Joint Optimization of Task Offloading Content Caching and Resource Allocation in Vehicular Edge Computing

CHAOGANG TANG, School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, China, and Mine Digitization Engineering Research Center of the Ministry of Education, Xuzhou, China
HUAMING WU, Center for Applied Mathematics, Tianjin University, Tianjin, China
RUIDONG LI, Institute of Science and Engineering, Kanazawa University, Kanazawa, Japan
JOEL J. P. C. RODRIGUES, Federal University of Piauí (UFPI), Teresina - PI, Brazil

In Vehicular Edge Computing (VEC) environments, the increasingly complicated functional and non-functional requirements from vehicular applications such as MetaVehicles usually incur larger sizes of task-input data, which not only increase the transmission delay of task-input data via the front-haul links but also degrade the quality of experience for users, even if computation tasks can be offloaded and executed at the network edge. In this article, we put forward a caching-enabled task offloading strategy, by caching and reusing the universal context data at the edge server, to avoid duplicated data transmission in VEC systems. The goal is to minimize the overall response latency for all the tasks, by jointly optimizing task offloading, content caching, and resource allocation decisions in VEC. The optimization problem is formulated as a Mixed-Integer Nonlinear Programming (MINLP) problem. To efficiently solve this problem, we decompose this problem into two subproblems, namely, the computing Resource Allocation (RA) problem and the Joint Offloading and Caching (JOC) problem. The corresponding algorithms are put forward to solve the content caching and task offloading problems, respectively. Numeric evaluation reveals that our strategies and algorithms can achieve better performance in minimizing the overall response latency, in comparison with other approaches.

CCS Concepts: • **Human-centered computing** → **Mobile computing**; • **Networks** → *Mobile ad hoc networks*; • **Computer systems organization** → *Sensor networks*;

Additional Key Words and Phrases: vehicular edge computing, task offloading, content caching, resource allocation

ACM Reference format:
Chaogang Tang, Huaming Wu, Ruidong Li, and Joel J. P. C. Rodrigues. 2025. Joint Optimization of Task Offloading Content Caching and Resource Allocation in Vehicular Edge Computing. *ACM Trans. Autonom. Adapt. Syst.* 20, 4, Article 35 (November 2025), 24 pages.
https://doi.org/10.1145/3732782

This work was supported by the National Natural Science Foundation of China (Grant No. 62071327) and Tianjin Science and Technology Planning Project (Grant No. 22ZYYYJC00020), and partially funded by Brazilian National Council for Scientific and Technological Development—CNPq, via Grant No. 306607/2023-9.
Authors' Contact Information: Chaogang Tang, School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, China, and Mine Digitization Engineering Research Center of the Ministry of Education, Xuzhou, China; e-mail: cgtang@cumt.edu.cn; Huaming Wu (corresponding author), Center for Applied Mathematics, Tianjin University, Tianjin, China; e-mail: whming@tju.edu.cn; Ruidong Li, Institute of Science and Engineering, Kanazawa University, Kanazawa, Japan; e-mail: liruidong@ieee.org; Joel J. P. C. Rodrigues, Federal University of Piauí (UFPI), Teresina - PI, Brazil; e-mail: joeljr@ieee.org.

ACM 1556-4703/2025/11-ART35
https://doi.org/10.1145/3732782

1 Introduction

The advent of 5G-enabled smart vehicles in China several years ago marked a significant milestone. That is, beyond delivering cost-effective infotainment services, smart vehicles have evolved into a social platform that fosters person-to-person and vehicle-to-vehicle interactions, even as drivers traverse the roads. Additionally, various intelligent driving systems and algorithms, such as Tesla's FSD and XPENG's XNGP, have been devised to achieve Level 3 and even Level 4 driving automation. These remarkable advancements owe their existence to the rapid progress in Information and Communication Technologies, the **Internet of Things (IoT)**, and AI technologies.

IoT-enabled smart vehicles continually generate vast amounts of data (or tasks), encompassing both time-critical information vital for driving safety and time-insensitive data related to infotainment services. The remote cloud center serves as an ideal platform for processing time-insensitive data and tasks. However, it falls short when handling time-critical tasks and data due to the extended data transmission times over the core (backbone) networks. In response to this challenge, **Vehicular Edge Computing (VEC)** has emerged as a promising computing paradigm, focusing on processing tasks and data in proximity to their source, i.e., the vehicles themselves.

Extensive literature has delved into topics such as task offloading, data processing, and the management of computing resources within VEC systems. Nonetheless, the dynamic topology inherent in **Vehicular Ad-hoc Networks (VANETs)**, the increasingly complicated functional requirements of service requestors, and the limited computing capabilities of edge servers all present substantial obstacles to achieving optimal performance, including throughput, response latency, and energy consumption in VEC systems and networks. Furthermore, current works mainly pay attention to the task offloading in VEC, through a hypothesis that the task-input data and related data libraries are both deposited on the vehicle side.

However, the increasingly complicated functional and non-functional requirements from vehicular applications such as MetaVehicles [13, 27] usually incur larger sizes of task-input data, which will increase the transmission delay of task-input data via the front-haul links. The task-input data usually consist of two distinct parts. One is the user-oriented data such as specific system settings and input parameters, and the other is the universal context information such as road conditions and traffic accidents. In most cases, the universal context information can be reused. For some metaverse-integrated vehicular applications, such context information may exist in the form of multimedia content, so the content size is often much larger than the size of the user-oriented data. The transmission of these contents will consume a mass of bandwidth resources of the front-haul networks [21, 23].

The reduction of redundant data transmission holds paramount importance for various autonomous and adaptive systems, particularly in the context of autonomous driving. For instance, even a slight delay in decision-making for autonomous driving can lead to immeasurable damages. Numerous task offloading requests are generated within these autonomous and adaptive systems, such as intelligent vehicles in the context of autonomous driving. In this article, we present a generalization of the task-input data by categorizing it into two distinct components: user-oriented data encompassing specific system settings and input parameters, and universal context data comprising information on road conditions and traffic accidents. Subsequently, we propose a caching-enabled task offloading strategy that considers a generalized scenario wherein the universal contents associated with the task-input data can be pre-cached at the edge server to eliminate redundant data transmission, thereby enhancing resource utilization efficiency in VEC systems. The goal of this article is to minimize the overall response latency for all the tasks, by jointly optimizing task offloading, content caching, and **Resource Allocation (RA)** decisions in VEC. Generally, the major contributions of this article can be outlined as follows:

— We present a generalization of the task-input data by categorizing it into two distinct components: user-oriented data and universal context data. Considering that minimizing redundant data transmission can significantly enhance the performance of autonomous and adaptive systems, we propose leveraging caching mechanisms at the edge server to reuse universal context data.
— A **Joint Task Offloading, Content Caching, and Resource Allocation (JTOCA)** optimization problem is put forward in this article, and the optimization goal is to minimize the overall response latency for all the tasks in the optimization period. Particularly, we model the JTOCA problem as a **Mixed-Integer Nonlinear Programming (MINLP)** Problem, which usually takes exponential time to find the optimal solution.
— Considering the rigorous latency requirements of vehicular tasks and the challenges in solving the JTOCA problem, we decompose this problem into two subproblems, i.e., the computing RA problem and the **Joint Offloading and Caching (JOC)** problem. The RA problem is a convex optimization problem and the JOC problem is a combinatorial optimization problem. To lower the difficulty in solving the JOC problem, we propose two heuristic algorithms to solve the content caching and task offloading problems, respectively.
— Numeric evaluation is extensively conducted in the simulation to investigate our strategy from multiple angles. Several existing strategies are also selected as the comparison approaches in the simulation. The experimental results reveal that our strategies and algorithms can achieve better performance in minimizing the overall response latency, in comparison with other approaches.

The rest of the article is organized as follows: Section 2 reviews the state-of-the-art literature on this topic. The system model and the formulated optimization problem are presented in Sections 3 and 4, respectively. The low-complexity algorithm designed for solving the optimization problem is presented in Section 5. The algorithm evaluation is presented in Section 6 and followed by the conclusion presented in Section 7.

2 Related Work

The research revolving around VEC has been conducted for recent years [11, 18, 37]. The main focus is how to spread the computation tasks among vehicles, edge servers, or the cloud center in a typical end-edge-cloud layered VEC architecture, aiming to improve the efficiency of the VEC system, regarding throughput, response latency, energy consumption, networking resources, and so on. To improve the task offloading efficiency, both content-oriented caching [1, 2, 5, 16, 36] and task-oriented caching technologies [15, 24, 31, 33] have been put forward in the past few years.

The rapid proliferation of smart vehicles enables frequent transmission, exchange, and sharing of massive contents, and various stringent requirements are also posed during this process. Vehicular edge caching can reduce content delivery latency, but the constrained storage capacity of edge nodes and the dynamic of VANETs as a barrier hinders the efficiency of vehicular edge caching. Thus, a social-aware vehicular edge caching mechanism is put forward in [36], which can adaptively tune the caching capability of **Roadside Unit (RSU)**, based on social information such as user preferences and service availability. The digital twin technology is adopted to build the social relation model. And deep learning-based approach is used to make optimal caching decisions. Numerical results have proved its advantages regarding caching utility compared to other existing approaches.

Yang and Liu [32] proposed to cache those popular contents at RSU in advance, to cater to the dynamics of vehicles and frequent content request changes. Also, they insist that current content-sharing approaches can no longer protect privacy in VEC environments. To tackle this

issue, an asynchronous **Federated Learning (FL)**-based strategy is designed for caching decision-making and privacy-protecting. The simulation proves that their strategy can outperform current benchmark schemes. The cloud infrastructures can no longer satisfy emerging mobile applications with low-latency requirements, and the multi-access edge computing paradigm provides an efficient solution for such applications, by deploying computing resources close to the users. New issues emerge, such as limited computing resources and high fluctuation of workloads. Accordingly, authors in [4] put forward a serverless-based framework, called NEPTUNE, that can fulfill multiple functions such as serverless function placement, resource contention resolution, CPU and GPU RA, and so on. A prototype built upon an extended version of Kubernetes is developed to evaluate the performance. The advantages regarding response latency and network overhead, compared to other baseline solutions, are investigated and proven in the simulation.

For multi-task and multi-scenario task offloading in VEC environments, a unified solution was proposed in [7] to optimize the response latency for all the tasks under multiple task offloading scenarios. Particularly, a Seq2seq-based Meta Reinforcement Learning algorithm is designed. First, an attention-based bidirectional gated recurrent unit is used to make the offloading decisions. Then, a meta reinforcement learning trains the policy offline, which is used for new offloading scenarios. Evaluation is carried out based on the task generator DAGGEN and realistic vehicular traces, and the results show the efficiency of the approach, e.g., the response latency for the tasks is reduced by 11.36% in comparison to a greedy algorithm. Thanks to flexible deployment and high availability, **Unmanned Aerial Vehicles (UAV)** can assist the VEC systems, e.g., by building wireless connections to the RSU and further providing computing services in VEC. Liu et al. [20] proposed to cache both contents and services at the RSU and UAV for response latency reduction and bandwidth resources saving. The hybrid data caching and task offloading problem is formulated, aiming to minimize the whole response latency for the tasks. In particular, a DQN-based solution is put forward to raise the UAV utility. A caching scheme for RSU and UAV is also designed, which enables data replacement of RSU and UAV independently. They experimentally prove that the strategy has a high convergence rate and efficient task completion delay.

It is becoming increasingly popular to apply UAVs to traditional cellular networks, aiming to provide continuous connectivity in various extreme scenarios. Anokye et al. [3] proposed cache contents at UAVs to improve the system performance. Particularly, the optimization problem is modeled as a Markov Decision Problem, which is solved by a dueling reinforcement learning-based algorithm. In VEC, FL can guarantee vehicles' privacy by only sharing local vehicle models. However, the training model may encounter challenges in updating successfully due to potential disruptions in wireless connectivity caused by the high mobility of vehicles. To tackle this issue, Wu et al. [30] proposed a cooperative caching scheme using asynchronous federated deep reinforcement learning.

Li et al. [19] put forward a collaborative task offloading and service caching replacement strategy. Tasks can be offloaded and services can be cached in a collaborative way between adjacent RSUs. They formulate the optimization problem as a mixed integer programming problem and solve it by combining an iterative algorithm with DRL. In this process, privacy issues may emerge, and works such as [10, 14] pay attention to cybersecurity defense and network intrusion detection.

Although content caching approaches can improve users' **Quality of Experience (QoE)**, few works take into account the users' preferences when designing the popularity function to calculate the content popularity. Intuitively, content caching could be more efficient when considering users' preferences, since it could improve the cache hits to a certain extent. Accordingly, a new caching strategy in mobile edge computing is proposed in [35], which combines the content popularity and user preferences in edge computing. Specifically, they try to mine the patterns between different

contents by analyzing the user-content matrix. A convolutional neural network model is also proposed for predicting users' preferences. In comparison to the baseline approaches, their strategy can achieve a higher cache hit ratio and faster response latency.

As **Mobile Augmented Reality (MAR)** applications, characterized by time-sensitive and computation-intensive tasks, are becoming increasingly popular, it is very challenging to handle this computation, even in mobile edge computing environments. Li et al. [17] investigated the joint task offloading and content caching problem for these MAR applications and tasks. For instance, they formulate the problem to maximize the hit ratio and minimize the response latency at the same time. A multi-objective artificial bee colony-based algorithm is put forward to solve this optimization problem. Simulation results reveal that the proposed algorithm can achieve better performance.

Apart from the content-oriented caching in VEC, task/service-oriented caching in VEC has also attracted extensive attention in the past few years [6, 8, 22, 34]. In contrast, task-oriented caching tries to cache task or service-related source code and database library at the edge server in advance, aiming to shorten the response latency for the offloaded tasks in edge computing environments. Task/service caching is a complicated process that consumes not only storage resources but also computing resources, e.g., for instantiating virtual machines and service instances. Chu et al. [6] tried to maximize the QoE for users in edge computing environments, and they consider service caching, RA, and task offloading as the decision variables. Solving the formulated MINLP problem is highly challenging. To address this, the authors propose a two-stage algorithm designed to obtain an approximately optimal solution. The simulation results demonstrate the effectiveness and robustness of the proposed approach.

There exists a situation where tasks with dependency relationships are offloaded to the network edge in VEC. To improve the performance of such VEC systems, Shen et al. [22] investigated this joint problem of task offloading and service caching in VEC. Particularly, vehicles offload their tasks to RSU for execution using the software-defined network, and RSU can cache and reuse the corresponding services for subsequent offloaded tasks. They try to optimize the offloading efficiency, which is defined as a weighted sum of response latency and energy consumption for vehicular tasks. A semi-distributed algorithm is put forward for this problem, and the simulation shows that their approach is better than the comparison baselines. A VEC framework was proposed in [12], which consists of two different modules in terms of functionality and roles played in the framework. For instance, one module focuses on microservice caching in VEC networks, and the cluster-based caching technique is adopted to cache those frequently requested microservices in VEC. The other module, on the other hand, focuses on the task offloading and execution in VEC. For instance, it leverages the computational capabilities of the edge servers and vehicles to achieve appropriate computation spreading between them in a collaborative manner.

It shall be noted that service caching can cause inescapable latency and energy costs. In view of this, Wang and Du [28] studied the joint service caching and task offloading problem, and their goal is to minimize the cost of edge services. A collaborative service caching mechanism is put forward to motivate service caching and sharing, with the help of caching coalitions. The experimental results reveal that the approach outperforms other comparison baselines in terms of response latency and edge network profits.

Different from the aforementioned works, we combine content caching and task offloading in VEC environments, from the viewpoint of input data reuse. The corresponding contents can be moved from the cloud center and cached to the edge server in VEC to shorten the response latency and energy consumption of vehicles.

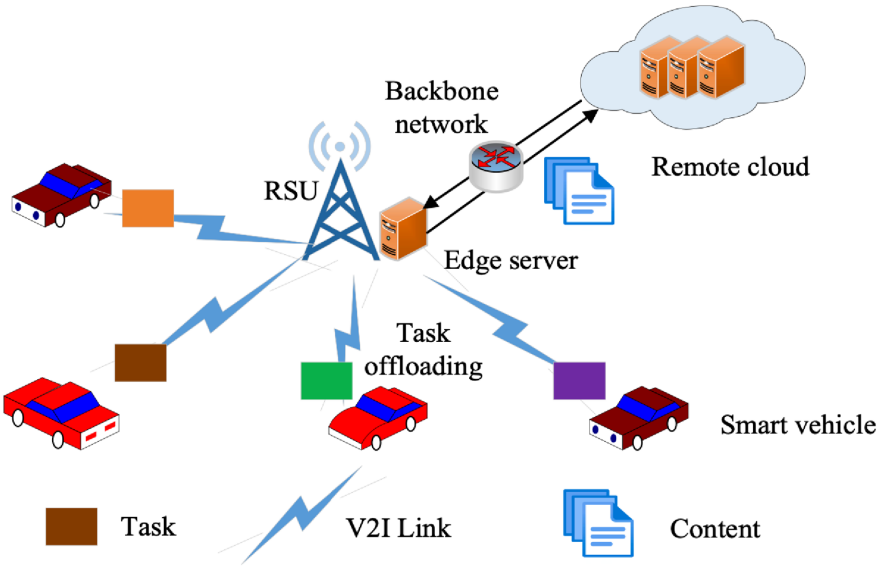


Fig. 1. Application scenario.

3 System Model

We consider a cooperative task offloading and content caching model in VEC, as illustrated in Figure 1. There are three kinds of entities, i.e., one RSU, smart vehicles, and the cloud center, respectively. The RSU, denoted by R, is deployed with an edge server. Denote by $\mathcal{N} = \{1, \cdots, N\}$ the set of vehicles in the model, where N is the number of vehicles. These vehicles can interact with R using **Vehicle-to-Infrastructure (V2I)** communication technologies. We assume that the edge server is empowered with caching capabilities, such that the second part of the task-input data can be cached to reduce the overall response latency of tasks. Denote by $\mathcal{K} = \{1, \cdots, K\}$ the set of cacheable contents and K is the total number of contents. Let c_k $(k \in \mathcal{K})$ represent the size of content k. These contents can be provided by **Content Providers (CP)** and stored in the cloud center. Each vehicle n is assumed to have one task for execution, and the task generated by vehicle n can be expressed as a tuple of four parameters, i.e., $M_n = (\alpha_n, c_{n,k}, s_n, d_n)$, in which α_n is the size of the first part of the task-input data, $c_{n,k}$ is the size of the second part of the task-input data that requires the content k, s_n denotes the amount of computation required for accomplishing the task, and d_n is the deadline for the task M_n. The accomplishment of task M_n needs to obtain the part $c_{n,k}$ from CP in normal circumstances. Moreover, we have $c_{i,k} = c_{j,k} = c_k$ $(i \neq j, \forall i, j \in \mathcal{N})$. In other words, different vehicles requesting the same content have the same size as the second part of the task-input data.

For easy presentation, we refer to the edge server and RSU R for the same meaning and use them interchangeably hereafter. Also, some key notations are summarized in Table 1. Assume that the tasks are indivisible and can be processed locally at the vehicles or offloaded to the edge server. We define a binary variable ζ_n to denote whether the task M_n is offloaded or not. If the task is offloaded to the edge for execution, $\zeta_n = 1$, and 0, otherwise. In addition, no matter where the task is processed, the second part of the task-input data should be obtained from the cloud as mentioned earlier. However, this way will consume a vast mass of network resources, increase the response latency of tasks, and degrade the QoE of vehicles. To avoid duplicated data transmission and mitigate network congestion, we can cache these contents at the edge. In particular, we define

Table 1. Notations

Notation	Description
R	RSU
$\mathcal{N}$	The set of vehicles
N	The number of vehicles
$\mathcal{C}$	The cacheable content set
c_k	The size of the content k
K	The number of contents
M_n	The task generated by vehicle n
α_n	The size of the first part of input data for vehicle n
$c_{n,k}$	The size of the second part of input data
s_n	The computation required for accomplishing the task
d_n	The deadline for the task M_n
C	The total caching capability of the edge server
W_{dl}	The downlink channel bandwidth between n and R
W_{up}	The uplink channel bandwidth between n and R
P	The transmission power of R
P_n	The transmission power of vehicle n
G_n	The channel gain between n and RSU R
δ^2	The noise power
η	The propagation rate from the cloud center to R
κ	The effective switched capacitance coefficient
ι	The number of cycles for per-bit computation
f_n	The local computing capability of vehicle n
$f_{R,n}$	The computing resources allocated to M_n by R
β_n^t	The vehicle n's preference towards the response latency
β_n^e	The vehicle n's preference towards energy consumption
χ_k	Denote whether the content k is cached at R
ζ_n	Denote whether the task M_n is offloaded to R

$\chi_k (\in \{0, 1\})$ as a binary variable to represent whether the content k is cached at the edge server. If k is cached at the edge server, $\chi_k = 1$. Consider that the caching capability of the edge server is limited, and we have the following constraint condition:

$$\sum_{k \in \mathcal{K}} \chi_k c_k \leq C, \tag{1}$$

where C is the total caching capability of the edge server.

3.1 Local Computing Model

Let f_n denote the local computing capability of vehicle n. If n chooses to perform the task M_n locally, it needs to obtain the content k from the outside. If the content is cached at the edge server, the transmission rate from R to n can be given as:

$$r_{n,k}^{dl} = W_{dl} \log \left(1 + \frac{PG_n}{\delta^2}\right), \tag{2}$$

where W_{dl}, P, and G_n are the downlink channel bandwidth between n and R, transmission power of R, and channel gain between n and RSU R, respectively. δ^2 denotes the noise power. Note that

we adopt the orthogonal spectrum for data transmission between R and n to avoid co-channel interference. The transmission delay for the content k from R to n can be calculated as:

$$T_{n,k}^{dl,c} = \frac{c_{n,k}}{r_{n,k}^{dl}}. \tag{3}$$

If the content is not cached at the edge server, R needs to obtain this content from the cloud and then transmit it to n. Accordingly, the transmission delay for the content k from the cloud center to n can be calculated as:

$$T_{n,k}^{dl,uc} = \frac{c_{n,k}}{r_{n,k}^{dl}} + \frac{c_{n,k}}{\eta}, \tag{4}$$

where η is the estimated propagation rate from the cloud center to R via the backbone network based on statistical data and empirical knowledge. Combining the variable χ_k, the generalized transmission delay taken to obtain the content k from the outside can be expressed as:

$$T_{n,k}^{dl} = \chi_k T_{n,k}^{dl,c} + (1 - \chi_k) T_{n,k}^{dl,uc}. \tag{5}$$

When the task-input data are complete, the vehicle n can process the task immediately. The time taken to accomplish the task can be given as:

$$T_{n,k}^{exe} = \frac{s_n}{f_n}. \tag{6}$$

The response latency for local computing can be given as:

$$T_n^{loc} = T_{n,k}^{exe} + T_{n,k}^{dl}. \tag{7}$$

The energy consumption for calculating the task M_n at vehicle n can be given as:

$$E_n^{loc} = \kappa \iota (f_n)^2 s_n, \tag{8}$$

where κ and ι are the effective switched capacitance coefficient, and the number of cycles required for the computation per bit.

3.2 Task Offloading Model

If the vehicle n offloads task M_n to the edge server, the response delay comprises four parts: (1) the time taken to transmit α_n to R, (2) the time taken to obtain $c_{n,k}$, (3) the time taken to execute the task at R, and (4) the time taken to deliver the output back to vehicle n from R. Considering the fact that the output size is much smaller than the input and the downlink rate for data transmission is also much larger than the uplink rate, we in this article ignore the fourth part of the response delay [11]. Next, we introduce how to calculate the first three parts of the response delay.

The data rate for task offloading from n to R can be calculated as:

$$r_n^{off} = W_{up} \log \left(1 + \frac{P_n G_n}{\delta^2}\right), \tag{9}$$

where W_{up}, P_n, G_n, and δ^2 are the uplink channel bandwidth between n and R, transmission power of vehicle n, the channel gain between n and RSU R, and the noise power, respectively. Thus, the time taken to transmit the task-input data to R can be expressed as α_n / r_n^{off}.

If the content k is cached at the edge server, the task execution starts upon the arrival of the input data (i.e., α_n). The offloading time in this case can be expressed as:

$$T_n^{off,c} = \frac{\alpha_n}{r_n^{off}}. \tag{10}$$

On the other hand, if the content k is not cached at the edge server, R needs to obtain this content from the cloud. The propagation delay taken to obtain k from the cloud center can be calculated as:

$$T_{n,k}^{pro,uc} = \frac{c_k}{\eta}. \tag{11}$$

The task execution cannot be started until both parts of the input data arrive at the edge server. In this case, the offloading time can be given as:

$$T_n^{off,uc} = \max\left\{\frac{\alpha_n}{r_n^{off}}, T_{n,k}^{pro,uc}\right\}. \tag{12}$$

Combining the variable χ_k, the generalized transmission delay for transmitting the task-input data can be expressed as:

$$T_{n,k}^{off} = \chi_k T_n^{off,c} + (1-\chi_k) T_n^{off,uc}. \tag{13}$$

Therefore, the energy consumption for vehicle n to offload the task M_n to the edge can be expressed as:

$$E_n^{off} = P_n \frac{\alpha_n}{r_n^{off}}. \tag{14}$$

Let $f_{R,n}$ denote the amount of computing resources allocated to task M_n by R. The calculation delay is given as follows:

$$T_n^{exe} = \frac{s_n}{f_{R,n}}. \tag{15}$$

Accordingly, the response latency for the offloaded task M_n is:

$$T_n^{edge} = T_{n,k}^{off} + T_n^{exe}. \tag{16}$$

Combining the variable ζ_n for vehicle n, the response latency for the task M_n can be generalized as:

$$T_n = (1-\zeta_n) T_n^{loc} + \zeta_n T_n^{edge}. \tag{17}$$

In the meanwhile, the energy consumption for vehicle n can be generalized as:

$$E_n = (1-\zeta_n) E_n^{loc} + \zeta_n E_n^{off}. \tag{18}$$

3.3 Vehicular Task Utility

Various metrics can be adopted to evaluate the performance of the VEC systems from different angles. For instance, from the perspective of vehicles, the response latency of the tasks is much more important than other evaluation metrics such as energy consumption, especially when the driver in the vehicle seeks the ultra-low-latency QoE for some computationally intensive tasks. The reduction of response latency for vehicular tasks is also the original intention of VEC. On the other hand, energy consumption as another evaluation metric, especially for electric vehicles, has also aroused extensive attention in the past few years. Accordingly, in this article, we define the task utility for M_n, which incorporates the two factors as follows:

$$J_n = \beta_n^t T_n + \beta_n^e E_n, \tag{19}$$

where $\beta_n^t, \beta_n^e \in [0,1]$, and $\beta_n^t + \beta_n^e = 1, \forall n \in \mathcal{N}$. The two parameters denote vehicle n's preferences towards the response latency and energy consumption, respectively. For instance, if vehicle n pursues the ultra-low-latency QoE, n can increases β_n^t and decreases β_n^e. If n cares about the energy consumption more than anything, it can increase β_n^e and decrease β_n^t, with the aim of reducing the

energy consumption at the expense of longer response latency. In an extreme case where n only cares about the response latency, then, $\beta_n^t = 1$ and $\beta_n^e = 0$.

4 Problem Formulation

The objective of this article is to minimize the task utility for all the tasks in the optimization period, by jointly optimizing task offloading, content caching, and RA in VEC. Before going further, we define three kinds of decision policies, i.e., the offloading policy ζ, the caching policy χ, and the computing RA policy $\mathcal{F}$, respectively. In addition, denote by $\mathcal{N}_o$ the set of vehicles offloading their tasks to the edge, i.e., $\mathcal{N}_o = \{n|\zeta_n = 1, \forall n \in \mathcal{N}\}$. Then, the three policies can be given as $\zeta = \{\zeta_n|\forall n \in \mathcal{N}\}$, $\chi = \{\chi_k|k \in \mathcal{K}\}$, and $\mathcal{F} = \{f_{R,n}|\forall n \in \mathcal{N}_o\}$. Note that if the vehicle n does not offload its task to the edge, the edge server does not allocate any computing resources to it, i.e., $f_{R,n} = 0, \forall n \notin \mathcal{N}_o$. Hence, the utility of the VEC system in this article can be defined as:

$$J(\zeta, \chi, \mathcal{F}) = \sum_{n \in \mathcal{N}} J_n. \tag{20}$$

Our JTOCA optimization problem in VEC can be formulated as:

$$\begin{aligned}
\mathcal{P}: \ & \min_{\zeta, \chi, \mathcal{F}} \quad J(\zeta, \chi, \mathcal{F}) \\
\text{s.t.} \ & f_n \le f_n^{max}, \quad \forall n \in \mathcal{N}, && (21)\\
& T_n \le d_n, \quad \forall n \in \mathcal{N}, && (22)\\
& \sum_{n \in \mathcal{N}} f_{R,n} \le f_{R,max}, && (23)\\
& \sum_{k \in \mathcal{K}} \chi_k c_k \le C, && (24)\\
& \zeta_n, \chi_k \in \{0, 1\}, \quad \forall n \in \mathcal{N}, \ \forall k \in \mathcal{K}, && (25)\\
& f_{R,n} \ge 0, \quad \forall n \in \mathcal{N}, && (26)
\end{aligned}$$

where the inequality (21) ensures that the local computing resources allocated to the task M_n should not exceed the maximal computing capability of n, when the task is executed locally. In the meanwhile, the task M_n should be accomplished before its deadline, which is specified by the inequality (22). In addition, the computing resources allocated to the offloaded tasks should not exceed the maximal computing capability of the edge server, which is guaranteed by the inequality (23). The inequality (24) denotes that the size of cached contents should not exceed the caching capability of the edge server.

Remark: Our optimization problem JTOCA is an MINLP problem, which usually takes exponential time to find the optimal solution. Such high time complexity is practically prohibitive in a large-scale VEC network. Considering the large number of vehicular tasks and cacheable contents, we aim to design a suboptimal solution with low time complexity that can be applied to solving JTOCA practically in this article.

4.1 Problem Decomposition

By analyzing the dependency relationships of the decision policies in the objective function and constraints of JTOCA, we have following two observations. First, the second term on the **Right-Hand Side (RHS)** of Equation (19) only depends on the offloading decision ζ_n of task M_n, i.e., the energy consumption of vehicle n for task M_n is independent of the content caching and RA decisions made by the edge server. Second, the three decision policies are tightly coupled in the objective function as well as the constraints. To decouple the decision variables ζ, χ, and $\mathcal{F}$ from

each other, we need to decompose the original optimization problem $\mathcal{P}$. Note that temporarily fixing the offloading and caching variables $\{\zeta_n|\forall n \in \mathcal{N}\}$ and $\{\chi_k|k \in \mathcal{K}\}$, the JTOCA problem can be rewritten as:

$$\begin{aligned} \mathcal{P}1: \ & \min_{\zeta,\chi}(\min_{\mathcal{F}} \ J(\zeta,\chi,\mathcal{F})) \\ \text{s.t.} \ & (21)-(26) \end{aligned} \tag{27}$$

where the constraints (22), (23), and (26) on the RA at the edge are decoupled from the constraints (Equations (21), (24), and (25)) on the task offloading and content caching. Hence, solving the problem $\mathcal{P}1$ is equivalent to solving the following JOC problem.

$$\begin{aligned} \mathcal{P}2: \ & \min_{\zeta,\chi} \ J^*(\zeta,\chi) \\ \text{s.t.} \ & (21),(24),(25) \end{aligned} \tag{28}$$

where $J^*(\zeta,\chi)$ is the function with optimal value regarding the RA problem, given below,

$$\begin{aligned} & J^*(\zeta,\chi) = \min_{\mathcal{F}} \ J^*(\zeta,\chi,\mathcal{F}) \\ \text{s.t.} \ & (22),(23),(26) \end{aligned} \tag{29}$$

Thus, we can transform the original problem $\mathcal{P}$ to problems $\mathcal{P}1$ and $\mathcal{P}2$, and such a transformation does not affect the optimality of the solution [26]. Accordingly, we can obtain the solution to $\mathcal{P}$ by solving the problems $\mathcal{P}1$ and $\mathcal{P}2$, respectively. Namely, we can solve the JTOCA problem by solving the JOC and RA problems.

5 Algorithm Design for JTOCA Problem

In this section, we try to solve the JTOCA problem by designing an approach of low complexity. Specifically, we solve the decomposed JOC and RA problems, respectively, to obtain the solution to the JTOCA problem.

Given feasible task offloading decision ζ and content caching decision χ that both satisfy their own constraints, we can expand the expression of the objective function (Equation (20)) as:

$$J(\zeta,\chi,\mathcal{F}) = \sum_{n\in\mathcal{N}} \beta_n^t[T_n^{loc} + \zeta_n(T_{n,k}^{off} - T_n^{loc})] + \beta_n^e E_n + M(\zeta,\chi,\mathcal{F}), \tag{30}$$

where $M(\zeta,\chi,\mathcal{F}) = \sum_{n\in\mathcal{N}_o} \beta_n^t \frac{s_n}{f_{R,n}}$. We observe that the first term on the RHS of Equation (30) is totally independent of the computing RA decision $\mathcal{F}$, while the second term on the RHS of Equation (30) (i.e., $M(\zeta,\chi,\mathcal{F})$) can be regarded as the function of the RA decision $\mathcal{F}$, given the task offloading decision and content caching decision.

5.1 Computing RA

Given feasible offloading decision and content caching decision, the subproblem RA (i.e., $J^*(\zeta,\chi)$) can be equivalently transformed into the optimization of $M(\zeta,\chi,\mathcal{F})$ based on the above analysis, i.e.,

$$\begin{aligned} RA: \ & \min_{\mathcal{F}} \ M(\zeta,\chi,\mathcal{F}) \\ \text{s.t.} \ & (22),(23),(26) \end{aligned} \tag{31}$$

Lemma 1. *The optimization problem RA is convex.*

Proof. Please refer to Appendix A. □

We can obtain the optimal RA scheme by solving the problem RA via **Karush–Kuhn–Tucker (KKT)** conditions. We construct the Lagrangian function for the problem RA $L(\mathcal{F}, v, \mu)$ as follows:

$$\begin{aligned} L(\mathcal{F}, v, \mu) &= \sum_{n\in\mathcal{N}_o} \beta_n^t \frac{s_n}{f_{R,n}} + \sum_{n\in\mathcal{N}} v_n(T_n - d_n) + \mu\left(\sum_{n\in\mathcal{N}} f_{R,n} - f_{R,max}\right) \\ &= \sum_{n\in\mathcal{N}_o} \beta_n^t \frac{s_n}{f_{R,n}} + \sum_{n\in\mathcal{N}_o} v_n(T_n - d_n) + \sum_{n\in\mathcal{N}/\mathcal{N}_o} v_n(T_n - d_n) + \mu\left(\sum_{n\in\mathcal{N}_o} f_{R,n} - f_{R,max}\right), \end{aligned}$$

where $v = \{v_n | v_n \geq 0, \forall n \in \mathcal{N}\}$ and $\mu(\geq 0)$ are the Lagrangian multipliers. Take the first partial derivative of $L(\mathcal{F}, v, \mu)$ regarding $\mathcal{F}$ and let it equal zero, i.e.,

$$\frac{\partial L(\mathcal{F}, v, \mu)}{\partial \mathcal{F}} = 0,$$

and we can get

$$\frac{\partial L(\mathcal{F}, v, \mu)}{\partial f_{R,n}} = \mu^* - \frac{\beta_n^t s_n + v_n^* s_n}{f_{R,n}^2} = 0, \ \forall n \in \mathcal{N}_o.$$

Hence, the optimal RA strategy $f_{R,n}^*$ can be calculated as:

$$f_{R,n}^* = \sqrt{\frac{\beta_n^t s_n + v_n^* s_n}{\mu^*}}, \quad \forall n \in \mathcal{N}_o. \tag{32}$$

The Lagrangian multiplier μ^* satisfies,

$$\sum_{n\in\mathcal{N}_o} f_{R,n}^* = f_{R,max}, \tag{33}$$

which is guaranteed by the following lemma, i.e.,

Lemma 2. *The problem RA can be optimized if and only if the computing resources at the edge server are fully allocated.*

Proof. Assume that the problem RA can be optimized when the computing resources at the edge server are not fully allocated. That is, $\sum_{n\in\mathcal{N}_o} f_{R,n}^* < f_{R,max}$ holds. Then, we allocate the remainder of the computing resources (i.e., $f_{R,max} - \sum_{n\in\mathcal{N}_o} f_{R,n}^*$) to an arbitrary task M_n ($n \in \mathcal{N}_o$). Given the objective function $M(\zeta, \chi, \mathcal{F}) = \sum_{n\in\mathcal{N}_o} \beta_n^t \frac{s_n}{f_{R,n}}$, we find that the optimal value of $M(\zeta, \chi, \mathcal{F})$ can be further decreased since the task M_n is allocated with more computing resources. That contradicts the assumption that the RA problem can be optimized when the computing resources at the edge server are not fully allocated. Accordingly, the proof is completed. □

By substituting Equation (32) into Equation (33), we can obtain μ^* as follows:

$$\mu^* = \left(\frac{\sum_{n\in\mathcal{N}_o} \sqrt{\beta_n^t s_n + v_n^* s_n}}{f_{R,max}}\right)^2. \tag{34}$$

In addition, $v^* = \{v_n^* | \forall n \in \mathcal{N}\}$ satisfies $v_n^*(T_n - d_n) = 0$, and $T_n \leq d_n$, $\forall n \in \mathcal{N}$, according to the KKT conditions. So, the value of $f_{R,n}^*$ depends on the following two cases.

Case 1: The optimal value for the computing RA $f_{R,n}^*$, $\forall n \in \mathcal{N}$, strictly satisfies the inequation $T_n < d_n$, i.e., $f_{R,n}^*$ is located in the interior of feasible zone of the inequation $T_n \leq d_n$, and we have

$v_n^* = 0$. Thus, combining Equations (32) and (34), we can obtain $f_{R,n}^*$ as:

$$f_{R,n}^* = \frac{f_{R,max}\sqrt{\beta_n^t s_n}}{\sum_{m\in\mathcal{N}_o}\sqrt{\beta_m^t s_m}}, \quad \forall n \in \mathcal{N}_o. \tag{35}$$

Case 2: The optimal value for the computing RA $f_{R,n}^*, \forall n \in \mathcal{N}$, satisfies the equation $T_n = d_n$, i.e., $f_{R,n}^*$ is located in the border of feasible zone of the inequality $T_n \le d_n$. In this case, the following equation holds,

$$T_{n,k}^{off} + \frac{s_n}{f_{R,n}^*} = d_n, \quad \forall n \in \mathcal{N}_o, \tag{36}$$

and we can obtain the optimal value $f_{R,n}^*$ directly by solving Equation (36),

$$f_{R,n}^* = \frac{s_n}{d_n - T_{n,k}^{off}}, \quad \forall n \in \mathcal{N}_o. \tag{37}$$

Hence, the optimal computing RA policy $\mathcal{F}^*$ can be expressed as:

$$f_{R,n}^* = \begin{cases} \max\left\{\frac{f_{R,max}\sqrt{\beta_n^t s_n}}{\sum_{m\in\mathcal{N}_o}\sqrt{\beta_m^t s_m}}, \frac{s_n}{d_n - T_{n,k}^{off}}\right\}, & n \in \mathcal{N}_o, \\ 0, & n \in \mathcal{N}/\mathcal{N}_o. \end{cases} \tag{38}$$

5.2 JOC Problem

After solving the problem RA, we pay attention to the JOC problem. This problem jointly determines the task offloading and content caching policies as shown in the problem $\mathcal{P}2$, given the computing RA policy $\mathcal{F}$. As presented earlier, we can obtain the optimal RA policy $\mathcal{F}^*$, given the task offloading and content caching decisions ζ and χ, respectively. That is,

$$J(\zeta,\chi) = \sum_{n\in\mathcal{N}} \beta_n^t [T_n^{loc} + \zeta_n (T_{n,k}^{off} - T_n^{loc})] + \beta_n^e E_n + M(\zeta,\chi,\mathcal{F}^*), \tag{39}$$

where $M(\zeta,\chi,\mathcal{F}^*) = \sum_{n\in\mathcal{N}_o} \beta_n^t \frac{s_n}{f_{R,n}^*}$. Thus, the JOC problem in $\mathcal{P}2$, subject to the constraints (Equations (21), (24), and (25)), can be rewritten as:

$$\min_{\zeta,\chi} J(\zeta,\chi). \tag{40}$$

As observed from this optimization problem, both the offloading and caching decisions are binary. An exhaustive search approach is the most straightforward method to obtain the optimal solution; however, it requires exponential time, specifically $O(2^{N+K})$, over the potential solution space. This is computationally prohibitive, particularly in large-scale VEC networks. Therefore, a heuristic algorithm is necessary to find a suboptimal solution to problem (Equation (40)) within polynomial time.

We notice that the two decisions are tightly coupled with each other. The caching decision is usually made based on the offloading requests, and in turn, the offloading decision is also affected by the cached contents at the edge server. To decouple them from each other, we propose a two-stage decision-making process in this article, with one stage determining the content caching policy and the other stage responsible for determining the offloading decision based on the content caching policy.

Lemma 3. *Taking no account of task offloading and computing RA, the optimization problem in* (Equation (40)) *w.r.t. caching decision χ is NP-hard.*

Proof. The proof is simple and thus omitted here. Please refer to the works [25, 37] for details. □

Accordingly, in stage one, it is necessary to design a heuristic algorithm to address the content caching in polynomial time. As is well known, the key to significantly improving the cache hit ratio is to raise the frequency of cached contents that are requested. Accordingly, it is always beneficial to cache those most frequently requested contents in the VEC system. On the other hand, caching those contents with larger sizes at the edge server can substantially reduce the network burden and response delay which also accords with our optimization objective in this article. Combining the two factors, we define the content popularity of the content k as follows:

$$\Gamma(k) = w_1 \frac{R_k - \min_{m\in\mathcal{K}} R_m}{\max_{m\in\mathcal{K}} R_m - \min_{m\in\mathcal{K}} R_m} + w_2 \frac{c_k - \min_{m\in\mathcal{K}} c_m}{\max_{m\in\mathcal{K}} c_m - \min_{m\in\mathcal{K}} c_m}, \tag{41}$$

where $R_k = \sum_{n\in N} \mathbb{I}\{m == k | c_{n,m} \in M_n\}$, indicating the number of tasks requesting the content k. $\mathbb{I}\{\cdot\}$ is the indicator function. $\mathbb{I}\{x\} = 1$, if x is true, and $\mathbb{I}\{x\} = 0$, otherwise. $c_{n,m} \in M_n$ means the task M_n requests the content m with the content size $c_{n,m}$. w_1 and w_2 are the weights to indicate the tradeoff between the two factors and $w_1 + w_2 = 1$. For instance, if the distribution of requests on the contents is more uneven than the distribution of content sizes, we can increase w_1 properly to represent that the cache hit ratio should be paid more attention. On the other hand, we can increase w_2 to underline the importance of content sizes if the distribution of content sizes is more uneven than the distribution of requests on the contents. We then adopt a heuristic greedy approach to determine the content caching decision χ, which can be outlined as follows. First, RSU acquires the task information from the beacon information exchanged with vehicles, when they communicate with each other via V2I technologies. We assume that RSU can also acquire the information on the cacheable contents $\mathcal{K}$. Then, RSU calculates the content popularity $\Gamma(k)$ for each content k and sorts the contents in descending order of the content popularity. Given the caching capability C, our heuristic greedy rule is that RSU always chooses to cache the content with the largest value of content popularity so far. The procedure repeats until no more contents can be cached. Specifically, the details are shown in Algorithm 1. A maximum heap H is used to store the sorted contents. The algorithm repeatedly retrieves the content from H to check whether the current content k can be cached (lines 8–12). If the program loop (line 8) ceases, it means that the current content k cannot be cached, because of the caching memory violation. Considering the possible residual memory capacity at the edge, we should continue to check whether other contents with smaller sizes can be cached (lines 15–21). Obviously, the time complexity of this algorithm is $O(K \log K)$, incurred by the sorting algorithm (line 6).

After determining the content caching decision, we can solve the task offloading issue in stage two. Similar to the caching decision, the task offloading decision variable ζ is also binary. In VEC, the high mobility of vehicles and the deadlines of vehicular tasks require that the time spent on the offloading decision-making should be almost real time. Unfortunately, even given caching decision χ^* and computing RA $\mathcal{F}^*$, we still require the time complexity of $O(2^N)$ (e.g., adopting the exhaustive searching) to optimally solve the combinatorial optimization problem (Equation (40)). Such an approach with high time complexity brings prohibitive costs in large-scale VEC networks.

To tackle the above issue, we put forward a low-complexity heuristic algorithm to solve the task offloading problem. Before proceeding further, we need to discuss the following two cases when determining whether the task should be offloaded.

Case 1: The vehicle n cannot totally undertake the computation of its task M_n. This case occasionally takes place when the computing capability of n is not sufficient. This case will give rise to the violation of the deadline. In this case, it is no doubt that the task M_n will be offloaded to the edge, since the offloading decision is binary.

Algorithm 1: Heuristic Greedy Approach for Content Caching

Input: Required parameters
Output: Content caching decision χ

Initialize a maximum heap H;
Initialize $\chi : \{\chi_k = 0\}_{k=1}^{K}$, w_1 and w_2;
Acquire $\{M_n | n \in \mathcal{N}\}, \mathcal{K}$;
Calculate R_k for each content $k \in \mathcal{K}$;
Calculate $\Gamma(k)$ for each $k \in \mathcal{K}$ based on Eq. (41);
Sort the contents in descending order of $\Gamma(k), \forall k \in \mathcal{K}$;
Store sorted contents using H;
while $C \geq 0$ **do**
    $k = H.pop()$;
    $\chi_k = 1$;
    $C- = c_k$;
end
$\chi_k = 0$;
$C+ = c_k$;
while H is not empty **do**
    $k = H.pop()$;
    if $C - c_k \geq 0$ **then**
        $\chi_k = 1$;
        $C- = c_k$;
    end
end
Return χ;

Case 2: The vehicle n can undertake the computation since it has sufficient computing resources and can thus satisfy the rigorous latency requirement of n. In this case, whether M_n is offloaded or not depends on which way can bring the smaller value of the objective function.

Generally, the heuristic algorithm is depicted in Algorithm 2. We first determine those tasks that fall into the category of Case 1 (lines 1–2). Denote by $\mathcal{S}_o$ the set of these tasks. As shown in the algorithm, $\mathcal{S}_o$ can be expressed as $\mathcal{S}_o = \mathcal{N}/\mathcal{S}$, where $\mathcal{S}$ denotes the set of tasks falling into the category of Case 2 described above. Then, the offloading decision ζ can be initialized based on $\mathcal{S}_o$. That is to say, $\zeta = \{\zeta_n = 1 | \forall M_n \in \mathcal{S}_o\} \cup \{\zeta_n = 0 | \forall M_n \in \mathcal{S}\}$. Denote by OPT the optimal value of the objective function w.r.t. (Equation (40)) we have found so far. We then repeat the following procedure. The heuristic rule is that in each iteration, we find the task M_n from $\mathcal{S}$ that can bring about the minimal value of the objective function, in comparison with other tasks from $\mathcal{S}$, if the task M_n is offloaded for execution. If the task is found, then it is removed from $\mathcal{S}$ and added to $\mathcal{S}_o$. Meanwhile, the task offloading decision ζ and the optimal value OPT are updated, respectively. The procedure repeats until the variable OPT does not decrease.

Inspired by the works [26], we can further improve the performance of the algorithm by the *exchange* operation (lines 12–20). Specifically, suppose that two tasks M_k and M_l come from $\mathcal{S}_o$ and $\mathcal{S}$, respectively. If we swap the offloading decisions of M_k and M_l (i.e., $\zeta_k : 1 \rightarrow 0$ and $\zeta_l : 0 \rightarrow 1$), the value of the objective function decreases, namely, $J(\zeta|_{\zeta_k=1,\zeta_l=0}, \chi^*, \mathcal{F}^*) > J(\zeta|_{\zeta_k=0,\zeta_l=1}, \chi^*, \mathcal{F}^*)$. In

Algorithm 2: Heuristic Task Offloading Algorithm for VEC

Input: Required parameters
Output: Task Offloading Decision ζ

Calculate T_n^{loc} via Eq. (7), $\forall n \in \mathcal{N}$;
Determine task sets $\mathcal{S} = \{M_n | T_n^{loc} \leq d_n, \forall n \in \mathcal{N}\}$ and $\mathcal{S}_o = \mathcal{N}/\mathcal{S}$;
Initialize ζ based on $\mathcal{S}_o$;
Initialize OPT;
// Record the optimal value found so far w.r.t. (40)
repeat
    Find $M_n \in \mathcal{S}$ such that $\zeta_n = \arg\min J(\zeta, \chi^*, \mathcal{F}^*)$;
    $\zeta_n = 1$;
    Update OPT;
    $\mathcal{S} \leftarrow \mathcal{S}/M_n$;
    $\mathcal{S}_o \leftarrow \mathcal{S}_o \cup M_n$;
until OPT not decrease or $\mathcal{S}$ is empty;
for each task $M_l \in \mathcal{S}$ **do**
    Find the index pair (k^*, l^*) by solving: $(k^*, l^*) = \arg\max\{\Delta J_{k,l}(\zeta) | M_k \in \mathcal{S}_o, M_l \in \mathcal{S}\}$;
    if $\Delta J_{k^*,l^*}(\zeta) > 0$ **then**
        // Conduct exchange operation
        $\zeta_k = 0$ and $\zeta_l = 1$;
        $\mathcal{S}_o \leftarrow \mathcal{S}_o/M_k \cup M_l$;
        $\mathcal{S} \leftarrow \mathcal{S}/M_l \cup M_k$;
        Update OPT;
    end
end
return ζ;

this case, the algorithm will conduct the *exchange* operation by swapping their offloading decisions to further optimize the objective function, on the condition that the constraints are satisfied. In particular, we define $\Delta J_{k,l}(\zeta) = J(\zeta|_{\zeta_k=1,\zeta_l=0}, \chi^*, \mathcal{F}^*) - J(\zeta|_{\zeta_k=0,\zeta_l=1}, \chi^*, \mathcal{F}^*)$ as the optimality gap by swapping the offloading decisions of M_k and M_l. conducting the *exchange* operation. The algorithm searches the index pair (k, l) that can maximize the optimality gap $\Delta J_{k,l}(\zeta)$. Then, the *exchange* operation is conducted on (k, l) as follows. The task M_k is removed from $\mathcal{S}_o$ and added to $\mathcal{S}$, and the task M_l is removed from $\mathcal{S}$ and added to $\mathcal{S}_o$. Meanwhile, the algorithm updates the offloading decisions by setting $\zeta_k = 0$ and $\zeta_l = 1$, respectively.

Remark: The time complexity of this heuristic algorithm can be analyzed as follows. Obviously, the *Repeat* loop and *For* loop can incur plenty of time expenditure. Considering the worst case where the initial $\mathcal{S}_o$ is empty, it is necessary to traverse N tasks in $\mathcal{S}$ to find the optimal M_n that satisfies $\zeta_n = \arg\min J(\zeta, \chi^*, \mathcal{F}^*)$ at the beginning. The task is then removed from $\mathcal{S}$, and the procedure repeats. In the worst case for the number of *Repeat* loops, it is beneficial to offload each task in $\mathcal{S}$. Thus, the procedure repeats until the set $\mathcal{S}$ is empty. Accordingly, the total time is $N + (N-1) + \cdots + 1 = \sum_i^N i = O(N^2)$. On the other hand, the *For* loop aims to find replaceable tasks in $\mathcal{S}_o$ to enhance the performance of the algorithm. Given a task M_l, it is necessary to traverse all the tasks in $\mathcal{S}_o$ to find the optimal index pair satisfying $(k^*, l^*) = \arg\max\{\Delta J_{k,l}(\zeta) | M_k \in \mathcal{S}_o, M_l \in \mathcal{S}\}$, which takes the time complexity of $|\mathcal{S}| \cdot |\mathcal{S}_o|$. Since $|\mathcal{S}| \cdot |\mathcal{S}_o| \leq (\frac{|\mathcal{S}|+|\mathcal{S}_o|}{2})^2 = O(N^2)$, the time

Table 2. Parameter Settings

Parameter	Value	Parameter	Value
N	[20,100]	c_k	[1,500]KB
K	[15,100]	α_n	[10,60]KB
s_n	[40,70]	C	1,500KB
η	30 KB/s	$\kappa\iota$	1e-7
f_n	[1,000, 2,000]	$f_{R,max}$	15,000
β_n^t	0.7	β_n^e	0.3
w_1	0.8	w_2	0.2
Pop_{size}	100	$Step_{max}$	50
Crossover probability	0.2	Mutation probability	0.02

complexity of this part is also of $O(N^2)$. As a result, the algorithm takes the time complexity of $O(N^2)$ to make the offloading decisions.

6 Performance Evaluation

In this section, we conduct extensive simulations to evaluate the proposed JTOCA optimization strategy. The simulation consists of two processes. The effects of involved parameters are investigated in the first process, and the performance evaluation in comparison to other approaches is conducted in the second process. All the simulation is carried out on a desktop computer with a 2.5 GHz Intel(R) Core(TM) i5-13400F CPU, 16 GB of RAM, Microsoft Windows 11 Operating System, and Python 3.7. Some key parameters involved in the simulation are outlined in Table 2. For instance, the number of cacheable contents ranges between 15 and 100, and the number of vehicles ranges between 50 and 100. Each vehicular task M_n is generated randomly, and the deadlines for task accomplishment are appropriately set to guarantee that there always exist some tasks that must be offloaded to avoid constraint violation. In addition, we evaluate the performance of our strategy against the following approaches.

— ***Genetic Algorithm (GA)***: The JOC problem includes two discrete variables ζ and χ, and GA is well positioned for solving such problems, as in [18]. In the simulation, we concatenate the two variables to form the chromosomes and further generate the population with 100 individuals.
— *Request Maximizing Caching and Offloading*: This approach adopts a greedy rule to make caching decisions and then adopts the proposed Algorithm 2 for task offloading. The greedy rule is that the content that is the most frequently requested by vehicular tasks is always to be cached first [29]. In the simulation, we denote this approach by "Request_Max."
— *Content Maximizing Caching and Offloading*: This approach also adopts a greedy rule to make caching decisions and then adopts the proposed Algorithm 2 for task offloading. Different from "Request_Max," the adopted greedy rule is that the content with the largest data size is always to be cached first. The principle behind it is that caching such contents can drastically reduce the transmission delay over the backhaul network as well as the front-haul network. We denote this approach by "Content_Max" in the simulation.
— *Hybrid Approach*: The approach combines GA with the proposed algorithm for task offloading. Particularly, it makes the content caching decision based on GA and adopts Algorithm 2 to make the task offloading decision. Compared to the first baseline that concatenates caching

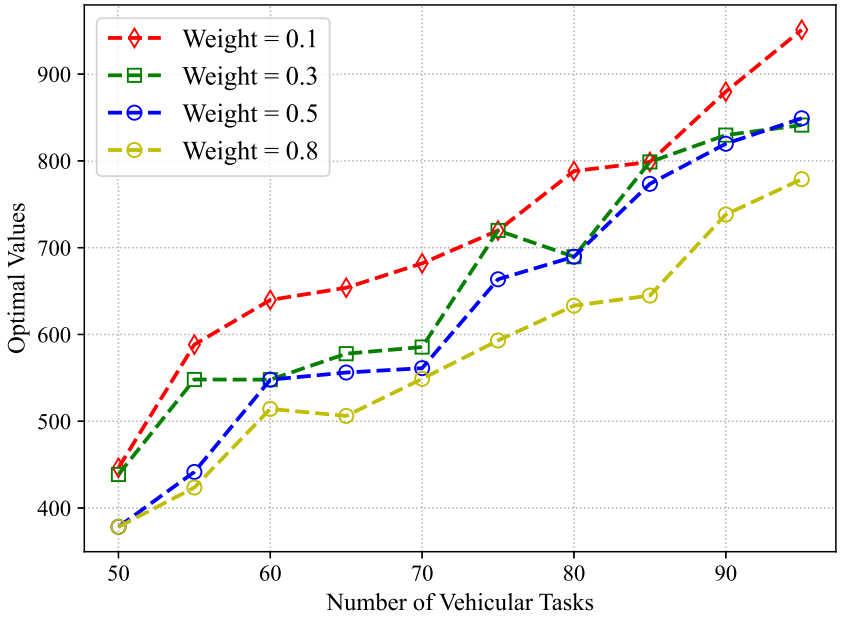


Fig. 2. The impact of w_1 and w_2.

and offloading decisions to encode the chromosomes, individually encoding the caching decision can accelerate the convergence rate of GA.

6.1 Impact of Parameters

As discussed earlier, w_1 and w_2 are the weights to indicate the tradeoff between the content requesting frequency and the content size. To determine the appropriate weights, we have conducted a series of simulations and the simulation results are shown in Figure 2. In the simulation, the number of vehicular tasks ranges from 50 to 100 and the number of cacheable contents is set to 15. The x-coordinate denotes the number of tasks and the y-coordinate denotes the optimal values (i.e., $J(\zeta, \chi, \mathcal{F})$). The term "Weight" in this figure denotes the weight w_1, and $w_2 = 1 - w_1$.

The simulation results reveal that paying more attention to the content requesting frequency can bring about better performance in terms of the optimal value. For instance, the average optimal value reduces by about 19.4%, when w_1 is set to 0.8 instead of 0.1. Intuitively, caching those most frequently requested contents at the edge can greatly shorten the response latency and the simulation results accord with the expectation. Accordingly, if not stated otherwise, we set w_1 to 0.8 as the default value in the following experiments.

We have investigated the influence of the caching capability of the edge server on the performance in the following simulation. The simulation results are shown in Figure 3, where the x-coordinate denotes the number of tasks and the y-coordinate denotes the optimal values. It is observed that the more powerful the caching capability, the better the performance, no matter how the number of tasks varies. This result also accords with the expectation, as caching more content at the edge can cater to more task offloading requests. In the following experimental evaluation, the caching capability at the edge server takes 1,500 KB as the default value, unless otherwise stated.

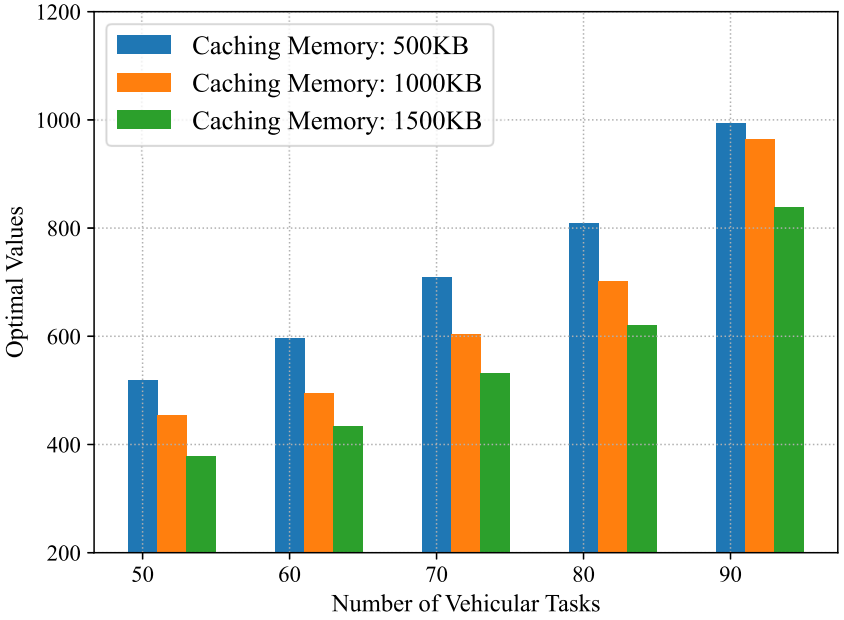
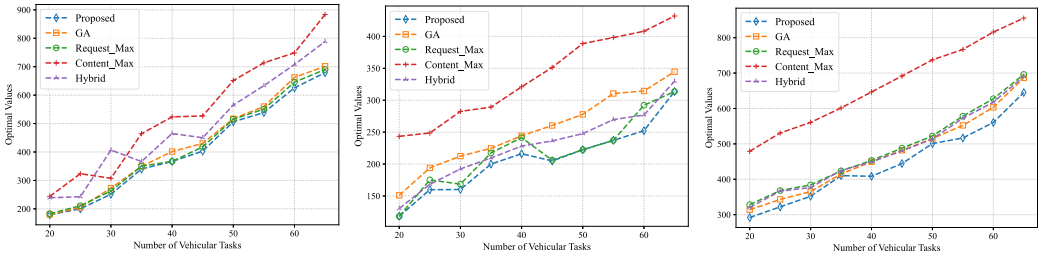


Fig. 3. Caching capacity at the edge.

6.2 Performance Comparison

We have conducted extensive simulations to evaluate the performance of our strategy. The simulation results for the performance comparison are reported in this section. First, we compare our approach with the four aforementioned baselines in a small-scale VEC network, since the GA-based approaches, including "Hybrid," are time-consuming in reality. Particularly, the number of cacheable contents is set to 15, and the number of vehicular tasks ranges between [20, 70]. Considering the influence of content requests upon performance, we investigate three cases in terms of the dynamics of content requests. The first case is that each vehicular task correlates with the required content randomly. The second case is that most of the vehicular tasks correlate with the required contents with small data sizes. For instance, we assume that the vehicular tasks request small-sized contents with a probability of 80% in the simulation. The last case is that most of the vehicular tasks correlate with the required contents with large data sizes. Similarly, we assume that the vehicular tasks request large-sized contents with a probability of 80% in the simulation. By considering these cases, we want to investigate the robustness and applicability of our approach.

The simulation results are depicted in Figure 4. The simulation results in Figure 4(a)–(c) correspond to the above three cases, respectively. From the figure, several conclusions can be made as follows. Generally, the proposed JTOCA optimization strategy has the best performance regarding the optimal values in the three cases. The "Content_Max" approach has the worst performance in the three cases, which can be attributed to the fact that this approach ignores the significant importance of content offloading frequency. As a matter of fact, individually caching those contents with large data sizes makes no sense, if these contents are not requested by vehicular tasks at all. In contrast, the "Request_Max" approach can reach the second-best performance among these approaches, even if it regards the content offloading frequency as the only evaluation metric in content caching. In addition, the iteration-based GA and "Hybrid" approaches are better than the "Content_Max" approach, but worse than the "Request_Max" approach and our approach. As for the performance



(a) Random content requests (b) Requests focus on small-size contents (c) Requests focus on large-size contents

Fig. 4. Performance evaluation for different approaches.

comparison between GA and the "Hybrid" approach, there are no obvious patterns as the number of vehicular tasks increases. For instance, the "Hybrid" approach is better than GA when the content requests focus on the contents with small sizes, as observed in Figure 4(b), and the "Hybrid" approach is worse than GA when the content requests focus on the contents with small sizes, as observed in Figure 4(a) or (c).

In the following simulation, we investigate the response latency of these approaches. Specifically, for the iteration-based GA and "Hybrid" approach, the population size (i.e., Pop_{size}) and the number of iterations (i.e., $Step_{max}$) are set to 100 and 50, respectively. The results are depicted in Figure 5, where the x-coordinate denotes the number of vehicular tasks and the y-coordinate denotes the response latency. Obviously, the response latency for GA and "Hybrid" increases rapidly as the number of vehicular tasks increases. The time taken to make caching and offloading decisions in time-critical VEC networks is prohibitive, especially when the number of contents and tasks is very large. In addition, "Hybrid" is better than GA regarding response latency since it only encodes the caching decision as an individual, thus reducing the length of the chromosome to a great extent and accelerating the convergence rate in the simulation.

In the next simulation, we evaluate the performance of our JTOCA optimization strategy in a large-scale VEC network that consists of a large number of cacheable contents. Considering the time-consuming inherent property of iteration-based approaches, we do not choose GA and "Hybrid" as the comparison baselines any longer in the following evaluation. Specifically, the set of contents is composed of two distinct parts. One part consists of randomly generated contents within the interval [1, 100] KB, and the other part consists of randomly generated contents within the interval [300, 500] KB. The caching capability is set to 4,500KB in this experiment. The number of contents is 100, and the number of vehicular tasks ranges from 20 to 70. Similarly, we also discuss three cases as mentioned above.

The simulation results are reported in Figure 6. Among the three approaches, our approach can reach the best performance in terms of the optimal values, no matter how the number of vehicular tasks varies. The "Content_Max" approach is the worst among the three approaches, even if the number of tasks is small. In contrast, the "Request_Max" approach always obtains the second-best performance in the simulation, which proves that the content requesting frequency is an important evaluation metric when making the content caching decision.

6.3 Real-World Application Scenarios

With the rapid advancement of the Internet of Vehicles, **New Energy Vehicles (NEVs)**, such as Tesla and XPENG, are increasingly equipped with high-precision sensors (e.g., LiDAR, cameras, mmWave radar) and vehicle-to-everything connectivity technologies, supported by high-performance onboard computing systems (e.g., NVIDIA Orin). Multimedia-oriented vehicular tasks,

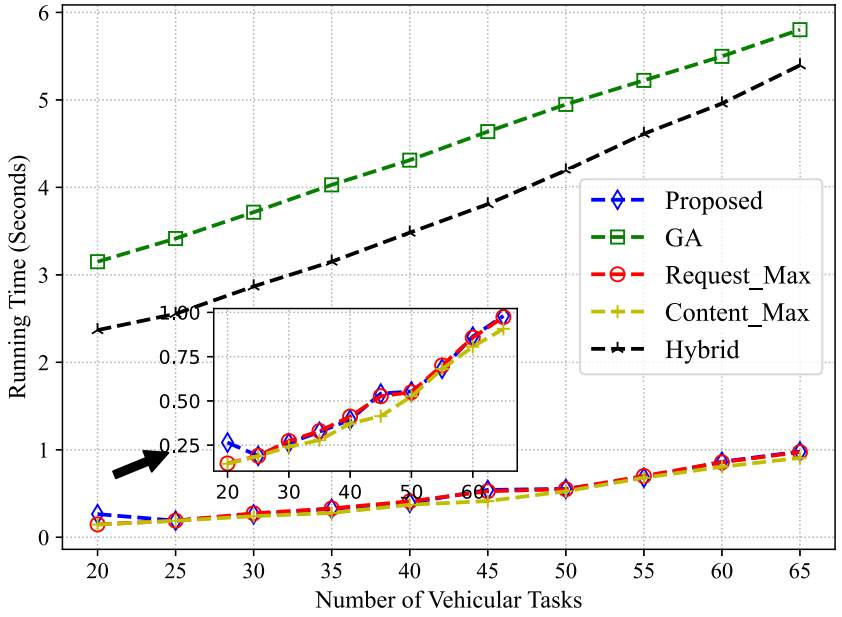
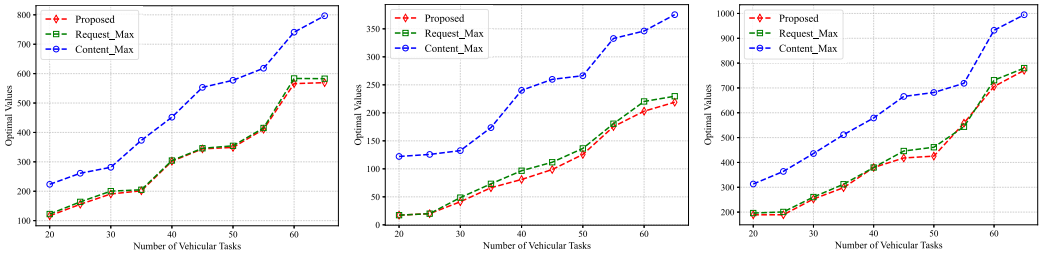


Fig. 5. Running time comparison among approaches.



(a) Random content requests (b) Requests focus on small-size contents (c) Requests focus on large-size contents

Fig. 6. Evaluation on the optimal values for different approaches.

such as real-time **High-Definition (HD)** map rendering (e.g., 3D semantic maps) and dynamic path planning in complex urban environments, require substantial computing resources and energy consumption, thereby posing significant challenges to the onboard systems. In this context, caching-enabled task offloading technologies can effectively alleviate the computational burden on local onboard systems of vehicles. For example, data related to HD map rendering can be cached at RSUs, and vehicular tasks requiring these data can be offloaded for execution at RSUs.

On the other hand, NEV smart cabins are transforming into what is referred to as "the third living space" [9], integrating multi-screen interactivity, AR navigation, and VR cloud gaming. However, achieving high-resolution rendering and low-latency interaction necessitates advanced onboard hardware. Alternatively, data associated with AR/VR graphic rendering can be cached at edge servers, allowing vehicular tasks requiring these data to be offloaded to RSUs. Subsequently, compressed and low-latency video streams are generated at the network edge, thereby reducing the computational loads on vehicles. Generally, recent advancements in the engineering domain

have enabled the practical implementation of theoretical insights regarding task offloading and RA within VEC systems.

7 Conclusion

Computation task offloading from vehicles has been a hot topic in VEC in the past few years. Different perspectives lead to different solutions for improving the performance of VEC systems in terms of response latency, energy consumption, and spectrum resources. In this article, we have proposed a content caching-assisted task offloading strategy in VEC, and the principle behind this is that the task-input data usually include a reusable part, i.e., the universal context data. Several heuristic rule-based algorithms are proposed to tackle the formulated MINLP problem. A series of experiments is conducted to evaluate the performance of our strategy, and the simulation results reveal its advantages in comparison to other baseline approaches. For future work, we plan to design more intelligent algorithms, such as deep reinforcement learning-based approaches, to solve similar problems.

References

[1] Bahman Abolhassani, John Tadrous, and Atilla Eryilmaz. 2022. Single vs distributed edge caching for dynamic content. *IEEE/ACM Trans. Netw.* 30, 2 (2022), 669–682.

[2] Rafat Aghazadeh, Ali Shahidinejad, and Mostafa Ghobaei-Arani. 2023. Proactive content caching in edge computing environment: A review. *Softw. Pract. Exp* 53, 3 (2023), 811–855.

[3] Stephen Anokye, Daniel Ayepah-Mensah, Abegaz Mohammed Seid, Gordon Owusu Boateng, and Guolin Sun. 2022. Deep reinforcement learning-based mobility-aware UAV content caching and placement in mobile edge networks. *IEEE Syst. J.* 16, 1 (2022), 275–286.

[4] Luciano Baresi, Davide Yi Xian Hu, Giovanni Quattrocchi, and Luca Terracciano. 2024. *NEPTUNE*: A comprehensive framework for managing serverless functions at the edge. *ACM Trans. Auton. Adapt. Syst*. 19, 1 (2024), 1–32.

[5] Elif Bozkaya-Aras. 2024. Blockchain-based secure content caching and computation for edge computing. *IEEE Access* 12 (2024), 47619–47629.

[6] Weibo Chu, Xinming Jia, Zhiwen Yu, John C. S. Lui, and Yi Lin. 2024. Joint service caching, resource allocation and task offloading for MEC-based networks: A multi-layer optimization approach. *IEEE Trans. Mob. Comput*. 23, 4 (2024), 2958–2975.

[7] Penglin Dai, Yaorong Huang, Kaiwen Hu, Xiao Wu, Huanlai Xing, and Zhaofei Yu. 2024. Meta reinforcement learning for multi-task offloading in vehicular edge computing. *IEEE Trans. Mob. Comput*. 23, 3 (2024), 2123–2138.

[8] Xingxia Dai, Zhu Xiao, Hongbo Jiang, Mamoun Alazab, John C. S. Lui, Geyong Min, Schahram Dustdar, and Jiangchuan Liu. 2023. Task offloading for cloud-assisted fog computing with dynamic service caching in enterprise management systems. *IEEE Trans. Ind. Inform.* 19, 1 (2023), 662–672.

[9] Dattatreya Prashanth Halady. 2016. Future automotive interiors—The 3rd living space. *SID Int. Symp. Dig. Tec. Pap.* 47, 1 (2016), 263–266.

[10] Lei Du, Zhaoquan Gu, Ye Wang, Le Wang, and Yan Jia. 2024. A few-shot class-incremental learning method for network intrusion detection. *IEEE Trans. Netw. Serv. Manag*. 21, 2 (2024), 2389–2401.

[11] Wenhao Fan, Shenmeng Li, Jie Liu, Yi Su, Fan Wu, and Yuanan Liu. 2023. Joint task offloading and resource allocation for accuracy-aware machine-learning-based IIoT applications. *IEEE Internet Things J.* 10, 4 (2023), 3305–3321.

[12] Ahmed S. Ghorab, Raed S. Rasheed, and Hatem M. Hammad. 2023. Joint microservices caching and task offloading framework in VEC based on deep reinforcement learning. *Int. J. Interact. Mob. Technol.* 17, 8 (2023), 78–99.

[13] Ilgin Gökasar, Dragan Pamucar, Muhammet Deveci, Brij B. Gupta, Luis Martínez, and Oscar Castillo. 2023. Metaverse integration alternatives of connected autonomous vehicles with self-powered sensors using fuzzy decision making model. *Inf. Sci.* 642 (2023), 119192.

[14] Yan Jia, Zhaoquan Gu, Lei Du, Yu Long, Ye Wang, Jianxin Li, and Yanchun Zhang. 2023. Artificial intelligence enabled cyber security defense for smart cities: A novel attack detection framework based on the MDATA model. *Knowl. Based Syst*. 276 (2023), 110781.

[15] Suling Lai, Linyu Huang, Qian Ning, and Chengping Zhao. 2024. Mobility-aware task offloading in MEC with task migration and result caching. *Ad Hoc Netw.* 156 (2024), 103411.

[16] Hui Li, Xiuhua Li, Chuan Sun, Fang Fang, Qilin Fan, Xiaofei Wang, and Victor C. M. Leung. 2024. Intelligent content caching and user association in mobile edge computing networks for smart cities. *IEEE Trans. Netw. Sci. Eng*. 11, 1 (2024), 994–1007.

[17] Yumei Li, Xiumin Zhu, Nianxin Li, Lingling Wang, Yawen Chen, Feng Yang, and Linbo Zhai. 2023. Collaborative content caching and task offloading in multi-access edge computing. *IEEE Trans. Veh. Technol.* 72, 4 (2023), 5367–5372.
[18] Zheng Li, Guosheng Li, Muhammad Bilal, Dongqing Liu, Tao Huang, and Xiaolong Xu. 2023. Blockchain-assisted server placement with elitist preserved genetic algorithm in edge computing. *IEEE Internet Things J.* 10, 24 (2023), 21401–21409.
[19] Zhen Li, Chao Yang, Xumin Huang, Weiliang Zeng, and Shengli Xie. 2023. CoOR: Collaborative task offloading and service caching replacement for vehicular edge computing networks. *IEEE Trans. Veh. Technol.* 72, 7 (2023), 9676–9681.
[20] Yinan Liu, Chao Yang, Xin Chen, and Fengyan Wu. 2024. Joint hybrid caching and replacement scheme for UAV-assisted vehicular edge computing networks. *IEEE Trans. Intell. Veh.* 9, 1 (2024), 866–878.
[21] Ranjan Pal, Nishanth Sastry, Emeka Obiodu, Sanjana Prabhu, and Konstantinos Psounis. 2023. EdgeMart: A sustainable networked OTT economy on the wireless edge for saving multimedia IP bandwidth. *ACM Trans. Auton. Adapt. Syst.* 18, 4 (2023), 1–25. 13:
[22] Qiaoqiao Shen, Bin-Jie Hu, and En-Jun Xia. 2022. Dependency-aware task offloading and service caching in vehicular edge computing. *IEEE Trans. Veh. Technol.* 71, 12 (2022), 13182–13197.
[23] Chaogang Tang, Wei Chen, Chunsheng Zhu, Qing Li, and Hsiao-Hwa Chen. 2022. When cache meets vehicular edge computing: Architecture, key issues, and challenges. *IEEE Wirel. Commun.* 29, 4 (2022), 56–62.
[24] Chaogang Tang, Yubin Zhao, and Huaming Wu. 2023. Lyapunov-guided optimal service placement in vehicular edge computing. *China Commun.* 20, 3 (2023), 201–217.
[25] Chaogang Tang, Chunsheng Zhu, Ning Zhang, Mohsen Guizani, and Joel J. P. C. Rodrigues. 2022. SDN-assisted mobile edge computing for collaborative computation offloading in industrial internet of things. *IEEE Internet Things J.* 9, 23 (2022), 24253–24263.
[26] Tuyen X. Tran and Dario Pompili. 2019. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Trans. Veh. Technol.* 68, 1 (2019), 856–868.
[27] Fei-Yue Wang. 2022. MetaVehicles in the metaverse: Moving to a new phase for intelligent vehicles and smart mobility. *IEEE Trans. Intell. Veh.* 7, 1 (2022), 1–5.
[28] Zichen Wang and Hongwei Du. 2023. Collaborative coalitions-based joint service caching and task offloading for edge networks. *Theor. Comput. Sci* 940, Part (2023), 52–65.
[29] Xianglin Wei, Jianwei Liu, Yangang Wang, Chaogang Tang, and Yongyang Hu. 2021. Wireless edge caching based on content similarity in dynamic environments. *J. Syst. Archit.* 115 (2021), 102000.
[30] Qiong Wu, Yu Zhao, Qiang Fan, Pingyi Fan, Jiangzhou Wang, and Cui Zhang. 2023. Mobility-aware cooperative caching in vehicular edge computing based on asynchronous federated and deep reinforcement learning. *IEEE J. Sel. Top. Signal Process.* 17, 1 (2023), 66–81.
[31] Jia Yan, Suzhi Bi, Lingjie Duan, and Ying-Jun Angela Zhang. 2021. Pricing-driven service caching and task offloading in mobile edge computing. *IEEE Trans. Wirel. Commun.* 20, 7 (2021), 4495–4512.
[32] Wentao Yang and Zhibin Liu. 2024. Efficient vehicular edge computing: A novel approach with asynchronous federated and deep reinforcement learning for content caching in VEC. *IEEE Access* 12 (2024), 13196–13212.
[33] Yang Yang, Lei Feng, Yao Sun, Yangyang Li, Fanqin Zhou, Wenjing Li, and Shangguang Wang. 2024. Decentralized cooperative caching and offloading for virtual reality task based on GAN-powered multi-agent reinforcement learning. *IEEE Trans. Serv. Comput.* 17, 1 (2024), 291–305.
[34] Zhixiu Yao, Shichao Xia, Yun Li, and Guangfu Wu. 2023. Cooperative task offloading and service caching for digital twin edge networks: A graph attention multi-agent reinforcement learning approach. *IEEE J. Sel. Areas Commun.* 41, 11 (2023), 3401–3413.
[35] Muhammad Yasir, Sardar Khaliq Uz Zaman, Tahir Maqsood, Faisal Rehman, and Saad Mustafa. 2023. CoPUP: Content popularity and user preferences aware content caching framework in mobile edge computing. *Clust. Comput.* 26, 1 (2023), 267–281.
[36] Ke Zhang, Jiayu Cao, Sabita Maharjan, and Yan Zhang. 2022. Digital twin empowered content caching in social-aware vehicular edge networks. *IEEE Trans. Comput. Soc. Syst.* 9, 1 (2022), 239–251.
[37] Jiayun Zhou and Xinglin Zhang. 2022. Fairness-aware task offloading and resource allocation in cooperative mobile-edge computing. *IEEE Internet Things J.* 9, 5 (2022), 3812–3824.

Appendix

A Proof of the Lemma 1

First of all, we prove that the objective function $M(\zeta, \chi, \mathcal{F}) = \sum_{n \in \mathcal{N}_o} \beta_n^t \frac{s_n}{f_{R,n}}$ in RA is convex w.r.t. $\mathcal{F}$. Let $\mathcal{H}_n \triangleq \beta_n^t \frac{s_n}{f_{R,n}}, \forall n \in \mathcal{N}_o$ and we then prove that $\mathcal{H}_n$ is convex w.r.t. $\mathcal{F}$ as follows. The partial

derivatives by differentiating $\mathcal{H}_n$ w.r.t. $\mathcal{F}$ can be calculated as:

$$\frac{\partial \mathcal{H}_n}{\partial f_{R,i}} = \begin{cases} -\frac{\beta_n^t s_n}{f_{R,i}^2}, & i = n, \\ 0, & i \neq n. \end{cases}$$

Then, the second partial derivatives are given as:

$$\frac{\partial^2 \mathcal{H}_n}{\partial f_{R,i} \partial f_{R,j}} = \begin{cases} \frac{2\beta_n^t s_n}{f_{R,n}^3}, & i = n \text{ and } j = n, \\ 0, & \text{otherwise.} \end{cases}$$

Accordingly, the Hessian matrix of $\mathcal{H}_n$ can be given as:

$$\mathcal{M}(\mathcal{H}_n) = \begin{bmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \frac{2\beta_n^t s_n}{f_{R,n}^3} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix}.$$

We observe that $Y^T \mathcal{M} Y \geq 0$ always holds for an arbitrary nonzero vector Y, so $\mathcal{M}(\mathcal{H}_n)$ is positive semi-definite. Accordingly, $\mathcal{H}_n$ is convex w.r.t. $\mathcal{F}$, $\forall n \in \mathcal{N}_o$. Owing to the additivity attribute of convex function, $\sum_{n \in \mathcal{N}_o} \mathcal{H}_n$ is also convex w.r.t. $\mathcal{F}$. Thus, the objective function $M(\zeta, \chi, \mathcal{F})$ is convex.

Second, we need to prove that the constraints are also convex. Obviously, the constraints (Equations (23) and (26)) are convex. For the constraint (Equation (22)), its left-hand side T_n ($\forall n \in \mathcal{N}$) can be expanded as:

$$\begin{aligned} T_n &= (1 - \zeta_n) T_n^{loc} + \zeta_n (T_{n,k}^{off} + T_n^{exe}) \\ &= (1 - \zeta_n) T_n^{loc} + \zeta_n T_{n,k}^{off} + \zeta_n \frac{s_n}{f_{R,n}}. \end{aligned}$$

We need to discuss T_n depending on the following two cases. One case is that the vehicle n executes its task M_n locally, i.e., $n \notin \mathcal{N}_o$. We then have $T_n = T_n^{loc}$ that is independent of the RA policy $\mathcal{F}$. Obviously, T_n as a constant is convex regarding $\mathcal{F}$.

The other case is that the vehicle n offloads M_n to the edge, i.e., $n \in \mathcal{N}_o$. We have $T_n = T_{n,k}^{off} + s_n / f_{R,n}$. Only the last term on the RHS (i.e., $s_n / f_{R,n}$) depends on the RA $\mathcal{F}$. The second partial derivatives of T_n, w.r.t. $\mathcal{F}$, are given as:

$$\frac{\partial^2 T_n}{\partial f_{R,i} \partial f_{R,j}} = \begin{cases} \frac{2s_n}{f_{R,n}^3} > 0, & i = n \text{ and } j = n, \\ 0, & \text{otherwise.} \end{cases}$$

Similarly, the Hessian matrix of T_n regarding $\mathcal{F}$ can also be constructed. We have $X^T \mathcal{M} X \geq 0$ always holds for an arbitrary nonzero vector X, so $\mathcal{M}(T_n)$ is positive semi-definite. Accordingly, combining the two cases, T_n is convex w.r.t. $\mathcal{F}$, $\forall n \in \mathcal{N}$. Thus, the constraint (Equation (22)) is convex. As a result, the RA is a convex optimization problem, and the proof is completed.

Received 23 April 2024; revised 19 March 2025; accepted 17 April 2025