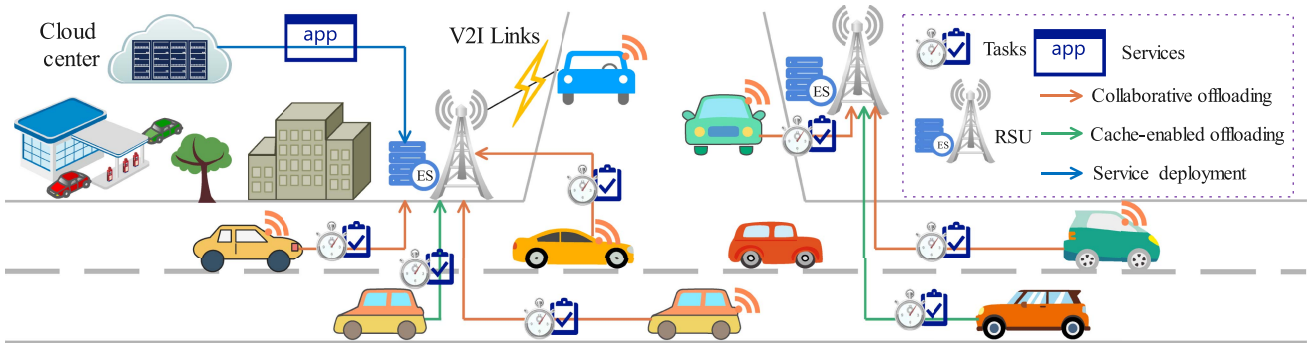


Fig. 1. The considered application scenario in this paper.

while $\alpha_{n,k} = 0$, otherwise. In this paper, we assume that each vehicular task can only be associated with one service. Consequently, the equation $\sum_{k=1}^{K} \alpha_{n,k} = 1$ $(\forall n \in \mathcal{N})$ holds. Define a binary vector $\boldsymbol{\lambda} = \{\lambda_1, \ldots, \lambda_N\}$ to indicate the offloading profile of N vehicular tasks in the optimization period, where $\lambda_n = 0$ indicates that task n is executed locally, and $\lambda_n = 1$ indicates that n is offloaded and executed at the RSU.

A. Caching Model

In VEC, task offloading to the edge server rather than the cloud center can significantly reduce response times and improve system efficiency. However, a large number of offloading requests may overwhelm the computing and communication resources of the edge server, leading to a degradation in users' QoE. To further reduce the response times for vehicular tasks, we propose caching services at RSUs and aim to shorten the task offloading delay. Specifically, define the binary vector $\boldsymbol{\delta} = \{\delta_1, \ldots, \delta_K\}$ to represent the caching profile of K vehicle services. Here, $\delta_k = 1$ $(k \in \mathcal{K})$ indicates that service k is cached at R, while $\delta_k = 0$ otherwise. Denote by C the maximum caching capacity

TABLE II
NOTATIONS AND DESCRIPTIONS

Symbols	Descriptions
N	The number of vehicles(tasks)
K	The number of services
R	The RSU equipped with the edge server
I_n	Task-input data size
D_n	The deadline for task n
s_k	The data size of service k
w_k	The required computing resources for service k
M_k	The number of components for service k
s_{k_i}	The data size of component k_i
C	The caching capacity of R
f_n	The processing frequency for vehicle n
κ	The maximal bandwidth resources of R
g_n	The wireless channel gain between n and R
p_n	The transmission power of vehicle n
r_n	The achievable data rate for task n
V_k	Set of tasks associated with service k
η	Time to integrate one unit bit of data
F_R	Maximal computation capability of R
δ_k	The caching decision of service k
λ_n	The offloading decision of task n
$\phi_{n,k,i}$	The component assignment decision
$f_{R,n}$	The computing resource allocation decision

of R, thereby constraining the number of services that can be cached as follows:

$$\sum_{k=1}^{K} \delta_k s_k \leq C. \tag{1}$$

B. Task Offloading Model

Let r_n denote the transmission rate for offloading the task n to R, and r_n actually depends upon the number of communication resources allocated to the task by R. For simplicity, we assume that all tasks offloaded to R equally share communication resources. Consequently, r_n can be expressed as follows [30]:

$$r_n = \frac{\kappa}{\sum_{i \in \mathcal{N}} \lambda_i} \log\left(1 + \frac{p_n g_n}{\sigma^2}\right), \tag{2}$$

where κ, p_n, g_n, and σ^2 are the total bandwidth resources of R, the transmission power of vehicle n, the channel gain between vehicle n and R, and the noise power, respectively.

Calculating the offload delay for the task n, indicated by T_n^{off}, is fundamentally dependent on the caching status of the associated service. The following discussion provides a detailed explanation.

1) Caching-Assisted Offloading Model: If the associated service k of task n is cached at the edge server (i.e., $\alpha_{n,k} = 1$ and $\delta_k = 1$), the vehicle only needs to offload the task-input data (e.g., input parameters) to the edge server R, as previously described. In this case, T_n^{off} can be expressed as:

$$T_n^{off} = \sum_{k=1}^{K} \alpha_{n,k} \delta_k \frac{I_n}{r_n}. \tag{3}$$

2) Collaborative Offloading Model: If the associated service k of task n (i.e., $\alpha_{n,k} = 1$) is not cached at the edge server, the vehicle needs to offload both the task-input data and the associated service to R. However, given that vehicle services can be divided into distinct components, vehicular tasks associated with the same service can collectively offload the service to R, for example, with each vehicle responsible for offloading a different component. In this paper, we refer to this process as *collaborative offloading* from the perspective of vehicles.

Let V_k denote the set of vehicular tasks associated with the service k, and the number of tasks in this set can be calculated as $|V_k| = \sum_{n=1}^{N} \alpha_{n,k}$. Once service k is determined to be collectively offloaded, there are M_k components that need to be collaboratively offloaded by $|V_k|$ vehicles. The question arises as to how to allocate these components to individual vehicles for collaborative offloading while satisfying diverse performance requirements. To address this, we introduce a binary variable $\phi_{n,k,i}$ ($n \in V_k, k \in \mathcal{K}, 1 \leq i \leq M_k$) to specify whether vehicle n offloads component i of service k to the edge server. Specifically, $\phi_{n,k,i} = 1$, if the vehicle n offloads the component i of the service k to the edge server R; and $\phi_{n,k,i} = 0$, otherwise. In this paper, we assume that given an arbitrary component j ($1 \leq j \leq M_k$) of service k ($\forall k \in \mathcal{K}$), it can only be offloaded by one vehicle; thus the following constraint always holds: $\sum_{i=1}^{|V_k|} \phi_{i,k,j} = 1$ ($i \in V_k$).

Note that a single vehicle may offload multiple components, or even all components, if it is the only vehicle associated with the specific service. Consequently, the time required to offload the components for task n can be analyzed based on the following two cases:

- *Case 1 ($|V_k| = 1$):* This implies that only vehicle n is associated with service k. In this scenario, there is no collaborative offloading available for vehicle n, which means it must offload both the task-input data and the entire service independently. Consequently, the offloading delay for the components can be expressed as:

$$T_n^{off,comp} = \sum_{k=1}^{K} \alpha_{n,k}(1 - \delta_k)\frac{s_k}{r_n}. \tag{4}$$

- *Case 2 ($|V_k| > 1$):* This indicates that in addition to vehicle n, there are other vehicles also associated with service k. In this case, these vehicles can implement the collaborative offloading strategy for task offloading. As a result, the offloading delay for the components can be formulated as:

$$T_n^{off,comp} = \sum_{k=1}^{K} \alpha_{n,k}\tau_k + \sum_{k=1}^{K} \alpha_{n,k}(1 - \delta_k) \sum_{i=1}^{M_k} \phi_{n,k,i} \frac{s_{k_i}}{r_n} \tag{5}$$

  where τ_k represents the time required to partition service k, which typically depends on the granularity of the partition. In this paper, it is assumed that different services may have distinct values for τ_k ($k \in \mathcal{K}$), and these values can be determined based on historical statistics.

To integrate the above two cases, we introduce an indicator variable ψ_k that denotes whether service k is exclusively associated with a single vehicular task. ψ_k can be defined as follows:

$$\psi_k = \begin{cases} 1 & |V_k| > 1 \\ 0 & |V_k| = 1 \end{cases} \tag{6}$$

Thus, the offloading delay for the components can be rewritten as:

$$\begin{aligned} T_n^{off,comp} = {} & (1 - \psi_k)\left(\sum_{k=1}^{K} \alpha_{n,k}(1 - \delta_k)\frac{s_k}{r_n}\right) \\ & + \psi_k\left(\sum_{k=1}^{K} \alpha_{n,k}\tau_k + \sum_{k=1}^{K} \alpha_{n,k}(1 - \delta_k) \sum_{i=1}^{M_k} \phi_{n,k,i}\frac{s_{k_i}}{r_n}\right). \end{aligned} \tag{7}$$

Apart from the components offloading, vehicle n also needs to offload its task-input data to the edge server, and the time taken for this process can be expressed as:

$$T_n^{off,para} = \sum_{k=1}^{K} \alpha_{n,k}(1 - \delta_k)\frac{I_n}{r_n}. \tag{8}$$

As a result, the offloading latency for task n, denoted by T_n^{off} is given as:

$$T_n^{off} = T_n^{off,comp} + T_n^{off,para}. \tag{9}$$

C. Computation Model

In this paper, vehicular tasks can be executed either locally on the vehicle or offloaded to the edge server, depending on the diverse performance requirements and resource availability. If the task n is executed locally (i.e., $\lambda_n = 0$), the computation delay, denoted by $T_n^{com,loc}$, is given as:

$$T_n^{com,loc} = \sum_{k=1}^{K} \alpha_{n,k} \frac{w_k}{f_n}, \tag{10}$$

where f_n represents the computing capability of vehicle n. It should be noted that the caching status of the associated service (e.g., k) does not influence the local execution of task n.

If task n is offloaded to edge server R for execution, the computation delay will be substantially influenced by the caching status of the associated service. We will present a systematic and detailed explanation of the methodology for calculating computation delay across different cases.

1) Associated Service Cached: If task n is offloaded for execution and the associated service is cached, R can initiate the execution process after completing the following preparations. First, R will allocate appropriate computing resources, such as by creating virtual machines (VMs) and virtual environments tailored for the offloaded tasks. The time required for these preparations is typically non-negligible, and services with different execution logic may have varying preparation times. For simplicity, let T_k^{init} represent the initialization time for creating VMs and virtual environments for service k. T_k^{init} can be determined based on the historical statistics. Then, the computation delay of task n is given as:

$$T_n^{com} = \sum_{k=1}^{K} \alpha_{n,k}\delta_k T_k^{init} + \sum_{k=1}^{K} \alpha_{n,k}\delta_k \frac{w_k}{f_{R,n}}, \tag{11}$$

where $f_{R,n}$ denotes the total computing resources allocated by R to task n.

2) Associated Service Not Cached: The preparations become significantly more complex when task n is offloaded for execution, but the associated service is not cached. First, vehicle n must offload both the task-input data and the corresponding components of the associated service. Even after the offloading process is completed, task n cannot be executed immediately because the edge server R must wait for all components to arrive and further reconstruct the entire service. Specifically, the time required to reconstruct the entire service can be expressed as $T_k^{intg} = \eta s_k$, where η represents the system-average time to integrate one unit bit of data and can be determined empirically through profiling. This linear integration time, although a simplified representation, can effectively capture the data-intensive aspect of integration without introducing additional complexity. In this case, the waiting time for task n can be given as:

$$T_n^{wt} = \sum_{k=1}^{K} \alpha_{n,k}(1-\delta_k) \max\left(\max_{m \in V_k} T_m^{off,comp} + T_k^{init} - T_n^{off}, 0 \right),$$

where the term $\max_{m \in V_k} T_m^{off,comp} + T_k^{init} - T_n^{off}$ represents the waiting time for task n until all components arrive at R. Note that the value of this term may be negative, which means that the offloading latency T_n^{off} for task n is greater than the integration and initialization time of the entire service. Thus, task n can be executed immediately without waiting. The aforementioned analysis of the preparations is based on the assumption that collaborative offloading is feasible. However, when only one task is associated with the service (i.e., $\psi_k = 0$), there is no collaborative offloading or reconstruction operation involved. Consequently, the computation delay for task n when the associated service is not cached can be expressed as follows:

$$\begin{aligned} T_n^{com} = (1-\psi_k) \Bigg(& \sum_{k=1}^{K} \alpha_{n,k}(1-\delta_k) T_k^{init} \\ & + \sum_{k=1}^{K} \alpha_{n,k}(1-\delta_k) \frac{w_k}{f_{R,n}} \Bigg) \\ & + \psi_k \left(T_n^{wt} + \sum_{k=1}^{K} \alpha_{n,k}(1-\delta_k) \frac{w_k}{f_{R,n}} \right). \end{aligned} \tag{12}$$

When task n is completed, the resulting output can be transmitted back to the vehicle. Given that the data size of the execution result is significantly smaller than that of the task input, we simplify the analysis by omitting the time associated with this transmission. By integrating the aforementioned analysis, the response latency for task n can be expressed as:

$$T_n = (1-\lambda_n) T_n^{com,loc} + \lambda_n (T_n^{off} + T_n^{com}). \tag{13}$$

Thus, the response latency for the VEC system can be expressed as:

$$T^{total} = \sum_{n=1}^{N} T_n. \tag{14}$$

IV. Problem Formulation

In this section, we formulate a caching-assisted collaborative task offloading optimization problem for VEC, aiming to minimize the overall response latency for all the vehicular tasks, by jointly optimizing the offloading policy, the caching policy, the component assignment policy, and the computing resource allocation policy. In addition to the aforementioned offloading and caching policies $\boldsymbol{\lambda}$, $\boldsymbol{\delta}$, let $\boldsymbol{\phi} = \{\phi_{n,k,i} | \forall n \in \mathcal{N}, \forall k \in \mathcal{K}, \forall i : 1 \le i \le M_k\}$ denote the component assignment policy, and $\boldsymbol{f_R} = \{f_{R,n} | \forall n \in \mathcal{N}\}$ denote the computing resource allocation policy. Define the objective function $J(\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\phi}, \boldsymbol{f_R}) = T^{total}$. Then, our optimization problem regarding $\boldsymbol{\lambda}$, $\boldsymbol{\delta}$, $\boldsymbol{\phi}$, and $\boldsymbol{f_R}$ can be formulated as follows:

$$\begin{aligned} \text{P0}: \ & \min_{\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\phi}, \boldsymbol{f_R}} \quad J(\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\phi}, \boldsymbol{f_R}) \\ \text{s.t.} \ & \sum_{k=1}^{K} \delta_k s_k \le C, \end{aligned} \tag{15a}$$

$$\sum_{k=1}^{K} \alpha_{n,k} = 1, \quad \forall n \in \mathcal{N}, \tag{15b}$$

$$\sum_{i=1}^{M_k} s_{k_i} = s_k, \quad \forall k \in \mathcal{K}, \tag{15c}$$

$$\sum_{i=1}^{|V_k|} \phi_{i,k,j} = 1, \ \forall i \in V_k, \forall k \in \mathcal{K}, \ 1 \le j \le M_k, \tag{15d}$$

$$T_n \le D_n, \quad \forall n \in \mathcal{N}, \tag{15e}$$

$$\sum_{n \in \mathcal{N}} \lambda_n f_{R,n} \le F_R, \quad \forall n \in \mathcal{N}, \tag{15f}$$

$$f_{R,n} \ge 0, \quad \forall n \in \mathcal{N}, \tag{15g}$$

$$\lambda_n, \delta_k \in \{0,1\}, \quad \forall n \in \mathcal{N}, \forall k \in \mathcal{K}, \tag{15h}$$

$$\phi_{i,k,j} \in \{0,1\}, \ \forall i \in V_k, \forall k \in \mathcal{K}, \ 1 \le j \le M_k, \tag{15i}$$

where the constraint (15e) specifies that task n must be completed prior to its deadline. The constraint (15f) ensures that the total allocated computing resources for all tasks do not surpass the maximum computing capacity R . Constraints (15g)–(15i) define the allowable value ranges for the decision variables. The meanings of other constraints, such as (15a)–(15d) have been elaborated upon in earlier sections.

The problem P0 is fundamentally a MINLP problem, making it highly challenging to solve P0 directly. Given the stringent latency requirements for vehicular tasks in large-scale VEC networks, algorithms with exponential time complexity for finding the optimal solution are practically infeasible. Consequently, in this paper, we propose to employ a suboptimal algorithm with low time complexity for addressing this problem.

The expression of the objective function $J(\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\psi}, \boldsymbol{f_R})$ can be expanded as:

$$\begin{aligned} J(\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\psi}, \boldsymbol{f_R}) &= \sum_{n=1}^{N} (1-\lambda_n) T_n^{com,loc} + \lambda_n \left(T_n^{off} + T_n^{com}\right) \\ &= \sum_{n \in \mathcal{N}_o} (T_n^{off} + T_n^{com}) + \sum_{n \in \mathcal{N} \setminus \mathcal{N}_o} T_n^{com,loc}, \end{aligned} \tag{16}$$

where $\mathcal{N}_o$ denotes the set of vehicular tasks that are offloaded to R for execution and $\mathcal{N}_o = \{n | \lambda_n = 1, \forall n \in \mathcal{N}\}$. The allocation of computing resources of R is exclusively applicable to tasks in $\mathcal{N}_o$. In addition, it can be observed that the decision variables $\boldsymbol{\lambda}$ and $\boldsymbol{\delta}$ are tightly coupled in both the objective function and the constraints. This is because task offloading and service caching interact with each other, where changes in one directly influence the other. Ideally, services associated with a higher number of vehicular tasks should be cached, while tasks whose requested services are already cached at R should be offloaded. Lastly, the allocation of service components among vehicles becomes a subsequent issue to address, following the determination of offloading and caching decisions.

Hence, to simplify the original problem P0, we decouple these variables from one another by decomposing it into three subproblems: the Task Offloading and Service Caching (TOSA) problem, the computing Resource Allocation (RA) problem, and the service Component Assignment (CA) problem.

V. Proposed Collaborative Task Offloading Scheme

In this section, we respectively solve the decomposed subproblems (i.e., TOSA, RA, CA) to obtain the solution of the original problem.

A. Computing Resource Allocation Problem

Given the feasible decisions for task offloading $\boldsymbol{\lambda}$, service caching $\boldsymbol{\delta}$, service component assignment $\boldsymbol{\phi}$, the original problem can be equivalently transformed into solving the following problem:

$$\begin{aligned} &\min_{\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\phi}} \ J^*(\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\phi}) \\ &\text{s.t.} \ \ (15a)\text{–}(15d), (15h), (15i). \end{aligned} \tag{17}$$

where $J^*(\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\phi})$ is the function with optimal value regarding the RA problem, and $J^*(\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\phi}) = \min_{\boldsymbol{f_R}} J(\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\phi}, \boldsymbol{f_R})$. Hence, the RA problem can be formulated as:

$$\begin{aligned} \text{RA}: \ &\min_{\boldsymbol{f_R}} \ J(\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\phi}, \boldsymbol{f_R}) \\ &\text{s.t.} \ \ (15e)\text{–}(15g). \end{aligned} \tag{18}$$

Given the feasible solutions for $\boldsymbol{\lambda}$, $\boldsymbol{\delta}$, and $\boldsymbol{\phi}$, the RA problem aims to acquire $J^*(\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\phi})$ by optimizing the computing resource allocation policy. For simplicity, let $G(\boldsymbol{f_R}) \triangleq \sum_{n \in \mathcal{N}_o} (T_n^{off} + T_n^{com})$, and the RA problem is actually to minimize $G(\boldsymbol{f_R})$. Based on the expanded expression of $G(\boldsymbol{f_R})$, we have the following lemma.

Lemma 1: Given the feasible solutions for $\boldsymbol{\lambda}$, $\boldsymbol{\delta}$, and $\boldsymbol{\phi}$, the RA optimization problem regarding $\boldsymbol{f_R}$ is convex.

Proof: By analyzing the expression $G(f_{R,1}, \ldots, f_{R,n}) = \sum_{n \in \mathcal{N}_o} (T_n^{off} + T_n^{com})$, we can observe that the computing resource allocation is only dependent on the term T_n^{com} and is independent of the term T_n^{off}. Thus, the second partial derivatives by differentiating $G(\boldsymbol{f_R})$ w.r.t. $\boldsymbol{f_R}$ is: $\partial^2 G(\boldsymbol{f_R}) / \partial f_{R,m} \partial f_{R,n} = 2 \sum_{k=1}^{K} \alpha_{n,k} w_k / f_{R,n}^3 > 0$, when $m = n$; and 0, otherwise. The Hessian matrix of $G(\boldsymbol{f_R})$, denoted by $\mathcal{M}(G(\boldsymbol{f_R}))$, can be easily constructed [31]. Then, we have $Y^T \mathcal{M} Y \ge 0$ for an arbitrary nonzero vector Y, which implies that $\mathcal{M}(G(\boldsymbol{f_R}))$ is positive semi-definite. Hence $G(\boldsymbol{f_R})$ is convex w.r.t. $f_{R,n}$. Thus, the objective function is convex regarding the computing resource allocation policy. Similarly, the left-hand side of the constraint (15e) can also be proven to be convex. Consequently, the RA problem is convex, and the proof is completed. ■

Lemma 2: The RA problem can be optimized only when the total amount of computing resources F_R is fully allocated.

The above lemma can be readily proven by contradiction; hence, we omit the detailed proof in this paper. Particularly, the Lagrangian function $L(\boldsymbol{f_R}, \boldsymbol{v}, \mu)$ for the RA problem is constructed as shown in (19) at the bottom of the next page, where $v = \{v_n | v_n \ge 0, \forall n \in \mathcal{N}\}$ and μ are the Lagrangian

multipliers. Define the Lagrange dual function $g(\boldsymbol{v}, \mu)$ as:

$$g(\boldsymbol{v}, \mu) = \min_{f_{R,n} \geq 0, \forall n \in \mathcal{N}} L(\boldsymbol{f_R}, \boldsymbol{v}, \mu).$$

Then, let the first partial derivative of $L(\boldsymbol{f_R}, \boldsymbol{v}, \mu)$ regarding $\boldsymbol{f_R}$ equal zero, i.e., $\partial L(\boldsymbol{f_R}, \boldsymbol{v}, \mu)/\partial \boldsymbol{f_R} = 0$, and we can obtain the optimal solution $f^*_{R,n}$ as:

$$f^*_{R,n} = \sqrt{\frac{(v_n + 1)\sum_{k=1}^{K} \alpha_{n,k} w_k}{\mu}}, \forall n \in \mathcal{N}_o. \tag{20}$$

Accordingly, the Lagrange dual function can be further rewritten as: $g(\boldsymbol{v}, \mu) = L(\boldsymbol{f^*_R}, \boldsymbol{v}, \mu)$. The dual problem for the RA problem can be formulated as:

$$\begin{aligned} \max_{\boldsymbol{v}, \mu} \quad & g(\boldsymbol{v}, \mu) \\ \text{s.t.} \quad & v_n \geq 0, \forall n \in \mathcal{N}. \end{aligned} \tag{21}$$

Obviously, Lemma 1 demonstrates that the RA problem satisfies Slater's Condition. As a result, we can effectively solve the dual problem using the gradient descent method, which offers a computationally efficient and practical approach to address this problem. Specifically, $\boldsymbol{v}$ and μ can be updated iteratively according to the following procedure:

$$\boldsymbol{v}(t+1) = [\boldsymbol{v}(t) + l_v \nabla_{\boldsymbol{v}} g(\boldsymbol{v}, \mu)]^+, \tag{22}$$

$$\mu(t+1) = \mu(t) + l_\mu \nabla_\mu g(\boldsymbol{v}, \mu), \tag{23}$$

where l_v and l_μ represent the learning rates for updating the Lagrangian multipliers, and the operator $[\cdot]^+$ is equivalent to the operator $\max(\cdot, 0)$ in this paper.

B. Service Component Assignment Problem

Given the vehicle service k and the set of vehicular tasks V_k, the CA problem for service k aims to achieve the minimal component offloading delay by allocating M_k components to $|V_k|$ vehicles. Given the offloading decision $\boldsymbol{\lambda}$ and caching status of service k (i.e., $\delta_k = 0$ and $\psi_k = 0$), the CA problem can be formulated as:

$$\begin{aligned} \text{CA}: \quad & \min_{\boldsymbol{\phi}, k} \; T_n^{off,comp} \\ \text{s.t.} \quad & \text{(15d), (15i)}. \end{aligned} \tag{24}$$

Lemma 3: For an arbitrary service k that is not cached but associated with multiple vehicular tasks, the corresponding CA problem is NP-hard.

Proof: The Component Assignment (CA) problem can be formulated as an instance of the Unrelated Parallel Machine Scheduling Problem (UPMSP), which is a well-known NP-hard problem [32]. Specifically, the UPMSP can be described as follows: given m jobs and n machines, each job can be processed without interruption on any machine, with varying processing times across different machines. The objective is to schedule the jobs on the machines such that the maximal completion time (makespan) is minimized.

The CA problem can be mapped into the UPMSP framework as follows: the component offloading delay corresponds to the processing time in the UPMSP. Each vehicle can offload one component at any given time, and each component can only be offloaded by one vehicle. This aligns with the UPMSP principle that each machine can process one job at any time, and each job can only be processed by one machine. Consequently, the CA problem addressed in this paper is NP-hard. ■

Given the service k, determining the optimal solution $\phi_{n,k,i}$ ($n \in V_k, 1 \leq i \leq M_k$) with exponential time may not be feasible in a real-time VEC environment. Hence, we propose a heuristic algorithm to address the CA problem. The edge server R has access to global information regarding the status of service k and task offloading requests. For each vehicular task in V_k, an offloading delay matrix Mtx can be constructed, where each row corresponds to a vehicle n in V_k, each column corresponds to a component i ($1 \leq i \leq M_k$), and the element at position (n, i) represents the offloading delay incurred by vehicle n offloading component i. The value of position (n, i) in Mtx denoted by $Mtx(n, i)$ can be expressed as: $Mtx(n, i) = s_{k_i}/r_n$ ($n \in V_k$ and $\psi_k = 1$).

The proposed algorithm is presented in Algorithm 1. According to the matrix Mtx, the offloading delay for each component i by each vehicle n can be determined. Then, the minimal delay time $Val(i)$ for each component i can be expressed as $Val(i) = \min_n\{Mtx(n, i) | n \in V_k, \psi_k = 1\}$ by selecting the smallest value from each column of Mtx. Subsequently, M_k components are sorted in descending order based on the values of $Val[i]$, and a new component assignment sequence for service k is constructed as $L_k = \{com_1, \ldots, com_{M_k}\}$. For each component com_i in L_k, we allocate it to the vehicle following a greedy strategy. Specifically, the greedy rule ensures that the current component is always assigned to the vehicle with the smallest current makespan. To this end, a makespan list $MK[\cdot]$ is initialized for each vehicle in V_k by setting $MK[n] = 0$ for all $n \in V_k$ at the beginning. For each component com_i in L_k, the algorithm assigns it to the vehicle n that satisfies $n = \arg\min_{\forall j \in V_k} MK[j] + Mtx(j, com_i)$.

Then, the current makespan for vehicle n is updated by $MK[n] + Mtx(n, com_i)$.

Remark. It takes the time of $\mathcal{O}(|V_k| \cdot M_k)$ to construct the matrix Mtx and additional $\mathcal{O}(|V_k| \cdot M_k)$ to calculate $Val[1:$

$$\begin{aligned} L(\boldsymbol{f_R}, \boldsymbol{v}, \mu) &= \sum_{n \in \mathcal{N}_o} \left(T_n^{off} + T_n^{com}\right) + \sum_{n=1}^{N} v_n (T_n - D_n) + \mu \left(\sum_{n \in \mathcal{N}_o} f_{R,n} - F_R \right) \\ &= \sum_{n \in \mathcal{N}_o} (T_n^{off} + T_n^{com}) + \sum_{n \in \mathcal{N}_o} v_n (T_n - D_n) + \sum_{n \in \mathcal{N} \setminus \mathcal{N}_o} v_n (T_n - D_n) + \mu \left(\sum_{n \in \mathcal{N}_o} f_{R,n} - F_R \right). \end{aligned} \tag{19}$$

Algorithm 1: Heuristic Greedy Algorithm for Component Assignment (GACA).

Input: V_k, ψ_k
Output: Component assignment strategy for service k
1 Given V_k, construct the matrix Mtx by $Mtx(n,i) = s_{k_i}/r_n$, $\forall n \in V_k, \forall i : 1 \leq i \leq M_k$;
2 **for** component $i = 1$ to M_k **do**
3 $\quad Val(i) = \min_n \{Mtx(n,i) | n \in V_k, \psi_k = 1\}$;
4 **end for**
5 Generate $L_k = \{com_1, \cdots, com_{M_k}\}$ by sorting M_k components in descending order;
6 Initialize makespan $MK[\cdot]$ for each vehicle n in V_k;
7 **for** each component com_i in L_k **do**
$\quad$ `// Implement the greedy strategy`
8 $\quad$ Select vehicle n by $n = \arg \min_{\forall j \in V_k} MK[j] + Mtx(j, com_i)$;
9 $\quad$ Allocate component com_i to vehicle n;
10 $\quad$ Update $MK[n]$ by $MK[n] += Mtx(n, com_i)$;
11 $\quad$ Update the CA decision by $\phi_{n,k,com_i} = 1$;
12 **end for**
13 **Return** component assignment strategy for service k;

$M_k]$. Sorting the components in descending order requires the time of $\mathcal{O}(M_k \log M_k)$. The program loop (lines 7–12) also takes the time of $\mathcal{O}(|V_k| \cdot M_k)$ to assign M_k components to vehicles in V_k. Accordingly, the algorithm complexity for solving the CA problem given service k ($\delta_k = 0$ and $\psi_k = 1$) is $\mathcal{O}(|V_k| \cdot M_k + M_k \log M_k)$. In the worst-case scenario, where none of the vehicle services are cached, the overall algorithm complexity is $\mathcal{O}(\sum_{k=1}^{K} |V_k| \cdot M_k + \sum_{k=1}^{K} M_k \cdot \log M_k)$. Compared to exponential time complexity, this heuristic algorithm with polynomial time complexity can significantly reduce the computational time for decision-making, thereby enhancing efficiency for solving the CA problem.

C. Task Offloading and Service Caching Problem

After addressing the RA and CA problems, we can shift our focus to solving the TOSA problem. Typically, task offloading and service caching are closely intertwined, as tasks are more likely to be offloaded when the associated service is cached, and services are more likely to be cached when they are frequently requested by vehicular tasks. Furthermore, since both task offloading and service caching variables are binary, searching for the optimal solution typically involves exponential time complexity, rendering it infeasible for real-time VEC environments.

In view of the exceptional perception and decision-making capabilities, proximal policy optimization (PPO) based algorithms have been increasingly applied across multiple domains. Hence, in this paper, we propose a greedy PPO (GrePPO) algorithm to tackle the TOSA problem. In the GrePPO algorithm framework, a greedy rule is employed to guide the service caching process, while the PPO algorithm is utilized to determine the task offloading strategy. For service caching, since the edge server has global information about the offloading requests and vehicle services, it can analyze the distribution of service demands. Once all the offloading requests are collected, the edge server determines the frequency of each service requested by vehicular tasks. Subsequently, R sorts these services in descending order based on their request frequencies. Services are cached sequentially until the cumulative data size of the cached services reaches or exceeds the caching capacity. This greedy approach primarily requires a time complexity of $O(K \log K)$ for sorting the services, thereby enabling efficient caching decisions.

On the other hand, the PPO algorithm consists of two key components: the actor network (i.e., the policy function) and the critic network (i.e., the value function). The actor network $\pi_\theta(a_t|s_t)$, which employs policy gradients, takes the current state s_t as input and directly outputs the action probabilities. In contrast, the value-based critic network $V_{\theta'}(s_t)$ assesses the quality of actions taken in the current state s_t, thereby facilitating the improvement of the actor network's performance. The basic elements in the PPO algorithm for the TOSA problem are defined as follows:

- *State Space:* $\mathcal{S}$ is a collection of all the environment states in the VEC system, and $\mathcal{S} \triangleq \{s_t = (S_\mathcal{N}, S_\mathcal{K}, S_R)\}$, $t \in \{0, 1, 2, \cdots\}$. $S_\mathcal{N}$ denotes the states of vehicles and related tasks. $S_\mathcal{N} \triangleq \{veh_n, task_n\}_{n \in \mathcal{N}}$, where $veh_n = \{loc_n, vel_n, dir_n, p_n, g_n, f_n\}$ denotes the comprehensive information of vehicle n, including its location, velocity, driving direction, transmission power, wireless channel gain, and local computing capability; $task_n = \{I_n, k, D_n\}$ describes the information of task n. $S_\mathcal{K} = \{s_k, w_k, M_k, \{s_{k_i} | 1 \leq i \leq M_k\}, V_k\}_{k \in \mathcal{K}}$ denotes the states of the vehicle services, and $S_R = \{C, f_R^{max}\}$ represents the states of the edge server R.
- *Action Space:* $\mathcal{A}$ represents the set of all possible actions within the VEC system that the agent can execute based on the observed state. For the TOSA problem, the agent determines which tasks to offload by observing the environmental state, and $\mathcal{A}$ can be defined as $\mathcal{A} \triangleq \{a_t = \boldsymbol{\lambda}\}$, $t \in \{0, 1, 2, \cdots\}$.
- *Reward Function:* The agent receives an immediate reward r_t upon taking an action in current state s_t. The value of r_t serves as a quantitative measure to evaluate the advantages and disadvantages of the action chosen in the current state. A higher r_t indicates that this action is more favorable compared to other possible actions. The immediate reward r_t is calculated using the reward function. In this paper, our goal is to minimize the objective function $J(\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\phi}, \boldsymbol{f_R})$, and thus the immediate reward r_t is defined as $r_t = -J(\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\phi}, \boldsymbol{f_R})$.

The PPO algorithm employs the advantage function to quantify the benefit of the action produced by the actor network in the current state, using k-step Generalized Advantage Estimation (GAE), as formally defined below.

$$\hat{A}_t = \sum_{i=0}^{k-1} (\gamma\xi)^k \vartheta_{t+k}, \tag{25}$$

$$\vartheta_t = r_t + \gamma V_{\theta'}(s_{t+1}) - V_{\theta'}(s_t), \tag{26}$$

where γ and ξ are the discount factor and the hyperparameter for GAE, respectively. ϑ_t is the TD error, representing the difference between the predicted and true values of critic networks. ϑ_t reflects the advantage of taking action a_t in current state s_t relative to the average. The PPO algorithm utilizes off-policy data to accelerate the model training. Particularly, it utilizes the importance sampling technique to reuse data collected by the old policy $\pi_{\theta_{old}}(a_t|s_t)$, thereby enhancing the sampling efficiency. By incorporating the importance sampling ratio, the actor network aims to maximize the clipped surrogate objective function, as defined below:

$$\mathcal{L}(\theta) = \mathbb{E}_t[\min(rio \cdot \hat{A}_t, clip(rio, 1-\epsilon, 1+\epsilon) \cdot \hat{A}_t)], \quad (27)$$

where rio is the importance sampling ratio and is defined as:

$$rio = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}.$$

The clip mechanism $clip(ratio, 1-\epsilon, 1+\epsilon)$ is employed to constrain the step size and ensure training stability. Specifically, when $ratio \leq 1-\xi$, it is clipped into $1-\xi$; and when $ratio \geq 1+\xi$, it is clipped to $1+\xi$.

The algorithm for describing the entire procedure is presented in Algorithm 2, consisting of two distinct phases: the training data generation phase and the model learning phase. In the data generation phase, the policy network $\pi_{\theta_{old}}(a_t|s_t)$ interacts with the VEC environment to generate a series of trajectories. These transitions are subsequently stored in the experience buffer B. In the model learning phase, a batch of $Batch\_Size$ transitions is sampled from the experience buffer. For each transition (s_t, a_t, r_t, s_{t+1}) in the batch, the TD error and GAE are computed, respectively. The state value evaluation for s_t is obtained using the critic network $V_{\theta'}(s_t)$. Subsequently, the actor network $\pi_\theta(a_t|s_t)$ and critic network $V_{\theta'}(s_t)$ are updated. To enhance the efficiency of model training, the training data within the batch are reused $Max\_Learning\_Num$ times. Finally, the old policy is updated with the trained policy, i.e., $\pi_{\theta_{old}}(a_t|s_t) = \pi_\theta(a_t|s_t)$. This iterative process continues until the model converges or the termination conditions are met.

VI. Performance Analysis

A. Parameters Settings

The proposed optimization strategy for caching-assisted collaborative task offloading in VEC is extensively evaluated through simulations in this section. In the simulation setup, RSUs and vehicles are randomly distributed. The communication coverage of an RSU is configured to 200 meters. The number of vehicles ranges from 10 to 40. By default, the number of cacheable services is set to 15. Other key parameters involved in the simulation are summarized in Table III. Furthermore, we evaluate the performance of our strategy in comparison to the following four approaches.

- PPO-based Offloading and Caching (POPC): This approach adopts a PPO-based algorithm to tackle the TOSA problem and the same strategy as ours for addressing the CA and RA problems [33]. The difference between our approach and POPC lies in the fact that POPC uses the actor network to generate both the task offloading and service caching decisions.
- DQN-based Offloading and Greedy Caching (DOGC): This approach employs a DQN-based algorithm to solve the task offloading problem and a greedy strategy to tackle the service caching problem [34]. The distinction of our approach compared to DOGC lies in using a DQN-based algorithm rather than a PPO-based algorithm for generating task offloading decisions.
- A2C-based Offloading and Greedy Caching (AOGC): This approach adopts an A2C-based approach to address the task offloading problem [35], and greedy strategy to address the service caching problem.
- Random Offloading and Greedy Caching (ROGC): This approach employs a random strategy for generating task offloading decisions and a greedy strategy for producing service caching decisions. This simplest approach is widely adopted as the baseline approach in existing literature [14].

TABLE III
Parameter Settings

Parameter	Value	Parameter	Value
N	$15 \sim 40$	M	4
vel_n	$5 \sim 10$ m/s	I_n	$300 \sim 500$ KB
D_n	$0.8 \sim 1.4$s	M_k	$8 \sim 12$
s_{k_i}	$3 \sim 5$MB	K	15
w_k	$400 \sim 600$ Mcycles	κ	800 MHz
C	500 MB	F_R	40 GHz
T_k^{init}	0.01s	f_n	$0.4 \sim 0.8$GHz

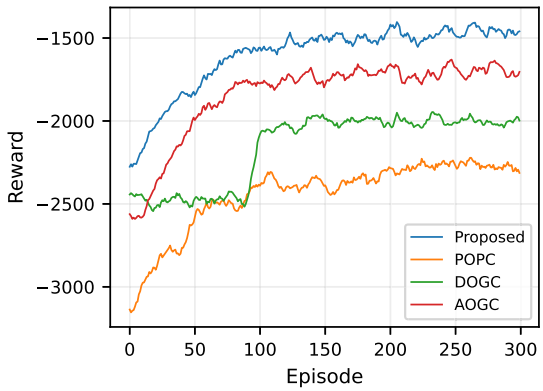


Fig. 2. Comparison between reward values across different algorithms.

B. Simulation Results and Analysis

The variations in reward values of different algorithms during the training process with respect to the number of training episodes are illustrated in Fig. 2. Clearly, our strategy achieves the highest reward value among the compared algorithms. After approximately 100 episodes, the reward value progressively converges to around -1500. Moreover, the curve exhibits minimal fluctuations after achieving convergence, indicating superior robustness relative to other algorithms. The reward value of POPC is the lowest among the four approaches. AOGC attains the second-highest reward value, suggesting that the A2C algorithm

Algorithm 2: Greedy PPO Algorithm for TOSA Problem (GrePPO).

Input: Parameters related to task and service information
Output: The task offloading profile

Initialize actor network $\pi_\theta(a_t|s_t)$ and $\pi_{\theta_{old}}(a_t|s_t) = \pi_\theta(a_t|s_t)$, critic network $V_{\theta'}(s_t)$, experience buffer B, maximum iterations MIT;
for $episode = 1, 2, 3, \cdots$ **do**
  Get initial state s_1 from the VEC environment;
  for $t = 1$ to EP_Max_Step **do**
    Get a_t by $a_t = \pi_{\theta_{old}}(a_t|s_t)$;
    Get $\boldsymbol{\lambda}$ by sampling the action a_t;
    Determine $\boldsymbol{\delta}$ using the greedy approach;
    Call Alg. 1 to address the CA problem;
    Address the RA problem, based on Eq. (22);
    Calculate r_t by $r_t = -J(\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{\phi}, \boldsymbol{f_R})$;
    Obtain the next state s_{t+1};
    Store the transition (s_t, a_t, r_t, s_{t+1}) to B;
    if $t\%Batch\_Size = 0$ **then**
      Retrieve $Batch\_Size$ transitions (s_t, a_t, r_t, s_{t+1}) from B;
      for $k = 1$ to $Max\_Learning\_Num$ **do**
        for each transition in Min_Batch **do**
          Get $V_{\theta'}(s_t)$ by critic network;
          Calculate TD error using Eq. (26);
          Calculate GAE according to Eq. (25);
          Update actor network $\pi_\theta(a_t|s_t)$ according to Eq. (27);
          Update critic network $V_{\theta'}(s_t)$ according to Eq. (25);
        end for
      end for
      $\pi_{\theta_{old}}(a_t|s_t) = \pi_\theta(a_t|s_t)$;
    end if
  end for
  Clear experience buffer B;
end for

combined with a greedy caching strategy is a more effective strategy compared to DOGC and POPC.

We conducted a series of experiments to evaluate the efficiency of our strategy using several performance metrics, such as the objective function value, task completion rate, and cache hit rate. Fig. 3 shows the performance comparison across different algorithms with the increasing number of vehicular tasks. Fig. 3(a) illustrates the influence of different numbers of tasks on the objective function values for algorithms. Our strategy can achieve the lowest values of the objective function among these approaches, no matter how the number of tasks varies. Furthermore, our strategy exhibits the smallest increase when the number of tasks increases. The advantage of our approach regarding the objective function values can be attributed to its superior mechanism for task offloading and service caching. Addressing these subproblems in our fashion can closely approximate the global optimum, even as the number of tasks increases and the system state space becomes more complex. ROGC achieves the worst performance among these approaches. The reason for this is acceptable and explainable, since it is the simplest benchmark. AOGC achieves the second-best performance, and POPC is worse than AOGC.

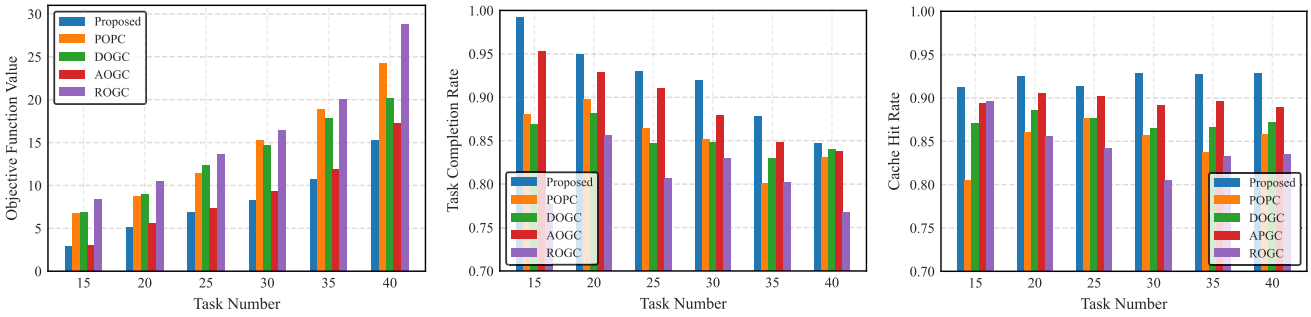
From the results shown in this figure, two valuable conclusions can be drawn. First, tackling the task offloading problem and the service caching problem separately tends to yield better outcomes compared to addressing them concurrently, such as by using the actor network to generate offloading and caching decisions simultaneously. Second, our strategy exhibits superior performance compared to the baseline approaches.

The simulation results presented in Fig. 3(b) compare the task completion rates of each algorithm with the increasing number of vehicular tasks. It is evident that our proposed strategy achieves a significantly higher task completion rate compared to other approaches, regardless of how the number of tasks fluctuates. In comparison, ROGC, as the simplest benchmark, exhibits the lowest performance in terms of task completion rate, indicating its limited adaptability to complex environments. AOGC ranks second among the five approaches in terms of performance. The simulation results further confirm that addressing task offloading and service caching problems independently tends to yield more favorable outcomes than tackling them as an integrated whole.

Fig. 3(c) presents the cache hit rates of each algorithm with the increasing number of vehicular tasks. Similar to the results presented in Fig. 3(a) and (b), our strategy and AOGC achieve the top two highest cache hit rates, which indicates that our cache decision-making strategy can effectively adapt to various changes in task volume. DOGC can maintain a relatively high cache hit rate, but the fluctuation is very large when the number of tasks changes. The cache hit rates of POPC and ROGC are relatively low. Particularly, the suboptimal cache hit performance of POPC may arise from the inherent complexity in coordinating the PPO algorithm for both offloading and caching decisions. The low cache hit rate of ROGC can likely be attributed to the absence of a robust linkage mechanism between its random offloading strategy and greedy caching strategy. The stochastic nature of its offloading decisions hinders ROGC's ability to effectively leverage service cache decisions, thus constraining its overall cache hit performance.

In summary, Fig. 3 illustrates the performance comparison of various algorithms across multiple metrics, including objective function values, task completion rates, and cache hit rates. As the number of tasks increases, our strategy not only achieves a higher performance level but also demonstrates stronger robustness, such as minimal performance degradation or stable performance under varying conditions. The simulation results emphasize the effectiveness and efficiency of our approach in terms of task offloading, resource allocation, and service caching, thereby enabling it to efficiently address task offloading and adapt to complex VEC environments.

Next, we investigate the influence of vehicle speeds on the task completion rate, as shown in the simulation results in



(a) Objective function value comparison (b) Task completion rate comparison (c) Cache hit rate comparison

Fig. 3. Performance comparison across different algorithms.

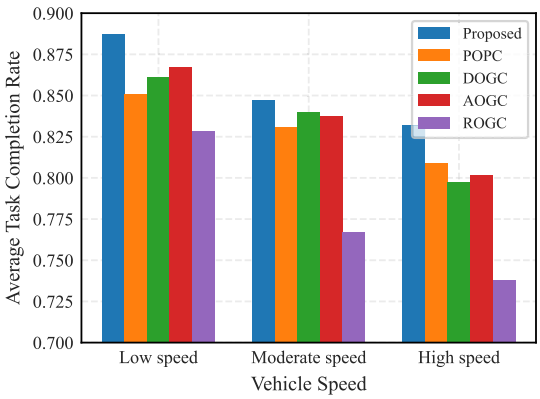


Fig. 4. Task success rate with different vehicle speeds.

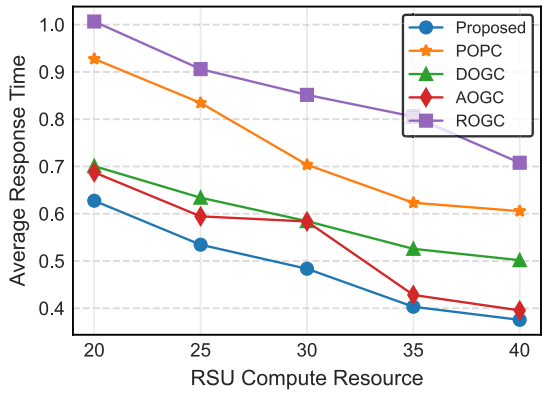


Fig. 5. Response time comparison with different RSU computing capabilities.

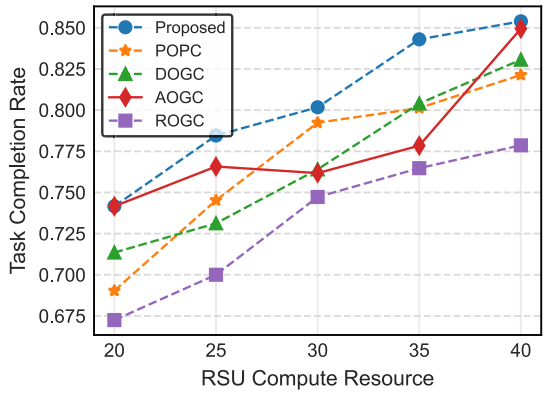
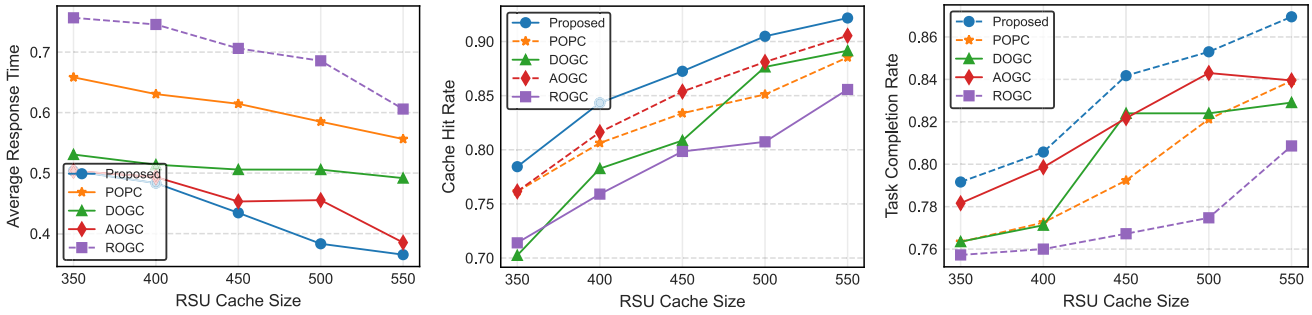


Fig. 6. Task completion rate comparison with different RSU computing capabilities.

Fig. 4. Vehicle speeds are categorized into three levels: low, medium, and high. The low-speed range (4–8 m/s) corresponds to vehicle movement in congested roads, residential areas, or signal-dense urban zones. The medium-speed range (8–12 m/s) represents typical mobility on urban arterial roads with relatively smooth traffic flow. The high-speed range (12–20 m/s) reflects conditions on urban expressways or elevated roads. This classification effectively captures the motion characteristics of vehicles in diverse urban traffic environments. As illustrated in the figure, the task completion rate for all approaches decreases significantly as vehicle speed increases. Nevertheless, our approach consistently achieves the highest task completion rate across all speed levels. Generally, higher vehicle speeds reduce the effective dwell time within the RSU's coverage area. A fast-moving vehicle may traverse the entire coverage zone within a single time slot. Consequently, even if the edge server completes task computation promptly, the vehicle might have already moved out of range by the time the results are ready for transmission. These transient disconnections due to rapid mobility prevent the successful delivery of computation results, thereby increasing task failure rates.

Fig. 5 illustrates the impact of RSU computing capabilities on the performance of various algorithms in terms of average response time (i.e., the objective function value). As shown, our strategy consistently achieves the shortest average response time across all tested RSU computing capabilities. Specifically, our approach not only exhibits the best performance compared to other approaches but also demonstrates the most significant reduction in response time as RSU computing capabilities improve. These results confirm that our strategy effectively exploits additional computing resources to accelerate task processing. Additionally, the average response times of POPC, DOGC, and AOGC decrease with the increasing RSU computing capabilities. Their performance falls between that of our strategy and ROGC, with significant variations observed across these three approaches.



(a) Average response time comparison (b) Cache hit rate comparison (c) Task completion rate comparison

Fig. 7. Performance comparison against different caching capabilities.

Fig. 6 demonstrates the trend of task completion rates across various algorithms as the computing capabilities of RSU increase. The significant superiority of our strategy in terms of task completion rate further substantiates the effectiveness of our approach in addressing the TOSA problem. Enhancing RSU computing capabilities indeed provides a robust hardware foundation for successful task execution, and our strategy can take advantage of these resources more efficiently compared to other approaches. In contrast, POPC, DOGC, and AOGC may exhibit limitations in solving the TOSA problem, which restricts their ability to utilize additional computing resources as effectively as ours to improve the task completion rate. Generally, tasks may fail due to suboptimal resource scheduling. The random offloading mechanism employed by ROGC is the main reason for its low task completion rate, as this approach neglects service caching and often fails to meet specified deadlines, resulting in the lowest task completion rate. Under varying configurations of RSU computing capabilities, our strategy consistently outperforms other approaches in two critical performance metrics: average response time and task completion rate. As RSU computing capability increases, the performance of all approaches improves, thereby confirming the positive influence of computing resources on the VEC system.

Fig. 7 examines the impact of caching capability on the performance of the algorithms. Fig. 7(a) illustrates the trend of average response time as the RSU cache size varies. Our strategy consistently achieves the lowest average response time across all approaches, regardless of changes in cache size. While the average response times of POPC, DOGC, and AOGC decrease with increasing cache size, they remain marginally higher than that of our strategy. Notably, ROGC consistently exhibits the highest average response time among all approaches. In general, larger RSU cache sizes allow for the storage of more service replicas, thereby improving the cache hit rate and decreasing the average response times.

The trend of the cache hit rate with varying cache sizes is illustrated in Fig. 7(b). Our strategy consistently achieves the highest cache hit rate compared to all other approaches. Typically, the cache hit rate increases as the cache size grows larger. This is expected, as an increase in cache capacity enhances the ability to store a wider variety of services, thereby increasing the likelihood of cache hits. In contrast, ROGC continues to demonstrate the lowest cache hit rate owing to its random task offloading strategy.

Fig. 7(c) illustrates the variation in task completion rate as RSU cache sizes change. Our strategy consistently achieves the highest task completion rate compared to all other approaches. While the task completion rates of POPC, DOGC, and AOGC also increase with larger cache sizes, their values remain significantly lower than our strategy. In contrast, ROGC consistently exhibits the lowest task completion rate due to the same reason analyzed above. In summary, by comprehensively evaluating various configurations of RSU cache sizes, our strategy demonstrates substantial superiority across three critical performance metrics: average response time, cache hit rate, and task completion rate. As the RSU cache size increases, the performance of all approaches generally improves, highlighting the significance of cache resources in enhancing VEC systems. Nevertheless, our strategy exploits these cache resources more efficiently compared to alternative strategies.

VII. Conclusion

In this paper, we explore the potential of collaborative offloading among vehicles whose tasks are associated with the same service. Specifically, for services not cached at the RSU, vehicles collaboratively perform task offloading, with each responsible for specific service component offloading. Consequently, even when services are not cached, the corresponding response latency in the VEC system can still be reduced. The primary objective of this paper is to minimize the overall response latency across the VEC system. Several algorithms are proposed to address the decomposed subproblems, respectively. The simulation results indicate that our solution outperforms existing methods in convergence speed, average response latency, and task completion rate. We intend to incorporate component dependencies in future work to enhance the robustness of our system model.

References

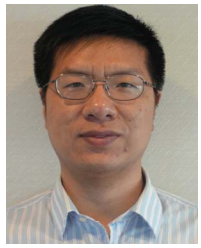
[1] Y. Liang, H. Tang, H. Wu, Y. Wang, and P. Jiao, "Lyapunov-guided offloading optimization based on soft actor-critic for ISAC-aided Internet of Vehicles," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 14708–14721, Dec. 2024.

[2] S. Wang et al., "Energy-efficient collaborative task offloading in multi-access edge computing based on deep reinforcement learning," *Ad Hoc Netw.*, vol. 169, 2025, Art. no. 103743.
[3] Z. Chen, Z. Huang, J. Zhang, H. Cheng, and J. Li, "Resource allocation and collaborative offloading in multi-UAV-assisted IoV with federated deep reinforcement learning," *IEEE Internet Things J.*, vol. 12, no. 5, pp. 4629–4640, Mar. 2025.
[4] H. Li, K. Xiong, Y. Lu, W. Chen, P. Fan, and K. B. Letaief, "Collaborative task offloading and resource allocation in small-cell MEC: A multi-agent PPO-based scheme," *IEEE Trans. Mobile Comput.*, vol. 24, no. 3, pp. 2346–2359, Mar. 2025.
[5] X. Li et al., "Cloud-edge-end collaborative intelligent service computation offloading: A digital twin driven edge coalition approach for industrial IoT," *IEEE Trans. Netw. Serv. Manag.*, vol. 21, no. 6, pp. 6318–6330, Dec. 2024.
[6] C. Barrios and M. Kumar, "Service caching and computation reuse strategies at the edge: A survey," *ACM Comput. Surv.*, vol. 56, no. 2, 2024, Art. no. 43.
[7] Y. Zhao et al., "Joint content caching, service placement, and task offloading in UAV-enabled mobile edge computing networks," *IEEE J. Sel. Areas Commun.*, vol. 43, no. 1, pp. 51–63, Jan. 2025.
[8] C. Tang, W. Chen, C. Zhu, Q. Li, and H. Chen, "When cache meets vehicular edge computing: Architecture, key issues, and challenges," *IEEE Wirel. Commun.*, vol. 29, no. 4, pp. 56–62, Aug. 2022.
[9] C. Tang, Y. Zhao, and H. Wu, "Lyapunov-guided optimal service placement in vehicular edge computing," *China Commun.*, vol. 20, no. 3, pp. 201–217, Mar. 2023.
[10] C. Tang, Z. Li, H. Wu, S. Xiao, and R. Li, "Deep reinforcement learning-based collaborative computation offloading for distributed vehicular edge computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 26, no. 12, pp. 22344–22356, Dec. 2025.
[11] J. Du, H. Wu, M. Xu, and R. Buyya, "Computation energy efficiency maximization for noma-based and wireless-powered mobile edge computing with backscatter communication," *IEEE Trans. Mobile Comput.*, vol. 23, no. 6, pp. 6954–6970, Jun. 2024.
[12] H. Wu, L. Tian, H. Tang, R. Li, and P. Jiao, "Graph convolutional reinforcement learning-guided joint trajectory optimization and task offloading for aerial edge computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 26, no. 10, pp. 17487–17498, Oct. 2025.
[13] M. Zhou, J. Tian, D. Li, T. Li, and J. Bian, "Collaborative service caching for delay minimization in vehicular edge computing networks," *Veh. Commun.*, vol. 52, 2025, Art. no. 100877.
[14] C. Tang, G. Yan, H. Wu, and C. Zhu, "Computation offloading and resource allocation in failure-aware vehicular edge computing," *IEEE Trans. Consum. Electron.*, vol. 70, no. 1, pp. 1877–1888, Feb. 2024.
[15] Y. Zhang, Z. Na, Z. Wen, A. Nallanathan, and W. Lu, "Joint service caching, computation offloading and resource allocation for dual-layer aerial Internet of Things," *Comput. Netw.*, vol. 257, 2025, Art. no. 110974.
[16] C. Ling et al., "Cooperative service caching in vehicular edge computing networks based on transportation correlation analysis," *IEEE Internet Things J.*, vol. 11, no. 12, pp. 22754–22767, Jun. 2024.
[17] C. Cheng, L. Zhai, X. Zhu, Y. Jia, and Y. Li, "Dynamic task offloading and service caching based on game theory in vehicular edge computing networks," *Comput. Commun.*, vol. 224, pp. 29–41, 2024.
[18] Y. Gao, H. Zhang, Q. Liu, and Y. Deng, "Joint task offloading and service caching for vehicular edge computing with energy constraints," in *Proc. 16th Int. Conf. Wireless Commun. Signal Process.*, Hefei, China, Oct. 2024, pp. 613–618.
[19] J. Shen, Y. Lin, Y. Zhang, W. Zhang, F. Shu, and J. Li, "Content caching-assisted vehicular edge computing using multi-agent graph attention reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 74, no. 2, pp. 3509–3514, Feb. 2025.
[20] J. Chen, X. Wang, and X. Shen, "Goal-driven trusted collaborator selection and task offloading in dynamic collaborative systems," *IEEE Internet Things J.*, vol. 12, no. 7, pp. 8537–8551, Apr. 2025.
[21] J. Xu, Y. Yao, X. Xu, W. Feng, and P. Li, "Joint optimization of task offloading and resource allocation of fog network by considering matching externalities and dynamics," *IEEE Trans. Mobile Comput.*, vol. 24, no. 4, pp. 2534–2550, Apr. 2025.
[22] W. Chu, X. Jia, Z. Yu, J. C. S. Lui, and Y. Lin, "Joint service caching, resource allocation and task offloading for MEC-based networks: A multi-layer optimization approach," *IEEE Trans. Mobile Comput.*, vol. 23, no. 4, pp. 2958–2975, Apr. 2024.
[23] Z. Li, C. Yang, X. Huang, W. Zeng, and S. Xie, "CoOR: Collaborative task offloading and service caching replacement for vehicular edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 72, no. 7, pp. 9676–9681, Jul. 2023.
[24] Z. Ma et al., "Dependent task offloading and service caching with state management for mobile edge computing," in *Proc. IEEE Int. Conf. Commun.*, Rome, Italy, IEEE, 2023, pp. 6249–6254.
[25] Q. Shen, B. Hu, and E. Xia, "Dependency-aware task offloading and service caching in vehicular edge computing," *IEEE Trans. Veh. Technol.*, vol. 71, no. 12, pp. 13182–13197, Dec. 2022.
[26] W. Fan et al., "Joint task offloading and service caching for multi-access edge computing in WiFi-cellular heterogeneous networks," *IEEE Trans. Wirel. Commun.*, vol. 21, no. 11, pp. 9653–9667, Nov. 2022.
[27] Z. Chai, Z. Chai, J. Ren, and D. Yuan, "Computation offloading and task caching in the cloud-edge collaborative IoVs: A multi-objective evolutionary algorithm," *Simul. Model. Pract. Theory*, vol. 141, 2025, Art. no. 103087.
[28] Z. Huang, Z. Kuang, B. Xu, Y. Bi, and A. Liu, "Dependency-aware task collaborative offloading and resource allocation in UAV enabled edge computing," *Peer Peer Netw. Appl.*, vol. 18, no. 3, 2025, Art. no. 118.
[29] Z. Ouyang et al., "A novel redundant service caching and task offloading method in mobile edge computing," in *Proc. Int. Conf. Web Serv.*, 2024, pp. 31–46.
[30] J. Zhou and X. Zhang, "Fairness-aware task offloading and resource allocation in cooperative mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 5, pp. 3812–3824, Mar. 2022.
[31] C. Tang, H. Wu, R. Li, and J. J. P. C. Rodrigues, "Joint optimization of task offloading content caching and resource allocation in vehicular edge computing," *ACM Trans. Auton. Adapt. Syst.*, vol. 20, no. 4, pp. 1–24, 2025.
[32] C. N. Potts, "Analysis of a linear programming heuristic for scheduling unrelated parallel machines," *Discret. Appl. Math.*, vol. 10, no. 2, pp. 155–164, 1985.
[33] E. Mustafa, J. Shuja, F. Rehman, A. Namoun, M. Bilal, and A. Iqbal, "Computation offloading in vehicular communications using PPO-based deep reinforcement learning," *J. Supercomput.*, vol. 81, no. 4, 2025, Art. no. 547.
[34] L. Zhai, Z. Lu, J. Sun, and X. Li, "Joint task offloading and computing resource allocation with DQN for task-dependency in multi-access edge computing," *Comput. Netw.*, vol. 263, 2025, Art. no. 111222.
[35] L. Wang, J. Li, M. Dai, and H. Zhang, "Energy efficient multi-task offloading in satellite- assisted vehicular edge computing networks: An improved soft-actor-critic approach," *IEEE Trans. Veh. Technol.*, vol. 74, no. 4, pp. 6473–6487, Apr. 2025.

Chaogang Tang (Member, IEEE) received the BS degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, and PhD degree from the School of Information Science and Technology, University of Science and Technology of China, Hefei, China, and the Department of Computer Science, City University of Hong Kong, under a joint PhD Program, in 2012. He is currently with the China University of Mining and Technology. His research interests include mobile cloud computing, fog computing, Internet of Things, and Big Data.

Shucai Wang received the BS degree in computer science and technology from Shandong Agricultural University, Tai'an, China, in 2024. He is currently working toward the MS degree with the School of Computer and Science Technology, China University of Mining and Technology, Xuzhou, China. His current research interests include edge/cloud computing and computation offloading in the Internet of Vehicles.

Huaming Wu (Senior Member, IEEE) received the BE and MS degrees in electrical engineering from the Harbin Institute of Technology, China in 2009 and 2011, respectively, and the Ph.D. degree with the highest honor in computer science with Freie Universität Berlin, Germany in 2015. He is currently a professor with the Center for Applied Mathematics, Tianjin University. His research interests include mobile cloud computing, edge computing, Internet of Things, deep learning, complex networks, and DNA storage.



Ruidong Li (Senior Member, IEEE) received the bachelor's degree in engineering from the Department of Information Science and Electronic Engineering, Zhejiang University, Zhejiang, China, in 2001, and the master's and Doctorate of Engineering degrees in computer science from the University of Tsukuba, Tsukuba, Japan. From 2008 to 2021, he was a senior researcher with the National Institute of Information and Communications Technology, Tokyo, Japan. He is currently an associate professor with Kanazawa University, Kanazawa, Japan.