Computation Offloading and Resource Allocation in Failure-Aware Vehicular Edge Computing

Chaogang Tang, *Member, IEEE*, Ge Yan, Huaming Wu, *Senior Member, IEEE*, and Chunsheng Zhu, *Member, IEEE*

***Abstract*—The advent of Intelligent Cyber-Physical Transportation Systems (ICTS) has not only accelerated the reformation and evolvement of smart transportation, but also ushered in a new era of vehicular applications. These applications typically impose stringent latency requirements and demand substantial computing resources. Vehicular edge computing (VEC) has emerged as an efficient solution to address these challenges, leveraging its inherent ability to provide ultra-low latency services. Existing studies primarily concentrate on either optimizing resource allocation or minimizing response latency, while ignoring the fact that the task execution in VEC is more susceptible to failures compared to cloud computing environments. Accordingly, we design a cost-efficient and failure-resistant task offloading strategy for VEC systems with the goal of minimizing the average response latency for all tasks. Specifically, our problem is modeled as a nonlinear multi-constraint continuous optimization problem, with tightly coupled optimization variables in the objective function and constraints. To tackle this issue, we initially decompose the optimization problem into per-slot optimization subproblems. Subsequently, we employ an effective algorithm with low time complexity to solve these subproblems in a slot-by-slot manner. We comprehensively evaluate the performance of our approach through extensive simulations, demonstrating that our method outperforms the baseline approaches in various aspects.**

***Index Terms*—ICTS, computation offloading, resource allocation, failure-aware, vehicular edge computing.**

I. Introduction

THE CYBER-PHYSICAL Transportation Systems (CPTS) aim to build bridges between the cyber system and physical system to realize higher efficiency in smart transportation. The rapid development of Intelligent Cyber-Physical Transportation Systems (ICTS) has significantly expedited the transformation and advancement of smart transportation. Its primary objective is to meet diverse demands, including traffic management, smart parking, and autonomous driving, by effectively integrating and leveraging data from multiple sources in both the physical space (sensors and actuators) and cyberspace (computation and communication) [1], [2].

The explosive growth of vehicular applications has posed tremendous pressure on smart vehicles, given their constrained computing and storage capacities. With the increasing amount of computation generated by vehicles, there is a surge in demand for computing resources [3]. However, owing to their inherent computational limitations, vehicles frequently find it necessary to offload specific computations to external entities within the ICTS framework. Roadside units (RSUs) serve as a perfect alternative for offloading and executing computations, leveraging their abundant communication and computation resources, which surpass those available in vehicles. This computing paradigm is referred to as Vehicular Edge Computing (VEC), where communication and computation resources are expanded from the remote cloud center to the logical network edge, represented by RSUs. This setup allows computations to be executed in close proximity to the data sources, such as smart vehicles.

As an indispensable part of ICTS, performance-enhanced RSU enables the efficient allocation of computational resources, offering scalability and flexibility through virtualization technologies. A variety of vehicular applications, especially those with rigorous latency requirements, can be outsourced and performed in VEC in the form of computational tasks. However, the precise allocation of computational resources is crucial in VEC, given that resources are relatively constrained compared to cloud computing. Numerous studies have focused on resource optimization from the perspective of the edge server [4], [5] and response latency optimization from the perspective of vehicular applications [6], [7]. Specifically, vehicles can offload a portion or all of their tasks to RSUs to accelerate task processing or conserve energy. This raises questions about whether, when, and to what extent computation should be offloaded to achieve these objectives. Conversely, VEC can process these tasks by distributing communication and computational resources among them. Consequently, questions arise about the allocation of resources to individual tasks to ensure their successful execution while meeting diverse performance requirements.

Unfortunately, most of the existing literature has ignored the fact that task execution in VEC is more prone to failures

Manuscript received 17 October 2023; revised 14 November 2023; accepted 9 December 2023. Date of publication 12 December 2023; date of current version 26 April 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62071327, and in part by the Tianjin Science and Technology Planning Project under Grant 22ZYYYJC00020. *(Corresponding author: Huaming Wu.)*

Chaogang Tang is with the School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China (e-mail: cgtang@cumt.edu.cn).

Ge Yan is with the School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221116, China (e-mail: TS21060201P31@cumt.edu.cn).

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

Chunsheng Zhu is with the College of Big Data and Internet, Shenzhen Technology University, Shenzhen 518118, China (e-mail: chunsheng.tom.zhu@gmail.com).

Digital Object Identifier 10.1109/TCE.2023.3342017

compared to cloud computing [8]. The resumption of task execution in VEC following failures is time-consuming, and, in more severe cases, some tasks may be unrecoverable, particularly in cases of hardware-level failures, such as edge server malfunctions. To simplify our discussion, we focus on recoverable failures in this paper, including temporary disconnections and software failures. In such scenarios, the VEC system incurs additional energy consumption and calculation delays for failure recovery. Task failures not only influence the quality of service (QoS) such as reliability and availability from the perspective of service providers, but also degrade the quality of experience (QoE) from the perspective of service requestors. As a result, it is worthwhile to investigate failure-resisted task offloading strategies in VEC systems, but current works seldom pay attention to such strategies.

Based on the aforementioned observations, the primary focus of this paper is to develop a cost-efficient strategy for computation offloading and resource allocation in failure-aware VEC systems. In particular, we strive to minimize the average response latency for all tasks within the optimization period. To achieve this, we take into account various types of failures and integrate them into the networking and computation models, respectively. To sum up, the contributions of this paper are threefold, given below:

- We investigate the failure-aware task offloading in the VEC system by incorporating task failures into the proposed networking and calculation models. We mathematically formulate the optimization problem, and endeavor to minimize the average response time of all the tasks over a long time-slotted horizon by jointly optimizing the task offloading and resource allocation decisions in this paper.
- The optimization variables related to task offloading and resource allocation are tightly coupled in both objective function and constraints. This interdependence adds complexity to the problem-solving process. To this end, we decompose the optimization problem into per-slot optimization subproblems through Lyapunov optimization. These subproblems are then resolved sequentially in a slot-by-slot way.
- We have conducted extensive simulations to evaluate the effectiveness of our approach. The results clearly demonstrate that our approach to task offloading and resource allocation in a failure-aware VEC system outperforms other strategies, particularly in terms of achieving a superior average response latency.

The remainder of this paper is organized as follows. The state-of-the-art literature is reviewed in Section II. We present our failure-tolerant VEC architecture in Section III. The optimization problem is described and formulated in Section IV and Section V, respectively. Section VI presents our algorithm for solving our optimization problem. The simulation results are presented in Section VII, followed by a conclusion in Section VIII.

II. Related Work

This section provides an in-depth review of the state-of-the-art literature concerning task offloading and resource allocation in VEC systems. Currently, there is an enormous amount of literature focusing on these aspects in VEC, encompassing various objectives such as latency minimization, energy optimization, cost optimization, or a weighted combination of them [9], [10], [11], [12], [13], [14].

A. Traditional Approaches

Lin et al. [11] studied online task offloading for heterogeneous VEC, considering the environmental dynamics. Through learning the relationship between historical observations and rewards, they aim to minimize the expected energy consumption for all the tasks with stringent latency requirements. The task popularity is introduced to their contextual clustering approach. Fan et al. [15] also paid attention to the task offloading in VEC, by jointly optimizing task offloading, bandwidth and computing resources in VEC. Several algorithms such as Generalized Benders Decomposition (GBD) and greedy algorithm are applied to guarantee the efficiency and low time complexity.

It is challenging to schedule the computing resources in 6G heterogeneous vehicular networks (HetVNET) while meeting customized QoE. In view of this, Hui et al. [9] provided personalized edge computing services for vehicles by designing a secure edge computing architecture based on smart contracts. A collaborative resource allocation scheme was proposed to help these infrastructures satisfy the QoE of vehicles. Hossain et al. [16] put forward a dynamic task offloading scheme using a non-cooperative game (NGTO). Each vehicle makes its decisions independently on where to offload their vehicular tasks, so as to maximize their own benefits. Each vehicle can adaptively adjust the task-offloading probability in the pursuit of utility maximization, and the game equilibrium can be achieved using their offloading algorithm.

B. AI-Based Approaches

In recent years, Artificial Intelligence (AI)-based technologies are increasingly applied to vehicular networks for a variety of purposes [7], [17], [18], [19], [20]. For instance, Karimi et al. [18] studied vehicular task offloading within the context of cooperation between Mobile Edge Computing (MEC) and cloud centers. They formulate the resource allocation problem with the objective of optimizing response latency. A deep reinforcement learning approach is adopted for approaching an optimal solution. The numerical results demonstrate the efficiency of their approach in terms of acceptance rate and time complexity.

To enhance edge intelligence in vehicular networks, the combination of federated learning and blockchain can be leveraged to enable time-sensitive vehicular applications to be executed in a distributed and time-efficient manner. Zhao et al. [7] put forward a federated dual deep Q-learning algorithm to cater to the dynamic and unpredictable nature of vehicular networks. In their approach, each vehicle deploys its own learning agent for state sensing, thereby improving scalability and flexibility. Lin et al. [19] proposed a computation offloading model based on the Markov Decision Process and introduced the integration of Deep Q-Network

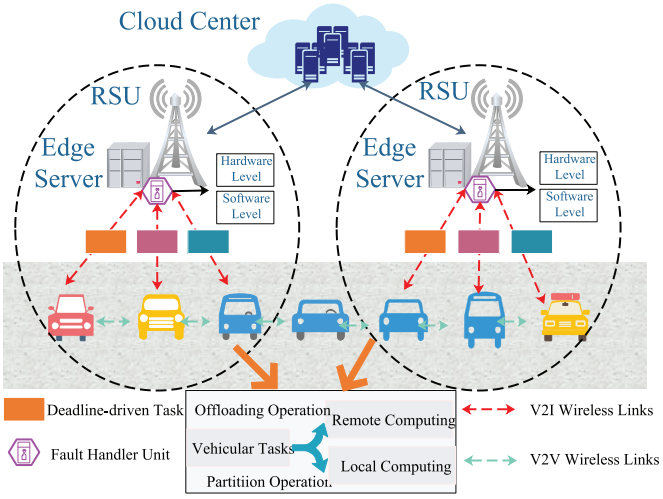


Fig. 1. The considered application scenario in this paper.

with the Simulated Annealing algorithm. Their aim is to design a cost-efficient and energy-aware computation offloading strategy in VEC systems. Similarly, Zhan et al. [20] also investigated computation offloading in VEC systems, with a focus on minimizing user cost, in the long run, using a deep reinforcement learning approach. Specifically, they design an offloading policy using a deep neural network trained by the proximal policy optimization algorithm. The effectiveness of their approach is evaluated through extensive simulations, which include comparisons with six baseline algorithms. The simulation results demonstrate the advantages of their approach in terms of reducing user cost.

C. Failure-Related Approaches

There are also several works that pay attention to failure-related issues in vehicular networks such as [21], [22], [23]. For instance, some AI-based algorithms are used to construct spanning trees in VANETs to handle the failure of nodes and fast-moving vehicles. A near-optimal spanning tree is constructed based on two phases in the RSU in [21]. Specifically, an artificial bee colony algorithm is adopted to construct the spanning tree in the first stage. In the second stage, they try to construct a near-minimum spanning tree with the maximum number of leaves.

Despite massive efforts made to improve the performance of VEC systems, the aforementioned works seldom take into account task failures during service provisioning in VEC. Owing to the resource scarcity at the edge servers compared to the cloud center, task failures occur on occasion, which seriously undermines the performance of VEC systems and degrades the QoE from the perspective of resource requestors. Hence, we believe that it is worth investigating failure-resisted task offloading strategies in VEC systems.

III. Proposed Failure-Tolerant VEC Architecture

Fig. 1 illustrates the system model considered in this paper, which extends the traditional VEC system by incorporating task failures. The model consists of smart vehicles, roadside units (RSUs), and a cloud center.

The cloud center serves as a centralized resource pool, offering dynamic allocation and elastic expansion of computing, storage, and network resources to support a wide range of services [24]. With its robust computing and storage capabilities, cloud computing is particularly suitable for non-real-time and long-term decision-making scenarios. However, accessing the cloud center often involves traversing the core network, resulting in substantial response delays that frequently violate the stringent latency requirements of vehicular applications.

Moreover, RSUs play a crucial role by serving as an intermediate geographical point between the cloud center and smart vehicles. This positioning enables the migration of resources from the cloud center to the logical network edge through the deployment of edge servers at RSUs. In contrast to the cloud center, RSUs possess the capability to handle computations from nearby vehicles, leading to reduced response latency and an enhanced QoE [25]. Additionally, the deployment of resources at RSUs can alleviate the burden on backhaul networks, thereby enhancing overall network performance.

Notwithstanding these advantages, task failures can occur in VEC systems owing to the aforementioned reasons. Therefore, we have deployed the fault handler unit, as shown in Fig. 1, and the corresponding functionalities are listed as follows:

- *Failure Monitor*: This unit is responsible for monitoring the failure events and managing the related log files. By recording and analyzing the log files, it can estimate failure-related parameters, e.g., failure rate and recovery rate.
- *Failure Category Recognition*: The unit is responsible for failure classification based on the failure information, including factors such as failure reasons, locations, and frequencies. For instance, as depicted in the figure, the failures can be categorized into hardware failures and software failures.
- *Failure Processing*: The most important role of this unit is to process the failures when they occur by employing some recovery mechanisms like checkpointing and rollback/roll-forward technologies [8], [26].

In this architecture, vehicular applications can be partially/totally outsourced and executed at RSUs, as shown in the figure. These vehicles, which offload and execute tasks, are commonly referred to as task vehicles.

IV. System Model

In our system model, we divide the optimization period $\mathcal{T}$ into T discrete time slots, such that $\mathcal{T} = \{1, 2, \ldots, T\}$, and each slot t has a duration of τ time units. The set of task vehicles is indexed by $\mathcal{N} = \{1, 2, \ldots, N\}$, where N is the number of task vehicles. We use $C_n(t)$ to denote the data size of the task generated by task vehicle n in the t-th time slot, and $d_n(t)$ represents the delay requirement of task vehicle n in slot t. Task failure occurs for task vehicle n in time slot t if the response latency for the task exceeds $d_n(t)$. Task vehicle n has the flexibility to offload a portion or all of its

TABLE I
NOTATIONS AND THEIR MEANINGS

Notation	Description
$x_n(t)$	Decision variable to indicate how much computation from vehicle n is offloaded in slot t
$f_E^n(t)$	Decision variable to indicate how much processing frequency of RSU is allocated to vehicle n in slot t
T	The number of time slots in optimization period
$C_n(t)$	The data size of task from vehicle n in slot t
$d_n(t)$	The delay requirement of vehicle n in slot t
$S_n(t)$	The size of task data from vehicle n to RSU in slot t
$J_n(t)$	The size of task data to be retransmitted by vehicle n in the next time slot t
τ	The time duration for each time slot
f_n	The processing frequency of vehicle n
ϑ	The effective switched capacitance coefficient
ς	The number of cycles needed to perform one task-input bit at vehicle n
B	The uplink channel bandwidth between task vehicle and RSU
$P_n(t)$	The transmission power of vehicle n in time slot t
$H_n(t)$	The channel gain between vehicle n and RSU in time slot t
δ^2	The noise power between task vehicle and RSU
$\lambda_n(t)$	The average rate of task arrival at RSU from vehicle n in time slot t
$\eta_n(t)$	The failure rate of tasks from vehicle n at RSU in time slot t
$\theta_n(t)$	The recovery rate for tasks from vehicle n in time slot t
ρ_t	The ratio of result size to the task data size in time slot t

computation to the RSU for execution. To represent this, we define a decision variable $x_n(t) \in [0, 1]$, which indicates the extent of computation offloaded in time slot t. Specifically, $x_n(t) = 0$ means that all the computation is performed locally at task vehicle n, and $x_n(t) = 1$ signifies that all computation is offloaded and executed at the edge server. In general, $x_n(t)C_n(t)$ represents the amount of computation offloaded to the RSU for execution in time slot t. In addition, we introduce some key notations that will be used throughout the paper, listed in Table I.

When a task is offloaded by task vehicle n, the response delay typically consists of three components: the transmission delay, the computation delay, and the feedback delay. Each type of delay is expressed based on its respective models, taking into account task failures. These models will be further elaborated in the subsequent sections.

A. Networking Model

At the beginning of time slot t, a task vehicle n generates $C_n(t)$ units of task data, which is then stored in its local buffer. When task vehicle n decides to offload computation, the transmission delay is governed by the following considerations. Let $S_n(t)$ represent the amount of task data that can be transmitted from task vehicle n to the RSU in time slot t. Task failure occurs if the response delay exceeds the deadline. In such cases, we denote $J_n(t+1)$ as the amount of task data that needs to be retransmitted by task vehicle n in the subsequent time slot $t+1$. As a result, the task data in the local buffer of task vehicle n can be regarded as a queue, with the corresponding backlog [27]:

$$Q_n(t+1) = \max\{Q_n(t) + x_n(t)C_n(t) - S_n(t), 0\} + J_n(t+1), \tag{1}$$

which implies that the remaining computation conducted locally by task vehicle n must be completed before the deadline. The time required for local computing at task vehicle n in time slot t, denoted as $\tau_n^{loc}(t)$, is calculated as follows:

$$\tau_n^{loc}(t) = \frac{(1 - x_n(t))C_n(t)}{f_n}, \tag{2}$$

where f_n is the processing frequency of task vehicle n. As mentioned earlier, the time $\tau_n^{loc}(t)$ should satisfy the following constraint:

$$\tau_n^{loc}(t) \le d_n(t) \quad \forall t \in \mathcal{T}. \tag{3}$$

We also consider the energy constraint in this paper. Specifically, the energy consumption for vehicle n, denoted as $e_n^{loc}(t)$, can be expressed as follows:

$$e_n^{loc}(t) = \vartheta\varsigma(1 - x_n(t))C_n(t)f_n^2, \tag{4}$$

where ϑ represents the effectively switched capacitance coefficient, and ς denotes the number of cycles required to process one task-input bit at task vehicle n.

We assume that vehicle n adopts an orthogonal spectrum for task offloading to avoid co-channel interference. The data rate for task offloading can be expressed as:

$$R_n(t) = B\log\left(1 + \frac{P_n(t)H_n(t)}{\delta^2}\right), \tag{5}$$

where B represents the uplink channel bandwidth between task vehicle n and the RSU, $P_n(t)$ is the transmission power of task vehicle n in slot t, $H_n(t)$ represents the channel gain between task vehicle n and the RSU in slot t, and δ^2 represents the noise power.

To determine $S_n(t)$ for task vehicle n in time slot t, it's important to consider that task failure must occur if the transmission delay for n exceeds either the delay requirement $d_n(t)$ or the per-slot duration τ. Generally, the task failure causes the backlog of task-input data, so the amount of task data ($S_n(t)$) to be offloaded in time slot t should exceed the amount of computation generated in the current slot for offloading ($x_n(t)d_n(t)$). Intuitively, $S_n(t)$ depends upon not only the data rate for task offloading but also the time required for task offloading. Moreover, each task vehicle n has its own delay requirement $d_n(t)$. When considering these factors and the variability of backlog for each task vehicle in different time slots, quantifying $S_n(t)$ becomes a complex task. To ease the discussion, we assume that the time used for computation offloading for each vehicle in each time slot is fixed [27] and denote it by τ_n^{trs}. Then, $S_n(t)$ can be expressed as:

$$S_n(t) = \tau_n^{trs}R_n(t). \tag{6}$$

The achievable throughput between task vehicle n and the RSU in time slot t can be calculated by:

$$D_n(t) = \min\{Q_n(t) + x_n(t)C_n(t), S_n(t)\}. \tag{7}$$

B. Calculation Model

With the arrival of tasks in each time slot, RSU will allocate necessary resources to these tasks before executing them. Unlike the previous work [28], we have considered task failures in this process, and further assume that these failures are recoverable. Then, the calculation delay of task vehicle n, denoted as $\tau_n^{clt}(t)$, is comprised of three components, namely, the queueing time $l_n^q(t)$, the service time $l_n^s(t)$, and the recovery time $l_n^r(t)$ if the task execution failure occurs.

The popularity of smart vehicles gives rise to a surge in vehicular tasks. In large-scale VEC systems, there may be hundreds of offloading requests from task vehicles within a short time interval. Given this scenario, it's essential to consider the queueing time in this paper, as it constitutes a significant portion of the response latency. Furthermore, we assume that the task arrivals at the RSU from task vehicle n follow a Poisson process with an average rate $\lambda_n(t)$ in time slot t. Additionally, information about other tasks from $\mathcal{N}\backslash\{n\}$ is required for obtaining the average queueing delay. According to the queueing model, the arrival rate of all tasks in time slot t, i.e., $\tilde{\lambda}(t) = \sum_{n=1}^{N} \lambda_n(t)$, also follows a Poisson process. Thus, the average queueing time for task vehicle n in the waiting queue in time slot t can be calculated as:

$$l_n^q(t) = \frac{\rho(t)}{\tilde{\mu}(t) - \tilde{\lambda}(t)}, \quad \forall n \in \mathcal{N}, \tag{8}$$

where $\tilde{\mu}(t)$ is the average service rate for the tasks executed at RSU in time slot t, and $\rho(t)$ is the service intensity and defined as the ratio of the average arrival rate $\tilde{\lambda}(t)$ to the average service rate $\tilde{\mu}(t)$. Note that $\tilde{\mu}(t)$ can be estimated through historical statistics by the RSU.

The service time, also referred to as the calculation delay in this paper, represents the time taken by the RSU to execute the offloaded task from task vehicle n. If there is no failure in this process, the service time, denoted as $l_n^s(t)$, can be expressed as follows:

$$l_n^s(t) = \frac{D_n(t)}{f_E^n(t)}, \tag{9}$$

where $f_E^n(t)$ is the processing frequency of the RSU allocated to task vehicle n in time slot t. However, if a failure occurs during task execution, the RSU needs recovery time to restart the task from task vehicle n. This is because the task failures in this paper are assumed to be recoverable, as mentioned earlier.

We assume that task failures at task vehicle n follow a Poisson process with the failure rate $\eta_n(t)$ [26]. Let $\mathcal{N}(t)$ denote the number of failures during the time $(0, t]$. Then, the probability that k failures occur for the task from vehicle n within the time duration $l_n^s(t)$ is:

$$\mathrm{P}\{\mathcal{N}\big(l_n^s(t)\big) = k\} = \frac{\big(\eta_n(t) l_n^s(t)\big)^k}{k!} e^{-\eta_n(t) l_n^s(t)}, \tag{10}$$

where $\mathbb{E}[\mathcal{N}(l_n^s(t))] = \eta_n(t) l_n^s(t)$. $\mathcal{R}_k(l_n^s(t))$ denotes the recovery time for the k-th failure when RSU executes the task from n. The recovery time is assumed to follow an exponential distribution with the recovery rate $\theta_n(t)$. It is worth mentioning that the recovery time only depends upon RSU. Meanwhile, we also assume that $\mathcal{N}(l_n^s(t))$ failures at the RSU are independent of each other. Thus, they are independent and identically distributed (i.i.d.) random variables. The total recovery time can be expressed as:

$$\mathcal{R}\big(l_n^s(t)\big) = \sum_{k=1}^{\mathcal{N}(l_n^s(t))} \mathcal{R}_k\big(l_n^s(t)\big), \tag{11}$$

where $\mathcal{R}(l_n^s(t))$ follows Gamma distribution, i.e., $\mathcal{R}(l_n^s(t)) \sim \Gamma(\eta_n(t) l_n^s(t), \theta_n(t))$. As a result, the recovery time $l_n^r(t)$ can be expressed as the mean of the total recovery time:

$$l_n^r(t) = \frac{\eta_n(t) l_n^s(t)}{\theta_n(t)}. \tag{12}$$

Thus, the calculation delay of task from n, denoted by $\tau_n^{clt}(t)$, can be expressed as:

$$\tau_n^{clt}(t) = l_n^q(t) + l_n^s(t) + l_n^r(t). \tag{13}$$

C. Result Feedback Model

The data size of the execution result typically depends on the intrinsic features of the task, making it challenging to quantify in most cases. Hence, current works tend to ignore the feedback delay, with an assumption that the task-output data size is negligible compared to the task data size. Nevertheless, in view of the possible failure during task offloading, we choose to quantify the data size of the result, by assuming that the ratio of the result data size to the task data size remains the same in each time slot [27], [29]. We denote this ratio as ρ_t in time slot t. If the RSU successfully completes the task from task vehicle n, it will return an amount of data size equal to $\rho_t D_n(t)$. Therefore, the time taken to deliver the result, i.e., the feedback delay denoted by $\tau_n^{fb}(t)$, is calculated as:

$$\tau_n^{fb}(t) = \frac{\rho_t D_n(t)}{R_{E,n}(t)}, \tag{14}$$

where $R_{E,n}$ is the transmission rate from RSU to n.

D. Task Data Retransmission Model

The total response delay for the task from vehicle n that is executed at the RSU in time slot t, denoted as $\tau_n(t)$, is calculated as follows:

$$\tau_n(t) = \tau_n^{trs}(t) + \tau_n^{clt}(t) + \tau_n^{fb}(t). \tag{15}$$

From the perspective of RSU (i.e., the edge server), if task failures occur, the task will be resumed by the RSU using checkpointing and rollback/roll-forward technologies. From the perspective of task vehicle n, retransmission of task data is necessary if task failures occur. In particular, task data retransmission for vehicle n is required if n does not receive the feedback before the deadline $d_n(t)$. In this scenario, we define the variable $J_n(t+1)$, representing the amount of task data that needs to be retransmitted by task vehicle n in the next time slot $t+1$, as given below:

$$J_n(t+1) = D_n(t)(1 - \mathbb{F}\{d_n(t) - \tau_n(t)\}), \tag{16}$$

where $\mathbb{F}\{\cdot\}$ is the unit-step function, i.e., $\mathbb{F}\{x\} = 1$ for $x \geq 0$, and 0, otherwise.

E. Energy Consumption Model

Task execution at the edge server incurs negligible energy consumption, which should be considered in this paper. This is important because both the computing capability and energy supply at the RSU are relatively limited compared to the cloud center. Specifically, we use $E_R(t)$ to represent the energy consumption of the RSU used for executing the offloaded tasks from the vehicle set $\mathcal{N}$ in time slot t. Then, $E(t)$ can be expressed as:

$$E_R(t) = \vartheta_E \varsigma_E \sum_{n=1}^{N} D_n(t) \left(f_E^n(t)\right)^2, \tag{17}$$

where ϑ_E represents the effectively switched capacitance coefficient for the RSU, and ς_E is the number of cycles required to process one task-input bit at the RSU.

V. Problem Formulation

In this paper, we aim to minimize the average response latency for all the tasks along the optimization period, by jointly optimizing offloading decision $\boldsymbol{x}$ and edge server frequency $\boldsymbol{f}_E$, while satisfying multiple constraints. We first define decision variables $\boldsymbol{x}(t) = \{x_n(t) | n = 1, \ldots, N\}, \forall t \in \mathcal{T}$, and $\boldsymbol{f}_E(t) = \{f_E^n(t) | n = 1, \ldots, N\}, \forall t \in \mathcal{T}$. Thus, the decision profile can be given as $\boldsymbol{x} = \{\boldsymbol{x}(t) | t = 1, \ldots, T\}$, and $\boldsymbol{f}_E = \{\boldsymbol{f}_E(t) | t = 1, \ldots, T\}$. Then, the optimization problem can be formulated as follows:

$$(\boldsymbol{\mathcal{P}_1}) \quad \min_{\boldsymbol{x}\boldsymbol{f}_E} \quad \frac{1}{T} \sum_{t=1}^{T} \sum_{n=1}^{N} \tau_n(t)$$

$$s.t. \quad \frac{1}{T} \sum_{t=1}^{T} E_R(t) \le E_{th}, \tag{18}$$

$$e_n^{loc}(t) \le E_n(t), \quad \forall t \in \mathcal{T}, \ \forall n \in \mathcal{N}, \tag{19}$$

$$\tau_n^{loc}(t) \le d_n(t), \quad \forall n \in \mathcal{N}, \ \forall t \in \mathcal{T}, \tag{20}$$

$$\sum_{n=1}^{N} f_E^n(t) \le f_E^{max}(t), \quad \forall t \in \mathcal{T}, \tag{21}$$

$$x_n(t) \in [0, 1], \quad \forall n \in \mathcal{N}, \ \forall t \in \mathcal{T}, \tag{22}$$

$$\rho_t \in [0, 1], \quad \forall t \in \mathcal{T}, \tag{23}$$

$$f_E^n(t) > 0, \quad \forall n \in \mathcal{N}, \ \forall t \in \mathcal{T}, \tag{24}$$

where constraint (18) denotes that the average energy consumption for computation carried out at the edge server across different time slots must be lower than the given threshold E_{th}. This ensures that the remaining energy can be allocated to other purposes, such as vehicle-to-infrastructure and pedestrian-to-infrastructure communications. Similarly, constraint (19) ensures that the energy consumption for local computation at vehicle n in time slot t must be lower than its threshold $E_n(t)$. Constraint (20) ensures that the time required for local computation at task vehicle n in time slot t should be accomplished before the deadline $d_n(t)$. Furthermore, constraint (21) ensures that the total amount of processing frequency allocated to these tasks cannot exceed the total computing capability of the RSU. Lastly, constraints (22)–(24) ensure the rationality of decision variables and certain constants. For instance, the offloading decision is bounded between 0 and 1, and the value of processing frequency allocated to offloaded tasks by the RSU must be non-negative.

The following reasons make our optimization problem $\boldsymbol{\mathcal{P}_1}$ difficult to solve. First, the problem $\boldsymbol{\mathcal{P}_1}$ is nonlinear due to the presence of the min(·) operation (see Eq. (7)) and the unit-step function $\mathbb{F}\{\cdot\}$ in the objective function. Second, the optimization variables for task offloading and resource allocation are tightly coupled in both the objective function and the constraints. Third, the long-term energy consumption constraint (18) couples the task offloading decision $\boldsymbol{x}$ with resource allocation decision $\boldsymbol{f}_E$ across different time slots. Last but least, optimally solving problem $\boldsymbol{\mathcal{P}_1}$ requires the information from all time slots, which can only be realized in an offline way. However, the information on future time slots can only be predicted in reality, which increases the difficulty in obtaining high-precision solutions. Accordingly, all these factors necessitate an online approach to solve this problem without relying on future information.

VI. Algorithm Design

Considering these challenges, we first convert the long-term energy constraint (18) into per-slot constraints using Lyapunov optimization techniques [25], [30]. By doing this, the optimization problem across different time slots can be converted into numerous subproblems, each focused on a single time slot. This allows us to tackle the optimization problem in a slot-by-slot manner, with the aim of obtaining the suboptimal solutions to $(\boldsymbol{\mathcal{P}_1})$.

A. Problem Transformation

In order to convert the long-term constraint (18) into per-slot constraints, we create a virtual energy-excess queue denoted as $Z(t)$, and the backlog is updated as follows:

$$Z(t+1) = \max\{Z(t) - E_{th}, 0\} + E_R(t). \tag{25}$$

where $Z(t)$ represents the variation in energy consumption at the RSU with respect to the threshold E_{th} in time slot t, and $Z(1) = 0$. Generally, the larger the value of $Z(t)$, the further the deviation.

Given the backlog $Z(t)$, the corresponding Lyapunov function with regards to (w.r.t.) $Z(t)$ can be defined as:

$$L(Z(t)) = \frac{Z^2(t)}{2}. \tag{26}$$

Lemma 1: Given four non-negative real numbers A, B, C and m, which satisfy $C = \max\{B - m, 0\} + A$, we have $C^2 \le A^2 + B^2 + m^2 - 2B(m - A)$.

Please refer to [31] for the details, we omit the proof here. Then, the Lyapunov drift $\triangle(Z(t))$ that indicates the increment of $L(Z(t))$ between two consecutive states is given as:

$$\triangle(Z(t)) = L(Z(t+1)) - L(Z(t)). \tag{27}$$

Based on Lemma 1, we have

$$\begin{aligned} Z^2(t+1) &= \{\max\{Z(t) - E_{th}, 0\} + E_R(t)\}^2 \\ &\le Z^2(t) + E_{th}^2 + E_R^2(t) + 2Z(t)(E_R(t) - E_{th}) \\ &\le Z^2(t) + E_{th}^2 + E_R^2(t) + 2Z(t)E_R(t). \end{aligned} \tag{28}$$

$$\begin{aligned}\triangle(Z(t)) &= \frac{1}{2}\Big(Z^2(t+1) - Z^2(t)\Big)\\ &\le \frac{1}{2}\Big(E_{th}^2 + E_R^2(t) + 2Z(t)E_R(t)\Big)\\ &\overset{\ddagger}{\le} H(t) + Z(t)E_R(t), \end{aligned} \tag{29}$$

where $H(t) = \frac{1}{2}(E_{th}^2 + E_R^2(t)) \le \frac{1}{2}(E_{th}^2 + E_{R,max}^2(t)) \triangleq H_{max}(t)$. $E_{R,max}(t)$ is the energy consumption constraint for RSU in time slot t which is independent of the size of task data and the processing frequency of RSU. Thus, the upper bound of $\triangle(Z(t))$ only depends upon the information in the current time slot. As such, we can transform the problem $(\boldsymbol{\mathcal{P}_2})$ into a series of per-slot deterministic optimization subproblems by incorporating the drift-plus-penalty term in each time slot. Furthermore, the subproblem within each time slot aims to minimize the weighted sum of the response latency for all the tasks, and energy consumption. In particular, the subproblem can be defined as:

$$\begin{aligned}(\boldsymbol{\mathcal{P}_2}) \quad & \min_{\boldsymbol{x}\boldsymbol{f}_E,\forall t} \quad Z(t)E_R(t) + V\tau(t)\\ s.t. \quad & (19)-(24), \end{aligned}$$

where $\tau(t) = \sum_{n=1}^{N}\tau_n(t)$, and the constraint (18) in $\boldsymbol{\mathcal{P}_1}$ has been incorporated into problem $\boldsymbol{\mathcal{P}_2}$ using the drift-plus-penalty term [25], and $V(>0)$ is the weight of response latency, used for adjusting the tradeoff between energy consumption control and response time minimization.

Theorem 1: The optimality gap that denotes the difference between the solutions of $\boldsymbol{\mathcal{P}_1}$ and $\boldsymbol{\mathcal{P}_2}$ is a function w.r.t. V, denoted by $F(V)$. Furthermore, if problem $\boldsymbol{\mathcal{P}_2}$ can be solved optimally in each time slot, $F(V)$ can be bounded by $O(1/V)$.

Please refer to [25] for the details, we omit the proof here. As a consequence, our original aim has shifted from the long-term optimization problem $\boldsymbol{\mathcal{P}_1}$ to the per-slot optimization problem $\boldsymbol{\mathcal{P}_2}$ via the problem transformation. According to this theorem, the optimal solution to $\boldsymbol{\mathcal{P}_1}$ can be obtained asymptotically provided that the problem $\boldsymbol{\mathcal{P}_2}$ per time slot is solved optimally.

B. Per-Slot Task Offloading and Resource Allocation

It can be seen that $\boldsymbol{\mathcal{P}_2}$ is a per-slot optimization problem, as the task offloading and resource allocation decisions for the current time slot t depend on information, specifically the backlog of task data, from the preceding time slot $t-1$. Nevertheless, this inherent sequential dependency does not hinder our ability to optimize $\boldsymbol{\mathcal{P}_2}$ in a slot-by-slot way. Notably, the variables $Z(t)$, $E_R(t)$ and $\tau(t)$ in $\boldsymbol{\mathcal{P}_2}$ are functions of $\boldsymbol{x}(t)$ and $\boldsymbol{f}_E(t)$. Let $\mathcal{G}_t(\boldsymbol{x}(t),\boldsymbol{f}_E(t)) = Z(\boldsymbol{x}(t-1),\boldsymbol{f}_E(t-1))E_R(\boldsymbol{x}(t),\boldsymbol{f}_E(t)) + V\tau(\boldsymbol{x}(t),\boldsymbol{f}_E(t))$, and then $\boldsymbol{\mathcal{P}_2}$ is actually to minimize $\mathcal{G}_t(\boldsymbol{x}(t),\boldsymbol{f}_E(t))$, while satisfying the above constraints (19)–(24).

Lemma 2: Given $\boldsymbol{f}_E(t)$, the optimization function in problem $\boldsymbol{\mathcal{P}_2}$, i.e., $\mathcal{G}_t(\boldsymbol{x}(t),\boldsymbol{f}_E(t))$, is non-decreasing, w.r.t. $\boldsymbol{x}(t)$.

Proof: Please refer to the Appendix A. ∎

From this lemma, we know that the most direct way to optimize problem $\boldsymbol{\mathcal{P}_2}$ is to reduce the amount of computation offloaded to RSU from the viewpoint of task vehicles. The inequality constraints (19) and (20) on the other hand restrict the amount of computation for local execution at the task vehicle side. From the above two inequality constraints, we can obtain two kinds of maximal amounts of computation allowed for local execution. Combining them, we can further acquire the maximal amount of computation allowed for local execution for task vehicle n, and thus the offloading decision variable, denoted by $x_n^*(t)$ can be given as:

$$x_n^*(t) = \max\left\{1 - \frac{f_n d_n(t)}{C_n(t)}, 1 - \frac{E_n(t)}{\vartheta\varsigma C_n(t) f_n^2}\right\}, \quad \forall n \in \mathcal{N}, \tag{30}$$

which means that vehicle n should offload at least $x_n^*(t)C_n(t)$ computation to RSU for execution, so as to satisfy the latency and energy constraints for local computing at n.

Lemma 3: Given $\boldsymbol{x}(t)$, the optimization function $\mathcal{G}_t(\boldsymbol{x}(t),\boldsymbol{f}_E(t))$ is convex, w.r.t. $\boldsymbol{f}_E(t)$.

Proof: Please refer to the Appendix B. ∎

In addition, the constraints (19)–(24) are all linear w.r.t. $\boldsymbol{f}_E(t)$, given the offloading decision $\boldsymbol{x}(t)$. Thus, the problem $\boldsymbol{\mathcal{P}_2}$ is a convex problem w.r.t. $\boldsymbol{f}_E(t)$. Accordingly, a dual decomposition method can be used for resource allocation. Given $\boldsymbol{x}(t)$, we construct the Lagrangian function for $\boldsymbol{\mathcal{P}_2}$ as:

$$\begin{aligned}\mathcal{L}(\boldsymbol{f}, v) = &\sum_{n=1}^{N}\mathbb{A}_n\big(f_E^n(t)\big)^2 + \frac{\mathbb{B}_n}{f_E^n(t)} + \mathbb{C}_n\\ &+ v\left(\sum_{n=1}^{N} f_E^n(t) - f_E^{max}(t)\right), \end{aligned} \tag{31}$$

where $\mathbb{A}_n = Z(\boldsymbol{x}(t-1),\boldsymbol{f}_E(t-1))\vartheta_E\varsigma_E D_n(t)$, $\mathbb{B}_n = VD_n(t)(1+\eta_n(t)/\theta_n(t))$, $\mathbb{C}_n = V(\tau_n^{trs}(t) + l_n^q(t) + \tau_n^{fb}(t))$, $\boldsymbol{f} = \boldsymbol{f}_E(t)$, and $v \ge 0$.

Based on the dual theory, the optimal solution to problem $\boldsymbol{\mathcal{P}_2}$ can be expressed as:

$$\begin{aligned}(\boldsymbol{f}^*, v^*) &= \arg\left\{\max_{v}\min_{\boldsymbol{f}}\mathcal{L}(\boldsymbol{f}, v)\right\}\\ &= \arg\left\{\max_{v}\min_{\boldsymbol{f}}\sum_{n=1}^{N}\mathbb{A}_n\big(f_E^n(t)\big)^2 + \frac{\mathbb{B}_n}{f_E^n(t)} + \mathbb{C}_n\right.\\ &\qquad \left. + v\left(\sum_{n=1}^{N} f_E^n(t) - f_E^{max}(t)\right)\right\}. \end{aligned} \tag{32}$$

We can apply the Lagrangian multiplier method to optimize $(\boldsymbol{f}, v)$ in an iterative way. In particular, the resource allocation profile $\boldsymbol{f}$ can be updated by a gradient descent method, while the Lagrange multiplier v is updated by a gradient ascent method, i.e.,

$$\boldsymbol{f}_{new} = \boldsymbol{f}_{old} - \xi_1\frac{\partial\mathcal{L}(\boldsymbol{f}, v)}{\partial\boldsymbol{f}}\bigg|_{f=f_{old}}, \tag{33}$$

$$v_{new} = v_{old} + \xi_2\frac{\partial\mathcal{L}(\boldsymbol{f}, v)}{\partial v}\bigg|_{v=v_{old}}, \tag{34}$$

where ξ_1 and ξ_2 are the learning rates. The corresponding algorithm is shown in Alg. 1.

Improvement: Let's denote the inner minimization as $g(v)$, which is expressed as follows:

Algorithm 1: Per-Slot Iterative Algorithm for Resource Allocation (IARA)

Input: Required parameters
Output: Resource allocation profile $\boldsymbol{f}_E(t)$
1 Acquire $\boldsymbol{J}(t)$, $\boldsymbol{Q}(t)$, $\boldsymbol{D}(t)$, $\boldsymbol{x}(t)$ and $Z(t)$;
2 Initialize the two variables $\boldsymbol{f}$ and v, respectively;
3 Set ξ_1 and ξ_2;
4 **repeat**
5 Update $\boldsymbol{f}$ based on Eq. (33);
6 Update v based on Eq. (34);
7 **until** stopping criterion satisfied;
8 Return $\boldsymbol{f}$;

$$
\begin{aligned}
g(v) &= \min_{\boldsymbol{f}} \sum_{n=1}^{N} \mathbb{A}_n \left(f_E^n(t)\right)^2 + \frac{\mathbb{B}_n}{f_E^n(t)} + \mathbb{C}_n \\
&\quad + v\left(\sum_{n=1}^{N} f_E^n(t) - f_E^{max}(t)\right) \\
&= \sum_{n=1}^{N} \min_{f_E^n(t)} \mathbb{A}_n \left(f_E^n(t)\right)^2 + \frac{\mathbb{B}_n}{f_E^n(t)} + v f_E^n(t) + \mathbb{C}_n - v f_E^{max}(t),
\end{aligned}
$$

where $g(v)$ can be decomposed into N convex subproblems. Owing to the convexity, the N subproblems can be solved in a distributed manner. Consequently, the update of the resource allocation profile, represented by Eq. (33), can be decomposed into N parallel operations, resulting in enhanced algorithm efficiency.

C. Algorithm Design

With the assistance of Alg. 1, we can efficiently determine the task offloading and resource allocation in each time slot by jointly optimizing the two decision variables $\boldsymbol{x}(t)$ and $\boldsymbol{f}_E(t)$. Then, we propose an online optimization algorithm for task offloading and resource allocation (TORA) that spans the entire optimization period, as illustrated in Alg. 2. Generally, TORA seeks the optimal solution to $\boldsymbol{\mathcal{P}_2}$ across various time slots. Specifically, the optimization problem is decomposed into T subproblems, each correspondingly addressed within a distinct time slot (lines 3–10). By solving problem $\boldsymbol{\mathcal{P}_2}$ (line 6), the algorithm obtains approximately optimal solutions, $\boldsymbol{x}^*(t)$ and $\boldsymbol{f}^*{}_E(t)$, for the given time slot t. TORA then calculates the total response latency for all tasks in each time slot (line 7) and accumulates the overall response latency across time slots (line 8). Finally, the mean response latency for all tasks (i.e., Avg) is returned.

In addition, the following procedure as denoted in Alg. 3 is adopted to obtain the optimal offloading decision and resource allocation profiles for each time slot (i.e., line 6 in Alg. 2). According to Lemma 2, minimizing $\mathcal{G}_t(\boldsymbol{x}(t), \boldsymbol{f}_E(t))$, from the perspective of task vehicles, requires reducing the amount of offloaded computation as much as possible, which inspires us to utilize the greedy heuristic for offloading decision optimization. For instance, pTORA calculates the maximal amount of computation allowed for local execution for each task vehicle n in time slot t, and then determines the offloading decision variable profile based on Eq. (30) (lines 1–4). Then, based on Lemma 3, the algorithm optimizes the resource allocation for the offloaded tasks at RSU by calling Alg. 1.

Algorithm 2: Optimization Algorithm for Task Offloading and Resource Allocation (TORA)

Input: Required parameters such as T, N, B, V and etc.
Output: Optimum solution to $\boldsymbol{\mathcal{P}_2}$
1 Construct initial queues $J_n(0)$, $Q_n(0)$, $D_n(0)$, $Z(0)$, $\forall n \in \mathcal{N}$;
2 $L = 0$;
3 **for** $t = 1$ to T **do**
4 Observe $C_n(t)$, $d_n(t)$, $\lambda_n(t)$, $\tilde{\mu}(t)$, $\eta_n(t)$, $\theta_n(t)$, $\forall n \in \mathcal{N}$;
5 Set E_{th}, $E_n(t)$, $E_{R,max}(t)$ and ρ_t;
6 Obtain the best solution $(\boldsymbol{x}^*(t), \boldsymbol{f}^*{}_E(t))$ to $\boldsymbol{\mathcal{P}_2}$, i.e., $(\boldsymbol{x}^*(t), \boldsymbol{f}^*{}_E(t)) = \arg\left\{\min_{\boldsymbol{x}(t), \boldsymbol{f}_E(t)} Z(t)E_R(t) + V\tau(t)\right\}$;
7 Calculate $L_t = \sum_{n=1}^{N} \tau_n(t)$, given $\boldsymbol{x}^*(t)$ and $\boldsymbol{f}^*{}_E(t)$;
8 $L = L + L_t$;
9 Update queues $J_n(t)$, $Q_n(t)$, $D_n(t)$, $Z(t)$, $\forall n \in \mathcal{N}$;
10 **end**
11 $Avg = L/T$;
12 return Avg;

Algorithm 3: Per-Slot Optimization for Task Offloading and Resource Allocation (pTORA)

Input: Required parameters for per-slot optimization
Output: $(\boldsymbol{x}^*(t), \boldsymbol{f}^*{}_E(t))$
1 **for** each $n \in \mathcal{N}$ **do**
2 Acquire $x_n^*(t)$ based on Eq. (30);
3 Calculate $\mathbb{A}_n$, $\mathbb{B}_n$, $\mathbb{C}_n$;
4 **end**
5 Construct the offloading decision $\boldsymbol{x}^*(t)$ for time slot t;
6 Acquire $\boldsymbol{f}^*{}_E(t)$ by calling Alg. 1;
7 return $(\boldsymbol{x}^*(t), \boldsymbol{f}^*{}_E(t))$;

D. Complexity Analysis

It shall be noted that the proposed algorithms IARA and pTORA both serve TORA. The performance of TORA is greatly affected by IARA and pTORA. Given the offloading decision profile in each time slot, IARA iteratively optimizes the Lagrange multiplier and the processing frequencies allocated to each task. It searches the solution by one loop (lines 4–7), which takes time of $O(K)$, where K is the total number of steps for the loop. In addition, pTORA requires the time of $O(N)$ to obtain $\boldsymbol{x}^*(t)$ in time slot t. Then, it requires the time of $O(K)$ to obtain $\boldsymbol{f}^*{}_E(t)$ using IARA. Thus, pTORA generally requires a total time of $O(N + K)$. On another hand, TORA optimizes the objective function along time slots, and the time complexity can be expressed as $O(T) \cdot O(N + K) = O(NT + TK)$. As a consequence, we can obtain the nearly optimal solution in polynomial time. Furthermore, extensive simulation is required to validate the efficiency and effectiveness of the algorithm.

TABLE II
PARAMETER SETTINGS

Parameter	Value	Parameter	Value
T	[0, 1000]	N	[20,200]
$C_n(t)$	[1, 51]	$\lambda_n(t)$	[10, 30]
$\tilde{\mu}(t)$	[30, 50]	$\eta_n(t)$	[20, 30]
$E_n(t)$	[2000, 5000]	λ_k^t	[100, 200]
$\theta_n(t)$	[3, 6]	$f_e^{max}(t)$	[100, 200]
V	[100, 1000]	$\vartheta_{E\varsigma E}$	1e-5
ρ_t	(0,1)	E_{th}	[1000, 20000]

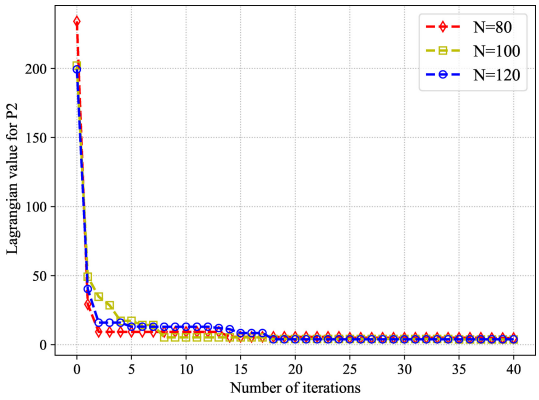


Fig. 2. The convergence ability of the algorithm IARA.

VII. PERFORMANCE EVALUATION

A. Experimental Settings

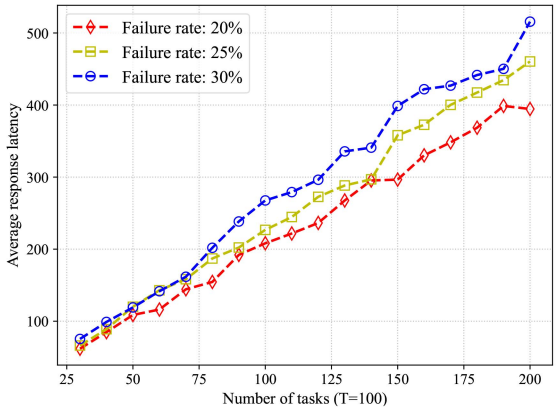
In this section, we conduct comprehensive simulations to verify the effectiveness and efficiency of our strategy. Specifically, the default values for some key parameters involved in the simulation are listed in Table II.

B. Convergence Performance

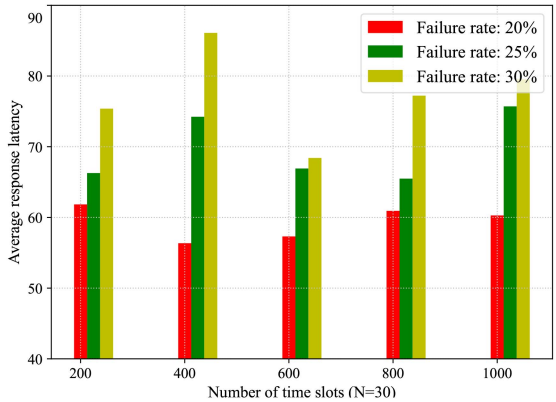
We commence the evaluation by assessing the convergence capability of our proposed algorithm, named IARA, which is a crucial factor in determining the efficiency of our strategy. The simulation results are visualized in Fig. 2, with the x-coordinate representing the number of iterations and the y-coordinate representing the Lagrangian values based on Eq. (31). Three different scenarios are examined in this experiment, where the number of tasks is set to 80, 100, and 120, respectively. The obtained results demonstrate that the algorithm exhibits rapid convergence within the first few iterations. Subsequently, the optimal values stabilize and fluctuate within a constrained range. Notably, after approximately 17 iterations, the optimal values no longer decrease, irrespective of the number of tasks. These simulation results provide compelling evidence that our proposed algorithm can achieve rapid convergence in the specific application scenario under consideration.

C. Impact of Failure Rates

We examine the impact of failure rates on the performance of our strategy as follows. In Fig. 3 (a), we present a performance comparison in terms of average response latency as the number of tasks increases, with the simulation set to



(a) Latency with different tasks

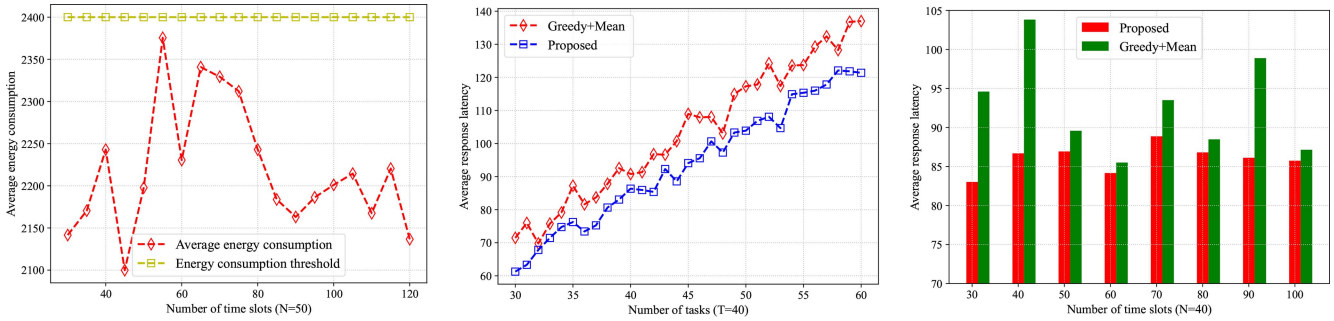


(b) Latency along time slots

Fig. 3. Performance evaluation different failure rates.

100 time slots. Three different kinds of task failure rates are studied, which are 20%, 25% and 30%, respectively. As indicated in the figure, an increase in task failure rates corresponds to an increase in average response latency. Moreover, average response latency increases with an increase in the number of tasks, regardless of the failure rates. This observation can be attributed to limited computing resources when compared to a cloud center. Consequently, queueing time rapidly escalates with a growing number of tasks, leading to increased average response latency for all tasks.

Fig. 3 (b) illustrates a performance comparison in terms of average response latency across different time slots, with the simulation configured for 30 tasks. Similar to the previous simulation, three kinds of task failure rates, i.e., 20%, 25% and 30%, are investigated. Two key observations are made: A higher task failure rate always incurs higher average response latency, no matter how the number of time slots varies. Also, there are no certain patterns for the average response latency as the number of time slots varies. The following possible reasons can result in this observation. For instance, the tasks are generated randomly in each time slot. The connection between two consecutive time slots is the task data that needs to be retransmitted (i.e., $J_n(t+1)$). In this context, if the amount of task data that needs to be retransmitted is large, the average response latency in the next time slot could be larger than that in the current time slot. Similarly, if the amount of



(a) Energy consumption comparison (b) Approaches with different tasks (c) Approaches with different task sizes

Fig. 4. Performance evaluation on average response latency for different approaches.

task data that needs to be retransmitted is very small or even negligible, the average response latency could be larger in the current time slot than that in the next time slot.

D. Violations of Average Energy Consumption

In the subsequent analysis, we assess whether there are any violations of the average energy consumption constraint after converting the time-slot-spanned optimization problem into per-slot optimization subproblems. The experimental results are depicted in Fig. 4 (a). In the simulation, the global energy consumption constraint (i.e., E_{th} in the constraint inequality (18)) is set to 2400. Several conclusions can be drawn as follows. First, regardless of the variation in the number of time slots, there are no violations of the energy consumption constraint. This finding validates the feasibility of converting the original optimization problem into a series of subproblems. It indicates that the per-slot optimization approach is effective in maintaining energy consumption within the specified constraints. Second, there are no deterministic relationships observed between the energy consumption in different time slots. The energy consumption in each time slot is influenced by the computational tasks assigned to that specific time slot. As discussed earlier, there are no deterministic relationships governing the computation tasks in different time slots. Consequently, the absence of deterministic relationships between energy consumption in different time slots is expected.

In summary, the simulation results not only confirm and validate our theoretical analysis but also provide support for the feasibility and effectiveness of the proposed approach. The absence of energy consumption violations and the lack of deterministic relationships between energy consumption in different time slots reinforce the practicality and robustness of our strategy. These findings contribute to a deeper understanding of the performance and behavior of our approach in real-world scenarios.

E. Performance Comparison

As for the performance comparison for different approaches, we adopt one approach called "Greedy+Mean" as the baseline. This approach combines the use of the greedy rule and the mean strategy. Specifically, the greedy rule and the mean strategy are employed to determine the amount of computation to be offloaded and how computing resources are allocated at the edge, respectively. As mentioned earlier, the maximal amount of computation allowed for local execution for task vehicles can be determined. Thus, the greedy rule aims to minimize the offloaded computation to the edge for each task, as defined in Eq. (30). Upon the arrival of tasks at the edge server, the edge first determines the number of tasks and then allocates computing resources to the tasks equally. The corresponding simulation results are depicted in Fig. 3.

Fig. 4 (b) provides a performance comparison in terms of the average response latency with varying numbers of tasks, using a simulation with 40 time slots. This figure clearly demonstrates that our approach obviously outperforms the baseline, regardless of the number of tasks. Additionally, it's notable that the average response latency increases for both approaches as the number of tasks increases. This rise in response latency is attributed to the limited computing resources at the edge. In other words, the average queueing time for each task can grow significantly with an increasing number of tasks. Given that queueing time is a significant component of response latency, the average response latency also increases as the number of tasks rises.

Fig. 4 (c) provides a performance comparison in terms of the average response latency across varying numbers of time slots, using a simulation with 40 tasks. This figure consistently shows that our approach is superior to the baseline, regardless of the number of time slots. At times, our approach significantly outperforms the baseline (e.g., with 40 or 90 time slots), while in other instances, it moderately surpasses the baseline (e.g., with 60 or 100 time slots). However, it's important to note that there are no deterministic relationships for the response latency in different time slots. This variability is due to reasons similar to those observed when investigating the effects of task failure rates on response latency. Task data is generated randomly in each time slot, and the backlog of computation from the previous time slot can influence the response latency in the current time slot. These factors can lead to fluctuations in computation between two consecutive time slots. Sometimes, the amount of computation in the previous time slot is greater than that in the current time slot, while other times it is not. Consequently, the average response latency exhibits this same pattern of variability.

VIII. Conclusion

VEC has gained significant attention in recent years, owing to its substantial advantages in satisfying the stringent latency requirements of vehicular applications and services. In this paper, we consider the joint optimization problem of task offloading and resource allocation, and design a cost-efficient and failure-resisted task offloading strategy, which distinguishes our work from existing research in VEC systems. Our primary objective is to minimize the average response latency for all the tasks. To tackle this problem, we decomposed the original problem into a sequence of subproblems, eliminating the long-term constraints. Furthermore, we introduced several algorithms to tackle these subproblems. Through extensive simulations, we have validated the feasibility, effectiveness, and efficiency of our strategy. For future work, we intend to explore more general scenarios, such as those involving multiple edge servers to support task offloading.

Appendix A
Proof of the Lemma 2

For simplicity, we rewrite $\mathcal{G}_t(\boldsymbol{x}(t),\boldsymbol{f}_E(t))$ as $\mathcal{G}_t(\boldsymbol{x},\boldsymbol{f}_E) = Z\cdot E_R(\boldsymbol{x},\boldsymbol{f}_E) + V\tau(\boldsymbol{x},\boldsymbol{f}_E)$, since $Z(t)$ only depends upon the information from the previous time slot $t-1$. As a result, $Z(t)$ is actually independent of $\boldsymbol{x}(t)$ and $\boldsymbol{f}_E(t)$, and we treat $Z(t)$ as a constant in slot t (i.e., Z) in the above equation. Therefore, we have

$$\begin{aligned}E_R(\boldsymbol{x},\boldsymbol{f}_E) &= \vartheta_E\varsigma_E\sum_{n=1}^{N} D_n(t)\left(f_E^n(t)\right)^2\\ &= \vartheta_E\varsigma_E\left[D_1(t)\left(f_E^1(t)\right)^2+\cdots+D_N(t)\left(f_E^N(t)\right)^2\right]\\ &= \vartheta_E\varsigma_E\Big[\min\{Q_1(t)+x_1(t)C_1(t),S_1(t)\}\left(f_E^1(t)\right)^2\\ &\quad+\cdots+\min\{Q_N(t)+x_N(t)C_N(t),S_N(t)\}\left(f_E^N(t)\right)^2\Big].\end{aligned}$$

Let $H(x_n(t)) = \min\{Q_n(t)+x_n(t)C_n(t), S_n(t)\}(f_E^n(t))^2, \forall n\in\mathcal{N}$. Obviously, $H(x_n(t))$ is a segmented function w.r.t. $x_n(t)$, i.e.,

$$H(x_n(t)) = \begin{cases} S_n(t)\left(f_E^n(t)\right)^2, & x_n(t)\geq \frac{S_n(t)-Q_n(t)}{C_n(t)}\\ (Q_n(t)+x_n(t)C_n(t))*\left(f_E^n(t)\right)^2, & \text{otherwise}\end{cases}$$

Since $S_n(t)(f_E^n(t))^2$ is independent of $x_n(t)$, $H(x_n(t))$ is non-decreasing w.r.t. $x_n(t)$. $\sum_{n=1}^{N} H(x_n(t))$ is also non-decreasing, and thus $E_R(\boldsymbol{x},\boldsymbol{f}_E)$ is non-decreasing w.r.t. $\boldsymbol{x}$, given $\boldsymbol{f}_E$. On the other hand, the other term $V\tau(\boldsymbol{x},\boldsymbol{f}_E)$ in $\mathcal{G}_t(\boldsymbol{x},\boldsymbol{f}_E)$ is also non-decreasing w.r.t. $\boldsymbol{x}$, given $\boldsymbol{f}_E$, according to Eqs. (13)–(15). Accordingly, given $\boldsymbol{f}_E(t)$, the optimization function in problem $\boldsymbol{\mathcal{P}_2}$ is non-increasing w.r.t. $\boldsymbol{x}(t)$, and the proof is completed.

Appendix B
Proof of the Lemma 3

Given $\boldsymbol{x}(t)$, the optimization function can be expanded as:

$$\begin{aligned}\mathcal{G}_t(\boldsymbol{x}(t),\boldsymbol{f}_E(t)) &= Z(\boldsymbol{x}(t-1),\boldsymbol{f}_E(t-1))E_R(\boldsymbol{x}(t),\boldsymbol{f}_E(t))\\ &\quad+V\tau(\boldsymbol{x}(t),\boldsymbol{f}_E(t))\\ &= \sum_{n=1}^{N} Z(\boldsymbol{x}(t-1),\boldsymbol{f}_E(t-1))\vartheta_E\varsigma_E D_n(t)\left(f_E^n(t)\right)^2\\ &\quad+V\left(\tau_n^{trs}(t)+l_n^q(t)+\tau_n^{fb}(t)+\left(1+\frac{\eta_n(t)}{\theta_n(t)}\right)\frac{D_n(t)}{f_E^n(t)}\right)\\ &= \sum_{n=1}^{N}\mathbb{A}_n\left(f_E^n(t)\right)^2+\frac{\mathbb{B}_n}{f_E^n(t)}+\mathbb{C}_n,\end{aligned}$$

where $\mathbb{A}_n \triangleq Z(\boldsymbol{x}(t-1),\boldsymbol{f}_E(t-1))\vartheta_E\varsigma_E D_n(t)$, $\mathbb{B}_n \triangleq VD_n(t)(1+\eta_n(t)/\theta_n(t))$, and $\mathbb{C}_n \triangleq V(\tau_n^{trs}(t)+l_n^q(t)+\tau_n^{fb}(t))$. Obviously, $\mathbb{A}_n$, $\mathbb{B}_n$, and $\mathbb{C}_n$ are all independent of $f_E^n(t)$, and thus can be regarded as constants, under the premise that $\boldsymbol{x}(t)$ is given.

Let $\mathbb{H}_n \triangleq \mathbb{A}_n(f_E^n(t))^2+\mathbb{B}_n/f_E^n(t)+\mathbb{C}_n, \forall n\in\mathcal{N}$ and we prove that $\mathbb{H}_n$ is convex w.r.t. $\boldsymbol{f}_E(t)$ as follows. The partial derivatives by differentiating $\mathbb{H}_n$ w.r.t. $\boldsymbol{f}_E(t)$ can be obtained as:

$$\frac{\partial\mathbb{H}_n}{\partial f_E^i(t)} = \begin{cases}2\mathbb{A}_n f_E^n(t)-\frac{\mathbb{B}_n}{\left(f_E^n(t)\right)^2}, & i=n,\\ 0, & i\neq n.\end{cases}$$

Then, the second partial derivatives are given as:

$$\frac{\partial^2\mathbb{H}_n}{\partial f_E^i(t)\partial f_E^j(t)} = \begin{cases}2\mathbb{A}_n+\frac{2\mathbb{B}_n}{\left(f_E^n(t)\right)^3}, & i=n \text{ and } j=n,\\ 0, & \text{otherwise.}\end{cases}$$

The Hessian matrix of $\mathbb{H}_n$ can be expressed as:

$$\mathcal{M}(\mathbb{H}_n) = \begin{bmatrix}0 & \cdots & \cdots & \cdots & 0\\ 0 & \cdots & \cdots & \cdots & 0\\ 0 & \cdots & \cdots & \cdots & 0\\ 0 & \cdots & 2\mathbb{A}_n+\frac{2\mathbb{B}_n}{\left(f_E^n(t)\right)^3} & \cdots & 0\\ \cdots & \cdots & \cdots & \cdots & \cdots\\ 0 & \cdots & \cdots & \cdots & 0\end{bmatrix}$$

It is noticeable that $X^T\mathcal{M}X \geq 0$ always holds for an arbitrary nonzero vector X, so $\mathcal{M}(\mathbb{H}_n)$ is positive semi-definite. Accordingly, $\mathbb{H}_n$ is convex w.r.t. $\boldsymbol{f}_E(t)$, $\forall n\in\mathcal{N}$. Owing to the additivity attribute of convex function, $\sum_{n=1}^{N}\mathbb{H}_n$ is also convex w.r.t. $\boldsymbol{f}_E(t)$. Thus, the proof is completed.

References

[1] R. W. Liu et al., "Intelligent data-driven vessel trajectory prediction in marine transportation cyber-physical system," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. & Commun. (GreenCom) IEEE Cyber, Phys. & Soc. Comput. (CPSCom) IEEE Smart Data (SmartData) IEEE Congr. Cybermat.*, 2021, pp. 314–321.

[2] Z. Guo, D. Meng, C. Chakraborty, X.-R. Fan, A. Bhardwaj, and K. Yu, "Autonomous behavioral decision for vehicular agents based on cyber-physical social intelligence," *IEEE Trans. Comput. Social Syst.*, vol. 10, no. 4, pp. 2111–2122, Aug. 2023.

[3] J. Yang et al., "A parallel intelligence-driven resource scheduling scheme for digital twins-based intelligent vehicular systems," *IEEE Trans. Intell. Veh.*, vol. 8, no. 4, pp. 2770–2785, Apr. 2023.

[4] Z. Zhang, N. Wang, H. Wu, C. Tang, and R. Li, "MR-DRO: A fast and efficient task offloading algorithm in heterogeneous edge/cloud computing environments," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 3165–3178, Feb. 2023.

[5] A. Bechihi, E. Panteley, P. Duhamel, and A. Bouttier, "A resource allocation algorithm for formation control of connected vehicles," *IEEE Control. Syst. Lett.*, vol. 7, pp. 307–312, 2023.

[6] J. Liang, J. Zhang, V. C. M. Leung, and X. Wu, "Distributed information exchange with low latency for decision making in vehicular fog computing," *IEEE Internet Things J.*, vol. 9, no. 19, pp. 18166–18181, Oct. 2022.

[7] N. Zhao, H. Wu, F. R. Yu, L. Wang, W. Zhang, and V. C. M. Leung, "Deep-reinforcement-learning-based latency minimization in edge intelligence over vehicular networks," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1300–1312, Jan. 2022.

[8] B. Yang, F. Tan, and Y. Dai, "Performance evaluation of cloud service considering fault recovery," *J. Supercomput.*, vol. 65, no. 1, pp. 426–444, 2013.

[9] Y. Hui et al., "Secure and personalized edge computing services in 6G heterogeneous vehicular networks," *IEEE Internet Things J.*, vol. 9, no. 8, pp. 5920–5931, Apr. 2022.
[10] C. Tang and H. Wu, "Joint optimization of task caching and computation offloading in vehicular edge computing," *Peer-to-Peer Netw. Appl.*, vol. 15, no. 2, pp. 854–869, 2022.
[11] Y. Lin, Y. Zhang, J. Li, F. Shu, and C. Li, "Popularity-aware online task offloading for heterogeneous vehicular edge computing using contextual clustering of bandits," *IEEE Internet Things J.*, vol. 9, no. 7, pp. 5422–5433, Apr. 2022.
[12] S. Liu, Q. Yang, S. Zhang, T. Wang, and N. N. Xiong, "MIDP: An MDP-based intelligent big data processing scheme for vehicular edge computing," *J. Parallel Distrib. Comput.*, vol. 167, pp. 1–17, Sep. 2022.
[13] S. Wang, J. Li, G. Wu, H. Chen, and S. Sun, "Joint optimization of task offloading and resource allocation based on differential privacy in vehicular edge computing," *IEEE Trans. Comput. Social Syst.*, vol. 9, no. 1, pp. 109–119, Feb. 2022.
[14] C. Tang, W. Chen, C. Zhu, Q. Li, and H. Chen, "When cache meets vehicular edge computing: Architecture, key issues, and challenges," *IEEE Wireless Commun.*, vol. 29, no. 4, pp. 56–62, Aug. 2022.
[15] W. Fan et al., "Joint task offloading and resource allocation for vehicular edge computing based on V2I and V2V modes," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 4, pp. 4277–4292, Apr. 2023.
[16] M. D. Hossain et al., "Dynamic task offloading for cloud-assisted vehicular edge computing networks: A non-cooperative game theoretic approach," *Sensors*, vol. 22, no. 10, p. 3678, 2022.
[17] X. Huang, L. He, X. Chen, L. Wang, and F. Li, "Revenue and energy efficiency-driven delay-constrained computing task offloading and resource allocation in a vehicular edge computing network: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 9, no. 11, pp. 8852–8868, Jun. 2022.
[18] E. Karimi, Y. P. Chen, and B. Akbari, "Task offloading in vehicular edge computing networks via deep reinforcement learning," *Comput. Commun.*, vol. 189, pp. 193–204, May 2022.
[19] B. Lin, K. Lin, C. Lin, Y. Lu, Z. Huang, and X. Chen, "Computation offloading strategy based on deep reinforcement learning for connected and autonomous vehicle in vehicular edge computing," *J. Cloud Comput.*, vol. 10, no. 1, p. 33, 2021.
[20] W. Zhan, C. Luo, J. Wang, G. Min, and H. Duan, "Deep reinforcement learning-based computation offloading in vehicular edge computing," in *Proc. IEEE Glob. Commun. Conf.*, Waikoloa, HI, USA, 2019, pp. 1–6.
[21] R. Danehchin, "Enhancing fault tolerance in vehicular ad-hoc networks using artificial bee colony algorithm-based spanning trees," *Int. J. Syst. Assuran. Eng. Manag.*, vol. 13, no. 4, pp. 1722–1732, 2022.
[22] A. Javed, A. Malhi, and K. Främling, "Edge computing-based fault-tolerant framework: A case study on vehicular networks," in *Proc. 16th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, 2020, pp. 1541–1548.
[23] R. Florin, A. G. Zadeh, P. Ghazizadeh, and S. Olariu, "Towards approximating the mean time to failure in vehicular clouds," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 7, pp. 2045–2054, Jul. 2018.
[24] C. Tang, S. Xia, Q. Li, W. Chen, and W. Fang, "Resource pooling in vehicular fog computing," *J. Cloud Comput.*, vol. 10, no. 1, p. 19, 2021.
[25] C. Tang, C. Zhu, H. Wu, Q. Li, and J. J. P. C. Rodrigues, "Toward response time minimization considering energy consumption in caching-assisted vehicular edge computing," *IEEE Internet Things J.*, vol. 9, no. 7, pp. 5051–5064, Apr. 2022.
[26] J. Yao and N. Ansari, "Fog resource provisioning in reliability-aware IoT networks," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8262–8269, Oct. 2019.
[27] H. Liao, Y. Mu, Z. Zhou, M. Sun, Z. Wang, and C. Pan, "Blockchain and learning-based secure and intelligent task offloading for vehicular fog computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 7, pp. 4051–4063, Jul. 2021.
[28] C. Tang, C. Zhu, N. Zhang, M. Guizani, and J. J. P. C. Rodrigues, "SDN-assisted mobile edge computing for collaborative computation offloading in Industrial Internet of Things," *IEEE Internet Things J.*, vol. 9, no. 23, pp. 24253–24263, Dec. 2022.
[29] Y. Sun, X. Guo, S. Zhou, Z. Jiang, X. Liu, and Z. Niu, "Learning-based task offloading for vehicular cloud computing systems," in *Proc. IEEE Int. Conf. Commun.*, 2018, pp. 1–7.
[30] Z. Zhou, Y. Guo, Y. He, X. Zhao, and W. M. Bazzi, "Access control and resource allocation for M2M communications in industrial automation," *IEEE Trans. Ind. Informat.*, vol. 15, no. 5, pp. 3093–3103, May 2019.
[31] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Found. Trends® Netw.*, vol. 1, no. 1, pp. 1–144, 2006.

Chaogang Tang (Member, IEEE) received the B.S. degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, and the Ph.D. degree from the School of Information Science and Technology, University of Science and Technology of China, Hefei, China, in 2012. He is currently with the China University of Mining and Technology. His research interests include vehicular edge computing Internet of Things, and big data.

Ge Yan received the bachelor's degree in engineering from Anhui Polytechnic University in 2021. He is currently pursuing the master's degree with the School of Information and Control Engineering, China University of Mining and Technology. His research interests include vehicular edge computing, and Internet of Things.

Huaming Wu (Senior Member, IEEE) received the B.E. and M.S. degrees in electrical engineering from Harbin Institute of Technology, China, in 2009 and 2011, respectively, and the Ph.D. degree (Highest Hons.) in computer science from Freie Universität Berlin, Germany, in 2015. He is currently an Associate Professor with the Center for Applied Mathematics, Tianjin University. His research interests include model-based evaluation, wireless and mobile network systems, mobile cloud computing, and deep learning.

Chunsheng Zhu (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from The University of British Columbia, Canada, in 2016. He is currently an Associate Professor with the College of Big Data and Internet, Shenzhen Technology University, China. His research interests include the Internet of Things, wireless sensor networks, cloud computing, big data, social networks, and security.