# Online distributed offloading of time-sensitive vehicular tasks in edge-cloud systems

Te-Yi Kan, and Konstantinos Psounis *Fellow, IEEE*

***Abstract*—With the integration of several types of sensors, vehicles are able to detect and identify objects in their surrounding, enabling them to provide vital road information to Automated Driving Systems (ADS) or drivers using Advanced Driving Assistance Systems (ADAS) to perform functions such as crash avoidance, crosswalk detection, navigation, etc. Recently, it has been proposed to identify, process, and even display information using machine learning (ML) modules. However, performing ML tasks within vehicles is challenging due to their resource-intensive and delay-sensitive nature. To this end, we investigate an edge-cloud-assisted system and propose a Threshold-based Online Distributed Offloading and Resource Allocation mechanism (TODORA) to reduce task duration by optimizing task offloading decisions and resource allocation. TODORA takes into account communication and other protocol-related overheads, making it applicable to real-world scenarios. Simulation results demonstrate the superiority of TODORA over the periodic optimal solution and state-of-the-art practical schemes. We also show that the computation overhead of TODORA is relatively low as compared to the overall task duration and delay tolerance of tasks, verifying its suitability in practical systems.**

***Index Terms*—Task offloading, resource allocation, edge computing, cloud computing**

## I. Introduction

As vehicle and sensor techniques have advanced, vehicles equipped with diverse sensors, such as LiDAR and cameras, can collect road information around them. This data is leveraged by Advanced Driving Assistance Systems (ADAS) or Automated Driving Systems (ADS) to improve drivers' safety and enhance user experience. Recently, machine learning (ML) modules (e.g., Deep Neural Networks) have emerged as effective methods in information-processing procedures that involve resource-intensive tasks, such as object detection and panoptic segmentation [1], [2]. To perform safety-oriented functions such as crash avoidance and crosswalk detection, it is of great importance to provide and display in-time information to ADAS or ADS. However, due to size constraints and limitations of cooling systems [3], vehicles typically cannot provide sufficient resources to complete all resource-intensive ML tasks within deadlines on their own. Consequently, edge computing systems have emerged as a solution to overcome the difficulties of vehicular ML tasks.

Edge computing system is a promising approach that can offer computing services to mobile devices while guaranteeing lower delay [4], [5]. Compared to conventional cloud computing systems in which a task should go through a long distance to and back from the cloud server, offloading tasks to edge computing systems, which are positioned at or near the network endpoints such as the roadside units (RSUs), can greatly reduce the propagation delay of tasks and thus is more suitable for applications where time is of the essence. With this upside, edge computing is also introduced to alleviate the computation load on vehicles and facilitate vehicular applications, such as path navigation and in-vehicle gaming [6], [7].

However, with increased users connecting and requesting computing services from edge systems, edge servers may still encounter a scarcity of resources. Not only computing resources but also radio resources (i.e., bandwidth) become insufficient as more and more devices desire to utilize edge computing services. With these challenges, there is a pressing need for an effective and efficient joint task offloading and resource allocation algorithm. Vehicular ML tasks, especially, require a well-designed offloading mechanism to adhere to its harsh delay constraints. Failing to meet the requirement of vehicular ML tasks would compromise the accuracy and timeliness of vital safety information, which may, in turn, increase driving risk.

In this paper, we consider a local-edge-cloud computing system where tasks can be executed on vehicles' onboard systems, offloaded to edge servers, or further migrated to the cloud. We further account for a realistic setting where tasks must contend for and share both computation and radio resources, and present a complete mechanism for solving the joint task scheduling and resource allocation problem. This comprehensive setup enables a more sophisticated approach to task scheduling, resulting in more efficient utilization of system resources and better support for time-critical ML tasks, as compared to prior works that focus only on local-edge offloading [8], [9] or edge-cloud task migration [10], [11], and those addressing only computation resource allocation [12], [13] or radio resource allocation [14], [15] in isolation.

While some prior studies have examined local-edge-cloud systems with joint computation and communication resource allocation [16], they typically focus only on execution delay, neglecting the overhead associated with managing such complex systems. In this work, we instead target the total task duration, which accounts for both execution delay and communication overhead from the request/response procedure. Our goal is to shorten the overall task duration to deliver in-time information while meeting task timeline constraints.

Furthermore, unlike prior works that typically focus on either binary offloading [9], [17] or partial offloading [14], [18], our study considers both approaches. In binary offloading,

The authors are with the Dept. of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089, USA (e-mail: teyikan@usc.edu, kpsounis@usc.edu).

tasks cannot be divided and must be executed entirely at a single location. This approach simplifies implementation on practical devices and avoids the additional overhead associated with task splitting and synthesis. On the other hand, partial offloading allows tasks to be split and executed across multiple computing layers, offering more efficient resource utilization.

The joint task scheduling and resource allocation problem is a mixed-integer nonlinear programming (MINLP) problem, which is known to be NP-hard. To tackle this, we first employ a standard solver to provide a periodic and centralized optimal solution, referred to as POPT. We theoretically demonstrate that the time complexity of POPT increases significantly with the system size (e.g., the number of vehicles and RSUs). To overcome this limitation, we propose a heuristic task offloading and resource allocation algorithm called TODORA, designed to reduce task duration in an online and distributed manner. TODORA is a threshold-based heuristic mechanism with $\mathcal{O}(1)$ complexity, making it suitable for implementation in practical systems.

Lastly, extensive simulations are conducted to evaluate the performance of TODORA, using vehicular ML tasks highlighted in the Waymo Open Dataset [19] and the corresponding real-world data histogram. A periodic optimal solution and an existing state-of-the-art online heuristic algorithm are implemented as reference benchmarks. The simulations cover various scenarios, including different vehicle computing capacities and RSU computing capacities, to name a few, and the results show that TODORA can achieve better performance compared to the benchmarks. With the consideration of communication overhead, the task duration is dramatically further reduced over prior work. The main innovations and contributions of this work can be briefly summarized as follows:

- We formulate a joint task offloading and resource allocation problem in a comprehensive local-edge-cloud computing system, considering both computation and radio resources.
- We minimize the overall task duration, accounting for both task execution time and request/response overhead, in contrast to previous studies that account only for execution time [14], [17].
- We consider both binary and partial offloading schemes. In contrast to prior work that usually focuses on only one [9], [18], our approach enables a comparative analysis and accommodates a broader range of application scenarios.
- We provide a periodic optimal solution using a standard solver and theoretically demonstrate its high time complexity, which limits its practical applicability.
- To overcome the high complexity of the optimal solution, we propose a novel, low-complexity offloading mechanism that operates without a centralized controller, unlike prior work [20], [21], and performs online decision-making to avoid the performance loss associated with periodic scheduling [13], [18]. These features enable fast, efficient decisions with minimal communication and computation delays.
- We perform extensive simulations using real-world vehicular ML tasks and data to validate the superiority of our proposed algorithm compared to the periodic optimal solution and the state-of-the-art approaches.

The remainder of this paper is organized as follows: A review of existing work is provided in Sec. II. Sec. III describes the system model and formulates the joint offloading and resource allocation problem under binary offloading, and Sec. IV provides the details of TODORA. Sec. V extends the system model and TODORA to accommodate partial offloading. Sec. VI presents the simulation results and discussion. Furthermore, Sec. VII lists the current limitations and future direction of our work. Lastly, Sec. VIII concludes this paper.

## II. Related Work

Edge computing is a promising paradigm for preventing mobile devices from overburdening onboard systems. It assists mobile devices in running resource-intensive and delay-sensitive tasks, such as object detection and depth estimation. However, compared with cloud computing systems, edge computing systems are more likely to suffer from a lack of computing resources. To this end, proper task offloading and resource allocation methods are needed to better leverage the computing resources of edge systems. There is a wealth of research on task offloading in edge computing systems, especially for vehicular applications.

Several prior works are based on offline settings [8], [10], [14], [17], [22], [23]. [8] and [22] both studied single-edge server frameworks and proposed heuristic algorithms that use vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) offloading to minimize delay, while [22] additionally proposed a generalized Bender decomposition and reformulation linearization-based algorithm. [10] further incorporated multi-hop V2V offloading and cloud computing into its system. In contrast to the above line of work that employs both V2V and V2I offloading, our work only utilizes V2I offloading. One reason not to offload tasks to other vehicles is that the computing power of different vehicles does not vary much. Another is the lack of incentives in the view of the plethora of ADAS and ADS tasks that each vehicle may have to execute in time on its onboard system. Last, there are potential security and privacy risks from sharing compute tasks across private vehicles. Thus, we doubt that future vehicles will share their compute resources with other vehicles. In [14], [17], [23], only V2I is considered. [23] investigates a joint task scheduling and resource allocation problem in the view of handoffs, i.e., when vehicles leave the service area of the RSU they currently offload tasks to. Different from the centralized mechanisms proposed in [8], [10], [22], [23], distributed algorithms that run in each RSU are designed in [14] and [17]. In both works, load distribution among RSUs was investigated to further decrease task delay. Although previously mentioned works perform well in minimizing delay, the offline setting is not realistic, and in this work, we focus on an online setting.

To address concerns with the offline setting, the authors of [12], [18] proposed periodic optimization algorithms that are more applicable to real-world scenarios. In [12], vehicles with idle computing resources form vehicular edge computing networks, and RSUs serve only for information exchange. In contrast, [18] allows task execution on both RSUs and local vehicles. Unlike [12], [18], our work further develops an online algorithm; thus, each vehicle request is resolved immediately without wasting time waiting for periodic decisions.

In addition to delay minimization, improving energy efficiency is another long-standing issue [13], [15]. Specifically, [13] considers both V2V and V2I offloading and develops an algorithm to minimize system energy consumption by optimizing the offloading strategy and the computation resource allocation. However, the proposed algorithm operates in a centralized and periodic fashion. [15] proposes an online mechanism to optimize offloading decisions and transmit power for higher energy efficiency in a dynamic environment. Although the proposed algorithm responds to user requests in an online manner, it relies on a centralized controller for coordination. While energy efficiency is important, the energy toll of vehicle-related ML tasks is smaller than that of other compute-intensive tasks, such as training large language models or executing blockchain computations. What is more, vehicle-related ML tasks must satisfy harsh time-critical deadlines since the in-time information provided by these tasks' results is key to drivers' safety. We thus focus on satisfying time-critical deadlines.

A large line of prior art considers specific objectives, such as maximizing utility and minimizing cost under an offline setting [9], [16], [24]–[27]. Game theoretical schemes were introduced in [9] and [16]. [9] proposes a two-stage offloading mechanism to increase the task completion rate and user utility. In [16], a centralized SDN controller is responsible for task scheduling and resource allocation. Pricing schemes for the edge and cloud servers were developed, and a Stackelberg game was formulated to model the competition and cooperation among each node in the system. [24]–[26] also adopt game theoretical methods but consider distributed algorithms. Decisions are made at each vehicle in [24] and [25] and at each RSU in [26]. Last, [27] also proposes a decentralized offloading and resource allocation method but focuses on cooperation among RSUs.

To better fit the dynamic nature of edge computing environments, periodic scenarios are investigated in [11], [20], [28]–[30]. [11] adopted a stochastic optimization technique to minimize system electricity costs while guaranteeing the task queue length. [20] applied deep reinforcement learning (DRL) to increase the task acceptance rate while considering the delay tolerance of tasks. [28] also utilizes DRL for task offloading decisions but adopts a distributed multi-agent scheme instead of a centralized mechanism, as in [11]. [29] and [30] also develop decentralized algorithms, where the former minimizes a weighted sum of delay, energy, and cost, while the latter focuses on delay and the price paid for offloading under task-specific requirements. Last, [21] devised an online algorithm that makes offloading decisions based on a predefined threshold to achieve lower delay and energy consumption. However, it relies on a central controller for information gathering and decision-making. Different from all these prior works, our scheme TODORA not only runs in an online manner but also gets rid of the centralized controller, enabling fully distributed execution on each RSU.

To summarize, a timeliness-aware offloading mechanism is essential for edge-assisted vehicular tasks. Unlike prior work, we propose an online, distributed offloading and resource allocation mechanism, TODORA. Additionally, TODORA accounts for the overhead of control messages when making offloading decisions, an aspect often overlooked in existing studies, enhancing its practicality for real-world deployment. TODORA operates within a comprehensive local-edge-cloud system, jointly managing both computation and radio resource allocation. This holistic approach maximizes system resource utilization and further improves the approach's applicability in real-world setups. Also, TODORA works under both binary and partial offloading. Table I provides a comparative summary of our work against other related works.

TABLE I
COMPARISON OF THE RELATED WORKS

| Paper | Offloading | Commun. Overhead | Distributed Scheduler | Decision | Resource Allocation | Local Computing | Cloud Computing |
|---|---|---|---|---|---|---|---|
| [8], [9] | Binary | × | × | Offline | × | ✓ | × |
| [22], [23] | Binary | × | × | Offline | ✓ | ✓ | × |
| [10] | Binary | × | × | Offline | ✓ | × | ✓ |
| [14] | Partial | × | ✓ | Offline | Only Radio | ✓ | × |
| [17] | Binary | × | ✓ | Offline | ✓ | ✓ | × |
| [12] | Binary | × | ✓ | Periodic | Only Comp. | × | × |
| [18] | Partial | × | × | Periodic | × | ✓ | × |
| [13] | Binary | × | × | Periodic | Only Comp. | ✓ | × |
| [15] | Partial | × | × | Online | Only Radio | ✓ | × |
| [16] | Partial | × | × | Offline | ✓ | ✓ | ✓ |
| [24] | Binary | × | × | Offline | Only Comp. | ✓ | × |
| [25] | Binary | × | ✓ | Offline | × | ✓ | × |
| [26] | Partial | × | ✓ | Offline | ✓ | ✓ | × |
| [27] | Binary | × | ✓ | Offline | ✓ | × | × |
| [11] | Binary | × | × | Periodic | × | × | ✓ |
| [20] | Binary | × | × | Periodic | × | ✓ | ✓ |
| [28] | Binary | × | ✓ | Periodic | × | × | ✓ |
| [29] | Partial | × | ✓ | Periodic | ✓ | × | ✓ |
| [30] | Binary | × | ✓ | Periodic | × | × | × |
| [21] | Binary | × | × | Online | × | ✓ | × |
| TODORA | Binary, Partial | ✓ | ✓ | Online | ✓ | ✓ | ✓ |

## III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we first present the system model in Sec. III-A to III-D, which encompasses the communication model between vehicles and roadside units (RSUs) as well as the computation model on edge servers and the cloud server. Subsequently, in Sec. III-E, we formulate the joint task offloading and resource allocation problem with the objective of minimizing the average task duration.

TABLE II
FRAMEWORK NOTATION

| Notation | Description |
|---|---|
| $\mathcal{T} = \{1, 2, \cdots, T\}, T$ | Set of time slots, size of $\mathcal{T}$ |
| $\mathcal{U} = \{1, 2, \cdots, U\}, U$ | Set of RSUs, size of $\mathcal{U}$ |
| $\mathcal{V} = \{1, 2, \cdots, V\}, V$ | Set of vehicles, size of $\mathcal{V}$ |
| $\mathcal{N} = \{1, 2, \cdots, N\}$, N | Set of tasks, size of $\mathcal{N}$ |
| $\tau$ | Duration of a time slot |
| $f_v^{\mathrm{V}}, f_u^{\mathrm{U}}, f^{\mathrm{C}}$ | FLOPS of vehicle $v$, RSU $u$, the cloud server |
| $\mathrm{RTT_v}, \mathrm{RTT_c}$ | RTT between the RSU and the vehicle, between the RSU and the cloud server |
| $B_u$ | Upload bandwidth of RSU $u$ |
| $u_v \in \mathcal{U}$ | RSU that vehicle $v$ connects to |
| $F_n, L_n, d_n^{\max}$ | Required computation resources, input size, delay tolerance of task $n$ |
| $t_n \in [0, T \cdot \tau]$ | The time that task $n$ is generated |
| $v_n \in \mathcal{V}$ | Vehicle that generates task $n$ |
| $\mathcal{S} = (S_n^{\mathrm{C}}, S_n^{\mathrm{U}}, S_n^{\mathrm{V}})$ | Offloading decision of task $n$ |
| $x_{n,t}$ | Transmission decision of task $n$ at time slot $t$ |
| $y_{n,t}$ | Computation resources allocation of task $n$ at time slot $t$ |
| $d_n$ | Task duration of task $n$ |

## A. System Model

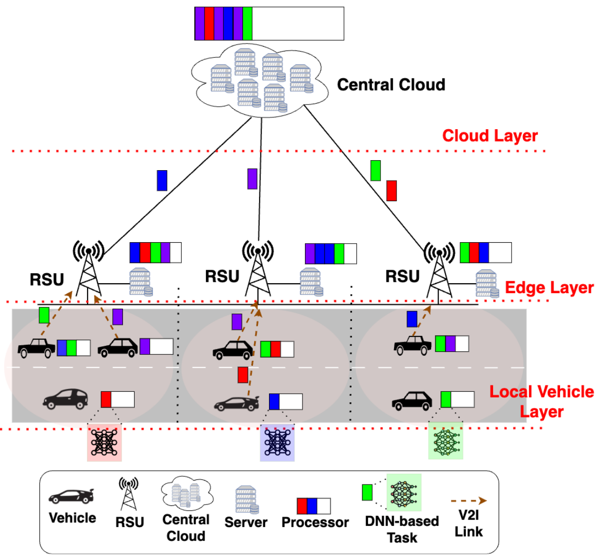


Fig. 1. Local-edge-cloud computing system for in-vehicle DNN-based ML tasks. This work considers a comprehensive setup where tasks share and contend for both computation and radio resources across the local, edge, and cloud layers. The system aims to minimize the total task duration, accounting for execution time, communication overhead from request/response procedures, and coordination overhead from periodic scheduling.

Fig. 1 shows an integral edge-cloud computing system composed of a cloud server and multiple RSUs equipped with edge computing capacities. We consider an LTE-V network in which a set of $V$ vehicles $\mathcal{V} = \{1, 2, \cdots, V\}$ and a set of $U$ RSUs $\mathcal{U} = \{1, 2, \cdots, U\}$ are deployed along a road. The road is partitioned into $U$ segments, and vehicles in each segment are served by one RSU located at the segment's center. We call the RSU that provides service to the vehicle $v \in \mathcal{V}$ as vehicle $v$'s serving RSU, and we denote it by $u_v$. Note that the serving RSU $u_v$ for vehicle $v$ may change over time as vehicles can perform handoffs between RSUs due to mobility. In this vehicular network, each vehicle $v$ moves at a velocity $\mathrm{vel}_v$, which follows a Gaussian distribution $N(\mu_v, \sigma_v)$. Since each RSU is equipped with an edge server, we use the terms "edge server" and "RSU" interchangeably. Specifically, "edge server" is stated when emphasizing the computation process, while "RSU" is used to describe the communication process. During a period of time, a set of $N$ delay-sensitive tasks $\mathcal{N} = \{1, 2, \cdots, N\}$ are generated by vehicles. Here, we assume the entire time period is divided into $T$ time slots. [1] $\mathcal{T} = \{1, 2, \cdots, T\}$ and $\tau$ denote the set of time slots and the duration of a time slot, respectively. Each RSU should collect information periodically, including available computation resources and the number of ongoing tasks, from the cloud servers and vehicles inside its service range to facilitate the decision-making procedure. It is assumed that vehicles report their information to their serving RSUs, and the cloud server distributes its information to all RSUs. To be easier to implement in real-world systems and eliminate the delay for task splitting and synthesis, this work focuses on binary offloading, meaning each task can be either fully executed locally or fully offloaded. It is worth noting that the offloading decision cannot be changed once it has been made. Let $\mathcal{S}_n = (S_n^{\mathrm{V}}, S_n^{\mathrm{U}}, S_n^{\mathrm{C}})$, where $S_n^{\mathrm{J}} \in \{0, 1\}, \mathrm{J} \in \{\mathrm{V}, \mathrm{U}, \mathrm{C}\}$, be a set of offloading decision variables of task $n$ indicating whether task $n$ is executed on the local vehicle ($S_n^{\mathrm{V}} = 1$), an edge server ($S_n^{\mathrm{U}} = 1$), or the cloud server ($S_n^{\mathrm{C}} = 1$). Since binary offloading is considered $\sum_{\mathrm{J}} S_n^{\mathrm{J}} = 1, \forall n$. We take into consideration the communication overhead of transmitting offloaded tasks, as well as the request/response procedure. The latter represents the delay in transmitting the offloading request from the vehicle to the decision maker and receiving the decision response back. In general, the control messages (i.e., request/response messages) are of small size. Their transmission delay and the associated resource allocation are negligible. Thus, we approximate their communication overhead only with the round trip time (RTT) between the vehicle and the decision maker.

[1]The present formulation concerns a periodic optimal algorithm, while the next section introduces an online distributed algorithm.

## B. Local Computing

If task $n$ is to be executed locally (i.e., $\mathcal{S}_n = (1, 0, 0)$), the task duration of task $n$ (denoted as $d_n^{\mathrm{V}}$ with $V$ indicating in-vehicle execution) can be expressed as:

$$d_n^{\mathrm{V}} = d_n^{\mathrm{request}} + t_n^{\mathrm{end}} - t_n^{\mathrm{start}} \tag{1}$$

$$s.t. \quad \int_{t_n^{\mathrm{start}}}^{t_n^{\mathrm{end}}} y_{n,t} f_v^{\mathrm{V}} dt = F_n, \tag{2}$$

where $d_n^{\mathrm{request}}$ represents the communication time for requesting and obtaining the offloading decision, $F_n$ is the required floating-point operations (FLOPs) of task $n$, $f_v^{\mathrm{V}}$ is the available floating-point operations per second (FLOPS) of vehicle $v$, and $t_n^{\mathrm{start}}/t_n^{\mathrm{end}}$ are the instances of time task $n$ starts/finishes execution. Constraint (2) guarantees that the total FLOPs allocated to task $n$ meet its computation requirement. It is assumed that a task can only be executed on the vehicle that generates it if it is to be processed locally (i.e., task $n$ is generated by vehicle $v$). Note that this work focuses on machine learning (ML) tasks, for which FLOPs serve as a widely-used metric to quantify computational requirements. Furthermore, we assume ML tasks are executed on GPUs and use FLOPS, a common performance indicator for GPUs, to represent the computation resources available on each entity.

## C. Edge Processing

There are three phases in edge processing: the request and response phase, the transmission phase, and the computation phase.

*1) Request and response phase:* In this phase, a vehicle sends an offloading request to the decision maker and waits for the response, which takes $d_n^{\mathrm{request}}$ time.

*2) Transmission phase:* In this phase, an offloaded task $n$, generated by vehicle $v_n$, is transmitted to its serving RSU $u_v$. When it comes to task transmission, we consider an LTE V2I network operating in TDMA mode, where the duration of an uplink frame is the same as the duration of a time slot

$\tau$. Considering typical LTE parameters, we assume that the frame size is fixed at 10 ms for all frames [31], and each frame can transmit up to 350 Kbit of data, corresponding to a bandwidth of $B = 35$ Mbps [32], [33]. In addition, we model the PHY layer by incorporating the probability of successful task delivery between a vehicle and the RSU, with $p_{v,u_v} \in [0,1]$ representing the probability of successful delivery from vehicle $v$ to its serving RSU $u_v$. This results in an effective data rate of $p_{v,u_v} \cdot B$. Following the analysis in [34], $p_{v,u_v}$ is a function of the distance between vehicle $v$ and RSU $u_v$, traffic density, and task arrival rate (See [34] for the details). Furthermore, we assume the use of state-of-the-art compression techniques, as proposed in [35], enabling the input data of typical ML tasks to fit within a single frame.

Let $x_{n,t} \in \{0,1\}$ be a decision variable indicating whether task $n$ is offloaded at frame $t$ to RSU $u_{v_n}$, which depends on whether a frame is available for this uplink transmission based on LTE uplink scheduling of transmissions from vehicles to this RSU. Let $d_n^{\mathrm{w}}$ denote the time waiting for an uplink transmission opportunity. Then $d_n^{\mathrm{w}} = t_n^{\mathrm{up}} - t_n^{\mathrm{q}}$, where $t_n^{\mathrm{up}}$ denotes the time task $n$ begins uploading (i.e. $x_{n,t_n^{\mathrm{up}}} = 1$) and $t_n^{\mathrm{q}}$ is the time task $n$ enters the transmission queue. Once the task begins uploading, it will take some time to complete uploading, which depends on LTE data rates affecting the transmission time and the associated propagation delay. We denote the transmission time by $d_n^{\mathrm{up}}$ and the associated propagation delay by $\mathrm{RTT_j}, \mathrm{j} \in \{\mathrm{v}, \mathrm{c}\}$. Note that while the propagation delay is small in the case of uploading to an RSU, it is not in the case of uploading to the cloud (See the following Cloud Processing subsection) where typical values are around 5 microseconds [33]. Thus, in our work, the communication delay for transmitting a task includes both the transmission ($d_n^{\mathrm{up}}$) and propagation delays (RTT). Once the task is uploaded and executed, the result needs to be downloaded. Similar to several prior works, e.g., [8], [17], we assume that the execution result can always fit in the current frame considering that it is small and that the downlink data rate is much larger than the uplink data rate (e.g., higher than 100 Mbps [33]). Thus, we assume there is no waiting time for a downlink opportunity.

Putting it all together, the time duration of task $n$ in the transmission phase is composed of the time waiting for uplink transmission $d_n^{\mathrm{w}}$, the uploading time $d_n^{\mathrm{up}}$, the downloading time $d_n^{\mathrm{down}}$, and the associated propagation delay $\mathrm{RTT_j}$.

*3) Computation phase:* In this phase, task $n$ is executed on the edge server $u = u_{v_n}$. Let $y_{n,t}$ be a decision variable indicating the percentage of computation capacity that is allocated to task $n$ on the serving RSU at time $t$. Similar to (1), the execution time on the edge server can be written as (superscript $U$ indicates RSU execution):

$$d_n^{\mathrm{exe,U}} = t_n^{\mathrm{end}} - t_n^{\mathrm{start}} \tag{3}$$

$$s.t. \quad \int_{t_n^{\mathrm{start}}}^{t_n^{\mathrm{end}}} y_{n,t} f_u^{\mathrm{U}} dt = F_n, \tag{4}$$

where $f_u^{\mathrm{U}}$ denotes the available FLOPS of edge server $u$, and $t_n^{\mathrm{start}}$ and $t_n^{\mathrm{end}}$ are the instances of time edge server $u$ begins and completes the task execution, respectively. Similar to (2), Constraint (4) guarantees that task $n$ obtains sufficient computation resources.

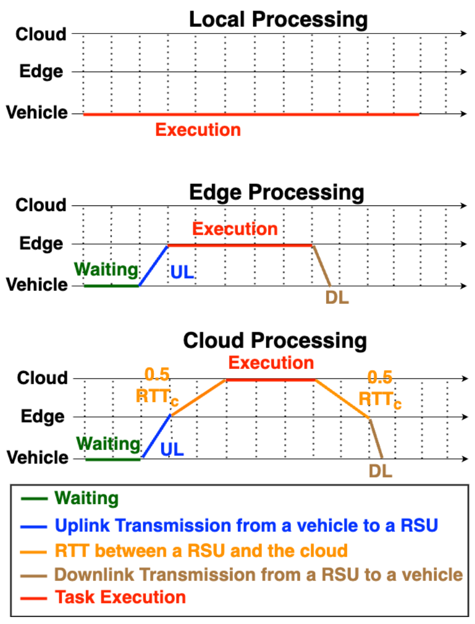


Fig. 2. Task duration in different execution scenarios. The horizon lines represent the locations (i.e., vehicle, edge, and cloud) where different stages occur, while the vertical lines indicate the start of each time slot.

As a result, the task duration of an edge-assisted task $d_n^{\mathrm{U}}$ can be expressed as: (also see the edge/RSU processing schematic as compared to the local/in-vehicle processing in Fig. 2)

$$d_n^{\mathrm{U}} = d_n^{\mathrm{request}} + d_n^{\mathrm{w}} + d_n^{\mathrm{up}} + \mathrm{RTT_v} + d_n^{\mathrm{exe,U}} + d_n^{\mathrm{down}}. \tag{5}$$

## *D. Cloud Processing*

The cloud processing procedure is similar to that of edge processing. The cloud server can be regarded as a server with a higher volume of computation resources compared to edge servers. However, offloading a task to the cloud server results in a sizable RTT between the RSU and the cloud, denoted by $\mathrm{RTT_c}$. Thus, the task duration of a task $n$ using cloud processing can be expressed as (see also the cloud processing schematic in Fig. 2):

$$d_n^{\mathrm{C}} = d_n^{\mathrm{request}} + d_n^{\mathrm{w}} + d_n^{\mathrm{up}} + \mathrm{RTT_v} + \mathrm{RTT_c} + d_n^{\mathrm{exe,C}} + d_n^{\mathrm{down}}, \tag{6}$$

where the upload/download times $d_n^{\mathrm{up}}$ and $d_n^{\mathrm{down}}$ are dominated by the much slower link speed between the vehicle and the RSU than that between the RSU and the cloud (the latter often being higher than 100 Gbps [36]), $d_n^{\mathrm{exe,C}}$ denotes the execution time of task $n$ on the cloud server, and $f^{\mathrm{C}}$ is the available FLOPS of the cloud server. The execution time should also satisfy the following equation:

$$d_n^{\mathrm{exe,C}} = t_n^{\mathrm{end}} - t_n^{\mathrm{start}} \tag{7}$$

$$s.t. \quad \int_{t_n^{\mathrm{start}}}^{t_n^{\mathrm{end}}} y_{n,t} f^{\mathrm{C}} dt = F_n, \tag{8}$$

where, $y_{n,t}$ is the percentage of computation resources allocated to task $n$ at time $t$. The primary notation of the framework is summarized in Table II.

### E. Problem Formulation

We minimize the average task duration by optimizing offloading decisions ($\{\mathcal{S}_n\}$), communication resource allocation ($\{x_{n,t}\}$), and computation resource allocation ($\{y_{n,t}\}$). The overall optimization can be formulated as follows:

$$
\begin{aligned}
\min \quad & \frac{1}{N}\sum_{n=1}^{N} d_n && \text{(9a)}\\
\text{variables:} \quad & \{\mathcal{S}_n\}, \{x_{n,t}\}, \{y_{n,t}\} && \text{(9b)}\\
\text{s.t.} \quad & d_n \le d_n^{\max}, \forall n, && \text{(9c)}\\
& S_n^{\mathrm{J}} \in \{0,1\}, \forall n, \forall \mathrm{J}, && \text{(9d)}\\
& \sum_{\mathrm{J}} S_n^{\mathrm{J}} = 1, \forall n, && \text{(9e)}\\
& x_{n,t} \in \{0,1\}, \forall n, \forall t, && \text{(9f)}\\
& S_n^{\mathrm{V}} + \sum_{t=t_n}^{T} x_{n,t} = 1, \forall n, && \text{(9g)}\\
& \sum_{n:u_{v_n}=u} \frac{x_{n,t}L_n}{B_u} \le \tau, \forall u, \forall t, && \text{(9h)}\\
& 0 \le y_{n,t} \le 1, \forall n, \forall t, && \text{(9i)}\\
& \sum_{n:v_n=v, S_n^{\mathrm{V}}=1} y_{n,t} \le 1, \forall v, \forall t, && \text{(9j)}\\
& \sum_{n:u_{v_n}=u, S_n^{\mathrm{U}}=1} y_{n,t} \le 1, \forall u, \forall t, && \text{(9k)}\\
& \sum_{n:S_n^{\mathrm{C}}=1} y_{n,t} \le 1, \forall t, && \text{(9l)}\\
& \text{Constraints (2), (4), and (8)}\\
& n \in \mathcal{N}, u \in \mathcal{U}, t \in \mathcal{T}
\end{aligned}
$$

where the objective and variables are provided in (9a) and (9b), respectively, and $d_n = S_n^{\mathrm{V}} d_n^{\mathrm{V}} + S_n^{\mathrm{U}} d_n^{\mathrm{U}} + S_n^{\mathrm{C}} d_n^{\mathrm{C}}$.

Constraint (9c) limits the task duration of a task below its delay tolerance. Constraints (9d) and (9e) ensure a binary offloading decision for each task. Constraints (9f) and (9g) ensure that a task will be executed locally or will be transmitted at exactly one frame, and Constraint (9h) guarantees that the tasks allocated to a frame do not exceed the frame size, where $L_n$ is the input data size of task $n$ and $B_u$ denotes the uplink bandwidth of RSU $u$. Last, Constraints (9i) - (9l) denote the computation capacity constraints at vehicles, edge servers, and the cloud server, respectively. (Recall that Constraints (2), (4), and (8) ensure that all tasks receive enough computing resources.)

Problem (9) is a mixed integer nonlinear programming problem (MINLP). In this work, we adopt solvers, such as SCIPY [37], provided in standard optimization software CVXPY [38] to solve this problem in a periodic manner to come up with a periodic optimal solution (denoted as POPT). However, solving MINLP with CVXPY becomes computationally expensive as the problem size increases, that is, as $T$, $U$, $V$, and $N$ grow. Specifically, the worst-case time complexity of solving a standard problem (i.e., no integral constraints) using CVXPY is $\mathcal{O}(n^4 + \log(1/\epsilon))$ [39], where $n$ is the number of constraints, and $\epsilon$ is a constant reflecting the solution accuracy of the solver. By relaxing Constraints (9d) and (9f), Problem (9) can be formulated as a standard problem with $n = (10+4T)N + (2U+V+1)T$ constraints. It is evident that the time complexity becomes prohibited large, given large $T$, $U$, $V$, and $N$ values. Due to the existence of integral constraints (9d) and (9f), the time complexity of Problem (9) will be even larger. Last, another shortcoming of POPT is that it is periodic-based rather than online and is centralized.

Motivated by the practical shortcomings of POPT, in the next section, we design an efficient and effective heuristic algorithm, which we refer to as TODORA, that can obtain a favorable solution to Problem (9) rapidly in an online, distributed fashion. TODORA is a low-complexity threshold-based algorithm that makes online offloading decisions at each RSU and can be implemented and operated in practical scenarios. In Sec. VI, we compare the performance of TODORA against POPT to assess whether avoiding the communication overhead with a central server and the delay introduced by periodic decisions results in better performance in practice despite the heuristic decisions.

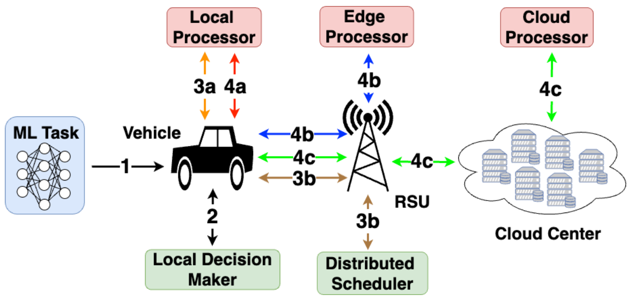


Fig. 3. Flow chart of TODORA. Boxes with different colors are used to highlight the distinct functions (i.e., processors, decision-makers, and schedulers) within each entity, and the data flow is explicitly drawn to indicate the final destination of each procedure.

## IV. Online and Distributed Algorithm

To provide in-time information to ADAS or ADS, it is of great importance to build a task scheduling and resource allocation mechanism to minimize the task duration of in-vehicle time-critical ML tasks. In addition, to ensure that the mechanism can be implemented in real-world scenarios, the communication and computation overhead of performing the scheduling and allocation processes should also be taken into account. With this in mind, we propose a two-stage Threshold-based Online Distributed Offloading Resource Allocation algorithm, which we call TODORA.

Instead of making decisions using a high-complexity framework, TODORA performs task scheduling and resource allocation by greedily examining task attributes with predefined thresholds. Due to its low-complexity and distributed nature, which eliminates the need for a central controller, the communication overhead of TODORA is much lower than that of POPT. Additionally, the threshold-based mechanism is more suitable for resolving online requests than POPT, as it can make a decision immediately after receiving a request without the need to wait for periodic decision points. In contrast to POPT, TODORA does not guarantee that a task will execute within the delay threshold. Instead, TODORA incorporates delay tolerance into its task scheduling and resource allocation mechanism to prioritize tasks with smaller delay tolerance $d_n^{\max}$ (more urgent tasks that need to be completed fast). A task will be marked 'delayed' if the delay tolerance is not satisfied. We discuss the design of TODORA in detail below.

*1) Local offloading decision:* Consider a task $n$ is generated at vehicle $v$, with corresponding serving RSU $u_v = u$ at time $t_n$. First, we assign a priority weight $p_n$ for each task $n$ equal to $1/\sqrt{d_n^{\max} F_n}$, which gives higher priority to tasks with smaller delay tolerance $d_n^{\max}$ and computation requirement $F_n$ (smaller tasks). This priority is used to share computation resources among tasks proportionally to their priority weight. Note that more urgent and smaller tasks receive a larger fraction of computation resources, motivated by well-established results of queuing theory, which dictate that earlier deadline first (EDF) schedulers maximize the number of tasks completed within their deadlines, and shortest job first (SFJ) schedulers minimize the average completion time of tasks [40]. We combine EDF and SJF principles since meeting deadlines and reducing completion time are both crucial for ADAS and ADS. In-time task results enable ADAS to assist drivers effectively and allow ADS to make safe decisions, and reducing task completion time offers additional reaction time, enhancing overall safety. Note that we apply the square root function to the term $d_n^{\max} F_n$ to moderate the influence of large values: This prevents excessive penalization of tasks with higher computation requirements or more relaxed delay constraints, which would occur under a pure reciprocal function (i.e., $1/d_n^{\max} F_n$). Additionally, the square root maintains the ability to distinguish between tasks, unlike the negative exponential function (i.e., $\exp(-d_n^{\max} F_n)$), which becomes indistinguishably close to zero when $d_n^{\max} F_n$ is large.

The first time vehicle $v$ connects to RSU $u$, RSU $u$ will inform vehicle $v$ of its available FLOPS ($f_u^{\mathrm{U}}$). In addition, at each response message to vehicle $v$, RSU $u$ will piggy-pack the sum of the priority values of all the tasks running on the RSU. This can serve as an estimate of the sum of the priority values of all the tasks running on the RSU at the time task $n$ may be assigned to be executed at the RSU $u$. We denote this sum by $P_u^{\mathrm{est}}$. With this information, vehicle $v$ will estimate the edge execution time of task $n$ as follows:

$$d_{n,\mathrm{exe}}^{\mathrm{U,est}} = F_n/(f_u^{\mathrm{U}} \cdot p_n/(P_u^{\mathrm{est}} + p_n)), \tag{10}$$

where $p_n/(P_u^{\mathrm{est}} + p_n)$ assigns task $n$ a portion of the RSU's FLOPS proportional to the task's priority weight. Once the estimate for the edge execution time is available, the algorithm may compare the sum of this estimate and the estimated upload time ($d_n^{\mathrm{up,est}}$) with $d_n^{\max}$ to assess whether offloading the task to the RSU may result in executing the task in time or not. (Note that vehicle $v$ can estimate the upload time since the input size of task $n$ and the ideal uplink data rate are known, and the effective data rate can be predicted based on the vehicle's velocity and distance to the serving RSU $u$.)

Next, vehicle $v$ will evaluate the estimated local execution time as follows:

$$d_n^{\mathrm{V,est}} = F_n/(f_v^{\mathrm{V}} \cdot p_n/(P_v + p_n)), \tag{11}$$

where $P_v$ denotes the sum of the priority values of all the tasks running on $v$. (Note that unlike $P_u^{\mathrm{est}}$, which is piggy-backed at the last response message and thus may be outdated when the edge execution time is estimated by the vehicle, $P_v$ is based on the priorities of the tasks running locally in the vehicle, and thus it is precise.)

Based on the above information, the vehicle can make a quick decision regarding whether to execute the task locally without even sending a request to the edge for the scheduler to decide the best execution plan for the task. Specifically, this quick local decision step proceeds as Algorithm 1 (see also steps 2, 3a, and 3b in Fig. 3). Note that since the vehicle

**Algorithm 1** Local offloading decision

```
1: if $d_n^{\mathrm{V,est}} < d_n^{\max}$ and $d_n^{\mathrm{up,est}} + d_{n,\mathrm{exe}}^{\mathrm{U,est}} \geq d_n^{\max}$ then
2:     Execute task $n$ locally on vehicle $v$.
3: else
4:     Send a task request to the distributed scheduler on RSU $u$.
5:     (See next Sec. IV-2 for details on the scheduler.)
6: end if
```

cannot precisely estimate the time it would take for task $n$ to be executed by the RSU and the result to be received by the vehicle ($d_{n,\mathrm{exe}}^{\mathrm{U,est}}$ is estimated using the potentially outdated $P_u^{\mathrm{est}}$ value, and the associated network delays cannot be precisely estimated by the vehicle either), it refrains from comparing estimated execution times locally versus via the RSU. Instead, it executes the task locally when it can do so within the maximum delay tolerance of the task, and there is a high probability that the upload time and execution time associated with offloading the task to the RSU would exceed that maximum delay tolerance. Otherwise, it offloads the task to the RSU for the scheduler to make a more informed decision.

*2) Distributed Scheduling:* At each RSU, there is a scheduler that makes scheduling decisions right after it receives a task request from a vehicle. Consider a task $n$ generated at vehicle $v$, with corresponding serving RSU $u_v = u$ at time $t_n$. Via a task request, vehicle $v$ will inform the scheduler of task attributes, its available FLOPS ($f_v^{\mathrm{V}}$), and the sum of the priorities of the tasks currently running in the vehicle, $P_v$. In addition, the cloud will periodically inform the scheduler of the sum of all running tasks' priority values on the cloud server, denoted by $P_c$. As discussed in Sec. III, it is assumed that the RSU and the cloud are connected via a high-speed wired link, ensuring a congestion-free connection, unlike the potentially congested wireless link between the vehicle and RSU. Consequently, the cloud can frequently update its information to the scheduler, and the $P_c$ value from the cloud is regarded as timely and accurate.

After receiving a task request, the scheduler will estimate the task execution time at the vehicle $d_n^{\mathrm{V,est}}$ using (11), and the task execution time at the edge (RSU) and the cloud as follows (these correspond to the time for steps 4a, 4b and 4c in Fig. 3):

$$\begin{aligned} d_n^{\mathrm{U,est}} &= d_n^{\mathrm{w,est}} + d_n^{\mathrm{up,est}} + \mathrm{RTT}_{\mathrm{v}}^{\mathrm{est}} \\ &\quad + F_n/(f_u^{\mathrm{U}} \cdot p_n/(P_u + p_n)) + d_n^{\mathrm{down}}, \qquad (12) \\ d_n^{\mathrm{C,est}} &= d_n^{\mathrm{w,est}} + d_n^{\mathrm{up,est}} + \mathrm{RTT}_{\mathrm{v}}^{\mathrm{est}} + \mathrm{RTT}_{\mathrm{c}}^{\mathrm{est}} \end{aligned}$$

$$+ F_n/(f^{\mathrm{C}} \cdot p_n/(P_c + p_n)) + d_n^{\mathrm{down}}, \tag{13}$$

where $d_n^{\mathrm{w,est}}$ is the estimated uplink waiting time (obtained using the average uplink waiting time of tasks that were offloaded to RSU $u$ from vehicle $v$ in the recent past), $f_v^{\mathrm{V}}, f_u^{\mathrm{U}}, f^{\mathrm{C}}$ are the available FLOPS of the in-vehicle, RSU, and cloud servers, $d_n^{\mathrm{down}}$ is the download time, and $\mathrm{RTT}_{\mathrm{c}}^{\mathrm{est}}$, $\mathrm{RTT}_{\mathrm{v}}^{\mathrm{est}}$ are the estimated RTT values between RSU $u$ and the cloud server, and between RSU $u$ and vehicle $v$, obtained using a standard exponential weighted moving average[2].

The scheduler will then select the option with the lowest estimated delay, and the decision will be sent back to vehicle $v$. Note that the scheduler at the RSU can make a more informed decision than the vehicle since the estimate in (12) is more reliable than that in (10) and the vehicle does not have information about the cloud.

*3) Airtime Resource Allocation:* When it comes to prioritizing transmissions, a task will be transmitted as soon as a frame is available. If multiple tasks are waiting for transmission, tasks with a higher $p_n$ value have a higher priority in being transmitted.

*4) Resource Allocation on RSU:* Parallel execution of tasks takes place when more than one task is to be executed at an RSU server. As already discussed, tasks share the computation resources of the RSU server proportional to their $p_n$ values. Specifically, task $n$ on a server of capacity $f$ will obtain $(p_n/P_u) \cdot f$ FLOPS, where $P_u$ is the sum of all running tasks' $p_n$ values.

*5) Resource Allocation on the Cloud:* It is assumed that the cloud server connects with all RSUs via optical fiber, which provides an extremely high data rate (higher than 100 Gbps [36]). Thus, we omit transmission delay as well as transmission resource allocation between the cloud server and RSUs. If tasks are further offloaded to the cloud server from RSUs, the same rule described in Sec. IV-4 is followed to provide the tasks with amounts of computing capacity proportional to their priority values.

*6) Handoff:* As discussed in Sec. III-A, a vehicle may move from one RSU to another. To accommodate this, a handoff mechanism is required for TODORA. Consider a task $n$ generated by vehicle $v$ at time $t_n$, with the serving RSU at the time of generation denoted as $u_v = u$. Upon its initial connection to RSU $u$, vehicle $v$ will inform the RSU $u$ of its relevant information, including its current location, heading direction, and speed, as described in [41]. Furthermore, with each request message sent to RSU $u$, vehicle $v$ will include its updated location and speed. Using this information, the scheduler at RSU $u$ can estimate vehicle $v$'s future position and predict when vehicle $v$ will leave its coverage area. Note that these location updates are lightweight control messages, typically only a few bytes in size [42]; thus, the associated communication overhead is minimal, and the resulting delay is negligible. The time $t_v^{u \to u'}$, at which vehicle $v$ switches to a new RSU $u'$, can be categorized into two scenarios:

- Before task $n$ is uploaded to RSU $u$: This includes the request and response phase and the upload waiting phase. In this case, vehicle $v$ will rerun the local offloading decision step based on the information provided by the newly connected RSU and regenerate the request if necessary. Note that the delay tolerance $d_n^{\max}$ should be updated by subtracting the elapsed time $(t_v^{u \to u'} - t_n)$.
- After task $n$ has been uploaded to RSU $u$: In this case, the task is executed as usual to obtain the result. Once completed, RSU $u$ transmits the result to the newly connected RSU $u'$ via a wired connection. Note that transmitting the result from RSU $u$ to RSU $u'$ incurs an additional delay due to this handoff process.

TODORA is a threshold-based algorithm that makes decisions by examining thresholds and comparing estimated values. This decision-making process is independent of system parameters, such as $N$, $U$, and $V$, making TODORA an $\mathcal{O}(1)$ algorithm. This characteristic makes TODORA suitable for real-world applications, as its decision time does not increase with the system workload. We validate this property through simulations in Sec. VI. A pseudocode of basic TODORA functionality is shown in Algorithm 2.

**Algorithm 2** Threshold-based online distributed offloading and resource allocation mechanism (TODORA)

```
Input: A task $n$ is generated from vehicle $v$; vehicle $v$ is served by
    RSU $u$; A central cloud center.
 1: Local offloading decision:
 2: Calculate the priority value $p_n = 1/\sqrt{d_n^{\max} F_n}$.
 3: Estimate the edge execution time and local execution time using
    (10) and (11), respectively.
 4: Run Algorithm 1.
 5: Distributed Scheduling:
 6: Estimate $d_n^{\mathrm{V,est}}$, $d_n^{\mathrm{U,est}}$, and $d_n^{\mathrm{C,est}}$ using (11), (12), and (13),
    respectively.
 7: Compare $d_n^{\mathrm{V,est}}$, $d_n^{\mathrm{U,est}}$, and $d_n^{\mathrm{C,est}}$.
 8: if $d_n^{\mathrm{V,est}}$ is the lowest one then
 9:     Assign task $n$ to vehicle $v$.
10: else if $d_n^{\mathrm{U,est}}$ is the lowest one then
11:     Assign task $n$ to RSU $u$.
12:     Airtime resource allocation:
13:     Task $n$ will be transmitted as soon as a frame is available.
14:     If multiple tasks are waiting for transmission, tasks of higher
15:     $p_n$ value will be transmitted first.
16:     Resource Allocation on RSU:
17:     Calculate $P_u$, which is the sum of all running tasks' $p_n$
18:     values.
19:     Allocate $(p_n/P_u) \cdot f_u^{\mathrm{U}}$ FLOPS to task $n$.
20: else
21:     Assign task $n$ to the Cloud.
22:     Airtime resource allocation:
23:     Follow the same rule from Line 12 to Line 15.
24:     Resource Allocation on the Cloud:
25:     Calculate $P_c$, which is the sum of all running tasks' $p_n$ values.
26:     Allocate $(p_n/P_c) \cdot f^{\mathrm{C}}$ FLOPS to task $n$.
27: end if
```

## V. Partial Offloading

While this work primarily focuses on binary offloading, the optimization problem can be extended to support partial offloading. Additionally, the proposed algorithm, TODORA, can be adapted to accommodate partial offloading scenarios with a few modifications. In this section, we formulate the

[2]$\mathrm{RTT}_{\mathrm{j}}^{\mathrm{est}} = (1-\alpha) \cdot \mathrm{RTT}_{\mathrm{j}}^{\mathrm{est}} + \alpha \cdot \mathrm{RTT}_{\mathrm{j}}^{\mathrm{sample}}, \mathrm{j} \in \{\mathrm{c}, \mathrm{v}\}$, where the left-hand side of the above equation is the current estimated value, and the right-hand side is the weighted sum of the value from the previous estimation and the current sample value.

optimization problem for partial offloading (see Sec.V-A) and discuss how TODORA can be extended to make decisions for partial offloading (see Sec. V-B).

## A. Partial Offloading Problem

In the context of partial offloading, a task can be split into multiple parts/subtasks and executed concurrently across different locations, including the local vehicle, RSU, and cloud server. We reuse $\mathcal{S}_n = (S_n^{\mathrm{V}}, S_n^{\mathrm{U}}, S_n^{\mathrm{C}})$ to represent the partial offloading decision variables for task $n$, where $S_n^{\mathrm{J}} \in [0,1]$ and $\mathrm{J} \in \{\mathrm{V},\mathrm{U},\mathrm{C}\}$. These variables indicate the portions of task $n$ assigned to the local vehicle ($S_n^{\mathrm{V}}$), the edge server ($S_n^{\mathrm{U}}$), and the cloud server ($S_n^{\mathrm{C}}$), respectively. To ensure the entire task is processed, the constraint $\sum_{\mathrm{J}} S_n^{\mathrm{J}} = 1, \forall n$ must hold. Note that each $S_n^{\mathrm{J}}$ determines the input size ($L_n^{\mathrm{J}} = S_n^{\mathrm{J}} L_n$) and the required FLOPs ($F_n^{\mathrm{J}} = S_n^{\mathrm{J}} F_n$) of the corresponding subtask $n^{\mathrm{J}}$, where $L_n$ and $F_n$ represent the total input size and FLOPs required for the original task $n$, respectively. Additionally, we define $y_{n,t}^{\mathrm{V}}$, $y_{n,t}^{\mathrm{U}}$, and $y_{n,t}^{\mathrm{C}}$ as the fractions of computational resources allocated to the corresponding subtasks of task $n$ on the vehicle, RSU, and cloud server at time $t$.

With partial offloading, the optimization problem remains similar to (9), with Constraints (9d), (9g), and (9h) modified as follows:

$$S_n^{\mathrm{J}} \in [0,1], \forall n, \forall \mathrm{J}, \tag{14}$$

$$1 - S_n^{\mathrm{V}} \leq \sum_{t=t_n}^{T} x_{n,t} \leq 1, \forall n, \tag{15}$$

$$\sum_{n:u_{v_n}=u} \frac{x_{n,t}(1-S_n^{\mathrm{V}})L_n}{B_u} \leq \tau, \forall u, \forall t. \tag{16}$$

Additionally, task $n$'s duration under partial offloading is given by $d_n = \max\{d_n^{\mathrm{V}}, d_n^{\mathrm{U}}, d_n^{\mathrm{C}}\}$.

Similar to (9), the optimization problem under partial offloading is a MINLP problem. As such, solvers provided by CVXPY are employed to solve the problem in a periodic manner, as described in Sec. III-E. The periodic optimal solution for partial offloading is denoted as POPTP.

## B. TODORA for Partial Offloading (TODORAP)

In this section, we modify TODORA to support partial offloading. We refer to this modified version as TODORAP.

Consider a task $n$ generated at vehicle $v$, which is served by RSU $u$, at time $t = t_n$. Instead of making a local offloading decision, vehicle $v$ immediately sends a request to the scheduler at RSU $u$. The scheduler then retrieves the attributes of task $n$ and the information of vehicle $v$. After receiving the task request, the scheduler proceeds to solve equations (17) and (18) to determine the portion of the task that will execute in the vehicle:

$$\text{solve:} \quad d_n^{\mathrm{V,est}} = d_n^{\mathrm{U,est}}, \qquad \text{s.t.} \quad S_n^{\mathrm{V}} + S_n^{\mathrm{U}} = 1 \tag{17}$$

$$\text{solve:} \quad d_n^{\mathrm{V,est}} = d_n^{\mathrm{C,est}}, \qquad \text{s.t.} \quad S_n^{\mathrm{V}} + S_n^{\mathrm{C}} = 1 \tag{18}$$

where $d_n^{\mathrm{V,est}}$, $d_n^{\mathrm{U,est}}$, and $d_n^{\mathrm{C,est}}$ can be written as:

$$d_n^{\mathrm{V,est}} = F_n^{\mathrm{V}}/(f_v^{\mathrm{V}} \cdot p_n^{\mathrm{V}}/(P_v + p_n^{\mathrm{V}})), \tag{19}$$

$$\begin{aligned} d_n^{\mathrm{U,est}} &= d_n^{\mathrm{w,est}} + d_n^{\mathrm{up,est}} + \mathrm{RTT}_{\mathrm{v}}^{\mathrm{est}} \\ &\quad + F_n^{\mathrm{U}}/(f_u^{\mathrm{U}} \cdot p_n^{\mathrm{U}}/(P_u + p_n^{\mathrm{U}})) + d_n^{\mathrm{down}}, \end{aligned} \tag{20}$$

$$\begin{aligned} d_n^{\mathrm{C,est}} &= d_n^{\mathrm{w,est}} + d_n^{\mathrm{up,est}} + \mathrm{RTT}_{\mathrm{v}}^{\mathrm{est}} + \mathrm{RTT}_{\mathrm{c}}^{\mathrm{est}} \\ &\quad + F_n^{\mathrm{C}}/(f^{\mathrm{C}} \cdot p_n^{\mathrm{C}}/(P_c + p_n^{\mathrm{C}})) + d_n^{\mathrm{down}}. \end{aligned} \tag{21}$$

Note that $p_n^{\mathrm{J}} = 1/\sqrt{d_n^{\max} F_n^{\mathrm{J}}}, \mathrm{J} \in \{\mathrm{V},\mathrm{U},\mathrm{C}\}$ represents the adjusted priority value for task $n$, reflecting the actual FLOPs allocated to the corresponding subtask. The remaining variables are determined using the same procedures outlined in Sec. IV.

Once the scheduler solves (17) and (18), it compares $d_n^{\mathrm{U,est}}$ and $d_n^{\mathrm{C,est}}$ to determine whether the remaining portion of the task will be executed in the RSU or the cloud. If $d_n^{\mathrm{U,est}} \leq d_n^{\mathrm{C,est}}$, the scheduler sets $\mathcal{S}_n = (1 - S_n^{\mathrm{U}}, S_n^{\mathrm{U}}, 0)$. Otherwise, $\mathcal{S}_n$ is set to $(1 - S_n^{\mathrm{C}}, 0, S_n^{\mathrm{C}})$. Thus, the vehicle splits task $n$ into two subtasks: one executed locally and the other offloaded to the corresponding server, either the edge server on RSU $u$ or the cloud server. These subtasks are treated as individual tasks and follow the same procedures described in Sec. IV to allocate airtime and computation resources. The details of TODORAP are summarized in Algorithm 3.

**Algorithm 3** TODORA for Partial Offloading (TODORAP)

```
Input: A task $n$ is generated from vehicle $v$; vehicle $v$ is served by
    RSU $u$; A central cloud center.
 1: Send a task request to the distributed scheduler on RSU $u$.
 2: The scheduler at RSU $u$ solves (17) and (18).
 3: Compare $d_n^{\mathrm{U,est}}$ and $d_n^{\mathrm{C,est}}$.
 4: if $d_n^{\mathrm{U,est}} \leq d_n^{\mathrm{C,est}}$ then
 5:     Split task $n$ into two subtasks, $n^{\mathrm{V}}$ and $n^{\mathrm{U}}$.
 6:     Assign the priority values $p_n^{\mathrm{V}} = 1/\sqrt{d_n^{\max} F_n^{\mathrm{V}}}$ and
 7:     $p_n^{\mathrm{U}} = 1/\sqrt{d_n^{\max} F_n^{\mathrm{U}}}$ to subtask $n^{\mathrm{V}}$ and $n^{\mathrm{U}}$, respectively.
 8:     Follow the same rule from Line 12 to Line 19 in Algorithm 2
 9:     to allocate airtime and computation resources to subtask $n^{\mathrm{U}}$.
10: else
11:     Split task $n$ into two subtasks, $n^{\mathrm{V}}$ and $n^{\mathrm{C}}$.
12:     Assign the priority values $p_n^{\mathrm{V}} = 1/\sqrt{d_n^{\max} F_n^{\mathrm{V}}}$ and
13:     $p_n^{\mathrm{C}} = 1/\sqrt{d_n^{\max} F_n^{\mathrm{C}}}$ to subtask $n^{\mathrm{V}}$ and $n^{\mathrm{C}}$, respectively.
14:     Follow the same rule from Line 22 to Line 26 in Algorithm 2
15:     to allocate airtime and computation resources to subtask $n^{\mathrm{C}}$.
16: end if
```

## VI. Simulation Results

In this section, we discuss the simulation environment and the reference schemes (i.e., POPT and the state-of-the-art scheme) in Sec VI-A and Sec VI-B, respectively. Then, we present performance results in Sec VI-C.

## A. Simulation environment

We implement TODORA, POPT, and a state-of-the-art scheme [21] in Python 3.8.3 and run experiments on a server equipped with Ubuntu 20.04.1 LTS and Threadripper 3970X system (32c/64t @ 3.7 GHz). We simulate an integrated edge-cloud system consisting of a cloud server and 3 RSUs, with adjacent RSUs spaced 1 mile apart. We do not consider more than 3 RSUs as it is very unlikely for a vehicle that obeys highway speed limits [43] to come across more than a couple of RSUs during the execution of a task (which is in the order of tens of microseconds). Following the setup in [33], we

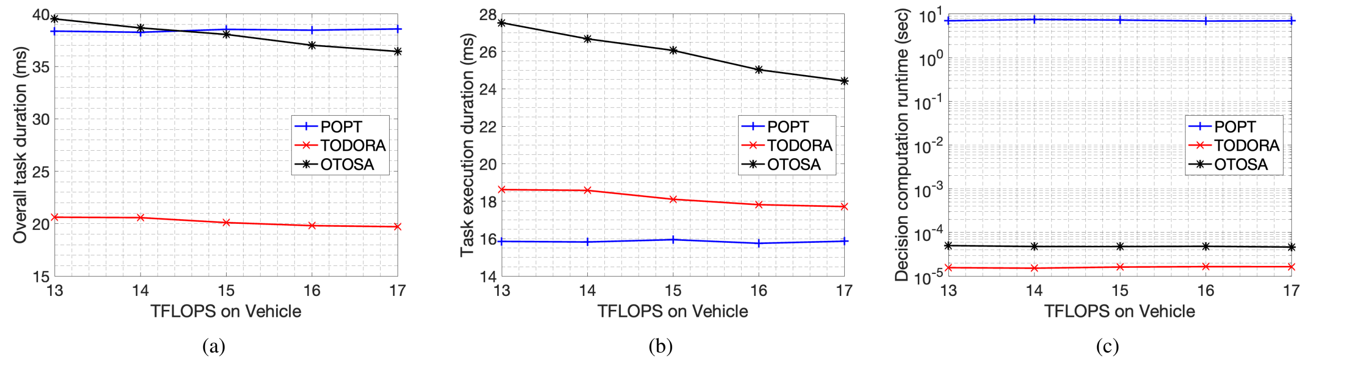


Fig. 4. (a) Average overall task duration versus FLOPS on a vehicle, (b) Average task execution duration versus FLOPS on a vehicle, (c) Decision computation runtime versus FLOPS on a vehicle.

set the average RTT between the cloud server and each RSU ($\mathrm{RTT}_c$) to 10 ms, and between a vehicle and its associated RSU ($\mathrm{RTT}_v$) to 2 ms. The propagation delay between two adjacent RSUs is set to 1 ms, as reported in [21].

We consider a bidirectional road system with two lanes and a daily traffic volume per lane ranging from 1000 to 5000 vehicles, based on statistics from the Federal Highway Administration [44]. Four types of ML tasks are considered: object detection, object tracking, panoptic segmentation, and pose estimation. These tasks are common in ADAS and ADS and have been identified as critical challenges in the Waymo Open Dataset [19]. Image data from the Waymo Open Dataset is used as the input histogram for the tasks. It is assumed that each image is first converted to JPEG format with medium quality (quality = 60) and resized to $640 \times 480$ pixels. Additionally, state-of-the-art ML modules are assumed to be adopted for each task type: YOLOv5 [1] for object detection, Siam R-CNN [45] for object tracking, MaskFormer [2] for panoptic segmentation, and ViTPose++ [46] for pose estimation. The required FLOPs for these tasks are derived from experiments conducted on a server equipped with an Nvidia RTX A6000 GPU, using image data from the Waymo Open Dataset as input. Task generation follows a Poisson process with an average inter-arrival time of 60 ms. The key simulation parameters are adopted from the settings in [19], [32], [33], [44], [47] and are summarized in Table III.

TABLE III
SUMMARY OF SIMULATION PARAMETERS

| Simulation parameters | Value |
|---|---|
| Number of time slots $T$ | 400000 |
| Number of RSUs $U$ | 3 |
| FLOPS on a vehicle $f_v^{\mathrm{V}}$ | $[13-17]$ TFLOPS |
| FLOPS on a RSU $f_u^{\mathrm{U}}$ | $[60-100]$ TFLOPS |
| FLOPS an the cloud $f^{\mathrm{C}}$ | 600 TFLOPS |
| Input data size $L_n$ | $[6-44]$ KB |
| Required FLOPs $F_n$ | $[200-1000]$ GFLOPs |
| Bandwidth $B_u$ | 35 Mbit |
| Velocity of a vehicle $\mathrm{vel}_v$ | $\mathcal{N}(65,1)$ mph |
| Duration of a time slot $\tau$ | 10 ms |

## B. Reference Algorithms

We compare TODORA against two algorithms:

- Periodic Optimal (POPT): The optimal solution described in Sec. III-E. It works periodically and relies on a central controller to provide optimal offloading decisions and resource allocation every 2 time slot. To obtain the optimal solution, we rely on the SCIPY solver [37] provided in CVXPY to solve Problem (9).
- Online centralized scheduler (OTOSA): The online scheduler from [21]. We modified the OTOSA algorithm in [21] to fit our local-edge-cloud system. To the best of our knowledge, OTOSA is the closest work that can make offloading decisions in an online manner to achieve lower delay. It is also a greedy threshold-based algorithm; however, it relies on a central controller to perform offloading and lacks the consideration of communication overhead caused by the requesting process. Comparing against this algorithm helps demonstrate the impact of communication overhead and distributed controllers.

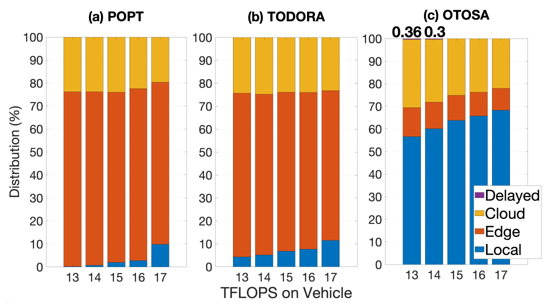
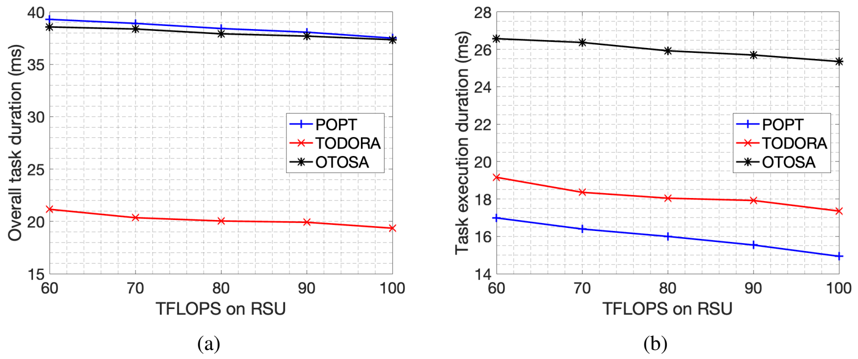


Fig. 5. Task distribution versus FLOPS on vehicle: (a) Task distribution of POPT, (b) Task distribution of TODORA, (c) Task distribution of OTOSA. (Since delayed task portions are not clearly visible, we annotate their percentages above the corresponding columns.)

## C. Performance Results

*1) The impact of vehicle computing resources:* We conduct experiments under various vehicle computing capacities. Following the setup in [47], where vehicles are equipped with GPUs comparable to the NVIDIA RTX 2060 Super GPU, we set the computing resources to $[13-17]$ TFLOPS. As shown in Fig. 4(a), TODORA outperforms the other two algorithms under all scenarios. This is because both OTOSA and POPT rely on a central scheduler, incurring substantial communication overhead. Moreover, POPT performs periodic scheduling, introducing further delay due to waiting for decision points.



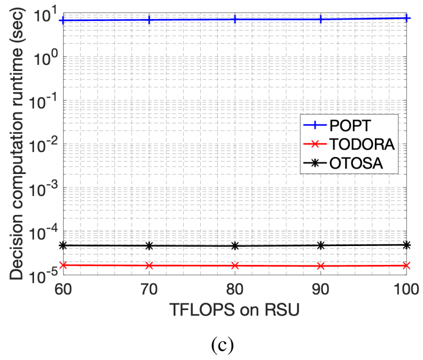


Fig. 6. (a) Average overall task duration versus FLOPS on RSU, (b) Average task execution duration versus FLOPS on RSU, (c) Decision computation runtime versus FLOPS on RSU.

To further explore the impact of the communication overhead and the loss due to the periodic mechanism, we examine the task execution duration in Fig. 4(b), where both factors are eliminated. As expected, this oracle version of POPT performs the best since its central scheduler can gather all the system information and compute optimal task assignments periodically. Apparently, the performance loss from the communication overhead (approximately 12 ms) and the periodic mechanism (around 10 ms) is considerable, suggesting that it is crucial to take these two factors into account while designing an algorithm. In comparison, the communication overhead for TODORA is only approximately 2 ms, highlighting its effectiveness in minimizing communication overhead.

Compared to OTOSA, TODORA achieves a lower task execution duration thanks to its task priority assignment strategy and its mechanism for exchanging system status information and estimating task execution delay. The priority assignment in TODORA combines principles from both EDF and SJF scheduling, allowing it to adapt effectively to diverse task characteristics (i.e., resource requirements and delay tolerance). Furthermore, the information exchange and delay estimation mechanism enables TODORA to better assess current network and processing conditions. As a result, TODORA achieves a task distribution similar to the optimal solution (See Fig. 5). OTOSA, in contrast, prioritizes computation power when selecting between edge and cloud servers while offloading tasks, leading to more tasks being offloaded to the cloud. However, this offloading comes with a substantial propagation delay between the cloud and vehicles. Thus, rather than offloading, a majority of tasks stay executed on local vehicles (See Fig. 5). This strategy not only leads to performance degradation when RSU resources significantly exceed those of vehicles but also makes OTOSA highly sensitive to vehicle computing capacity. As shown in Fig. 4(b), the average task execution duration of OTOSA decreases much more dramatically with increasing vehicle resources compared to TODORA and POPT. The latter two algorithms tend to offload most tasks to edge servers to exploit edge computing, which provides high computation resources with small propagation delays.

Fig. 5 depicts the distribution of tasks. TODORA shows a similar trend with POPT, thanks to its priority assignment, information exchange, and delay estimation mechanisms, but it tends to execute more tasks locally. This is likely because tasks with urgent deadlines or higher expected offloading delays will be directly executed on vehicles without requesting offloading; this can help eliminate the communication overhead. Fig. 5 also verifies that with increasing vehicle computing resources, all schemes schedule more tasks locally, with OTOSA scheduling much more. This makes sense: given enough resources on vehicles, tasks would be executed locally and avoid delays associated with offloading. Furthermore, it can be seen that OTOSA introduces a small number of delayed tasks when vehicles have insufficient resources ($0.36\%$ and $0.3\%$ for 13 and 14 TFLOPS, respectively), as when too many tasks are executed locally, the vehicle resources are not enough. (Recall that delayed tasks refer to tasks that are not executed in time.)

As discussed in Sec III-E, a drawback of POPT is its high computation time for making offloading decisions, as it utilizes standard solvers to solve Problem (9). In Fig. 4(c), we numerically verify this claim: POPT takes more than 1 seconds to obtain the decision for each task. TODORA and OTOSA, in contrast, take less than 1 ms since they are both low-complexity threshold mechanisms, suggesting that they can be operated in practical systems to satisfy the timeliness constraints. In addition, the runtime of all schemes is approximately constant over all scenarios. This is reasonable as TODORA and OTOSA make their decision by examining thresholds, which is independent of the variation of vehicle computing resources. Similarly, as analyzed in Sec. III-E, the complexity for POPT to solve Problem (9) does not depend on vehicle computing resources.

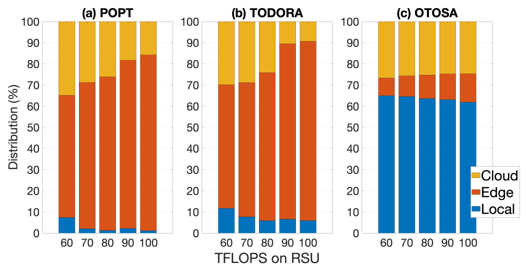


Fig. 7. Task distribution versus FLOPS on RSU: (a) Task distribution of POPT, (b) Task distribution of TODORA, (c) Task distribution of OTOSA.

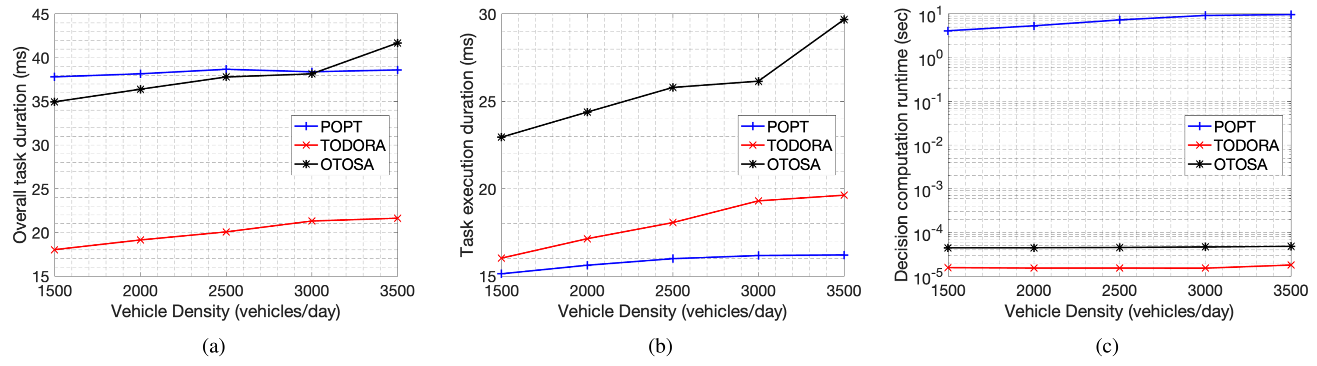


Fig. 8. (a) Average overall task duration versus vehicle density, (b) Average task execution duration versus vehicle density, (c) Decision computation runtime versus vehicle density.

*2) The impact of RSU computing resources:* In this section, we evaluate the impact of RSU computing resources. Each edge server is envisioned to be equipped with multiple server-grade GPUs (e.g., Nvidia RTX A6000). The computing resources of an edge server are then set to $[60-100]$ TFLOPS. As shown in Fig. 6(a), TODORA achieves the best performance in every scenario thanks to better decision-making. As expected, the average task duration of all schemes drops with increasing RSU FLOPS.

Fig. 6(b) shows the task execution duration. As expected, POPT has the least task execution duration. However, as already seen, this performance gain cannot offset the performance loss due to the communication overhead and the periodic mechanism. Also, the decrement of task execution duration of OTOSA is smaller than the other two schemes as OTOSA schedules fewer tasks to edge servers (See Fig. 7).

Looking at Fig. 7, as the RSU resources increase, edge servers execute more tasks, which aligns with expectations. Similar to the previous discussion, TODORA allocates more tasks on vehicles when vehicles have comparable computing capacities with RSUs thanks to its decision mechanism. OTOSA, in contrast, tends to assign a large number of tasks to local vehicles and is more likely to route offloaded tasks to the cloud server instead of edge servers. Since OTOSA schedules fewer tasks on edge servers, it is expected that OTOSA will be less sensitive to the change in edge servers' resources, which is consistent with Fig. 6(a) and Fig. 6(b).

Fig. 6(c) plots the runtime as a function of RSU FLOPS. The trend for all schemes is similar to that in Fig 4(c); that is, changing FLOPS has no impact on their runtime.

*3) The impact of vehicle density:* As the vehicle density grows, there will be more tasks in the system, stretching the availability of resources. Even though the traffic density of highway lanes can be as high as $5000$ (see Sec. VI-A), we show results for a maximum density of $3500$ as higher values make it hard for POPT to converge. Fig. 8(a) shows that, as expected, the average task duration increases with density. TODORA achieves the best performance among all schemes as before. Similarly to before, POPT provides the lowest task execution duration, as we can observe in Fig. 8(b).

As shown in Fig. 9, tasks are executed on local vehicles more frequently as vehicle density increases, likely due to a combination of saturated edge resources and congested communication channels when the system handles a higher number of tasks. Edge servers become unable to support additional tasks when too many vehicles request their resources. Furthermore, the limited wireless bandwidth provided by RSUs results in a significant increase in upload waiting time as more tasks are offloaded. Further analysis reveals that the upload waiting time increases approximately $4.6$ times when vehicle density rises from $1500$ to $3500$ (a factor of $\sim 2.33$).

Fig. 8(c) shows that the computation overhead of POPT increases dramatically with vehicle density. This is because more vehicles and tasks lead to more decision variables, and, as per our discussion in Sec. III-E, the time complexity to solve Problem (9) is proportional to the number of decision variables. This suggests that POPT is not scalable and not suitable for high-vehicle-density areas, such as urban highways. TODORA, on the other hand, shows great scalability.

*4) The impact of task arrival rate:* In the previous experiments, we have kept the task arrival rate per vehicle fixed and varied the vehicle density. Here, we do the opposite. As the task arrival rate increases (i.e., the average inter-arrival time decreases), there will be more tasks in the system. We investigate the performance of TODORA and OTOSA as a function of the average task inter-arrival time. The performance of POPT is not included since the large number of tasks leads to an extremely high time complexity for POPT. Additionally, the delay tolerance constraint (9c) cannot be enforced at a high task arrival rate. As shown in Fig. 10, TODORA still outperforms OTOSA. As expected, the average overall task duration of both schemes increases as the average inter-arrival time decreases. This increase is primarily attributed to the deficiency of communication resources, specifically the limited wireless bandwidth. As a result, the average upload waiting time rises by approximately $9$ ms.

Fig. 11 plots the task distribution of TODORA and OTOSA. Both TODORA and OTOSA tend to execute tasks locally under a short inter-arrival time. This is likely due to resource scarcity on RSUs and the deficiency of the wireless bandwidth. It can be seen that TODORA dramatically increases the proportion of locally-processed tasks under such conditions. This leads to resource contention on vehicles, resulting in longer overall task durations. On the other hand, offloaded tasks

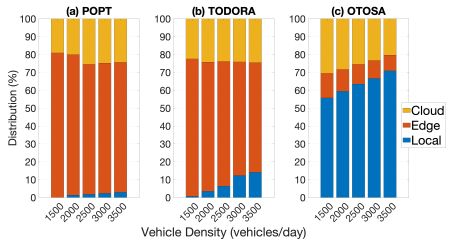


Fig. 9. Task distribution versus vehicle density: (a) Task distribution of POPT, (b) Task distribution of TODORA, (c) Task distribution of OTOSA.

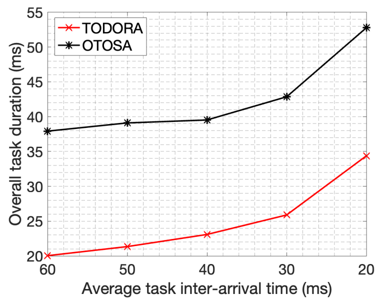


Fig. 10. Average overall task duration versus average task inter-arrival time.

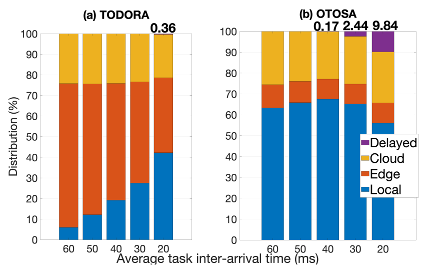


Fig. 11. Task distribution versus average task inter-arrival time: (a) Task distribution of TODORA, (b) Task distribution of OTOSA. (Since some delayed task portions are not clearly visible, we annotate their percentages above the corresponding columns.)

experience a high upload waiting time, which significantly contributes to the overall task duration, as illustrated in Fig. 10.

Another observation is that the percentage of tasks experiencing handoff increases as the inter-arrival time decreases. This is expected because the higher system load leads to longer task durations which increases the chance that a vehicle handoffs to a new RSU before the task finishes. However, even when the inter-arrival time is as short as 20 ms, fewer than $1\%$ of tasks undergo handoff. Moreover, the associated overhead (i.e., the delay incurred when transferring the task result to the newly connected RSU) is minimal, particularly as the overall task duration increases. For TODORA, the handoff overhead is approximately $1$ ms, while the overall task duration is around $34$ ms. Given that no more than $1\%$ of tasks experience handoff, the contribution to overall task duration is bounded by $0.01$ ms, indicating that the impact of handoff is marginal.

TODORA introduces very few delayed tasks when the inter-arrival time is $20$ ms (only $0.36\%$ of tasks are delayed). This is because TODORA continues to allocate tasks to local vehicles, even though the limited computing resources on vehicles cannot support too many tasks simultaneously. Despite this, TODORA successfully satisfies the maximum delay tolerance for more than $99\%$ of tasks. In contrast, OTOSA introduces a sizable number of delayed tasks at high task arrival rates ($0.17\%$, $2.44\%$, and $9.84\%$ of tasks are delayed when the inter-arrival time is $40$, $30$, and $20$ ms, respectively). The superior performance of TODORA can be attributed to reduced communication overhead and more effective offloading decisions, which increase system capacity and enable higher input rates to be processed without exceeding resource limits. Note that while OTOSA also increases the number of tasks executed locally as the inter-arrival time decreases, the portion of locally processed tasks appears to decline under extremely short inter-arrival times (i.e., less than $30$ ms), as seen in Fig. 11. This apparent mismatch occurs because most delayed tasks were originally designated for local execution, but resource scarcity on vehicles causes them to turn into delayed tasks.

*5) The impact of the number of RSUs:* In this section, we evaluate the impact of the number of RSUs. As shown in Fig. 12, TODORA outperforms both POPT and OTOSA across all scenarios, highlighting its scalability. Notably, the overall task duration for TODORA and POPT remains nearly constant, whereas the duration for OTOSA increases with the number of RSUs. This is due to their scheduling strategies: TODORA and POPT primarily schedule tasks on edge servers. Increasing the number of RSUs leads to more vehicles and tasks, resulting in more workloads, but workloads are distributed across more RSUs, keeping the task duration roughly constant. In contrast, OTOSA tends to offload tasks to the cloud server. With more RSUs, the cloud server handles more workloads, leading to longer task durations.

Fig. 13 shows that the computation overhead of POPT increases sharply with the number of RSUs, while that of TODORA remains constant. This observation is consistent with our analysis in Secs. III-E and IV. The time complexity of POPT is proportional to $U$, while TODORA is an $O(1)$ algorithm. This demonstrates the scalability of TODORA.

*6) The impact of task dependency:* So far, we have assumed all tasks are independent: Each task may start execution as soon as it is generated. However, there may be situations where a task execution depends on the output of a previously executed task. In this section, we investigate the performance of TODORA and OTOSA over dependent tasks via simulations. Two scenarios are evaluated: (i) the previous scenario with independent tasks only (Scenario 1) and (ii) a scenario with both independent and dependent tasks (Scenario 2).

Consider two sets of tasks: The first consists of tasks that can be executed independently, whereas the second consists of dependent tasks that must be scheduled after a previous task is completed. The dependency among the tasks follows a simple sequential task topology of two tasks (i.e., one task is the independent task, and the other is the dependent task), and tasks are generated in pairs. To be consistent with prior experiments, the arrival rate of pairs of tasks is set to half the arrival rate of independent tasks in prior experiments. Additionally, both TODORA and OTOSA schedule a dependent task after receiving the output of the corresponding independent task, and the delay tolerance for a dependent task corresponds to the maximum delay tolerance after it can be scheduled.

Fig. 14 plots the average overall task duration of Scenario 1 (solid lines) and Scenario 2 (dotted lines) for TODORA and OTOSA, as a function of vehicle density. Similarly, TODORA outperforms OTOSA in both scenarios.

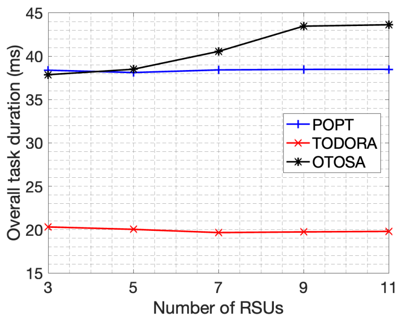


Fig. 12. Average overall task duration versus the number of RSUs.

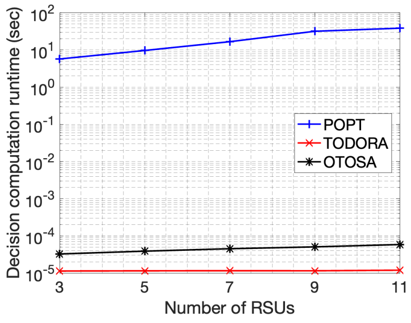


Fig. 13. Decision computation runtime versus the number of RSUs.

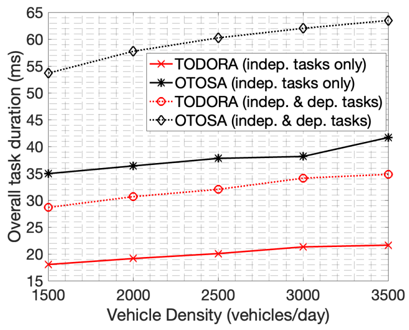


Fig. 14. Average overall task duration of Scenario 1 (indep. tasks only) and of Scenario 2 (indep. & dep. tasks) versus vehicle density.

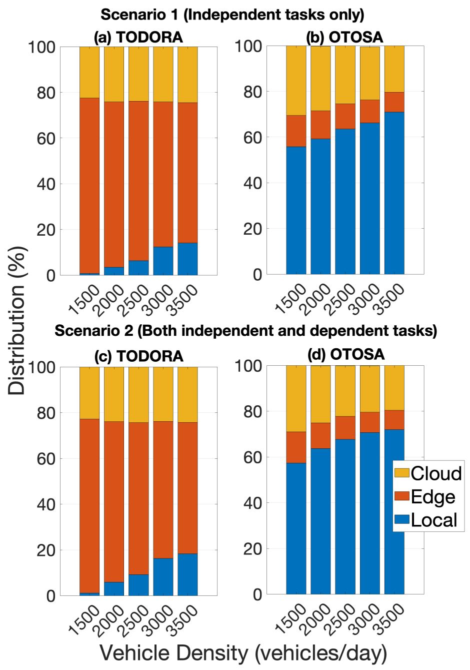


Fig. 15. Task distribution of Scenario 1 (Independent tasks only) and of Scenario 2 (Both independent and dependent tasks) versus vehicle density: (a) Task distribution of TODORA in Scenario 1, (b) Task distribution of OTOSA in Scenario 1, (c) Task distribution of TODORA in Scenario 2, (d) Task distribution of OTOSA in Scenario 2.

As expected, the average overall task duration of Scenario 2 is longer for both TODORA and OTOSA, as dependent tasks have to wait for the termination of a previous task before being scheduled, leading to a longer overall task duration. Specifically, the waiting time equals the average overall task duration of the independent tasks. Additionally, the average duration of the dependent tasks themselves closely mirrors that of the independent tasks. As a result, the average overall task duration of the dependent tasks will be around twice that of the independent tasks. With $50\%$ of tasks being dependent tasks, the average overall task duration of Scenario 2 will be approximately 1.5 times that of Scenario 1.

Fig. 15 compares the task distribution of Scenario 1 versus that of Scenario 2 for TODORA and OTOSA. As observed, the task distribution of both scenarios is similar for both TODORA and OTOSA. This is likely because the dependency constraint primarily affects the time that dependent tasks can be scheduled, introducing an additional waiting time for dependent tasks. On the other hand, the task attributes of dependent tasks remain consistent with those of independent tasks, resulting in similar task distribution in both scenarios.

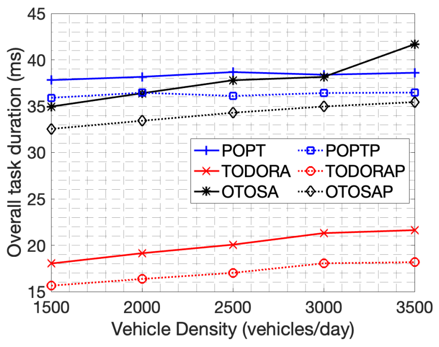


Fig. 16. Average overall task duration of Binary Offloading and of Partial Offloading versus vehicle density.

*7) The impact of partial offloading:* Thus far, we have focused on binary offloading, where tasks are assumed to be indivisible and must be fully executed locally or fully offloaded. However, in some cases, tasks can be divided into smaller subtasks and partially offloaded to remote servers. In this section, we simulate the partial offloading schemes introduced in Sec. V (i.e., TODORAP and POPTP). Additionally, we slightly modify OTOSA (referred to as OTOSAP) to support partial offloading. We compare these schemes with their counterparts designed for binary offloading (i.e., TODORA, OTOSA, and POPT). Fig. 16 shows the average overall task duration for binary offloading (solid lines) and partial offloading (dotted lines) under varying vehicle densities. First, focusing on partial offloading schemes, TODORAP outperforms all other schemes as it reduces communication overhead and eliminates periodic losses.

Additionally, we compare partial offloading with binary offloading across all schemes. As expected, partial offloading yields shorter task durations for all schemes due to the advantage of parallel execution: with partial offloading, a task can be split into smaller subtasks based on the available resources on the vehicle, RSU, and cloud server, thereby better utilizing the available resources. Note that, in our work, the reported task duration for partial offloading includes the overhead associated with task splitting and result synthesis, which is often neglected in prior studies (e.g., [14]). As shown in Fig. 16, the overall task duration exceeds 15 ms even for TODORA, while the associated overhead is only 0.9 ms. (This overhead is measured based on experiments conducted on our Threadripper 3970X server.) This indicates that the overhead accounts for a small portion of the overall task duration, underscoring the practicality of partial offloading.

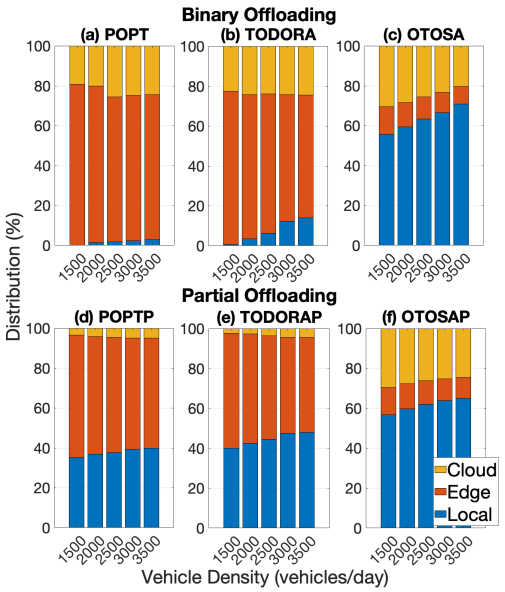


Fig. 17. Task distribution of Binary Offloading and of Partial Offloading versus vehicle density: (a) Task distribution of POPT, (b) Task distribution of TODORA, (c) Task distribution of OTOSA, (d) Task distribution of POPTP, (e) Task distribution of TODORAP, (f) Task distribution of OTOSAP.

Fig. 17 compares the task distribution between binary offloading and partial offloading.[3] As shown in the figure, a larger portion of tasks is executed locally under partial offloading. This is likely because, even if vehicle computation resources are insufficient to complete an entire task quickly, they can still assist in executing portions of the task, thereby reducing overall task duration. Similar to the binary offloading scenario, all schemes tend to allocate more tasks locally to avoid increased upload waiting times as vehicle density increases. Interestingly, compared to TODORA under binary offloading, TODORAP achieves a task distribution more similar to the periodic optimal solution (i.e., POPTP). This indicates that TODORAP provides decisions closer to the optimal solution than TODORA.

## VII. Discussion

In this work, we propose a heuristic algorithm for task offloading and resource allocation with the aim of minimizing the average task duration of ML tasks. TODORA provides a favorable performance as shown in Sec. VI-C.

[3]In partial offloading, a task may be split and executed across multiple locations. If task $n$'s partial offloading decision is $\mathcal{S}_n = (0.4, 0.2, 0.4)$, it contributes to 0.4 local tasks, 0.2 edge tasks, and 0.4 cloud tasks.

Limitations to be further investigated in future work include the following: First, in this work, we focus on GPU when considering the allocation of compute resources. However, executing ML tasks requires various types of resources, such as CPU, GPU, and memory. To this end, multi-resource allocation may be useful. Second, we assume that we know the exact number of required FLOPs. In practice, however, ML tasks generally have elastic resource demands, and it may not be possible to know the exact demands or even the distribution of the demands in advance. To this end, an efficient and effective method to estimate the demands in an online fashion is another area that needs further exploration; see, for example, [48]. Last, while our simulation-based evaluation is insightful and grounded in real-world ML tasks and datasets, there are several potential limitations when transitioning TODORA to real-world deployment. For instance, simulations do not fully capture the computation overhead. Although we numerically demonstrate that the decision runtime of TODORA is extremely low, in real-world deployment, this runtime may fluctuate due to varying workloads on the server where the scheduler is implemented. Additionally, while our network setup accounts for the uncertainty of vehicular networks by incorporating packet loss, it only considers network load from vehicles. In practice, network traffic from other devices may also impact vehicular network performance. As a result, real-world systems may require further experimental validation using a testbed and real-world tasks before deployment. However, conducting such large-scale evaluations is often impractical within an academic setting.

## VIII. Conclusion

To meet the stringent timing demands of in-vehicle ML tasks, we propose an online, distributed approach called TODORA that optimizes offloading and resource allocation to reduce task duration. Simulation results show that TODORA outperforms both periodic optimal and state-of-the-art methods.

## References

[1] ultralytics/yolov5: v5.0 - yolov5-p6 1280 models, aws, supervise.ly andyoutubeintegrations. [Online]. Available: https://zenodo.org/records/7347926

[2] B. Cheng, A. Schwing, and A. Kirillov, "Per-pixel classification is not all you need for semantic segmentation," in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 17 864–17 875.

[3] S.-C. Lin *et al.*, "The architectural implications of autonomous driving: Constraints and acceleration," *SIGPLAN Not.*, vol. 53, no. 2, p. 751–766, mar 2018.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, 2016.

[5] W. Yu *et al.*, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[6] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proc. IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019.

[7] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile Netw. Appl.*, vol. 26, no. 3, p. 1145–1168, 2021.

[8] M. S. Bute, P. Fan, G. Liu, F. Abbas, and Z. Ding, "A Collaborative Task Offloading Scheme in Vehicular Edge Computing," *2021 IEEE 93rd Veh. Technol. Conf. (VTC2021-Spring)*, vol. 00, pp. 1–5, 2021.

[9] Y. Hui, Z. Su, T. H. Luan, C. Li, G. Mao, and W. Wu, “A Game Theoretic Scheme for Collaborative Vehicular Task Offloading in 5G HetNets,” *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 16 044–16 056, 2020.

[10] L. Liu, J. Feng, X. Mu, Q. Pei, D. Lan, and M. Xiao, “Asynchronous Deep Reinforcement Learning for Collaborative Task Computing and On-Demand Resource Allocation in Vehicular Edge Computing,” *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 15 513–15 526, 2023.

[11] E. Karimi, Y. Chen, and B. Akbari, “Task offloading in vehicular edge computing networks via deep reinforcement learning,” *Computer Communications*, vol. 189, pp. 193–204, 2022.

[12] Y. Hou, Z. Wei, R. Zhang, X. Cheng, and L. Yang, “Hierarchical Task Offloading for Vehicular Fog Computing Based on Multi-Agent Deep Reinforcement Learning,” *IEEE Trans. Wireless Commun.*, vol. PP, no. 99, pp. 1–1, 2023.

[13] P. Qin, Y. Fu, G. Tang, X. Zhao, and S. Geng, “Learning Based Energy Efficient Task Offloading for Vehicular Collaborative Edge Computing,” *IEEE Trans. Veh. Technol.*, vol. 71, no. 8, pp. 8398–8413, 2022.

[14] W. Fan *et al.*, “Game-Based Task Offloading and Resource Allocation for Vehicular Edge Computing With Edge-Edge Cooperation,” *IEEE Trans. Veh. Technol.*, vol. 72, no. 6, pp. 7857–7870, 2023.

[15] J. Su, Z. Liu, Y.-a. Xie, Y. Li, K. Ma, and X. Guan, “UEE-Delay Balanced Online Resource Optimization for Cooperative MEC-enabled Task Offloading in Dynamic Vehicular Networks,” *IEEE Internet Things J.*, vol. PP, no. 99, pp. 1–1, 2023.

[16] Z. Zhang and F. Zeng, “Efficient Task Allocation for Computation Offloading in Vehicular Edge Computing,” *IEEE Internet Things J.*, vol. 10, no. 6, pp. 5595–5606, 2023.

[17] S. Li, N. Zhang, H. Chen, S. Lin, O. A. Dobre, and H. Wang, “Joint Road Side Units Selection and Resource Allocation in Vehicular Edge Computing,” *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13 190–13 204, 2021.

[18] Y.-J. Ku, S. Baidya, and S. Dey, “Adaptive Computation Partitioning and Offloading in Real-Time Sustainable Vehicular Edge Computing,” *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13 221–13 237, 2021.

[19] P. Sun *et al.*, “Scalability in perception for autonomous driving: Waymo open dataset,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, June 2020.

[20] Y. Chen, F. Zhao, X. Chen, and Y. Wu, “Efficient Multi-Vehicle Task Offloading for Mobile Edge Computing in 6G Networks,” *IEEE Trans. Veh. Technol.*, vol. 71, no. 5, pp. 4584–4595, 2022.

[21] H. D. Nguyen, S. Aoki, Y. Nishiyama, and K. Sezaki, “A Run-time Dynamic Computation Offloading Strategy in Vehicular Edge Computing,” *2021 IEEE 94th Veh. Technol. Conf. (VTC2021-Fall)*, vol. 00, pp. 1–7, 2021.

[22] 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, 2023.

[23] Z. Nan, S. Zhou, Y. Jia, and Z. Niu, “Joint Task Offloading and Resource Allocation for Vehicular Edge Computing With Result Feedback Delay,” *IEEE Trans. Wireless Commun.*, vol. 22, no. 10, pp. 6547–6561, 2023.

[24] S. Raza, S. Wang, M. Ahmed, M. R. Anwar, M. A. Mirza, and W. U. Khan, “Task Offloading and Resource Allocation for IoV Using 5G NR-V2X Communication,” *IEEE Internet Things J.*, vol. 9, no. 13, pp. 10 397–10 410, 2022.

[25] Q. Luo, C. Li, T. H. Luan, W. Shi, and W. Wu, “Self-Learning Based Computation Offloading for Internet of Vehicles: Model and Algorithm,” *IEEE Trans. Wireless Commun.*, vol. 20, no. 9, pp. 5913–5925, 2021.

[26] Q. Luo, C. Li, T. H. Luan, and W. Shi, “Minimizing the Delay and Cost of Computation Offloading for Vehicular Edge Computing,” *IEEE Trans. Services Comput.*, vol. 15, no. 5, pp. 2897–2909, 2022.

[27] K. Tan, L. Feng, G. Dn, and M. Trngren, “Decentralized Convex Optimization for Joint Task Offloading and Resource Allocation of Vehicular Edge Computing Systems,” *IEEE Trans. Veh. Technol.*, vol. 71, no. 12, pp. 13 226–13 241, 2022.

[28] M. Z. Alam and A. Jamalipour, “Multi-Agent DRL-Based Hungarian Algorithm (MADRLHA) for Task Offloading in Multi-Access Edge Computing Internet of Vehicles (IoVs),” *IEEE Trans. Wireless Commun.*, vol. 21, no. 9, pp. 7641–7652, 2022.

[29] Z. Liu, P. Dai, H. Xing, Z. Yu, and W. Zhang, “A Distributed Algorithm for Task Offloading in Vehicular Networks With Hybrid Fog/Cloud Computing,” *IEEE Trans. Syst., Man, Cybern*, vol. 52, no. 7, pp. 4388–4401, 2022.

[30] M. S. Bute, P. Fan, L. Zhang, and F. Abbas, “An Efficient Distributed Task Offloading Scheme for Vehicular Edge Computing Networks,” *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13 149–13 161, 2021.

[31] “Ts 36.201- evolved universal terrestrial radio access (e-utra); lte physical layer; general description,” *Technical report*, 2017.

[32] H. Qiu, P.-H. Huang, N. Asavisanu, X. Liu, K. Psounis, and R. Govindan, “Autocast: scalable infrastructure-less cooperative perception for distributed collaborative driving,” in *Proc. 20th Annu. Int. Conf. Mobile Systems, Applications and Services*, 2022, p. 128–141.

[33] D. Xu *et al.*, “Tutti: coupling 5g ran and mobile edge computing for latency-critical video analytics,” in *Proc. 28th Annu. Int. Conf. Mobile Computing And Networking*, 2022, p. 729–742.

[34] M. Gonzalez-Martín, M. Sepulcre, R. Molina-Masegosa, and J. Gozalvez, “Analytical models of the performance of c-v2x mode 4 vehicular communications,” *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1155–1166, 2019.

[35] J. Meng, Z. J. Kong, Y. C. Hu, M. G. Choi, and D. Lal, “Do we need sophisticated system design for edge-assisted augmented reality?” in *Proc. 5th Int. Workshop on Edge Systems, Analytics and Networking*, 2022, p. 7–12.

[36] “World record optical fiber transmission capacity doubles to 22.9 petabits per second,” *Technical report*, 2023. [Online]. Available: https://www.nict.go.jp/en/press/2023/11/30-1.html

[37] P. Virtanen *et al.*, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, vol. 17, pp. 261–272, 2020.

[38] S. Diamond and S. Boyd, “CVXPY: A Python-embedded modeling language for convex optimization,” *J. Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[39] Z.-q. Luo, W.-k. Ma, A. M.-c. So, Y. Ye, and S. Zhang, “Semidefinite relaxation of quadratic optimization problems,” *IEEE Signal Process. Mag.*, vol. 27, no. 3, pp. 20–34, 2010.

[40] S. M. Ross, *Introduction to Probability Models*, 6th ed. San Diego, CA, USA: Academic Press, 1997.

[41] G. M. N. Ali, P. H. J. Chong, S. K. Samantha, and E. Chan, “Efficient data dissemination in cooperative multi-RSU vehicular ad hoc networks (VANETs),” *J. Syst. Softw.*, vol. 117, pp. 508–527, 2016.

[42] “Intelligent transport systems (its); vehicular communications; basic set of applications; part 2: Specification of cooperative awareness basic service,” *Technical report*, 2014.

[43] National traffic speeds survey iii: 2015. [Online]. Available: https://www.nhtsa.gov/sites/nhtsa.gov/files/documents/812485_national-traffic-speeds-survey-iii-2015.pdf

[44] Highway statistics series. [Online]. Available: https://www.fhwa.dot.gov/policyinformation/statistics/2021/hm62.cfm

[45] P. Voigtlaender, J. Luiten, P. H. Torr, and B. Leibe, “Siam r-cnn: Visual tracking by re-detection,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, June 2020.

[46] Y. Xu, J. Zhang, Q. Zhang, and D. Tao, “Vitpose++: Vision transformer for generic body pose estimation,” *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 46, no. 2, pp. 1212–1230, 2024.

[47] Y. He, L. Ma, Z. Jiang, Y. Tang, and G. Xing, “Vi-eye: semantic-based 3d point cloud registration for infrastructure-assisted autonomous driving,” in *Proc. the 27th Annu. Int. Conf. Mobile Computing and Networking*. New York, NY, USA: Association for Computing Machinery, 2021, p. 573–586.

[48] B. Herzog, S. Reif, J. Hemp, T. Hönig, and W. Schröder-Preikschat, “Resource-demand estimation for edge tensor processing units,” *ACM Trans. Embed. Comput. Syst.*, vol. 21, no. 5, oct 2022.

**Te-Yi Kan** received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, in 2019, and the M.S. degree in electrical and computer engineering from the University of California, CA, USA, in 2020. From 2020 to 2022, he was a Research Assistant with the Academia Sinica, Taiwan. He is currently a Ph.D. student in electrical and computer engineering at the University of Southern California. His research interests include wireless communication systems, wireless networks, vehicular networking, and edge computing.

**Konstantinos Psounis** received the BS degree from the Department of Electrical and Computer Engineering, National Technical University of Athens, Greece, in 1997, and the M.S. and Ph.D. degrees in Electrical Engineering from Stanford, CA, USA, in 1999 and 2002, respectively. He is a Professor of Electrical and Computer Engineering and of Computer Science with the University of Southern California. He is an IEEE Fellow and a Distinguished Member of the ACM.