UNIVERSITY OF CALIFORNIA SAN DIEGO

Enabling Delay-Sensitive Modern Vehicular Applications by Using Resource Limited Vehicular Edge Computing Systems

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Electrical Engineering - Communication Theory & Systems

by

Yu-Jen Ku

Committee in charge:

Professor Sujit Dey, Chair
Professor Dinesh Bharadia
Professor Farinaz Koushanfar
Professor Xinyu Zhang
Professor Jishen Zhao

2023

The Dissertation of Yu-Jen Ku is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First of all, I would like to express my deepest appreciation to my advisor Professor Sujit Dey for his professional guidance and feedback on this study and for his support as the chair of my committee. His enthusiasm for scientific research has inspired me throughout my doctoral journey.

I would also like to thank the rest of my committee members, Professor Dinesh Bharadia, Professor Farinaz Koushanfar, Professor Xinyu Zhang, and Professor Jishen Zhao, for their insightful comments and suggestions.

Besides my committee, I would like to express my sincere gratitude to Professor Sabur Baidya from the University of Louisville for his collaborative effort and support during my thesis study. I will never forget the days when we had endless late-night meetings and experiments in the lab. Many thanks to Dr. Po-Han Chiang too, who brought me to this lab and helped me to complete the first part of this study. I am also grateful to all my lab mates for all the great projects that we have done during these years.

To conclude, I cannot forget to thank my family and friends for all the unconditional support they gave throughout my doctoral study. I would not have been able to complete my thesis without them.

Chapter 1, in full, is a reprint of the material as it appears in IEEE Transactions on Vehicular Technology 2020, Yu-Jen Ku, Po-Han Chiang, and Sujit Dey. The dissertation author was the primary investigator and author of this paper.

Chapter 2, in full, is a reprint of the material as it appears in IEEE Transactions on Vehicular Technology 2021, Yu-Jen Ku, Sabur Baidya, and Sujit Dey. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, has been accepted for publication of the material as it may appear in IEEE Transactions on Vehicular Technology 2023, Yu-Jen Ku, Sabur Baidya, and Sujit Dey. The dissertation author was the primary investigator and author of this paper.

VITA

2014	Bachelor of Science, National Taiwan University
2015-2016	Research Assistant, National Taiwan University
2016–2022	Research Assistant, University of California San Diego
2023	Doctor of Philosophy, University of California San Diego

PUBLICATIONS

Y. -J. Ku, S. Baidya and S. Dey, "Uncertainty-aware Task Offloading for Multi-vehicle Perception Fusion over Vehicular Edge Computing," to appear in *IEEE Transactions on Vehicular Technology*

Y. -J. Ku, B. Flowers, S. Thornton, S. Baidya and S. Dey, "Adaptive C-V2X Sidelink Communications for Vehicular Applications Beyond Safety Messages," *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, Helsinki, Finland, 2022, pp. 1-6

Y. -J. Ku, S. Baidya and S. Dey, "Adaptive Computation Partitioning and Offloading in Real-Time Sustainable Vehicular Edge Computing," in *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 13221-13237, Dec. 2021

Y. -J. Ku, S. Sapra, S. Baidya and S. Dey, "State of Energy Prediction in Renewable Energy-driven Mobile Edge Computing using CNN-LSTM Networks," *2020 IEEE Green Energy and Smart Systems Conference (IGESSC)*, Long Beach, CA, USA, 2020, pp. 1-7

S. Baidya, Y. -J. Ku, H. Zhao, J. Zhao and S. Dey, "Vehicular and Edge Computing for Emerging Connected and Autonomous Vehicle Applications," *2020 57th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2020, pp. 1-6

Y. -J. Ku, P. -H. Chiang and S. Dey, "Real-Time QoS Optimization for Vehicular Edge Computing With Off-Grid Roadside Units," in *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 11975-11991, Oct. 2020

Y. -J. Ku and S. Dey, "Sustainable Vehicular Edge Computing Using Local and Solar-Powered Roadside Unit Resources," *2019 IEEE 90th Vehicular Technology Conference (VTC-Fall)*, Honolulu, HI, USA, Sep. 2019, pp. 1-7

Y. -J. Ku, P. -H. Chiang and S. Dey, "Quality of Service Optimization for Vehicular Edge Computing with Solar-Powered Road Side Units," *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, Hangzhou, China, 2018, pp. 1-10

ABSTRACT OF THE DISSERTATION

Enabling Delay-Sensitive Modern Vehicular Applications by Using Resource Limited Vehicular Edge Computing Systems

by

Yu-Jen Ku

Doctor of Philosophy in Electrical Engineering - Communication Theory & Systems

University of California San Diego, 2023

Professor Sujit Dey, Chair

This study aims at using Road-side Unit (RSU)-assisted Vehicular Edge Computing (VEC) systems to support nowadays compute-intensive and delay-sensitive vehicular applications. Vehicles can offload these applications to the VEC server at the RSU to ease the burden on their limited onboard computing resources. Although VEC servers usually have higher computing capacity than vehicles, their computing resource, as well as the RSU's communication and energy resources, are not unlimited due to deployment constraints and operating costs. Our goal is to find the optimal resource allocation strategies for the RSU-assisted VEC systems to enable low-latency compute-intensive vehicular applications for vehicles.

In the first part of the study, we consider Solar-powered RSU-assisted VEC systems, where the RSUs are solely powered by solar energy. Firstly, we aim to minimize service disruption of the offloaded vehicular applications under the intermittent solar power supply. We propose a two-phase approach that jointly optimizes solar energy usage and storage, user association, and RSU's computing and communication resource allocation for the involved RSUs and vehicles in the VEC system. Secondly, we further reduce the application's execution delay by a framework that uses computing resources from both the VEC server and the vehicle's local computing (VLC) unit for application execution through task partitioning and offloading. Furthermore, the framework is able to adjust the VEC server's platform configuration and balance between the vehicular application's execution delay and its application-level performance, according to the available computing, communication, and solar energy resources of the Solar-powered RSU.

In the second part of the study, we focus on using the RSU-assisted VEC system to support an emerging advanced vehicular application, the multi-vehicle perception fusion. It is challenging to effectively allocate RSU's computing and communication resources to support the multi-vehicle perception fusion application due to its complex and uncertain task composition natures. To minimize the end-to-end delay of the above application, we present a real-time mechanism that jointly determines the optimal RSU's computing and communication resource allocation, as well as task partitioning and scheduling strategies, which are adaptive to the dynamic task composition of the application.

Introduction

Advanced Driver Assistance Systems (ADAS) in modern vehicles involve multiple vehicular applications, many of which are compute-intensive and delay-sensitive Artificial Intelligence (AI) driven algorithms. Performing these complex applications on some vehicles may not be feasible due to the application's high computing complexity and the vehicle's limited computing capacity. A Road-side Unit (RSU)-assisted Vehicular Edge Computing (VEC) system, where the RSUs are equipped with small cell base stations (SBSs) and VEC servers, can be used to support these applications through task offloading. However, the computing, communication, and energy resources of RSU-assisted VEC system might be limited due to deployment constraints and operating costs. It is important to study how to effectively utilize these resources to support compute-intensive and delay-sensitive vehicular applications for vehicles.

In Chapter 1, we consider a VEC system consisting of Solar-powered RSUs to support vehicular applications from multiple vehicles, where RSUs are solely powered by solar energy. Vehicles can offload the applications to the Solar-powered RSUs for low-latency application executions. Although Solar-powered RSUs are promising as they are sustainable and easy to deploy, they may suffer from a high risk of power deficiency due to the intermittent nature of solar power. We aim to reduce the service disruption for the offloaded continuous delay-sensitive vehicular applications due to the sporadic solar power supply. We propose a two-phase approach that jointly optimizes solar energy usage and storage, user association, as well as computing and communication resource allocation decisions to minimize service disruption. Our study shows the effectiveness of the proposed approach to reduce service disruption in a dense Solar-powered

RSU environment, which is simulated with real-world urban vehicular traffic data and solar generation profile.

In Chapter 2, we further consider the potential improvement of the application's execution delay by dynamically adjusting the vehicular application-level performance (e.g., object detection accuracy) and by jointly utilizing the computing resources from both the VEC server and the vehicular local computing (VLC) units on the involved vehicles through task partitioning and offloading. We also consider adaptively changing the system configuration of the VEC server according to the available solar power supply. We aim to minimize the end-to-end delay of the vehicular application while maximizing its application-level performance. We propose a dynamic programming-based algorithm that jointly makes the task partitioning and offloading, as well as system and application-level performance adaptation decisions, in real-time. To be more realistic, we implement an object detection vehicular application on a low-power edge computing platform and establish corresponding empirical models for the energy consumption, execution delay, and application-level performance (i.e., object detection accuracy) of the above Solar-powered RSUs-assisted VEC system. Analysis based on the empirical models shows the ability of our proposed approach to minimize the application's end-to-end delay while maximizing its detection accuracy.

In the last chapter, we study the problem of using an RSU-assisted VEC system to support a more comprehensive vehicular application, a multi-vehicle perception fusion application, which involves multiple vehicles' individual vehicular perception tasks and additional fusion tasks at the end. Achieving this edge-collaborated perception fusion with task partitioning and offloading involves multiple challenges: (i) frequently varying uplink channel conditions, (ii) sequential and parallel task dependencies, and (iii) uncertain task composition induced by unknown object detection outputs in the dynamic driving environment. We propose a real-time mechanism to minimize the end-to-end delay of the application by addressing these challenges. Based on the real-time channel conditions, VEC and VLC server capacities, and the number of detected objects, the proposed mechanism jointly determines the bandwidth allocation, task partitioning,

offloading, and execution scheduling for all the involved tasks of the multi-vehicle perception fusion application. Numerical results with real-world traffic data simulation show that the proposed approach significantly reduces the end-to-end delay of the fusion application compared to existing techniques.

Chapter 1

Real-time QoS Optimization for Vehicular Edge Computing with Off-Grid Roadside Units

1.1 Introduction

This chapter presents our work to study service disruption minimization for offloading vehicular applications to solar-powered Roadside Units (RSUs)-assisted Vehicular Edge Computing (VEC) systems. RSUs equipped with small cell base stations (SBSs) are evolving as a key infrastructure to support connected vehicles. Due to the low latency and high throughput, communications provided by SBSs to connected vehicles, RSUs can enable or extend various vehicular applications, such as autonomous driving, road safety, infotainment, and collaboration services [1]. Further, when augmented with Mobile Edge Computing (MEC) servers, the RSUs can fulfill the computation-intensive needs of vehicular applications, while maintaining low latency, through offloading vehicle users' (VUs') computing tasks to RSUs. The scenario has been defined in literature as *Vehicular Edge Computing* [2, 3].

In 2020, SBSs are projected to consume 4.4 TWh of energy and emit 2.3 million tons of carbon dioxide equivalent (CO_{2e}) [4, 5]. Furthermore, dense deployments of RSUs are expected in order to support the massive growth of emerging connected vehicles and their high throughput requirements [6], leading to further power consumption and carbon emissions. One promising

solution is the use of renewable energy (RE) in wireless communications [7]. In order to enhance the sustainability of RSUs by easing their grid power consumption, we proposed the idea of Solar-powered Roadside Units (SRSUs) in [8], which consist of SBS, MEC, and a self-sustained solar system.

The main challenge of adopting RE in an SRSU network is the intermittent and fluctuating nature of RE (i.e., solar energy) generation [9]. RE-powered VEC must consider the SRSU's communication and computing resources as opportunistic due to the intermittent harvested RE. Further, RE-powered VEC must also consider the VU's high mobility and low application latency requirement. In this work, we consider that VUs offload their applications (e.g., object recognition and collision prediction using camera or lidar data) to the MEC server of the associated SRSU. For these time-sensitive and computation-intensive applications, VUs will send the raw data to SRSU and receive the processed results with ultra-low latency. Such applications will inevitably suffer from service degradation when the communication and/or computing capacity of SRSU is limited. In this work, we aim to minimize Quality of Service (QoS) loss in a dense SRSU network. We define QoS loss as a weighted sum of instances of (i) service outage (when no SRSU can serve the VU) and (ii) service disruption (when the VU is handed over to another SRSU), over total number of VUs.

In our preliminary work [8], we proposed an offline QoS Loss Minimization Algorithm (QLM) to heuristically minimize the weighted QoS loss using SRSUs. However, QLM assumes accurate predictions of SRSUs' solar generations and VUs' offloading demands. The impact of prediction error on the performance of QLM was not discussed. Moreover, the offline solution provided by QLM cannot adapt to dynamic solar generation and offloading demands. Finally, QLM assumes unlimited battery capacity in order to provide an analytic solution, which is not viable in real-world SRSU deployment.

In this work, given: (i) predictions of SRSUs' solar generations and power consumptions, (ii) current VUs' locations, wireless channel conditions, and offloading demands, and (iii) current SRSUs' stored energy, communication, and computing resources, we propose to jointly solve

solar energy scheduling, VU-SRSU association, and SRSU resource allocation problems. We propose to solve this problem in two phases: (i) solar energy scheduling phase, which determines battery charging/discharging for SRSUs in advance in order to schedule the available solar energy in each time slot, and (ii) user association and resource allocation phase, which decides VU-SRSU association and SRSU resource allocation in real-time to minimize the weighted QoS loss, based on the available energy determined from the first phase. Compared to QLM, the proposed solution adapts the solar generations and offloading demands dynamically in real-time. Our simulation results show that this approach produces up to a 54% reduction in the weighted QoS loss compared to our preliminary work in [8].

The contributions of this paper are summarized as follows:

(a) To the best of our knowledge, this is the first work to address the problem of using SRSUs in vehicular edge computing. Specifically, the paper considers the problem of SRSU edge computing and small cell communication resource allocation problems given the real-time offloading demands of the fast moving VUs as well as the limited solar energy availabilities of SRSUs.

(b) For the first time, service outage incurred when no SRSU can serve a VU and service disruption caused by VU handover between SRSUs are considered in defining QoS. We propose a weighted QoS objective function to incorporate preference between these two factors.

(c) To optimize the weighted QoS, we propose a two-phase approach consisting of an offline solar energy scheduling (battery charging/discharging scheduling) phase and an online user association and SRSU resource allocation phase. The proposed approach is real-time adaptive to offloading demands, locations, and channel conditions of VUs, as well as SRSU resource availabilities.

(d) To demonstrate the feasibility and effectiveness of the proposed technique, we develop a

simulation framework consisting of real-world solar generation [10], urban traffic profiles [11], and offloading demands. The simulation results show that the proposed approach significantly reduces the weighted QoS loss compared to existing techniques.

The rest of the paper is organized as follows. We review the related work in Section 1.2. In Section 1.3, the overview of our system model and problem formulation is presented. In Section 1.4 we introduce the proposed two-phase approach. The simulation results are presented in Section 1.5 and we conclude in Section 1.6.

1.2 Related Work

There have been various studies addressing either RE-powered wireless communication system [12–14] or RE-powered edge and cloud server network [15, 16]. However, they do not jointly consider both wireless communication and edge computing resources. For RE-powered MEC system, to jointly consider these resources while using RE as the only power supply, Mao et al. [17] address the fluctuating RE challenges for computation task offloading between a single BS-user link. Xu et al. [18, 19] characterize multiple aspects of RE-powered MEC system by Markov Decision Process (MDP) states and propose an online learning-based algorithm to minimize system delay, battery depreciation, and backup power supply cost. The above techniques only consider single-BS scenario, while our work considers load-balancing and intercell interference in the multi-BS scenario.

[20–22] address the challenges of RE-powered multi-BS system, where each BS is equipped with a MEC server. [20] and [21] provide online solutions to control MEC capacity based on Lyapunov optimization [23]. In [20], Chen et al. aim at minimizing system delay through workload balancing among BSs under their long-term energy availability constraint, which does not consider the real-time availability of RE. In [21], Wu et al. minimize the drop rate of computation task and downlink data traffic due to excessive delay or lack of RE. The authors propose a workload balancing and data traffic admission control solution. However, they

model the computation task and the downlink data traffic separately. In VEC, delay constraint of vehicular applications usually jointly constrains both task execution and data transmission delay. Therefore, in this work, we consider a joint delay constraint consisting of execution and transmission delay. In [22], Gou et al. maximize the number of offloading users by an algorithm that iteratively decides SBS coverage, channel allocation, and MEC computing allocation. However, compared to our proposed technique, the iterative nature of the solution is not real-time adaptive to the current RE availability, VU traffic, and offloading demand.

The above studies do not consider challenges specific to characteristics of VUs, such as high mobility, fast-changing channel condition, and ultra-low delay constraint. On the contrary, RE-powered Vehicle-to-Everything (V2X) studies [24–26] take these VU characteristics into consideration. Yang et al. [24] and Atoui et al. [25, 26] both consider a straight stretch of road with RE-powered RSU deployed along it. Based on vehicles' locations and velocities, they schedule the uplink [24] and downlink [25, 26] data transmission between BSs and vehicles to maximize both network throughput [24] and the number of served vehicles [25,26]. These studies focus on data transmission and do not consider the challenges for computation task offloading in VEC. Also, these studies require vehicle to buffer the data and transmit at the scheduled time slot, which is not feasible for time-sensitive vehicular applications that our research considers.

Without the use of RE, there are a few papers integrating both MEC and V2X with in-grid RSUs [2, 3]. In [3], Zhang et al. leverage vehicle-to-vehicle (V2V) technology and propose a predictive task offloading scheme to address the communication overhead when a vehicle is moving between different RSUs. In [2], Dai et al. balance the offloading tasks from vehicles by jointly considering vehicle mobility, transmission rate, and MEC computing capacity to minimize task completion delay. These two studies do not consider RE and how to utilize the opportunistic MEC computing and V2X communication resources given limited RE power supply is not discussed.

1.3 System Model and Problem Formulation

In this section, we will first introduce our system model. Then we define the weighted QoS loss and formulate a QoS loss minimization problem. For ease of reference, we list the key notations of our system model in Table 1.1.

1.3.1 Network and Channel Model

We consider an SRSU network with a set of SRSUs B. Each SRSU consists of a communication module SBS and a computation module MEC server. For the sake of notation brevity, we will use SBS b and MEC b to represent the SBS and MEC server in SRSU $b \in B$, respectively. The total operation time is equally divided into T time slots. The duration of each time slot is τ. At the t^{th} time slot, there is a set of VUs $I^t = \{1, 2, ..., l^t\}$ in the network, where $l^t = |I^t|$ is the number of VUs in I^t. We denote the location of VU $i \in I^t$ as a_i^t. At the t^{th} time slot, let $\eta_{D,bi}^t$ be the signal-to-interference-noise ratio (SINR) of downlink transmission from SBS b to VU i. $\eta_{D,bi}^t$ is given by,

$$\eta_{D,bi}^t = \frac{p_b * g_{b,i}^t}{N_0 + \sum_{b' \neq b} p_b' * g_{b'i}^t} \tag{1.1}$$

where $g_{b'i}^t$ denotes the downlink channel gain, p_b is the transmit power of SBS b and N_0 is the noise level. b' is the interfering SBS, which operates the same frequency bands as SBS b.

Let $r_{D,bi}^t$ be the achievable downlink transmission rate from SBS b to VU i per subcarrier,

$$r_{D,bi} = W * log_2(1 + \eta_{D,bi}^t) \tag{1.2}$$

where W is the bandwidth per subcarrier. Similarly, we denote p_i as the transmit power of VU i and h_{ib}^t as the uplink channel gain. The uplink transmission rate from VU i to SBS b per subcarrier can thus be represented as,

Table 1.1. Summary of key notations and abbreviation in chapter 1

Notation	Description	Notation	Description
B	Index set of SRSU in the network	x_{bi}^t	Association indicator of VU i and SRSU b
I^t	Index set of VU in the network	a_i^t	Location of VU
$K_{D,b}$	Available downlink subcarriers of SRSU b	P_b^t	Power consumption of SRSU b
$K_{U,b}$	Available uplink subcarriers of SRSU b	L_b^t	Scheduled solar energy for SRSU b
U_b	Maximum computing speed of MEC b	E_b^t	Battery level of SRSU b
γ	SINR threshold for user association	S_b^t	Generated solar energy of SRSU b
ω_i^t	Data generation rate of the on-board sensor of VU i	u_{bi}^t	Computing speed of MEC b allocated to VU i
c_i^t	Computing resource required for processing the uploaded data of VU i	$k_{U,bi}^t$	Number of uplink subcarriers of SBS b allocated to VU i
d_i^t	Maximum delay of *delay sensitive data*	$k_{DS,bi}^t$	Number of subcarriers of SBS b allocated to VU i for *delay sensitive downlink data*
ε_i^t	Data rate of *delay tolerant downlink data*	$k_{DT,bi}^t$	Number of subcarriers of SBS b allocated to VU i for *delay tolerant downlink data*
δ_i^t	Size of data processing result	E^{max}	Maximum battery capacity
θ_i^t	Maximum delay of *delay tolerant downlink data*		

$$r_{U,bi} = W * log_2(1 + \frac{p_i h_{ib}^t}{N_0}) \tag{1.3}$$

where the interference from other VUs is negligible with frequency reuse and bandwidth allocation techniques [27].

Note that in vehicular communication, the channel condition between SRSU and VU changes rapidly due to mobility of VU. Therefore, we assume the duration of time slot τ to be small enough so that the channel condition is unchanged within the time slot.

1.3.2 Workload Model

In this work, we consider the case that VU has no spare computing capacity, which is the case for current vehicles and will be so for a vast majority of vehicles in the near future. Therefore, each VU will offload all the computation tasks of its vehicular applications. We refer to these tasks as workloads. At the t^{th} time slot, each VU will generate a workload to be offloaded, which is modeled by the following parameters. First, ω_i^t is the data generation rate of the on-board sensor (e.g., camera or Lidar) on VU i, which will be uplink transmitted to the MEC server. Second, c_i^t is the computing resource required for processing the uploaded data, which is quantized as number of machine instructions. Third, δ_i^t is the processing result (e.g., an alert/guidance message), which will be downloaded by VU i. Fourth, d_i^t is the delay requirement from MEC server receives the data to VU i receives the result. Finally, VU may request to download extra information from the MEC server or the Internet, which has data size ε_i^t and delay constraint θ_i^t. Note that the MEC processing result is critical to driving safety and needs low latency, therefore, d_i^t is much smaller than θ_i^t. We refer to the MEC processed data as *delay sensitive downlink data*, and the extra information as *delay tolerant downlink data*.

1.3.3 SRSU Association and Resource Utilization

Let $x_{bi}^t = \{0,1\}$ be the user association indicator at the t^{th} time slot. $x_{bi}^t = 1$ if VU i is associated with SRSU b (its data processing tasks are thus offloaded to SRSU b), and $x_{bi}^t = 0$

otherwise. At each time slot, we assume each VU can only associate with one SRSU. A MEC server, on the other hand, can serve workloads from different VUs by using techniques like Virtual Machine (VM) [28]. Also note that workload cannot be offloaded between different SRSUs. To satisfy the workload demand, SRSU needs to allocate adequate amounts of computing and communication resources to each associated VU. In our case, the connection between VU and SBS will create two bearers, one default bearer and one Guaranteed Bit Rate (GBR) bearer (i.e., dedicated bearer) [29]. Note that the *delay tolerant downlink data* is transmitted through the default bearer, we let $k^t_{DT,bi}$ be the number of downlink subcarriers allocated to VU i by SBS b for this bearer at the t^{th} time slot. On the other hand, the offloaded data and the *delay sensitive downlink data* are transmitted through the GBR bearer. We denote $k^t_{U,bi}$ and $k^t_{DS,bi}$ as the number of uplink and downlink subcarriers, respectively, used for the GBR bearer between VU i by SBS b. We also denote u^t_{bi} as the computing speed, which is quantized as machine instructions per second, of the VM server created for VU i by MEC b. To ensure that the data generated by the on-board sensor will not be dropped due to VU's memory buffer overflowing, the average uplink transmission rate of VU i should be greater than (or equal to) the data generation rate ω^t_i of the on-board sensor. The uplink subcarriers allocated to VU i, henceforth, should satisfy the following constraint,

$$\sum_{b\in B} x^t_{bi} r^t_{U,bi} k^t_{U,bi} \geq \sum_{b\in B} x^t_{bi} \omega^t_i \tag{1.4}$$

To satisfy the downlink delay constraint, the number of subcarriers allocated to VU i for the *delay tolerant downlink data* should satisfy,

$$\sum_{b\in B} x^t_{bi} r^t_{D,bi} k^t_{DT,bi} \geq \sum_{b\in B} x^t_{bi} \frac{\varepsilon^t_i}{\theta^t_i} \tag{1.5}$$

Note that the *delay sensitive downlink data* need to be processed and transmitted in low latency. Hence, the computing speed of VM server and downlink subcarriers allocated to VU i

by SRSU b should satisfy the following,

$$\sum_{b\in B} x_{bi}^t \left(\frac{c_i^t}{u_{bi}^t} + \frac{\delta_i^t}{r_{D,bi}^t k_{DS,bi}^t}\right) \leq \sum_{b\in B} x_{bi}^t d_i^t \tag{1.6}$$

On the other hand, the computing and communication resources of each SRSU are limited, which is constrained by the following three equations,

$$\sum_{i\in I^t} x_{bi}^t u_{bi}^t \leq U_b, \tag{1.7}$$

$$\sum_{i\in I^t} x_{bi}^t k_{U,bi}^t \leq K_{U,b}, \tag{1.8}$$

$$\sum_{i\in I^t} x_{bi}^t (k_{DS,bi}^t + k_{DT,bi}^t) \leq K_{D,b} \tag{1.9}$$

where U_b is the maximum number of machine instructions the processor of MEC b can execute per second [30]. $K_{U,b}$ and $K_{D,b}$ are SBS b's maximum number of available sub-carriers for uplink and downlink transmission, respectively.

1.3.4 Power Consumption Model

Power consumption of each SRSU is modeled by the power consumption of MEC plus the power consumption of SBS. At the t^{th} time slot, we denote $P_{S,b}^t$ as the power consumption of MEC b, which linearly increases with the overall processor's computing speed [28]. Let $p_{M,b}$ be the idle power of MEC b and $p_{C,b}$ be the power consumption for each unit utilization of the processor's speed of MEC b. $P_{S,b}^t$ can then be represented by the following equation,

$$P_{S,b}^t = \tau p_{M,b} + \tau p_{C,b} \sum_{i\in I^t} x_{bi}^t u_{bi}^t. \tag{1.10}$$

Besides, power consumption of SRSU also includes energy consumed by the SBS.

The energy consumption of SBS is the energy consumed by operating uplink and downlink transmissions. Power consumption of uplink transmission is the circuit power for demodulation and baseband processing. It increases linearly with the number of active subcarriers [31]. Secondly, operating downlink transmission consumes circuit and RF related power; both are linearly increasing with the number of active downlink subcarriers [32]. Hence, the power consumption of SBS at the t^{th} time slot can be expressed as:

$$P^t_{X,b} = \tau \sum_{i \in I^t} x^t_{bi}(p_{D,b}(\frac{\delta^t_i}{r^t_{D,bi}} + k^t_{DT,bi}) + p_{U,b}k^t_{U,bi}) + \tau p_{N,b}. \tag{1.11}$$

where $p_{N,b}$ is the idle power of SBS b, $p_{U,b}$ is the circuit power consumption per active uplink subcarrier, and $p_{D,b}$ is the joint circuit and transmission power consumption per active downlink subcarrier. The overall power consumption of SRSU b at the t^{th} time slot can, therefore, be represented as, $P^t_b = P^t_{S,b} + P^t_{X,b}$.

1.3.5 Solar Generation and Battery Model

At the t^{th} time slot, let S^t_b be the amount of energy harvested from the solar panel of SRSU b. We assume S^t_b is available at the beginning of the t^{th} time slot and will be immediately stored without any loss of energy. The battery level of SRSU b is denoted as E^t_b, which is constrained by energy causality and battery capacity. We assume battery is lossless and let $E^max \in (0,\infty)$ denote the battery capacity. Therefore, the battery level E^t_b should satisfy,

$$0 \leq E^t_b = E^{t-1}_b + S^t_b - P^t_b \leq E^{max}. \tag{1.12}$$

1.3.6 QoS Model

The evaluation of QoS in this paper is defined according to the instance of service outage and service disruption on workloads.

Service Outage

Because the energy, computing, and communication resources are limited, SRSUs may not be able to serve a VU while satisfying this VU's workload requirements 1.4-1.6. Because there is no computing capacity in a VU, service outage happens when its workload cannot be offloaded to any SRSU in the network. We denote the number of VUs experiencing service outage at the t^{th} time slot as C^t_{drop}, which can be calculated as,

$$C^t_{drop} = \sum_{i \in I^t} (1 - \sum_{b \in B} x^t_{bi}). \tag{1.13}$$

and the service outage rate is $\frac{C^t_{drop}}{l^t}$, where $l^t = |I^t|$ is the total number of VUs in the network at the t^{th} time slot.

Service Disruption

Service disruption happens to a VU when an SRSU hands it over to another SRSU. The handover can take place when a VU is leaving an SRSU's coverage or when we actively change its associated SRSU. During the handover, the VU's workload cannot be offloaded, leading to service disruption. We denote the number of VUs experiencing service disruption at the t^{th} time slot as $C^t_{handover}$, which can be calculated as,

$$C^t_{handover} = \sum_{i \in I^t} (\sum_{b \in B} x^t_{bi})(1 - \sum_{b \in B} x^t_{bi} x^{t-1}_{bi}). \tag{1.14}$$

and the service disruption rate is $\frac{C^t_{handover}}{l^t}$.

The level of impact of the above two cases, service outage and service disruption, on driving experience is different. In the first case, the VU will be left unserved during the whole time slot. However, in the second case, the duration of handover disruption may be small. Once the VU is successfully associated with the next SRSU, it can then be served by the MEC server during the remaining period of the current time slot. Therefore, we introduce a weighted factor

$\kappa < 1$ on the service disruption rate to capture the different impacts on VUs between these cases. We then define the weighted QoS loss of the t^{th} time slot as $L_t = \frac{C^t_{drop}+C^t_{handover}}{l^t}$, and the weighted QoS loss of the total operation time as,

$$\mathcal{L} = \frac{\sum_{t=1}^{T} C^t_{drop} + \kappa C^t_{handover}}{\sum_{t=1}^{T} l^t}. \tag{1.15}$$

By properly adjusting κ, solving Problem 1.16 can effectively optimize QoS for VUs, depending on the network policy.

1.3.7 Problem Formulation

Our objective is to determine the user association x^t_{bi}, and the resource allocation u^t_{bi}, $k^t_{U,bi}$, $k^t_{DS,bi}$, and $k^t_{DT,bi}$ for VU i to minimize the weighted QoS loss of the total operation time. The decision is made at the beginning of each time slot based on the current SRSUs' available energy, computing, and computation resources, as well as VUs' locations, workload demands, and wireless channel conditions. The optimization problem is formulated as,

$$\underset{x^t_{bi},u^t_{bi},k^t_{U,bi},k^t_{DS,bi},and k^t_{DT,bi} \forall i \in I^t, \forall t}{\text{minimize}} \quad \mathcal{L} \tag{1.16a}$$

$$\text{subject to} (1.4) - (1.9), (1.12) \tag{1.16b}$$

$$\sum_{b \in B} x^t_{bi} \leq 1 \quad \forall i \in I^t, t \in [1,T] \tag{1.16c}$$

$$x^t_{bi} = \{0,1\} \quad \forall i \in I^t, t \in [1,T] \tag{1.16d}$$

$$\sum_{b \in B} x^t_{bi} \eta^t_{D,bi} \geq \sum_{b \in B} x^t_{bi} \gamma \quad \forall i \in I^t, t \in [1,T] \tag{1.16e}$$

Constraint 1.16c, together with 1.16d, state that the workload is not separable and cannot be offloaded to multiple SRSUs simultaneously. Moreover, constraint 1.16e limits a VU to only offload its workload to the SRSU that provides enough downlink SINR, with the threshold being set by γ.

Furthermore, we assume to have the knowledge of the predicted profiles of SRSU's solar

energy generation and power consumption in advance. These data will help us plan the utilization of solar energy (i.e., the battery charging/discharging scheduling strategy) for each SRSU. SRSU power consumption and solar generation profiles are shown to be predictable in [10, 33]. We will list the prediction performance in Section 1.5.2 and further discuss the effect of prediction error on the optimization problem.

1.4 Solution Methodology

The solution of Problem 1.16 involves decisions in two dimensions, as shown in Fig. 1.1. In the spatial dimension, feasible solutions of user association and resource allocation at each time slot should be decided to minimize the weighted QoS loss. However, the decision at each time slot is coupled with the temporal solar energy availability. As an example, if SRSU A in 1.1a uses most of its solar energy (shown in the blue bar) in the t^{th} time slot to serve as many VU as possible, 3 VUs at the $(t+2)^{th}$ time slot will experience service outage due to the lack of solar energy. But if SRSU A reserves some energy and lets SRSU B serve more VUs than it served in 1.1a, SRSU A will have enough energy to serve all its VUs at the $(t+2)^{th}$ time slot, as 1.1b shows. Based on this observation, we follow the logic of [14, 34, 35] to schedule the utilization of renewable energy for each time slot in advance so that multiple BSs will not run out of renewable energy simultaneously. We therefore propose a two-phase QoS loss Minimization Algorithm (TQMA). TQMA solves P1 in two phases corresponding to the two dimensions: (i) solar energy scheduling phase (temporal dimension), and (ii) user association and resource allocation phase (spatial dimension). The process flow of TQMA is depicted in Fig. 1.2. Note that Phase 1 is executed offline based on the predicted profiles of SRSUs' solar generations and power consumptions, and Phase 2 is executed online based on current (i) VUs' workloads, locations, and transmission rates, and (ii) SRSUs' available communication, computing, and scheduled solar energy resources.

Fig. 1.3 shows the overview of the SRSU-assisted vehicular edge computing network

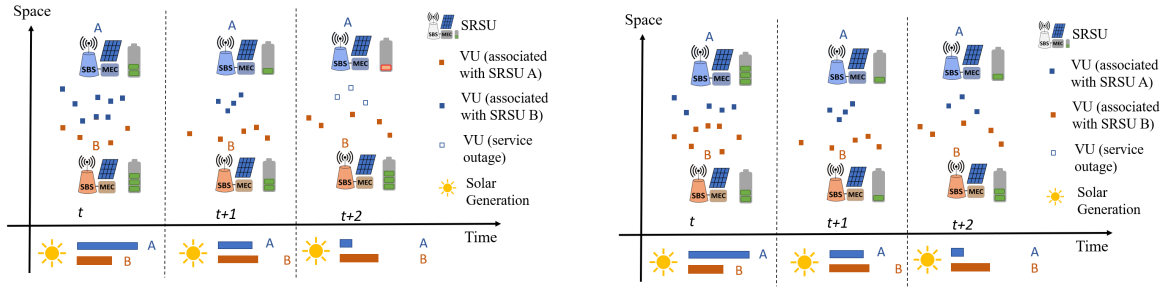

Figure 1.1. Two dimensions that are involved in solving Problem 1.16: offline solar energy scheduling (temporal dimension), and online user association and resource allocation (spatial dimension); also showing two scenarios describing the impact of energy scheduling (a) left, the condition with the absence of not performing energy scheduling at SRSU *A* and (b) right, the condition when performing energy scheduling at SRSU *A*.

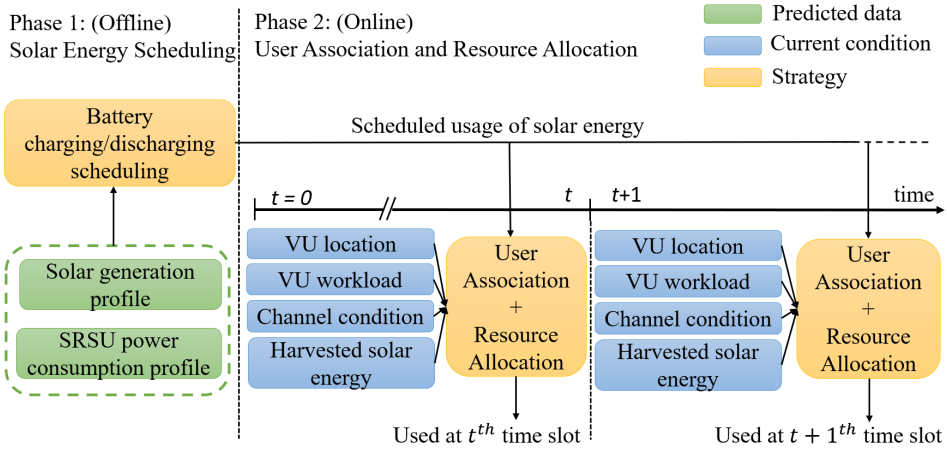

Figure 1.2. The proposed two-phase approach, TQMA, to solve Problem 1.16

and the information flows for Phase 2 of TQMA. At the beginning of each time slot, each VU will send the workload offloading request (blue arrows), including all the workload parameters, to the SRSU it associated with. Each SRSU will then send all the required information for Phase 2 decision to the SRSU network coordinator (green arrow). The SRSU network coordinator will make the Phase 2 decision and forward the resulting user association and SRSU resource allocation decisions back to SRSUs (purple arrows). Note that while the offloaded tasks are

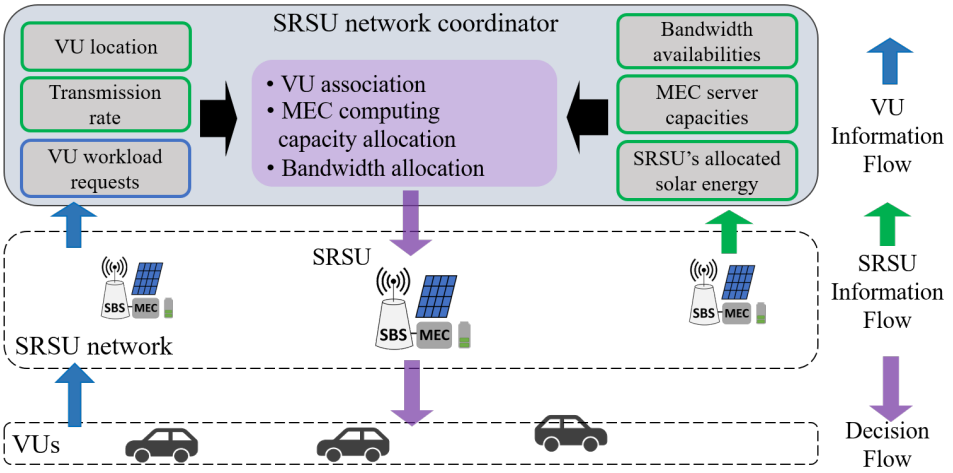

Figure 1.3. Overview of the SRSU-assisted vehicular edge computing system, including request and decision flows

executed on the MECs associated with the SRSUs, the network coordinator and hence the proposed TQMA algorithm will be run in a separate server.

1.4.1 Phase 1 and Solar Energy Scheduling Algorithm (SESA)

We denote L_b^t as the scheduled solar energy of SRSU b at the t^{th} time slot, which will be regarded as the maximum allowable amount of energy for SRSU b to utilize at the t^{th} time slot. We also define $\pi_b^t = L_b^t/\hat{P}_b^t$ as SRSU b's Solar Utilization Ratio (SUR) for the t^{th} time slot, where $\hat{P}_b^t$ is the predicted SRSU power consumption. For SRSU b, the objective of Phase 1 is to maximize the minimum value of SUR within the whole operation time by optimally arranging the value of $L_b^t, t \in [1,T]$. Note that L_b^t needs to follow the energy causality constraint, $0 \leq \sum_{t'=1}^{t} \hat{S}_b^t - \sum_{t'=1}^{t} L_b^t \leq E^{max}, t \in [1,T]$, where $\hat{S}_b^t$ is the predicted solar generation profile for SRSU b.

The rationale is to distribute the solar energy at each time slot proportional to the SRSU's expected power consumption. This will prevent all SRSUs from having energy surplus and deficit at the same time. Therefore, neighboring SRSUs can better balance their power consumption based on their energy availability in Phase 2. Moreover, this can also prevent SRSUs from fully depleting their batteries during the hours when solar energy is not being generated.

Algorithm 1: SESA

Input:
Predicted solar generation profile $\{\hat{S}_b^t | t \in [1,T]\}$
Predicted SRSU power consumption profile $\{\hat{P}_b^t | t \in [1,T]\}$
Battery capacity E^{max}
Output:
Scheduled solar energy $L = \{L_b^t | t \in [1,T]\}$

1 $\beta \leftarrow zeros(1,T)$;
2 $L_b^t \leftarrow \hat{S}_b^t, \forall t \in [1,T], t_f \leftarrow t^{end}$;
3 **for** $t \leftarrow t^{end}$ **to** 1 **do**
4 update β, L, t_f using *DistributeEnergy*(β, L, t, t_f, b)
5 return $\{L_b^t | t \in [1,T]\}$

It is inevitable that imperfect predictions will lead to a non-optimal L_b^t, $t \in [1,T]$ when applied to actual solar generation and SRSU power consumption. We will discuss the effect of prediction error on performance in Section 1.5.2.

To arrange L_b^t, we propose the algorithm, SESA, which is shown in Algorithm 1. To begin with, we initialize L_b^t as $\hat{S}_b^t$ for each time slot $t \in [1,T]$. Let β_b^t be the expected battery level of SRSU b at the t^{th} time slot, which is initialized as zero. Let t_f be the last time slot that we can schedule the solar energy to. t_f is initialized as T in line 2 of SESA. To satisfy the energy causality constraint, we will start to schedule the solar energy iteratively from the last time slot to the beginning. At each iteration, we execute *DistributeEnergy* in Algorithm 2 for the current time slot t. In *DistributeEnergy*, we will decide how much energy to be scheduled to each future time slot of t. We will first calculate the SUR π_b^t for t and the average SUR $\bar{\pi}$ for the time slots between t and t_f. If $\pi_b^t > \bar{\pi}$, we will decrease the value of L_b^t until the new π_b^t equals $\bar{\pi}$. The remaining energy will be distributed to time slots $t' \in (t, t_f]$. Each time slot t' will receive $\varepsilon^{t'}$ amount of energy that will be added to $L_b^{t'}$. We assume $\varepsilon^{t'}$ is proportional to the required energy for $\pi_b^{t'}$ to reach $\bar{\pi}$ for t'. The above steps are listed in lines 1-6 of *DistributeEnergy*.

However, during the scheduling process, the expected battery level may achieve the maximum capacity at any time slot between t and t_f. Assume the maximum capacity is achieved at t'', no more energy can be stored and scheduled from t to any time slot after t''. Let $t^* \in (t, t_f]$

Algorithm 2: DistributeEnergy

Input: β, L, t_s, t_f, b
Output: β, L, t_f

1 calculate $\bar{\pi} \leftarrow \frac{\sum_{t=t_s}^{t_f} L_b^t}{\sum_{t=t_s}^{t_f} \hat{P}_b^t}$;
2 calculate $\pi^t, \forall t \in [t_s, t_f]$;
3 **if** $\pi^{t_s} > \bar{\pi}$ *and* $t_f > t_s$ **then**
4 $\quad J \leftarrow \{t | \pi^t < \bar{\pi}, t \in (t_s, t_f]\}$;
5 $\quad \Delta \leftarrow L_b^{t_s} - \bar{\pi}\hat{P}_b^{t_s}, \beta' \leftarrow \beta, \varepsilon \leftarrow zeros(1, T)$;
6 $\quad$ calculate $\varepsilon^t, \forall t \in J$;
7 $\quad$ calculate $\beta'^t \leftarrow \beta'^t + \sum_{t'=t+1}^{t_f} \varepsilon^{t'}, \forall t \in [t_s, t_f)$;
8 $\quad \tilde{T} \leftarrow \{t | \beta'^t \geq E^{max}, \forall t \in [t_1, t_f)\}$;
9 $\quad$ **if** $\tilde{\mathrm{T}} \notin \phi$ **then**
10 $\quad\quad t^* \leftarrow \min_{t \in \tilde{T}} t, \varepsilon \leftarrow (E^{max} - \beta'^{t^*})$;
11 $\quad\quad \beta^t \leftarrow \beta^t + \varepsilon^*, \forall t \in [t_s, t^*]$;
12 $\quad\quad L_b^{t_s} \leftarrow L_b^{t_s} - \varepsilon^*, L_b^{t^*+1} \leftarrow L_b^{t^*+1} + \varepsilon^*$;
13 $\quad\quad$ update β, L from DistributeEnergy$(\beta, L, t^* + 1, t_f, b)$;
14 $\quad\quad t_f \leftarrow t^*$;
15 $\quad\quad$ update β, L, t_f from DistributeEnergy(β, L, t_s, t_f, b);
16 $\quad$ **else**
17 $\quad\quad \beta \leftarrow \beta', L_b^{t_s} \leftarrow L_b^{t_s} - \Delta$;
18 $\quad\quad L_b^t \leftarrow L_b^t + \varepsilon^t, \forall t \in (t_s, t_f]$;
19 $\quad\quad$ return β, L, t_f;
20 **else**
21 $\quad$ return β, L, t_f;

be the earliest time slot that achieves the maximum battery capacity after $\varepsilon^{t'}$ is added to each time slot $t' \in (t, t_f]$. We then set its expected battery level $\beta_b^{t^*}$ to full and add the corresponding solar energy to $L_b^{t^*}$. After that, we split $(t, t_f]$ into two segments: $(t, t^*]$ and $(t^*, t_f]$, and recursively apply *DistributeEnergy* to these segments. The recursive process, which is shown in lines 13-15 of DistributeEnergy, ends when t^* doesn't exist within the new segment. Finally, we update the value of t_f and $\beta^t, t \in [1, T]$ in lines 14 and 17 of *DistributeEnergy*, then proceed to the next iteration. SESA will return L_b^t, $t \in [1, T]$, until the solar energy scheduling process is executed for all the time slots.

Therefore, at each time slot, SRSU b will drain $L_b^t - \hat{S}_b^t$ amount of energy from the battery if $L_b^t - \hat{S}_b^t \geq 0$, or store $\hat{S}_b^t - L_b^t$ amount of energy to the battery, otherwise.

The complexity of SESA is $O(T^3)$, where T is the number of time slots. Since SESA is executed offline before the whole operation time starts, the complexity will not affect the real-time feasibility of our technique.

1.4.2 Phase 2 and the MRGAP Problem

In Phase 2, we formulate a user association and SRSU resource allocation problem to minimize the weighted QoS loss $\mathcal{L}_t$ at each time slot. At the t^{th} time slot, the above problem can be formulated as,

$$\underset{X^t,\psi^t}{\text{minimize}} \frac{C^t_{drop} + \kappa C^t_{handover}}{l^t} \tag{1.17a}$$

$$\text{subject to} (1.4) - (1.9) \tag{1.17b}$$

$$\sum_{b \in B} x^t_{bi} \leq 1 \quad \forall i \in I^t \tag{1.17c}$$

$$x^t_{bi} = \{0,1\} \quad \forall i \in I^t, \forall b \in B \tag{1.17d}$$

$$\sum_{b \in B} x^t_{bi} \eta^t_{D,bi} \geq \sum_{b \in B} x^t_{bi} \gamma \quad \forall i \in I^t \tag{1.17e}$$

$$P^t_b \leq \min(L^t_b, E^{t-1}_b + S^t_b) \quad \forall b \in B \tag{1.17f}$$

where $\psi^t = \{k^t_{U,bi}, k^t_{DT,bi}, k^t_{DS,bi}, u^t_{bi} | i \in I^t, b \in B\}$ and $X^t = \{x^t_{bi} | i \in I^t, b \in B\}$. Constraints 1.17c and 1.17d state that the workload is not separable and can only be offloaded to one SRSU. Constraint 1.17e limits a VU to only associate with the SRSU which provides enough signal strength (with the SINR threshold be γ). Due to prediction error, it is possible that an SRSU's available energy is less than L^t_b. Therefore, the power consumption of SRSU should be limited by the minimum between actual available energy $S^t_b + E^{t-1}_b$ and scheduled solar energy L^t_b, in Constraint 1.17f.

We next show that Problem 1.17a can be formulated as a variant of Multi-Resource Generalized Assignment Problem (MRGAP) [36]. MRGAP is originally proposed to minimize a

total cost when assigning items to containers under multiple resource constraints. Given N is a set of items, M is a set of containers, and K is a set of multiple resources provided by containers to the items, MRGAP is formulated as

$$\underset{X_{mn}, n\in N, m\in M}{\text{minimize}} \sum_{n\in N}\sum_{m\in M} z_{mn}x_{mn} \tag{1.18a}$$

$$\text{subject to} \sum_{m\in M} x_{mn} = 1, \forall n \in N \tag{1.18b}$$

$$x_{mn} = \{0,1\} \quad \forall n \in N, \forall m \in M \tag{1.18c}$$

$$\sum_{n\in N} v_{mnk}x_{mn}, \forall m \in M, k \in K \tag{1.18d}$$

where n is the index of the item, m is the index of the container, and k is the index of the resource. x_{mn} is the decision of whether to assign item n to container m. z_{mn} is the cost of assigning item n to container m, w_{mk} is the maximum capacity on resource k of container m, and v_{mnk} is the amount of resource k required to assign item n to container m. Finding the optimal solution of MRGAP is NP-Hard [37]. To map Problem 1.17a to MRGAP, we consider a special case where the assignment constraint 1.17f is relaxed to $\sum_{m\in M} x_{mn} \leq 1, \forall n \in N$, which allows items without any assignment. Different from conventional MRGAP, this special case always has a feasible solution.

Next, we show how Problem 1.17a is mapped to the relaxed case of MRGAP. Because Problem 1.17a has a constant denominator l^t, we rewrite the numerator of its objective function,

$$C^t_{drop} + \kappa C^t_{handover} = l^t + \sum_{i=I^t}\sum_{b=B} (-1 + \kappa - \kappa\Omega(x^t_{bi}, x^{t-1}_{bi}))x^t_{bi}. \tag{1.19}$$

where $\Omega(x,y)$ is an indicator function, it returns 1 if $x = y$, or otherwise returns 0 (See section 1.7.1). Minimizing Eq. 1.19 is equivalent to minimizing its second term (i.e. the summation of $-1 + \kappa - \kappa\Omega(x^t_{bi}, x^{t-1}_{bi})$, which can be mapped to z_{mn} in MRGAP. Let M be the SRSU set B, N be the VU set I^t, and K to contain resources of the (i) computing speed, (ii) downlink

subcarriers, (iii) uplink subcarriers, and (iv) energy. Let $v_{bi1}, v_{bi2}, v_{bi3}, and v_{bi4}$ be the amount of computing speed, the number of uplink subcarriers, the number of downlink subcarriers, and the corresponding power consumption allocated to VU i by SRSU b, respectively. Consequently, Problem 1.17a can be formulated as a special case of MRGAP with relaxed constraint 1.17f and additional constraints 1.17e, 1.17f, and 1.6.

Next, we develop a real-time heuristic algorithm H-URA for Problem 1.17a. To begin with, let v^t_{bik} denote the value of v_{bik} in the corresponding MRGAP problem of Problem 1.17a at the t^{th} time slot. We first show how many subcarriers for uplink and delay tolerant downlink data transmission are needed to serve VU i. The allocation of $k^t_{U,bi}$ and $k^t_{DT,bi}$ from SRSU b should follow constraints 1.4 and 1.5, respectively. Once these constraints are satisfied, there is no need to increase the value of $k^t_{U,bi}$ and $k^t_{DT,bi}$. The constraints in 1.4 and 1.5, thus, can be reduced to deterministic allocation decision,

$$k^t_{U,bi} = \lceil \frac{\omega^t_i}{r^t_{U,bi}} \rceil, k^t_{DT,bi} = \lceil \frac{\varepsilon^t_i}{\theta^t_i r^t_{D,bi}} \rceil \tag{1.20}$$

The value of v^t_{bi2} can, therefore, be set as $\lceil \frac{\omega^t_i}{r^t_{U,bi}}$ for VU i. On the other hand, the allocation of computing speed and downlink subcarriers for the delay sensitive downlink data should satisfy the joint delay constraint 1.6. Therefore, deterministic allocation decision does not exist. A reasonable way is to define v^t_{bi1} (required computing speed) and v^t_{bi3} (required downlink subcarriers) based on the availability of these two resources,

$$v^t_{bi1} = \lceil \frac{K_{D,b} + U_b}{K_{D,b}} (\frac{c^t_i}{d^t_i}) \rceil, v^t_{bi3} = \lceil \frac{K_{D,b} + U_b}{U_{b,b}} (\frac{\delta^t_i}{r^t_{D,bi} d^t_i}) + \frac{\varepsilon^t_i}{\theta^t_i r^t_{D,bi}} \rceil \tag{1.21}$$

Meanwhile, v^t_{bi4} is set to be the power consumption for SRSU b when utilizing v^t_{bi1}, v^t_{bi2}, and v^t_{bi3} amount of resources. With the value of $v^t_{bi1}, v^t_{bi2}, v^t_{bi3}$, and v^t_{bi4}, we propose to solve Problem 1.17a by heuristically solving the Lagrangian dual problem of its MRGAP form [38].

The Lagrangian dual of Problem 1.17a can be formulated as,

$$\underset{\lambda_b^t,\mu_b^t,\rho_b^t,\sigma_b^t \in \mathbb{R}_+, b\in B}{\text{maximize}} \quad \underset{x_{bi}^t, b\in B, i\in I^t}{\text{minimize}} \quad \sum_{i\in I^t}\sum_{b\in B} z_{bi}^t x_{bi}^t + \tag{1.22a}$$

$$\sum_{b\in B} \lambda_b^t (\sum_{i\in I^t} x_{bi}^t v_{bi1}^t - U_b) + \tag{1.22b}$$

$$\sum_{b\in B} \mu_b^t (\sum_{i\in I^t} x_{bi}^t v_{bi2}^t - K_{U,b}) + \tag{1.22c}$$

$$\sum_{b\in B} \rho_b^t (\sum_{i\in I^t} x_{bi}^t v_{bi3}^t - K_{D,b}) + \tag{1.22d}$$

$$\sum_{b\in B} \sigma_b^t (\sum_{i\in I^t} x_{bi}^t v_{bi4}^t - L'^t_b) \tag{1.22e}$$

$$\text{subject to} 1.17c - 1.17e \tag{1.22f}$$

where $L'^t_b = \min\,(L_b^t\,,\, E_b^{t-1} + S_b^t\,)$. $\lambda_b^t, \mu_b^t, \rho_b^t, \sigma_b^t$ are the Lagrangian multipliers for dualizing constraints 1.7-1.9 and 1.17f. The optimality of Problem 1.22f for 1.17a depends on the values of $\lambda_b^t, \mu_b^t, \rho_b^t, \sigma_b^t$. However, since the workload demands will change in different time slots, the optimal values of these Lagrangian multipliers will also change. Consequently, traditional searching-based methods [36, 38] to find the optimal Lagrangian multipliers are time-consuming since the solution is only applicable to the current time slot. Therefore, we propose to define the Lagrangian multipliers as follows,

$$\lambda_b^t = \gamma \frac{\sum_{i\in I^{t-1}} x_{bi}^{t-1} u_{bi}^{t-1}}{U_b}, \mu_b^t = \gamma \frac{\sum_{i\in I^{t-1}} x_{bi}^{t-1} k_{U,bi}^{t-1}}{k_{U,b}}, \rho_b^t = \gamma \frac{\sum_{i\in I^{t-1}} x_{bi}^{t-1} k_{D,bi}^{t-1}}{K_{D,b}}, \sigma_b^t = \gamma \frac{P_b^{t-1}}{L'^{t-1}_b} \tag{1.23}$$

where γ is a constant scaling factor. The rationale is as follows. Consider two SRSUs which have the same z_{bi}^t to VU i, we tend not to assign this VU to the SRSU whose resources are more likely to be fully utilized. The likelihood relies on the resource utilization condition at the previous time slot.

Lemma 1.4.1. *With fixed $\lambda_b^t, \mu_b^t, \rho_b^t, \sigma_b^t$, solving Problem 1.22f is equivalent to finding the SRSU which minimizes $q_{bi}^t = z_{bi}^t + \lambda_b^t v_{bi1}^t + \mu_b^t v_{bi2}^t + \rho_b^t v_{bi3}^t + \sigma_b^t v_{bi4}^t$ for each VU.*

Proof. See section 1.7.2. □

To further minimize the service disruption, we tend to assign VU to the SRSU that locates on its future path. We propose to use a Maximum Likelihood Markov Chain [39] to predict the probability of a VU's future location. First, we divide the network neighborhood into A non-overlapping areas. Each area is represented by a state in the Markov Chain. Second, we create an $|A|x|A|$ transition matrix $\hat{A}^t$ for this Markov Chain at the t^{th} time slot, where —A— is the size of A. We define $N^t_{s_1 s_2}$ as the total instances of VUs moving from area s_1 to area s_2 during any consecutive time slots before t. The state transition probability $\hat{A}^t_{s_1,s_2}$ can then be represented as $\hat{A}^t_{s_1,s_2} = N^t_{s_1 s_2} / \sum_{s \in A} N^t_{s_1 s}$. Let b_{s_2} be the SRSU which provides the best signal strength to the geological center of area s_2. If a VU is in area s_1, the probability that b_{s_2} is the next SRSU for this VU to associate in the next time slot is predicted as $\hat{A}^t_{s_1,s_2}$. This probability is then multiplied by κ and added to q^t_{bi} for each VU-SRSU pair. For each $s \in A$, the complexity of calculating $\sum_{s \in A} N^t_{s_1 s}$ is $O(|A|)$ and hence the complexity of updating $\hat{A}^t_{s_1,s_2}, s_1 \in A, s_2 \in A$ is $O(|A|^2)$. Note that in an SRSU network, the number of VU is usually larger than $|A|$. Therefore, $O(|A|^2) < O(l^2)$.

Based on lemma 1.4.1 and $\hat{A}^t$, we assign each VU to the SRSU which corresponds to the VU's minimal q^t_{bi}. However, this assignment may not be valid since we relax constraints 1.7-1.9, and 1.17f in Problem 1.17a. Therefore, we propose to make association decisions for VUs one by one while checking if the decision satisfies the relaxed constraints. We will pick the VU which has the largest difference between its best and second-best $q^t_{bi}, b \in B$, as the highest priority VU to make the association decision for. We then assign the VU to the SRSU that corresponds to the best q^t_{bi} if the constraints 1.7-1.9, 1.17e, 1.17f of Problem 1.17a can be satisfied, and proceed to the next VU.

To check if a VU association satisfies the constraints 1.7-1.9, 1.17e, 1.17f of Problem 1.17a and determine the optimal resource allocation decision, we adopt the procedure Minimize SRSU Power Consumption Algorithm (MPCA), which is proposed in our previous work [8].

Given a VU set ζ of an SRSU, MPCA will first check if the SRSU can serve all the workloads from ζ. If possible, then MPCA will allocate computing and communication resources to the VUs in ζ while minimizing the power consumption of the SRSU (with the rationale to save solar energy). MPCA determines the optimal resource allocation as follows. We have argued the optimal value of $k^t_{U,bi}$ and $k^t_{DT,bi}$. To show the optimal allocation of $k^t_{DS,bi}$ and u^t_{bi} in Eq. 1.25 for a given VU set ζ of SRSU b, we define the following terms for all the VUs in ζ,

$$\begin{gathered} l^t_i = \frac{\delta^t_i}{r^t_{D,bi} d^t_i}, \varphi^t_i = \frac{l^t_i c^t_i}{d^t_i}, \vartheta^t_b = \sum_{i \in \zeta} k^t_{DT,bi}, \\ H^t_b = \frac{\sum_{i \in \zeta} (\varphi^t_i)^{1/2}}{K_{D,b} - \vartheta^t_b - \sum_{i \in \zeta} l^t_i}, y^t_i = \frac{(c^t_i)^{1/2}}{d^t_i} + \frac{(l^t_i)^{1/2}}{(d^t_i)^{1/2}} H^t_b \end{gathered} \tag{1.24}$$

Then, the optimal resource allocation for u^t_{bi} and $k^t_{DS,bi}$ will be,

$$u^t_{bi} = \lceil y^t_i (c^t_i)^{1/2} \rceil, k^t_{DS,bi} = \lceil \frac{y^t_i (l^t_i d^t_i)^{1/2}}{H^t_b} \rceil, i \in \zeta \tag{1.25}$$

The above resource allocation solution to minimize power consumption of the SRSU can be solved by analyzing the problem's Karush–Kuhn–Tucker (KKT) conditions [40] or using convex optimization programming tools [41]. We omit the proof here for the sake of brevity.

MPCA returns 0 if the KKT conditions are violated or constraints 1.7-1.9, 1.17e, 1.17f are not satisfied. Otherwise, MPCA returns the optimal resource allocation decisions u^t_{bi}, $k^t_{U,bi}$, $k^t_{DT,bi}$, and $k^t_{DS,bi}$ for each VU in ζ.

Based on the above discussion, we propose H-URA for real-time user association and SRSU resource allocation, which is shown in Algorithm 3. The pseudocode of MPCA is also included in Algorithm 4. H-URA takes real-time VUs' locations workload demands, and channel conditions, as well as SRSUs' resource availabilities and Lagrangian multipliers as input. To begin with, Q^t in line 3 of H-URA records the value of q^t_{bi} for all the VU-SRSU pairs. The user association procedure is determined by the while loop in lines 4-18. H-URA will decide the highest priority VU to make the association decision for in lines 5-8. If H-URA determines VU

i^* as the highest priority VU and b^* is the SRSU corresponds to its minimal q^t_{bi}, then H-URA will consider associating i^* with b^*. H-URA will check if this association satisfies all the constraints of Problem 1.17a in lines 9 and 11 by using MPCA. If constraints are satisfied, H-URA will confirm the association, update the association indicator and resource allocation decisions in lines 12-15. Note that ζ' in line 8 is the set of VUs that have been associated with SRSU b^* by H-URA. The elements in Q^t_{bi} related to VU i^* will then be set as ∞ in line 16 so that VU i^* will not be considered again in the next iteration. If the constraints of Problem 1.17a cannot be satisfied, H-URA will set the value of $Q^t_{b^*i^*}$ as ∞ in line 18 and proceed to the next iteration. The iteration ends when all the VUs are associated with an SRSU or when all the elements in Q^t are ∞.

Note that in the worst case, the while loop will iterate lB times, which is the size of Q^t. For each iteration, in the worst case, the time complexity of lines 5-7 is lB while the complexity of other steps is less than or equal to l. On the other hand, the complexity of updating $\hat{A}^t$ is less than $O(l^2)$. Therefore, the time complexity of H-URA is $O(l^2B^2)$ for time slot t. Hence, H-URA is possible to be executed in real-time for reasonable sizes of the current VU set I^t and SRSU set B. This is validated with experimental results reported in the next section.

By combining the proposed SESA and H-URA algorithms, we present our proposed heuristic method to solve Problem 1.16, TQMA, as shown in Fig. 1.4. In Phase 1, SESA will schedule the solar energy for each time slot. Then, H-URA will be executed at each time slot to make user association the resource allocation decisions real-time in Phase 2.

1.5 Experimental Result

1.5.1 Simulation Framework

The objective of our simulation framework is to observe the weighted QoS loss performance of different solar energy scheduling, user association, and SRSU resource allocation strategies. In the simulation results below, we assume that VUs offload object detection tasks

Algorithm 3: H-URA

Input:
The scheduled solar energy, battery level and solar generation $L_b^t, E_b^{t-1}, S_b^t, \forall b \in B$
VU location $\{a_i^t\}$, and workload $\{\omega_i^t, c_i^t, \delta_i^t, d_i^t, \varepsilon_i^t, \theta_i^t\}, \forall i \in I^t$
Channel conditions $\{g_{bi}^t | i \in I^t, b \in B\}$
System Parameters $\gamma, E^{max}, K_{D,b}, K_{U,b}, and U_b, \forall b \in B$
Previous association indicators $x_{bi}^{t-1}, \forall i \in I^t, b \in B$
Next SRSU probability prediction $\hat{A}^t$
Lagrangian multipliers $\lambda_b^t, \mu_b^t, \rho_b^t, \sigma_b^t, \forall b \in B$
Output: User association X^t , and Resource allocation ψ^t

$L'^t_b \leftarrow min(L_b^t, E_b^{t-1} + S_b^t), \forall b \in B$;
$visit\_UE \leftarrow 0$;
$Q^t \leftarrow \{q_{bi}^t\}_{b \in B, i \in I^t}$;
while $visit\_UE \leq l$ *and* $\exists q_{bi}^t \neq \infty$ **do**
  for $\forall i \in I^t$ **do**
    $b_i^1 \leftarrow argmin_{b \in B} Q_{bi}^t$;
    $b_i^2 \leftarrow argmin_{b \in B \setminus \{b_i^1\}} Q_{bi}^t$;
  $i^* \leftarrow \max_i Q_{b_i^2 i}^t - Q_{b_i^1 i}^t, b^* \leftarrow b_i^1, \zeta' \leftarrow \{i | x_{b^* i}^t = 1\} + \{i^*\}$;
  $\{\tilde{u}_{b^*,i}^t, \tilde{k}_{DS,b^*i}^t, \tilde{k}_{U,b^*i}^t, \tilde{k}_{DT,b^*i}^t | i \in \zeta'\} \leftarrow MCPA(\zeta', b^*)$;
  calculate $P_{b^*}^t$ using 1.11;
  if $MCPA(\zeta', b^*) \neq 0$ *and* $P_{b^*}^t \leq L'^t_b$ **then**
    $x_{b^* i}^t \leftarrow 1$;
    for $\forall i \in \zeta'$ **do**
      $k_{U,b^*i}^t \leftarrow \tilde{k}_{U,b^*i}^t, k_{DT,b^*i}^t \leftarrow \tilde{k}_{DT,b^*i}^t$;
      $u_{b^* i}^t \leftarrow \tilde{u}_{b^*,i}^t, k_{DS,b^*i}^t \leftarrow \tilde{k}_{DS,b^*i}^t$;
    $Q_{bi^*}^t \leftarrow \infty, \forall b \in B$, $visit\_UE \leftarrow visit\_UE$+1 ;
  else
    $Q_{b^* i^*}^t \leftarrow \infty, \zeta' \leftarrow \zeta' \setminus \{i^*\}$;
return X^t, ψ^t

to SRSUs. In the meantime, some VUs will request to download videos as the *delay tolerant downlink data*. To simulate realistic VU movement and topology, we take a $1000 * 800$ (meters) rectangular area in Brooklyn, New York City, as shown in Fig. 1.5. We use historical vehicular traffic data in this area collected by New York State Department of Transportation [11]. Fig. 1.5 also shows the placement of 20 SRSUs used in our simulation environment. We list the related simulation parameters in Table 1.2. The duration of each time slot τ is 1 second. Because the duration of the handover process in LTE-A can be less than 100 ms [43], we set κ=0.1. Total

Algorithm 4: MPCA

Input: ζ, b

Output: 0 or $\{\tilde{u}_{bi}^t, \tilde{k}_{DS,bi}^t, \tilde{k}_{U,bi}^t, \tilde{k}_{DT,bi}^t | i \in \zeta\}$

for $\forall i \in \zeta$ **do**
  calculate $\tilde{u}_{bi}^t, \tilde{k}_{DS,bi}^t, \tilde{k}_{U,bi}^t, \tilde{k}_{DT,bi}^t$ using Eq. 1.20 and 1.25 ;
if *constraints 1.7-1.9, 1.17f are satisfied for SRSU b and every* $i \in \zeta$ *satisfies Eq. 1.17e and* $H_b^t > 0$ **then**
  return $\{\tilde{u}_{bi}^t, \tilde{k}_{DS,bi}^t, \tilde{k}_{U,bi}^t, \tilde{k}_{DT,bi}^t | i \in \zeta\}$;
  else
    return 0;

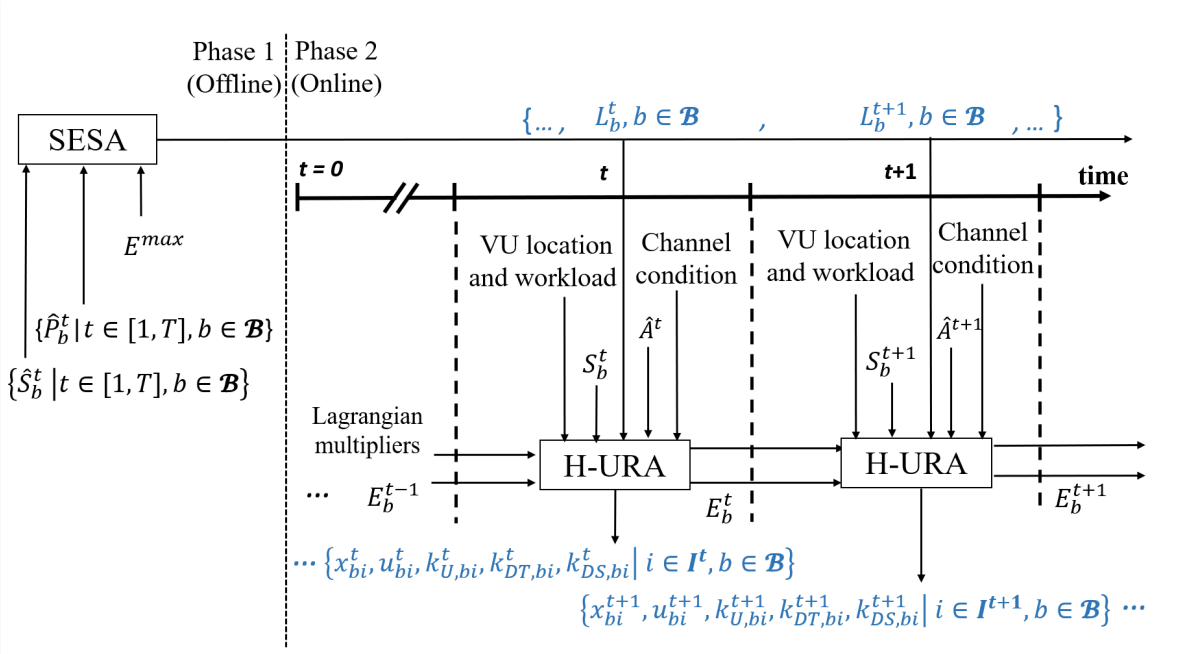

Figure 1.4. Breakdown of TQMA algorithm

simulation time is 24 hours, starting from 9 AM to include both day and night. Therefore, T is 86400.

At the beginning of each time slot, VUs enter the area from both ends of each street following a Poisson distribution with rate Θ. Each VU travels with predetermined route and speed. The travel route decision, speed, and Θ are set in a manner that the average traffic volume of each street satisfies the historical data in [11]. Furthermore, the channel model and the transmit

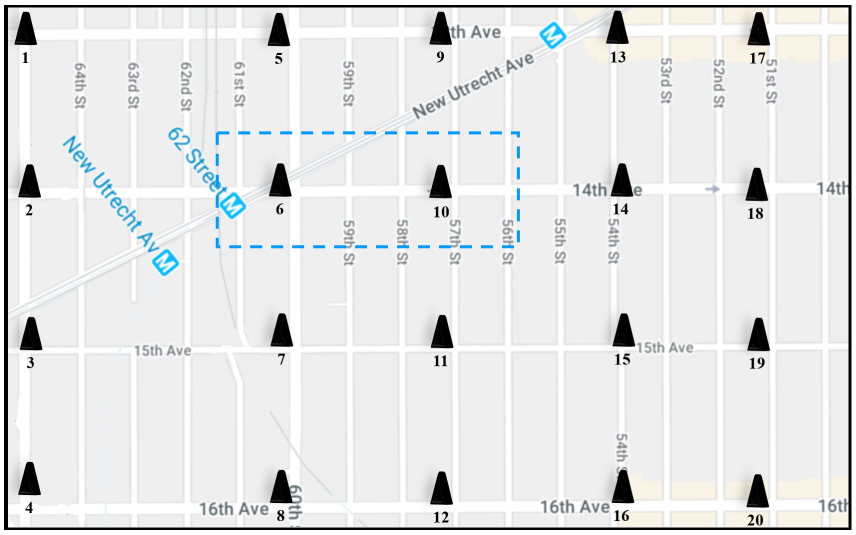

Figure 1.5. A neighborhood in Brooklyn, NY and SRSU deployment studied in this paper [42]

Table 1.2. Key parameters in simulation framework

Parameter	Value	Parameter	Value
$K_{D,b}$	710	$p_{U,b}$	0.0067 W/subcarrier
$K_{U,b}$	710	$p_{U,b}$	0.0067 W/subcarrier
U_b	4744 MIPS	$p_{N,b}$	10 W
$p_{M,b}$	4.8 W	γ	0 dB
$p_{C,b}$	6.25 W	N_0	-174 dBm/Hz
p_b	30 dBm	E_b^0	0Wh
p_i	23 dBm	E^{max}	600 Wh

Parameter	Value
g_{bi}^t, h_{ib}^t	Pathloss and slow fading: Manhattan grid layout (B1) in [44]; Fast fading: Nakagami-m distribution [45]

power of SRSUs and VUs are listed in Table 1.2 [46, 47]. We set $A = 40$ for the next SRSU prediction. To model the workload, we assume that each VU will upload an H.264 encoded video file with the data rate ω_i^t be uniformly distributed between 11 and 13.5 MB/s. It requires 10 million instructions per second (MIPS) as c_i^t for video processing, including decoding and object detection [8] at the MEC. We assume the size of the *delay sensitive downlink data* δ_i^t is uniformly distributed between 0.1 and 0.3 MB and the delay constraint d_i^t is 0.1s. In the meantime, VUs will have 0.25 probability to download a video file with size uniformly distributed between 7 and 9 MB as the *delay tolerant downlink data*, which has delay constraint θ_i^t = 1s.

We model the downlink and uplink channel gains, g_{bi}^t and h_{ib}^t, by using Manhattan grid layout (B1) in [44] as the pathloss and slow fading, and the Nakagami-m distribution [45] as the fast fading, which have been widely used by the industry [46, 48] and are shown to be sufficient to model the vehicular communication channel [45].

The subcarriers are allocated to VUs in groups, and each group has 12 subcarriers (i.e., W=180kHz/group) [49]. Multiple groups of subcarriers can be allocated to the same VU simultaneously. We assume each SRSU can utilize 710 subcarrier groups concurrently for each direction of transmission. To improve the inter-cell interference, we adopt the frequency reuse mapping technique [50] with reuse factor 3.

We model the MEC server of an SRSU by a Raspberry Pi 2 Model B [51], which is used to serve the offloaded workloads. Its corresponding computing resource and power consumption profiles are specified in Table 1.2.

For the solar generation profile, we use the data collected at multiple sites in UC San Diego [10]. We normalize the solar energy data and assume the solar panel size is 1 m^2 for each SRSU. We use the proposed algorithm in [10] to predict solar generation profiles 24 hours in advance.

To compare against SESA, we use a best-effort technique, denoted as the Best effort Solar Energy scheduling Algorithm (BSEA). BSEA consists of a best-effort solar energy scheduling strategy and the same user association and SRSU resource allocation technique (H-URA) as TQMA. BSEA allows each SRSU to serve the associated VUs without constrained by the scheduled solar energy.

Another comparison is the Green energy and delay Aware User association and Resource Allocation (GAURA) algorithm proposed by [14]. GAURA is a combination of battery charging/discharging scheduling, SBS transmit power control, and user association algorithms, which is the closest approach to TQMA compared to other works. We assume GAURA follows the same way of H-URA to allocate subcarriers for uplink and the delay tolerant downlink data transmission. On the other hand, to fulfill the delay constraint in 1.6, we assume that GAURA will

allocate $k^t_{DS,bi}$ downlink subcarriers and u^t_{bi} computing speed to VU i by the ratio: $u^t_{bi} = 4k^t_{DS,bi}$.

We also compare TQMA with our previous approach, QLM [8]. We assume that QLM has accurate predictions of VU's location and workload.

In the following sub-section, we will first present a performance comparison of our proposed TQMA with BSEA, GAURA, and QLM. Second, to show the efficiency of the Phase 2 algorithm, H-URA, a dynamic programming based Optimal User association and Resource allocation Algorithm (OPTA) [52] is implemented. Since [52] does not solve phase 1, we use the proposed SESA as the Phase 1 algorithm. We will compare the performance of TQMA and OPTA to show the efficiency of our proposed Phase 2 algorithm, H-URA. We introduce and analyze the complexity of OPTA in section 1.7.3. Third, to show the gap between the optimal solution and the proposed TQMA algorithm, we implement the exhaustive search method for Problem 1.16. The complexity analysis of the exhaustive search method is listed in section 1.7.4. Finally, we will show the effect of solar energy prediction error on the performance of TQMA.

1.5.2 Simulation Results

We have implemented the proposed TQMA algorithm using MATLAB on a computer with a 3.8 GHz CPU, which is used to perform the offline battery scheduling and online user association and resource allocation for all the SRSUs in a neighborhood, like shown in Fig. 1.5. Note that a TQMA instance will be responsible for the SRSUs and the VUs of each such neighborhood. Since the battery scheduling algorithm SESA is run offline, we focus here on the run-time performance of H-URA. From our simulation-based experiments, the worst-case run-time of H-URA algorithm for a time slot is less than 180 ms. This is well below the time interval of 1s H-URA is executed (each time slot). Note that the input information (e.g., VU locations, workloads, and harvested solar) will not change dramatically during the 180 ms run-time of H-URA. Hence, we can conclude that H-URA is real-time, validating our time complexity based assertion in Section 1.4.

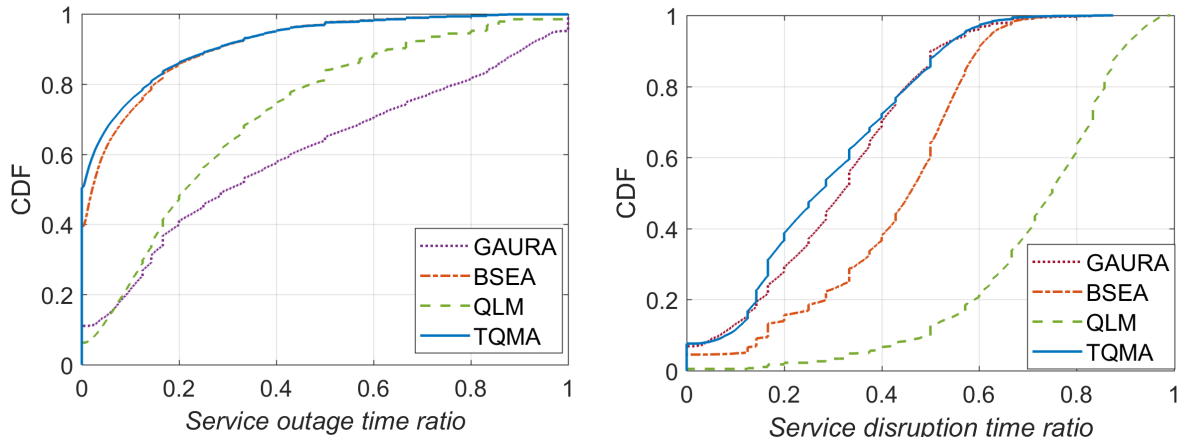

Figure 1.6. The empirical cumulative distribution function of (a) left, the service outage time ratio and (b) right, the service disruption time ratio for individual VUs.

Performance comparison of TQMA with other techniques

The weighted QoS loss performance of TQMA, BSEA, QLM, and GAURA are 0.125, 0.145, 0274, and 0.453, respectively. The performance of TQMA is the best compared to other techniques. To further discuss the effect of the above algorithms on individual VUs, we define *service outage time ratio* and *service disruption time ratio* for each VU as the following:

$$\textit{service outage time ratio} = \frac{\text{service outage time}}{\text{service request time}} \tag{1.26}$$

$$\textit{service disruption time ratio} = \frac{\text{service disruption time}}{\text{service request time}} \tag{1.27}$$

where the service outage time is the duration that this VU is experiencing the service outage, the service disruption time is the duration that this VU is experiencing the service disruption. The service request time is the duration that this VU is in the neighborhood and sending offloading demands.

In Fig. 1.6, we show the empirical cumulative distribution function (CDF) of the *service outage time ratio* and *service disruption time ratio* for the VUs. In Fig. 1.6a, 86.2% of the VUs

are served by the SRSUs for at least 80% of the service request time (*service outage time ratio* $<$ 0.2) by using TQMA. On the contrary, 85.8%, 47%, and 40% of the VUs are served by SRSUs for at least 80% of their service request time by using BSEA, QLM, and GAURA algorithms, respectively. The performance of BSEA is close to TQMA because they share the same H-URA algorithm. On the other hand, in Fig. 1.6b, we can see that 85.7% of the VUs have less than 50% of their service request time experiencing the service disruption (the *service disruption time ratio* $<$ 0.5) by using TQMA. Compared to TQMA, 9.6%, 59.6%, and 90.1% of the VUs have the *service disruption time ratio* $<$ 0.5 by using QLM, BSEA, and GAURA, respectively. QLM performs the worst because it will first consider associating a VU to the SRSU which provides the best signal strength, regardless of the VU's location, future movement, and the current associated SRSU. Compared to other algorithms, TQMA enables more VUs being served by SRSUs for longer duration while reducing their chances of experiencing service disruption.

Fig. 1.7 shows the weighted QoS loss performance comparison of the above algorithms under various system parameters (i.e., solar panel size, available computing speed, subcarrier groups, and battery capacity of SRSU). Fig. 1.7a shows the weighted QoS loss performance of these four algorithms under different solar energy availabilities, which are controlled by changing the solar panel size. TQMA has the best performance in terms of the weighted QoS loss among all the listed algorithms for different solar panel sizes. For instance, when the solar panel size equals 1 m^2, the performance of TQMA is 13.8% better than BSEA, 54.4% better than QLM, and 72.5% better than GAURA. The QoS loss of TQMA decreases while the solar panel size increases. However, the decrease starts to slow down and stops after the solar panel size exceeds 1.1 m^2. It is because the bottleneck of the performance becomes other limited resources after SRSU has enough solar energy.

From Fig. 1.7b, we can observe that the weighted QoS loss decreases when the available number of subcarrier groups of each SRSU increases. Again, TQMA outperforms other algorithms. The performance gap between TQMA and the second-best algorithm, BSEA, grows with the number of subcarrier groups. The gap grows from 0.0273 to 0.0353 when the number

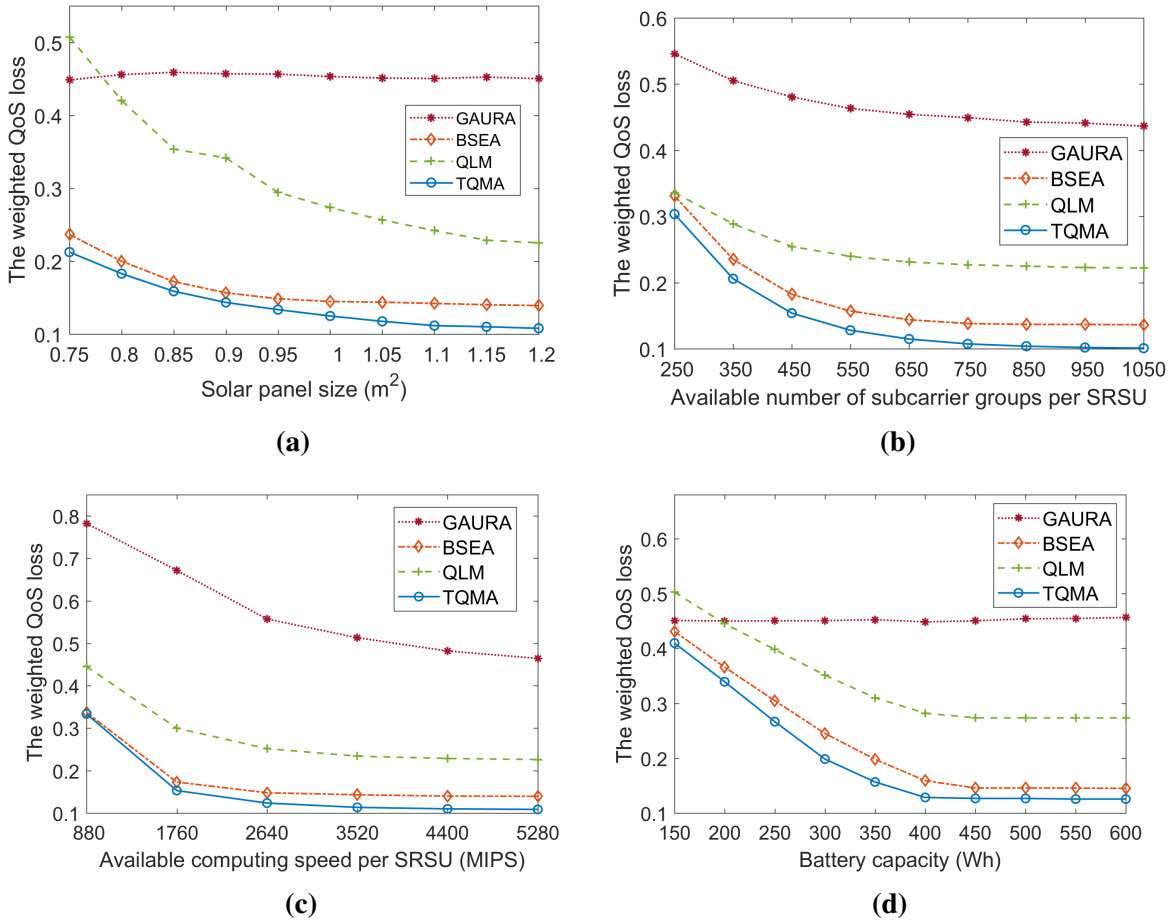

Figure 1.7. The weighted QoS loss performance of various algorithms on (a) upper left, different solar panel sizes, (b) upper right, different available subcarrier groups of SRSU, (c) lower left, different available computing speeds of SRSU, and (d) lower right, different battery capacities of SRSU.

of subcarrier groups increases from 250 to 1050, which shows that TQMA can more efficiently utilize these increased subcarrier resource. In Fig. 1.7c, the weighted QoS loss decreases when the available computing speed of each SRSU increases. Again, TQMA outperforms the other three algorithms under all conditions. Notice that the performance of TQMA improves slowly after the available computing speed exceeds 3520 MIPS. The weighted QoS loss only improves 0.0048 (i.e., 4%) from 3520 MIPS to 5280 MIPS. The performance of GAURA rises vastly in low available computing speed conditions, as its resource allocation mechanism (i.e., $u^t_{bi} = 4k^t_{DS,bi}$) will put a heavier burden on utilizing the computing speed than downlink subcarrier groups,

especially in low available computing speed conditions.

In Fig. 1.7d, the weighted QoS loss increases rapidly after the battery capacity decreases to a certain level. For TQMA, QLM, and BSEA, we can observe that the critical point is 400 Wh. The weighted QoS loss starts to increase below this capacity because the capacity cannot fulfill the SRSU's power demand at night when there is no solar energy generated. The results in Fig. 1.7 demonstrate the tradeoff between QoS and different resource availabilities, including solar panel sizes, battery capacities, MEC specifications, and configurations of SBS (subcarriers). This enables the service providers to identify what might be the best configurations of SRSU for expected solar generations and offloading demand profiles.

Performance Comparison with OPTA

In this comparison, we investigate the efficiency of our proposed Phase 2 algorithm, H-URA, by comparing TQMA to QLM and OPTA. To lower the complexity, we consider a smaller neighborhood surrounded by the dashed rectangle in Fig. 1.5. There are 2 SRSUs in this neighborhood and less than 14 VUs during peak hours. We equally divide the available computing speed into 5 levels and allocate them to each VU by levels. Subcarriers are divided into 5 groups. Fig. 1.8a shows the weighted QoS loss performance of QLM, TQMA, and OPTA when the solar panel size varies from 0.76 m^2 to 0.98 m^2. When the solar panel size is 0.9 m^2, the performance gap is 0.109 between TQMA and OPTA, while the gap between QLM and OPTA is 0.244. In terms of the peak time complexity (i.e., the recorded longest computation time for a time slot), TQMA takes 0.0938s while OPTA requires 333.5s when running on a 3.8 GHz CPU.

In Fig. 1.8b, we present the weighted QoS loss performance of these 3 algorithms on different average VU density scenarios. The average VU density is calculated as $\sum_t^T |I^t|/T$, where I^t is the VU set at the t^{th} time slot and T is the total number of time slots. We control the value of the average VU density by changing the vehicle generating rate Θ. In the meantime, Fig. 1.8c shows the corresponding peak time complexity. The gap between TQMA and OPTA increases linearly from 0.01 to 0.211 when the average VU density increases from 1.6 to 8.1.

However, the corresponding peak time complexity of OPTA increases exponentially from 78.7s to 1047s. Although OPTA's dynamic programming Phase 2 algorithm provides promising QoS performance under different solar energy availability and average VU density conditions, it is prohibitively expensive in terms of time complexity. On the contrary, our proposed Phase 2 algorithm H-URA can keep the peak time complexity low for real-time decision making while compromising somewhat on optimal QoS performance though significantly better than QLM.

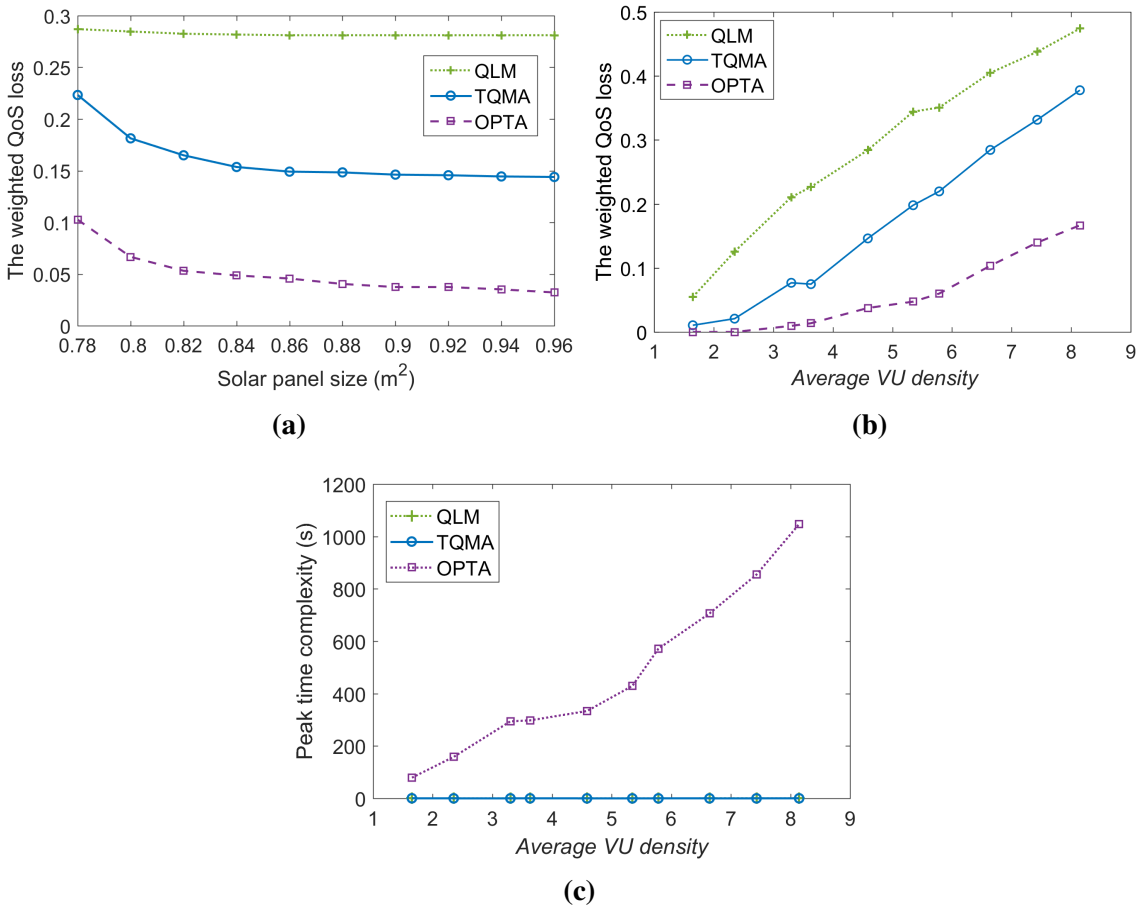

Figure 1.8. (a) the weighted QoS loss performance of various algorithms on different solar panel sizes, (b) the weighted QoS loss performance of various algorithms on different *Average VU densities*, and (c) the peak time complexity of various algorithms on different *Average VU densities*.

Performance Comparison with Exhaustive Search

In this experiment, we investigate the efficiency of our proposed TQMA algorithm for solving Problem 1.16 by comparing with an exhaustive search method, which finds the optimal solution for Problem 1.16. The exhaustive search method searches all the solar energy scheduling possibilities and uses dynamic programming algorithm (i.e. OPTA's Phase 2 algorithm) for user association and resource allocation for each solar energy scheduling possibility. Fig. 1.9 shows the performance comparison of BSEA, TQMA, OPTA, and the exhaustive search method. As shown in 1.7.4, the complexity of the exhaustive search method is $O(TU^B K_U^B K_D^{B+1} l_{max}^2 B^2 T!^{\hat{S}})$, where ! is the factorial function, $\hat{S}$ is the maximum harvested solar energy of a time slot, and l_{max} is the maximum number of VUs for a time slot. Due to the extremely high complexity, in this experiment we simulate only 4 time slots to represent a day (i.e., the gap between each slot is 6 hours). The granularity of the solar energy scheduling decision is 10 W. We consider the same neighborhood as in the previous subsection. Similar to the previous subsection, we control the value of the *average VU density* by changing the value of the vehicle generating rate Θ. We equally divide the available computing speed into 5 levels and allocate them to each VU by levels. The subcarriers are divided into 5 groups. Compared to BSEA, where no solar energy scheduling algorithm is implemented, TQMA's performance is closer to the optimal solution. The performance gap between TQMA and the optimal solution is 0.15 under regular traffic conditions (i.e. *average VU density* = 5.0). However, the peak time complexity of TQMA is 19.2ms, while the exhaustive search method requires 192,038s when running on a 3.8 GHz CPU. Therefore, finding the optimal solution is prohibitively expensive in terms of peak time complexity. To show their performances for high VU density scenarios, we increase Θ and create a 5.5 *average VU density* scenario. The weighted QoS loss gap between TQMA and the optimal solution is 0.14, which is almost the same as the gap when the average VU density is 5.0. But the peak time complexity of the exhaustive search method increases to 228,220 s while TQMA only requires 20.3 ms. Therefore, our proposed TQMA is more efficient in terms of both the

peak time complexity and the weighted QoS loss.

To further investigate the cause of the performance gap between TQMA and the exhaustive search method, we include the performance of OPTA in Fig. 1.9. OPTA achieves the same weighted QoS loss as the optimal value. Because TQMA and OPTA share the same solar energy scheduling algorithm, the performance of OPTA shows that the gap between TQMA and the optimal value is due to the heuristic user association and resource allocation. Moreover, OPTA also demonstrates an approach for narrowing the performance gap without sacrificing largely on the time complexity. Its peak time complexity is 144.7s under regular traffic conditions, which is between TQMA (i.e. 19.2 ms) and the exhausted search method (i.e. 192,038 s). Note that the performance of OPTA converges to the optimal value in Fig. 1.9 because this experiment is conducted under a limited-scale scenario. In fact, OPTA is not an optimal approach for Problem 1.16 as it considers only one solar energy scheduling possibility.

Effect of prediction error on TQMA

Finally, in this sub-section, we present the effect of the prediction error of solar generation on the performance of TQMA. For each experiment, we run TQMA two times with the same simulation settings. For the first time, we use the predicted solar generation profile for SESA. The second time, we use the exact solar generation profile (no prediction error) for SESA. The simulation results of two different days are shown in Table 1.3, where *SD* is the *service disruption rate* and *SO* is the *service outage rate*. For day number 1, we observe prosperous and less intermittent solar generation since the weather is mostly sunny. Therefore, the prediction error is very small. We observe that its weighted QoS loss, *SD*, and *SO* are very similar with and without solar prediction error (compared to no prediction error). The weighted QoS loss of using solar prediction increases by 0.5(4.2%) compared to no prediction case. On the other hand, for day number 2, we observe poor and highly intermittent solar generation since the weather is partly sunny and partly cloudy. Consequently, the prediction error is worse than day number 1. The weighted QoS loss of using solar prediction increases by 0.6(1.6%) compared to

Table 1.3. Performance with prediction error

	Solar Prediction Error			Performance		
Day 1	**MAE**	**MAPE(%)**	**RMSE**	**QoS loss**	*SO*	*SD*
Prediction Error	3.31	6.61%	5.73	12.5	8.74	37.5
No error	-	-	-	12.0	8.23	37.6
Day 2	**MAE**	**MAPE(%)**	**RMSE**	**QoS loss**	*SO*	*SD*
Prediction Error	8.63	49.8%	18.29	37.8	35.3	34.2
No error	-	-	-	37.2	34.8	25.1

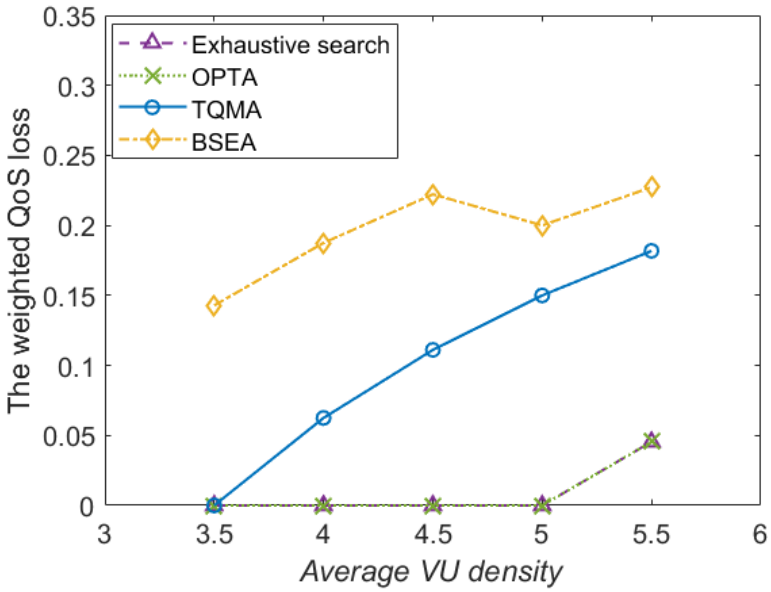

Figure 1.9. The weighted QoS loss performance of TQMA, OPTA, and BSEA, compared with the optimal solution using exhaustive search

no prediction error case. Its SO increases by 0.5 % and SD drops by 0.9%. In this case,*SD* drops because *SO* increases. If a VU is experiencing service outage, it will not be counted as service disruption. Although the prediction error increases, the performance drop of TQMA in terms of the increased weighted QoS loss is still under 5%.

1.6 Conclusion

In this paper, we propose a real-time QoS loss minimization algorithm to support the offloading of delay-sensitive vehicular applications in a Solar-powered RSU network. The algorithm involves a two-phase approach: (i) the solar energy scheduling phase and (ii) the user association and resource allocation phase. SESA and H-URA respectively are developed for these two phases. A complete algorithm, TQMA, is proposed by integrating the above two algorithms which our simulation shows to significantly reduce the weighted QoS loss for the total operation time compared to existing techniques under various resource availabilities. The results help service providers and city planners to identify adequate SRSU configurations for expected solar energy generation and offloading demands.

In the next chapter, we will present a framework that utilizes both the VEC server computing resource and the local computing resource of each vehicle to further reduce the execution delay of the vehicular applications through task partitioning and offloading. Additionally, the framework can adaptively adjust the VEC server's system configuration and the vehicle's application-level performance according to the available resources at the Solar-powered RSU.

Chapter 1, in full, is a reprint of the material as it appears in IEEE Transactions on Vehicular Technology 2020, Yu-Jen Ku, Po-Han Chiang, and Sujit Dey. The dissertation author was the primary investigator and author of this paper.

1.7 Appendix

1.7.1 Proof of Eq. 1.19

$$\begin{aligned} C^t_{drop} + \kappa C^t_{handover} &= \sum_{i=I^t}(1 - \sum_{b=B} x^t_{bi}) + \kappa \sum_{i=I^t}(\sum_{b=B} x^t_{bi})(1 - \sum_{b=B} x^{t-1}_{bi} x^t_{bi}) \\ &= l^t - \sum_{i=I^t}\sum_{b=B} x^t_{bi} + \kappa \sum_{i=I^t}(\sum_{b=B} x^t_{bi} - \sum_{b=B} x^t_{bi}\Omega(x^t_{bi}, x^{t-1}_{bi})) \\ &= l^t - \sum_{i=I^t}\sum_{b=B} x^t_{bi} + \kappa \sum_{i=I^t}\sum_{b=B} x^t_{bi} - \kappa \sum_{i=I^t}\sum_{b=B} x^t_{bi}\Omega(x^t_{bi}, x^{t-1}_{bi}) \\ &= l^t + \sum_{i=I^t}\sum_{b=B}(-1 + \kappa - \kappa\Omega(x^t_{bi}, x^{t-1}_{bi}))x^t_{bi} \end{aligned} \tag{1.28}$$

1.7.2 Proof of Lemma 1.4.1

With fixed Lagrangian multipliers $\lambda^t_b, \mu^t_b, \rho^t_b, \sigma^t_b$, problem in 1.22f is reduced to:

$$\underset{x^t_{bi}, b\in B, i\in I^t}{\text{minimize}} \quad \sum_{i\in I^t}\sum_{b\in B} z^t_{bi} x^t_{bi} + \tag{1.29a}$$

$$\sum_{b\in B} \lambda^t_b (\sum_{i\in I^t} x^t_{bi} v^t_{bi1} - U_b) + \tag{1.29b}$$

$$\sum_{b\in B} \mu^t_b (\sum_{i\in I^t} x^t_{bi} v^t_{bi2} - K_{U,b}) + \tag{1.29c}$$

$$\sum_{b\in B} \rho^t_b (\sum_{i\in I^t} x^t_{bi} v^t_{bi3} - K_{D,b}) + \tag{1.29d}$$

$$\sum_{b\in B} \sigma^t_b (\sum_{i\in I^t} x^t_{bi} v^t_{bi4} - L'^t_b) \tag{1.29e}$$

$$\text{subject to} 1.17c - 1.17e \tag{1.29f}$$

The objective function of 1.29f can then be rewritten as,

$$\sum_{i\in I^t}\sum_{b\in B} x^t_{bi}(z^t_{bi} + \lambda^t_b v^t_{bi1} + \mu^t_b v^t_{bi2} + \rho^t_b v^t_{bi3} + \sigma^t_b v^t_{bi4}) - \sum_{b\in B}(\lambda^t_b U_b + \mu^t_b K_{U,b} + \rho^t_b K_{D,b} + \sigma^t_b L'^t_b) \tag{1.30}$$

where the second term is a constant. Therefore, 1.29f is equal to,

$$\underset{x^t_{bi}, b\in B, i\in I^t}{\text{minimize}} \quad \sum_{i\in I^t}\sum_{b\in B} x^t_{bi} q^t_{bi} \tag{1.31a}$$

$$\text{subject to} 1.17c - 1.17e \tag{1.31b}$$

with $q^t_{bi} = z^t_{bi} + \lambda^t_b v^t_{bi1} + \mu^t_b v^t_{bi2} + \rho^t_b v^t_{bi3} + \sigma^t_b v^t_{bi4}$.

Note that q^t_{bi} and constraints 1.17c-1.17e are separate for different VUs. Therefore, the optimal solution of 1.29f (which is also the optimal solution of 1.22f) will be finding the SRSU which minimizes q^t_{bi} under constraints 1.17c-1.17e for each VU.

1.7.3 OPTA Algorithm

Since we have introduced SESA in Section 1.4, in this appendix, we analyze the complexity of OPTA's Phase 2 algorithm, which is based on dynamic programming. For a given instance of Phase 2, integers $i, n, \alpha_1, ..., \alpha_{3B}$, we use $f(i, n, \alpha_1, ..., \alpha_{3B})$ to represent the optimal value of 1.17 with B SRSUs, which considers the VU set $\{1, 2, ..., i\} \in I^t$ and allows at most n dropped VUs. Furthermore, each SRSU b utilizes exactly α_{3b-2} amount of computing speed, α_{3b-1} uplink subcarriers, and α_{3b} downlink subcarriers. To track the optimal user association and resource allocation decisions, we let $X(i, n, \alpha_1, ..., \alpha_{3B})$ and $\Psi(i, n, \alpha_1, ..., \alpha_{3B})$ be the corresponding user association and computing speed allocation of VU i for the instances $i, n, \alpha_1, ..., \alpha_{3B}$. We only track the allocation of computing speed because once we get x^t_{bi} from X, the optimal $k^t_{U,bi}, k^t_{DT,bi}$ can be derived by choosing the smallest possible values which satisfy workload constraints 1.4, 1.5. With the recorded u^t_{bi} in Ψ, we can calculate the optimal $k^t_{DS,bi}$ by delay constraint 1.6.

The core formula of OPTA is,

$$f(i,n,\alpha_1,...,\alpha_{3B}) = \begin{cases} \infty, & \text{if } n < 0 \\ \infty, & \text{if } \exists b \in B, \alpha_{3b-2} < 0 or \alpha_{3b-1} < 0 or \alpha_{3b} < 0 \\ 0, & \text{if } i = 0, n \geq 0, \alpha_{3b-2} \geq 0 or \alpha_{3b-1} \geq 0 or \alpha_{3b} \geq 0, \forall b \in B \\ \infty, & \text{if } \exists b \in B, P_b^t(\alpha_{3b-2}, \alpha_{3b-1}, \alpha_{3b}) < {L'}_b^t \\ min(A_1, A_2) & otherwise \end{cases} \tag{1.32}$$

where $P_b^t(\alpha_{3b-2}, \alpha_{3b-1}, \alpha_{3b})$ returns the corresponding power consumption of SRSU b for utilizing α_{3b-2} amount of computing speed, α_{3b-1} uplink subcarriers, and α_{3b} downlink subcarriers. ${L'}_b^t = min(L_b^t, E_b^{t-1} + S_b^t)$ is for SRSU b to follow constraint 1.17f. $A_1 = 1 + f(i,n,\alpha_1,...,\alpha_{3B})$ is the optimal value when choosing not to serve VU i. Finally, A_2 is the optimal value considering all possible values of $x_{bi}^t, u_{bi}^t b \in B$ for VU i, and can be defined as,

$$A_2 = \underset{b, x_{bi}^t, u_{bi}^t}{\text{minimize}} \quad z_{bi}^t + f(i-1,n,\alpha_1,...,\alpha_{3b-2} - u_{bi}^t, \alpha_{3b-1} - k_{U,bi}^t, \alpha_{3b} - k_{DT,bi}^t - k_{DS,bi}^t, ..., \alpha_{3B}) \tag{1.33}$$

with $k_{U,bi}^t$, $k_{DT,bi}^t$, and $k_{DS,bi}^t$ be the optimal numbers of uplink and downlink subcarriers correspond to x_{bi}^t and u_{bi}^t. Note that in Eq. 1.33, if $\eta_{D,bi}^t > \gamma$, $z_{bi}^t = -1 + \kappa - \kappa\Omega(x_{bi}^t, x_{bi}^{t-1})$, otherwise $z_{bi}^t = \infty$.

f is initialized by an arbitrarily large value. X and Ψ are initialized as zero matrices. We recursively calculate the elements in f for i from 1 to l, n from 1 to l, α_{3b-2} from 1 to U_b, α_{3b-1} from 1 to $K_{U,b}$, α_{3b} from 1 to $K_{D,b}$, $\forall b \in B$, until all the elements in f are updated. We record the corresponding optimal values of x_{bi}^t and u_{bi}^t in $X(i,n,\alpha_1,\ldots,\alpha_{3B})$ and $\Psi(i,n,\alpha_1,\ldots,\alpha_{3B})$, respectively. The optimal value of Problem 1.17 is then the smallest element in matrix $f(l,l,:,\ldots,:)$ (i.e., f with the specific indices, $i = l, n = l, 1 \leq \alpha_{3b-2} \leq U_b, 1 \leq \alpha_{3b-1} \leq K_{U,b}$, and $1 \leq \alpha_{3b} \leq K_{D,b} \forall b \in B$). We then calculate the optimal $x_{bi}^t, u_{bi}^t, k_{U,bi}^t, k_{DT,bi}^t$ and $k_{DS,bi}^t$ for VU

i iteratively from $i = l$ to $i = 1$, by using X, Ψ, and the indices correspond to the minimum element.

The time complexity of OPTA is $O(U^B K_U^B K_D^{B+1} l^2 B^2)$ if all the SRSUs have the same computing capacity U, number of uplink subcarriers K_U, and number of downlink subcarriers K_D. The complexity grows exponentially with the number of SRSUs in the network. Since the value of U, K_U, and K_D are usually very large, OPTA will be prohibitive in terms of run-time if there are more than 2 SRSUs in the network.

1.7.4 Complexity Analysis of the Exhaustive Search Method

Here we perform a complexity analysis of the exhaustive search method for Problem 1.16. The optimal solution of Problem 1.16 requires the solar energy to be optimally scheduled to each time slot, while the VUs are associated with the optimal SRSU and SRSU resources are optimally allocated. For the sake of simplicity of analysis, we assume each SRSU has the same value of downlink subcarriers (i.e., K_D), uplink subcarriers (i.e., K_U), and computing capacity (i.e., U). By dynamic programming analysis in section 1.7.3, the complexity of the Phase 2 problem is $O(U^B K_U^B K_D^{B+1} l^2 B^2)$ for each time slot, where B is the number of SRSU and l is the current number of VU. On the other hand, since energy is continuous, there are unlimited possibilities of how many portions of the generated solar energy can be used in the current time slot and how the rest of it can be scheduled in the future time slots, so as to the energy stored in the battery. For simplicity, we assume the granularity of energy is 1 W and the maximum harvested solar energy for each time slot is $\hat{S}$. For the t^{th} time slot, because every 1 W of the harvested solar energy can be scheduled to any time slot $t' \in [t, T]$, there are $O((T-t+1)^{\hat{S}})$ scheduling possibilities. Therefore, for the overall operation time, there are $O(\prod_{t=1}^{T} (T-t+1)^{\hat{S}}) = O(T!^{\hat{S}})$ possible solar energy scheduling strategies will be searched, where ! is the factorial function. Consequently, with $l_{max} = \max_t l^t$, the complexity of exhaustively searching the optimal solution of Problem 1.16 is $O(TU^B K_U^B K_D^{B+1} l_{max}^2 B^2 T!^{\hat{S}})$.

Chapter 2

Adaptive Computation Partitioning and Offloading in Real-time Sustainable Vehicular Edge Computing

2.1 Introduction

This chapter presents our study to further reduce vehicular applications' execution delay through task partitioning and offloading technologies. Additionally, we will present a framework that can adaptively adjust the vehicular edge computing (VEC) server's system configuration and the vehicle's application-level performance according to the available resources at the Solar-powered Road-side Unit (RSU). The rapid advancement in vehicular technology in recent years has enabled modern vehicles to be equipped with a wide range of vehicular applications, many of which are based on compute-intensive machine learning based algorithms. The vehicular local computing (VLC) units often cannot satisfy the computing demands of such applications, due to limited computing resources, or contention with other applications. A promising solution to resolve this problem is using the emerging new generation of RSUs, consisting of a small cell base station (SBS) and a VEC server [2, 3]. The VEC servers being one hop wireless distance away from the Vehicle Users (VUs), provide much lower communication delay compared to using conventional cloud computing resources, or even mobile edge computing units in wireless networks. VEC servers, although inferior to cloud or mobile edge computing servers, have more

computing capabilities than individual VLC resources in most vehicles. By VUs offloading the compute-intensive vehicular applications like object detection to RSUs, VUs can receive better service quality and improve driving experience.

However, transportation systems need dense deployment of RSUs, especially in urban areas, to support the high density of VUs. The dense deployment will significantly increase the cellular networks' energy consumption, thus worsening the carbon footprint. Recent studies have projected 110 million tons of carbon dioxide equivalent (CO_{2e}) emitted by the operation of base stations in global cellular networks in 2030 [4, 53]. Therefore, the future dense RSUs should be deployed without increasing the cellular network's greenhouse gas emission burden. In our previous work [54] , we proposed the use of Solar-powered RSU (SRSU), which consists of SBS, VEC server, and a self-sustained solar energy system. Note that in an SRSU, the generated solar energy is limited and fluctuating. If solar energy cannot meet the SRSU's power demand, the SRSU will need to reduce its computing and communication loads by preventing some of the VUs from offloading their applications to the VEC server.

In this work, we assume that each VU has a VLC node which can be supplemented with VEC resource to execute the VU's application. To efficiently utilize the computation resources of VLC node and VEC server, we consider a dynamic offloading scenario, where different subtasks of an application can be chosen to be either executed locally at the VLC node, or offloaded to a VEC server. The dynamic offloading decisions will depend on current computing, communication, and energy resources of the serving SRSU, as well as the VLC node capacity and channel condition for each VU.

In our previous work [54], we aimed at minimizing the disruption of vehicular applications due to the limited solar energy supply in real-time by optimally partitioning and executing the application tasks to either VLC nodes or VEC servers. The vehicular application is considered as disrupted when its delay requirement cannot be satisfied. However, the proposed method does not consider the potential latency improvement under an energy constraint by dynamically changing the system configuration of the VEC server. In one of our recent studies [55], we showed

the average computation delay can be further optimized satisfying a given energy constraint by appropriately configuring the CPU and GPU frequencies of the VEC server. Additionally, the proposed offloading method in [54] does not consider the potential delay improvement achievable by adapting application-level performance parameters. For example, machine vision based vehicular applications can lower their processed image quality for reducing the size of the data for offloading to further minimize the transmission delay, however, with possible impact on application-level performance, such as object detection accuracy.

In this paper, given the current channel condition and VLC node capacity of each VU, as well as the current computing, communication and energy resources of the SRSU, we aim at minimizing the end-to-end delay of the vehicular application and maximizing the application's performance. We propose to dynamically determine the task partitioning and offloading, VEC server's system configuration, and VUs' application-level performance adaptions. The decisions are calculated in real-time to accommodate the rapidly changing locations and channel conditions of the VUs. To show in real-world the benefit of our proposed method, we consider a vehicular object detection application, which is an essential building block for various complex vehicular applications such as Advanced driver-assistance systems (ADAS), path planning, and navigation. We implement the object detection application using SSD-MobileNetV2 [56] on an edge computing platform Nvidia Jetson TX2 Board [57]. With extensive experiments, we establish empirical energy consumption, end-to-end delay, and detection accuracy models, which are used in simulation-based evaluations for the proposed method. The simulation results show that our proposed approach can significantly reduce the end-to-end delay while maximizing the detection accuracy compared to existing strategies.

The main contributions of this work are summarized below.

(a) To the best of our knowledge, this is the first work to optimize delay and accuracy performance of a vehicular object detection application for the SRSU-assisted VEC system using task partitioning and offloading, as well as joint system and application-level

adaptations.

(b) Specifically, we develop a technique which determines in real-time the optimal VEC server's hardware configuration, image quality for detection, and task partitioning and offloading decisions for an SRSU-assisted VEC system.

(c) Using a real-world edge computing platform, we establish empirical models of the energy consumption, computing capacity, end-to-end delay, and accuracy for an SRSU-assisted VEC system.

(d) To demonstrate the effectiveness of the proposed technique, we develop a simulation framework consisting of real-world solar generation, urban traffic traces, and the above empirical models. The simulation results show that the proposed approach significantly improves the end-to-end delay and accuracy compared to existing techniques.

2.2 Related Work

There have been many studies on computation task offloading for vehicular edge computing [58–61]. These studies focus on task offloading strategies which leverage computing resources at the edge to minimize the task completion delay [58, 60, 61], while maximizing the edge computing resource utilization [59] or the number of offloaded tasks [58]. Their approaches address the challenges in highly varying bandwidths under the constraint on computation delay [60] or vehicle's energy consumption [61] for real-time applications. However, these techniques do not study the trade-off between leveraging VLC node and VEC server, and hence can not be used for task partitioning according to the computing capacities of both VLC node and VEC server. Since in our considered scenario, both the capacities of VLC node and VEC server are limited, these resources need to be carefully allocated to computation tasks to achieve real-time computation delay.

To facilitate computation capacity-aware task offloading, [62] proposed a learning-based task partitioning and scheduling algorithm which partitions and assigns subtasks among multiple

VEC servers to minimize the completion delay and handover-induced service disruption. The technique requires data exchange between multiple RSUs, which will cause prohibitively large communication delay when applied to our high-data volume vehicular perception applications. [63–65], on the other hand, study the optimal computation task partitioning between VEC server and VLC node by proposing joint task partitioning and offloading as well as SBS communication and VEC server computation resource allocation methods. Among these studies, [63] and [64] aim at minimizing the system cost in terms of utilized communication and computation resources, under delay constraints while [65] jointly minimizing the task execution delay and the utilized VEC server's computing resources. However, the partitioning techniques proposed in these studies cannot be applied to subtasks with task dependencies as they assume the computation tasks can be arbitrary partitioned, offloaded, and executed in parallel.

To consider task dependency during partitioning and offloading, [66, 67] divide the sequential convolution layers of a Deep Neural Network (DNN) into several independent subtasks. These subtasks can be executed in parallel on multiple edge nodes to minimize task completion time [66] or the utilized edge server memory [67] as well as the communication overhead for task offloading. However, the proposed parallel partitioning techniques cannot leverage the potential reduction of communication delay with sequential partitioning. On the other hand, [68] and [69] enable local devices to early stop a DNN and offload the result to edge server. The edge server can choose to adopt the result or further execute the rest layers of the DNN. [68] aims at minimizing the execution delay and [69] aims at minimizing the utilized communication and computing resources on local and edge devices while maximizing the object detection results transmitted to the edge server under communication resource constraint. [70] proposes a real-time task partitioning and bandwidth allocation strategy to maximize the throughput (i.e. the number of processed data per second) using limited edge computing and communication resources. Although these task partitioning methods consider both the computing capacities in VLC nodes and VEC servers during decision making, they are not applicable to VEC servers whose operations are not only constrained by the limited communication and

computing resources, but energy availability.

The authors in [71] propose a joint task offloading and user association strategy for the multi-user mobile edge computing system to minimize the overall energy consumption of the users and edge server. However, firstly, the proposed method does not consider task dependency as they assume multiple mutually independent tasks in each user. Secondly, the challenges of minimizing the SRSU's energy consumption is different from the challenges of operating the SRSU under limited energy availability, as the computing and communication resources of SRSU will also be constrained due to the lack of energy. Both [72] and [18] consider renewable energy powered edge server. In [72], task partitioning and offloading as well as the utilization of renewable energy is determined online using Lyapunov optimization to maximize the number of offloaded tasks. The authors in [18] propose an online learning technique for partitioning and offloading of the incoming tasks as well as autoscaling of the computing capacity for the edge servers to jointly minimize the application delay, battery deprecation, and back up power usage. The learning technique is used to predict the system's long-term channel rate and workload states. However, these techniques do not consider the task dependency graph, and the corresponding transmitted data size is linear to the partitioned load. In practice, the computation loads of subtasks in a task graph are discrete and do not possess such linearity relationship with the input data. Therefore, the proposed theoretical approaches can not be applied to our problem.

To the best of our knowledge, this is the first study to consider not only the compute and communication-intensive, delay-sensitive dependency-aware task partitioning and offloading with the collaboration of VLC node and VEC server, but the challenges of utilizing limited communication, computing, and energy resources of an SRSU.

2.3 Systems Overview

In this section, we introduce an overview of the SRSU-assisted VEC system, including vehicular applications, solar energy-driven communication, and computing paradigms. For ease

Table 2.1. Summary of key notations and abbreviations in chapter 2

Notation	Description	Notation	Description
τ	the duration of current time slot	$\mathcal{I}$	current VU set
b	the SRSU for study	i	the index of VU
η_{bi}	the current SNR of uplink transmission from VU i to SRSU b	I	total number of VUs in the VU set
$r_{b,i}$	uplink transmission rate from VU i to SRSU b	$\mathcal{W}_i$	bandwidth allocated by SBS b to VU i
$\mathcal{K}$	subtask set for the vehicular application	$\omega_{k,q,i}$	input data size of subtask k of VU i using application parameter q
k	index of the subtask	$\omega_{(k+1),q,i}$	the output data size of subtask k of VU i using application parameter q
$T_{k,i}$	computation delay of subtask k	$T_{tx,i}$	transmission delay for transmiting the required data for offloading
d_i	end-to-end delay of VU i	a_i	accuracy of the vehicular application of VU i
QoS_i	QoS utility of VU i	E_R	Energy consumption of the SRSU
E_S	Energy consumption of the VEC server	E_B	Energy consumption of the SBS
E_c	Energy consumed for executing the offloaded subtasks	E_t	the current available amount of solar energy
$c_{l,i}$	CPU-GPU configuration of VLC node of VU i	c_e	CPU-GPU configuration of the VEC server
y_i	offloading strategy of VU i	q_i	compression level of Vu i
$\mathcal{Q}$	the set of available compression levels for VU	$\mathcal{C}$	the set of available CPU-GPU configurations for the VEC server
h	number of VUs using *Full Offloading*	n	number of total offloading VUs

of reference, we list the key notations in Table 2.1.

2.3.1 Network and Channel Models:

We consider an SRSU-assisted VEC system consisting of one serving SRSU b and multiple served VUs. The SRSU is equipped with a communication module SBS and a computation module VEC server. The operation time is divided into multiple time slots. Note that the following modellings and discussions are within a single time slot t, for simplicity, we do not attach superscript t for each variable. We denote the duration of time slot t as τ. For each time slot, there exists a set of VUs $\mathcal{I} = \{1, 2, ..., I\}$ in the coverage area of the SRSU b and the VUs' locations will vary in different time slots due to the mobility of the vehicles. For each VU $i \in I$, we denote $\eta_{b,i} = \frac{\rho_i * g_{b,i}}{N_0}$ as the current signal-to-noise ratio (SNR) of uplink transmission from VU i to the SBS of SRSU b. ρ_i is the transmit power of VU i, $g_{b,i}$ is the uplink channel gain, and N_0 is the noise level. The uplink transmission rate from VU i to SBS b can be represented as,

$$r_{b,i} = W_i * log_2(1 + \eta_{b,i}) \tag{2.1}$$

where W_i is the bandwidth allocated by SBS to VU i and the interference from other VUs is negligible with the use of Orthogonal Frequency-Division Multiplexing (OFDM) technology. We ignore the inter-cell interference by assuming it is mitigated by inter-cell interference coordination (ICIC) technologies, e.g. Fractional Frequency Reuse (FFR) [73]. We assume the wireless communication between the SBS and the VUs use C-V2X protocols [74] and the available bandwidths are evenly distributed among the VUs which require uplink transmission. We model the uplink channel gain, $g_{b,i}$, by using B1 Manhattan grid layout [44] as the pathloss and slow fading, and the Nakagami-m distribution [45] as the fast fading, which have been widely used by the industry [46, 48] and are shown to be sufficient to model vehicular communication channels [45].

Compared to the uplink data of the vehicular applications, e.g., the captured images or

radar point clouds, whose data sizes are usually more than several KBytes (images) or even several MBytes (point clouds) [75], the data sizes of the vehicular applications' computation results are very small. The computation results, such as the bounding boxes for object detection, classification, and fusion as well as basic safety messages (BSM) for collision detection, are less than 1 KBytes in terms of data size. Moreover, the downlink data rate is usually higher than the uplink data rate due to the higher transmission power [47]. Therefore, we ignore the impact of downlink data transmission in our study.

We assume the duration of time slot, τ, to be small enough so that $r_{b,i}$ is unchanged within one time slot [54]. Note that $r_{b,i}$ will still change across different time slots due to the mobility of VU.

2.3.2 Vehicular Task Model

Most vehicular applications involve computation tasks that can be expressed as a dependency graph of sequential subtasks. For example, Fig. 2.1 shows two dependency graphs of radar and camera-based vehicular applications. In Fig. 2.1(a), the detection range determination block decides the range of distance to perform the energy detection on the radar signal. Lane interval estimator and lane detector are applied if enough signal energy is detected. If a lane is present and the application detects possible departure due to the vehicle's speed, a departure warning will be sent to the driver [76]. In Fig. 2.1(b), the image captured by the camera will be decided and resized to be the input feature for the DNN-based object detection and classification, where we use SSD-MobileNetV2 [56] as an example. In Fig. 2.1(b), conv. is the convolution layer, which is the most common layer in SSD-MobileNetV2. If any object is detected, the application will return the coordination of the bounding boxes for the detected object.

We assume that at each time slot, every VU generates a computation task that consists of a set of $\mathcal{K} = \{1, 2, ..., K\}$ sequentially dependent subtasks, as shown in Fig. 2.2. That is, the data input of subtask k depends on the data output of subtask $k+1$. Therefore, subtask $k+1$ can start only after the completion of subtask k. For each subtask $k \in \mathcal{K}$ of VU i, we assume $\omega_{k,q,i}$

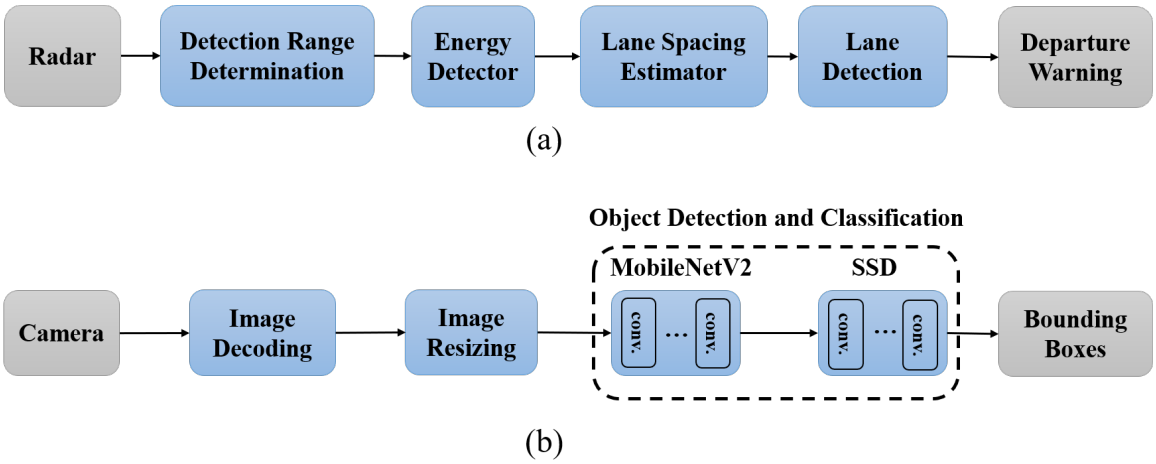

Figure 2.1. Task dependency graphs of (a). Radar signal-based lane departure warning system (b). DNN-based object detection and classification application for the cameras.

is the input data size and $\omega_{(k+1),q,i}$ is the output data size, where q is an application adaptation parameter. For example, if the considered vehicular application is vehicular machine vision, such as object detection and classification, q can be the encoding bitrate of the input image.

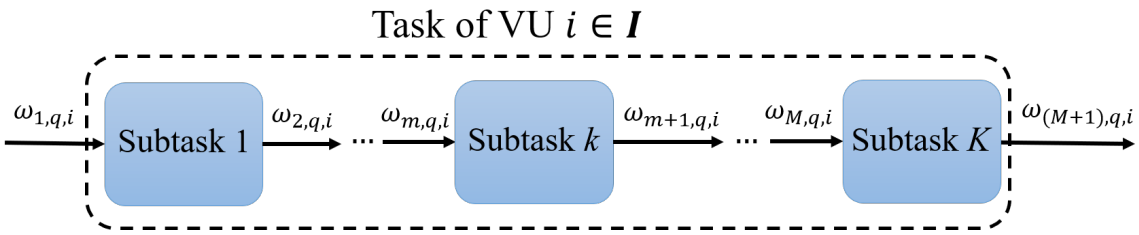

Figure 2.2. Subtask breakdown of a vehicular application

Each subtask can be executed locally in the VLC node or, offloaded and executed at the VEC server. In such cases of computation offloading to edge, data at the task-splitting point, e.g. subtask k', needs to be transmitted over the wireless communication channel, such that the first $\{1,2,..,k'\}$ subtasks are executed at the VLC node and the remaining $\{k'+1,...,K\}$ subtasks are executed at the VEC server.

2.3.3 Performance Metrics

Herein, we consider the performance metrics for object detection, which is a critical component in various complex vehicular applications as mentioned above. Therefore, we primarily focus on two object detection performance metrics - the end-to-end delay and the accuracy.

End-to-end delay The end-to-end delay in our study is defined as the summation of the computing delay of each subtask in $\mathcal{K}$ and the communication delay of transmitting the required data for computation offloading. Therefore, the end-to-end delay of VU i can be represented as,

$$d_i = \sum_{k=1}^{K} T_{k,i} + T_{tx,i} \tag{2.2}$$

where $T_{k,i}$ is the computing delay of subtask k. $T_{tx,i}$ is the transmission delay for offloading subtask k' and its subsequent subtasks to the VEC server, that is, for transmitting the input of subtask k' with data size $\omega_{k',q,i}$. Therefore, $T_{tx,i}$ can be defined as $\frac{\omega_{k',q,i}}{r_{b,i}}$. $T_{tx,i} = 0$ if all the subtasks of VU i are executed locally.

Note that as we focus on optimizing the end-to-end delay in this study, without loss of generality, we assume the data is processed frame-by-frame in the vehicular application, that is, subtask 1 starts processing the next input data after subtask K, which is the last subtask in $\mathcal{K}$, finishes processing the previous input. Therefore, queuing delay is negligible in the network.

Accuracy The accuracy of the object detection a_i of VU i can be represented as a function of the application level adaption parameter q, namely, $a_i = a(q_i)$, where q_i is the parameter q used by VU i. In this paper, we take compression level of the input image (i.e. the encoding bitrates of the jpeg compressed image) as an example of the application adaptation parameter. We measure the accuracy in terms of the intersection over union (IoU). IoU is the intersection over union of the areas of the bounding boxes of the detected objects in the input image with respect to the result of the uncompressed base image.

Assume for image m, the bounding box area of the detected objects in the base image

(i.e., at $q = 1$, the lowest possible compression level) is N_1^m, and the bounding box area of the detected objects in the corresponding input image (with compression level q') frame is $N_{q'}^m$. The accuracy of this input image is defined as: $(N_1^m \cap N_{q'}^m)/(N_1^m \cup N_{q'}^m)$. Without loss of generality, we define the overall accuracy of images at image quality q' by averaging the accuracy over M different frames,

$$a(q') = \frac{1}{M} \sum_{m=1}^{M} \frac{(N_1^m \cap N_{q'}^m)}{(N_1^m \cup N_{q'}^m)} \tag{2.3}$$

where M is a large whole number. Note that $(0 < a(q') < 1)$.

QoS utility The quality of service (QoS) of vehicular applications aims to have lower delay and higher accuracy. However, higher accuracy is usually achieved by larger data size, that potentially impacts the end-to-end delay, which is a function of data transmission time and computation delay at the VLC node and VEC server as expressed in Eq. 2.2. Therefore, the system needs to consider a trade-off between end-to-end delay and accuracy. In this paper, we define a joint performance metric, QoS utility, which we represent as a weighted function of the end-to-end delay and accuracy. For each VU i, the QoS utility is defined as,

$$QoS_i = \alpha \frac{d_n}{d_i} + (1 - \alpha) * a(q_i) \tag{2.4}$$

and the average QoS utility of all the current VUs is,

$$\hat{QoS} = \frac{1}{I} \sum_{i \in \mathcal{I}} QoS_i \tag{2.5}$$

where d_n is the term used to normalize d_i to the same range of $a(q_i)$. For example, if the value of $a(q_i)$ is between 0 and 1, d_n will be determined as the smallest possible value of d_i. α $(0 < \alpha < 1)$ is the trade-off factor between the end-to-end delay and accuracy, and is determined by the service provider. Higher value of α means the QoS utility emphasizes more on the performance

of end-to-end delay and vice versa. For example, for the SRSUs deployed along a highway, where end-to-end delay is critical to driving experience due to high vehicle speed, the service provider can choose higher value of α to focus more on reducing the end-to-end delay than very high accuracy. Meanwhile, because the moving patterns of vehicles on the highway are stable and easy to track across multiple consecutive frames, detection accuracy can be compensated by object tracking techniques so that the impact of the trade-off in accuracy will not affect the driving safety.

2.3.4 Energy Consumption and Harvesting at SRSU

The energy consumption of the SRSU E_R consists of the energy consumed by its VEC server and SBS. We denote E_S and E_B be the energy consumed by the VEC server and SBS, respectively. Therefore, $E_R = E_S + E_B$. Note that E_S depends on the load and CPU-GPU configuration c_e of VEC server. The load of VEC server is a function of the offloaded subtasks, therefore, E_S for the current time slot can be represented as,

$$E_S = E_{S,idle} + E_c \tag{2.6}$$

where $E_{S,idle}$ is the idle energy consumption and E_c is the energy consumed for executing the offloaded subtasks.

On the other hand, the energy consumption of the SBS can be represented as the following,

$$E_B = E_{B,idle} + \sum_i P(r_{b,i}) * T_{tx,i} \tag{2.7}$$

where $P(r_{b,i})$ is the base-band signal processing power consumption at SBS for uplink transmission at datarate $r_{b,i}$. $T_{tx,i}$ is the transmission time (i.e. the time when SBS is actively processing the uplink signal at datarate $r_{b,i}$).

At the beginning of each time slot, we let E_t be the amount of energy harvested from the solar panel of SRSU b and can be immediately used by the SRSU. Therefore, the energy

consumption of VEC server and SBS should satisfy,

$$E_R = E_B + E_S \leq E_t \tag{2.8}$$

In the following section, we build the empirical system models for the vehicular tasks, performance metrics and energy consumption with real-world control parameters, e.g., application adaptation parameter, computing configurations and load, that instills the non-linear behaviors in the aforementioned system models.

2.4 Empirical System Model

We emulate the SRSU-assisted VEC system by using a setup of Nvidia Jetson TX2 boards which are power-efficient embedded AI computing devices [57], and use NI USRP B210 radios for communications. We operate the Nvidia Jetson TX2s at different CPU-GPU configurations to emulate the different computing capacities of the VEC server and VLC nodes. We list the corresponding hardware configurations in Table 2.2, including CPU and GPU frequencies and the number of available CPU and GPU cores. Note that the CPU and GPU frequencies listed in Table 2.2 are chosen from the available frequencies allowed by the Nvidia Jetson TX2 board.

We mimic different computing capacities of the VEC server with two configurations, VEC_config1 and VEC_config2, by choosing two sets of available CPU-GPU configurations of the Nvidia Jetson TX2 board. Each set consists of a collection of CPU and GPU frequencies that the VEC server can tune to operate at depending on current performance and energy requirements.

Similarly, for VLC nodes, we emulate their computing capacity by choosing two different CPU-GPU configurations, VLC_config1 and VLC_config2. For the disparity of computing capacity between the edge and local computing devices, we choose the lowest available GPU frequency for both VLC_config1 and VLC_config2. Additionally, we assume VLC_config1 and VLC_config2 can only access two and one CPU cores, respectively, on the Nvidia Jetson TX2 board, while both edge configurations can access six CPU cores. Finally, for the CPU frequency

Table 2.2. Chosen CPU and GPU configurations to emulate the computing capacity of VLC node and VEC server.

Configuration name	VLC_ config1	VLC_ config2	VEC_ config1	VEC_ config2
CPU cores	2	1	6	6
CPU frequency (MHz)	960	652	{960, 1267, 1574}	{652, 824, 960}
GPU cores	1	1	1	1
GPU frequency (MHz)	114	114	{725, 1300}	{725, 1300}

of VLC_config1, we choose the lowest frequency listed among the available CPU frequencies of VEC_config1, and likewise, we determine the CPU frequency of VLC_config2 as the lowest CPU frequency listed for VEC_config2.

As mentioned earlier, while we use object detection using a vehicle camera as an example of the real-time vehicular application, our work can be easily extended to other types of vehicular applications as well. We use SSD-MobileNetV2 [56] for object detection due to its lightweight computations, favorable for real-time applications. In this section, first we provide the task model for object detection with SSD-MobileNetV2 and show the impact of corresponding application adaptation parameter, i.e. compression level, on the data size in the task pipeline and accuracy. Second, we will present the empirical model for the computing delay at the VLC node and VEC server w.r.t. different system settings. Third, we will demonstrate the empirical model for energy consumption of the SRSU, including energy consumption at the SBS and VEC server, by extending the theoretical models described in the previous section, considering different system configurations and load conditions.

2.4.1 Object Detection Task Model and Impact of Compression

The task graph considered for object detection is shown in Fig. 2.3. After the vehicle camera captures a 1080p image, the image is decoded and resized into a 2-dimensional 300 by 300 matrix input and forwarded to the neural network of SSD-MobileNetV2 for object detection.

For the sake of simplicity, we choose a combination of functional blocks as one subtask, as shown in the dotted boxes A and B (i.e., the following two subtasks, Decoding & Resizing and Neural Network Inference), for the rest of this study. However, note that our approach is not limited to two subtasks and is scalable for more subtask scenarios.

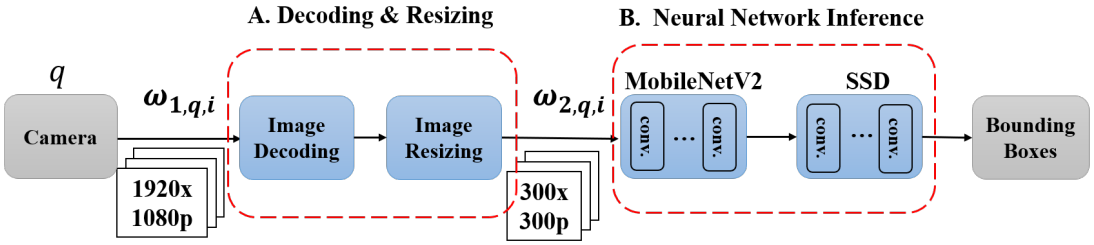

Figure 2.3. Task dependency graph of object detection using SSD-MobileNetV2, showing data size and compression level.

As the application adaptation parameter, we use compression level $q \in \mathcal{Q}$, which is applied to camera, to control the encoding bitrate and hence the data size of images. $\mathcal{Q}$ is the set of available compression levels. The impact of compression level is two-fold. Higher compression level reduces the size of the input image, thus reducing the dataflow transmission time to forward to the next node in the task pipeline, but it also affects the accuracy of the object detection.

Table 2.3 shows the data size along the processing flow of the object detection using a compressed 1080p jpeg image, under different values of q. Note that $\omega_{2,q,i}$ is the decoded 300x300 pixels image, therefore, its size is not impacted by the encoding bitrate compression level.

Table 2.4 shows the corresponding impact on accuracy for different values of q for a set of 70 image frames of 1080p resolution. We choose the base image when the compression level is 1 with 100% encoding bitrate. Thus the accuracy for $q = 1$ is 1.0 based on Eq. 2.3 and it decreases with the increasing values of q. Note that the lowest accuracy in Table 2.4 (i.e. q_i=4), is just 10% less than the accuracy of the base image. However, even with a 10% decrease, the accuracy of SSD-MobileNetV2 is still higher than some of the other object detectors,

Table 2.3. Input data size of each subtasks at different compression level

Compression level q	1	2	3	4
Encoding bitrate level	100%	75%	50%	25%
$\omega_{1,q,i}$ (KByte)	92	79	55	27
$\omega_{2,q,i}$ (KByte)	270	270	270	270

Table 2.4. Accuracy at different compression levels

q	1 (Base image)	2	3	4
Accuracy	1.0	0.97	0.93	0.9

e.g., YOLOV2 [77], which has been largely used for autonomous vehicles as mentioned in literature [78, 79]. Therefore, our study can still meet the same driving safety performance as other vehicular object detection studies even with the trade-off in accuracy.

2.4.2 Computing Delay and Impact of Computing Capacity

Here, we empirically model the computing delay for the object detection using the different computing capacities of VLC node and VEC server. Note that VLC node will run the application for a single VU and thus its computing capacity is impacted by CPU-GPU configurations only as mentioned in Table 2.2. However, the VEC server runs applications for multiple VUs and thus is impacted by both its CPU-GPU configurations and the load in terms of the number of application instances. Now, based on Fig. 2.3, there are two subtasks in the task graph. Therefore, $K = 2$, and T_1 is the DR delay (execution delay of Decoding and Resizing subtask) and T_2 is the inference delay (execution delay of neural network inference), which together constitutes the computing delay.

Computing Delay at the VLC Node

We implement the object detection subtasks in Fig. 2.3 on the Nvidia Jetson TX2 board and measure T_1 and T_2. Table 2.5 shows the observed computing delay of each subtask for processing an image frame under the VLC configurations listed in Table 2.2. Note that the

Table 2.5. Breakdown of the end-to-end delay of object detection using different VLC computing capacities

Computing delay (s)	VLC_config1	VLC_config2
T_1	0.036	0.085
T_2	0.095	0.103
Overall	0.131	0.188

minimum computing delay of the object detection at the VLC node is higher than 0.130 s, which is not fast enough for the 0.1 s requirements for vehicular applications to react to the fast changing traffic condition [80]. Therefore, offloading some of the subtasks to the VEC server, which has higher computing capacity than VLC node, can reduce the computing delay, and thus, potentially reduce the end-to-end delay. In the next subsection, we demonstrate the empirical model of the computing delay at the VEC server.

Computing Delay at the VEC Server

We model the computing delay of subtasks on the VEC server empirically by observing T_1 and T_2 under different VEC server load conditions. Fig. 2.4 shows the DR delay T_1 and the inference delay of neural network inference T_2, under the conditions of different VEC server capacities and different number of instances of DR and the neural network subtasks. Note that

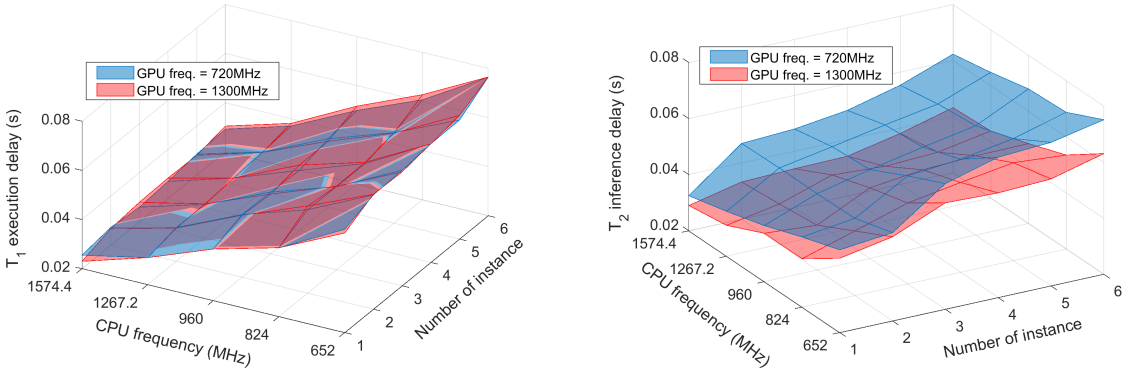

Figure 2.4. (a) DR delay T_1 and (b) Inference delay T_2, under different number of running instances and VEC server's computing capacities

the CPU and GPU frequencies used in Fig. 2.4 are listed in Table 2.2. In Fig. 2.4(a), the DR delay does not change between different GPU frequencies because the execution of DR does not use any GPU resource. On the other hand, Fig. 2.4(b) shows that the relationship between the inference delay and the increasing CPU-GPU frequencies as well as the number of application instances is not easy to represent by simple linear and quadratic models. Therefore, such knowledge of nonlinear correlation between delay and computing capacity shown in Fig. 2.4 is necessary for accurate delay performance optimization.

2.4.3 Energy Consumption Model

Energy consumption at the VEC server

In our empirical study, we observe the major factors that impact the energy consumption at the VEC server are the server's CPU-GPU configuration, the number of offloading VUs (i.e. the running application instances), and the offloaded computation loads. Fig. 2.5(a) shows the energy consumed per second of a Jetson TX2 board as VEC server while a number of VUs offload both DR and the neural network inference simultaneously. The measurement is taken under different CPU-GPU configurations. The energy consumption is linearly increasing with the CPU frequency and number of offloading VUs. However, the increasing rate varies when the Jetson board is operated with different GPU frequencies. In reference to Fig. 2.4a, we can see that although higher CPU and GPU frequencies lead to less computing delay, the corresponding energy consumption will be higher. Under the condition when the SRSU lacks of available energy, VEC server needs to reduce its operating CPU-GPU frequency while sacrificing the computing delay of the offloaded subtasks. Fig. 2.5(b) shows the energy consumed per second while multiple VUs offload only the neural network subtask. While it shows similar trend of increasing energy consumption as Fig. 2.5(a), its absolute value is less than Fig. 2.5(a) under fixed CPU-GPU frequency settings and number of instances, because only one of the subtasks (i.e. neural network inference) is executed.

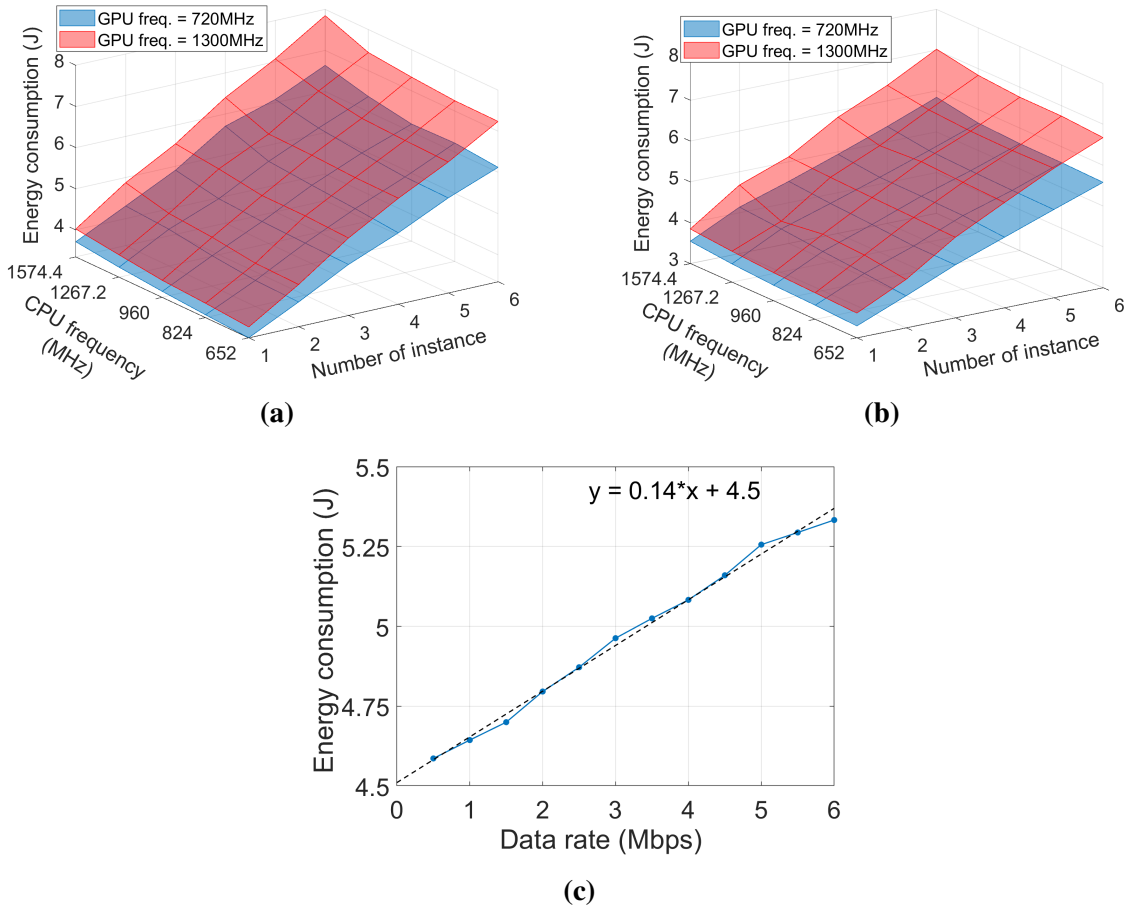

Figure 2.5. Energy consumption per second of (a) VEC server executes instances of both DR and the neural network inference and (b) VEC server executes various instances of the neural network inference under different VEC server's computing capacities, and (c) SBS under different uplink data rates

Energy consumption at the SBS

To measure the energy consumed by the wireless communication at the SRSU, we use the same experimental settings as in [81], with one Jetson board and one NI USRP B210 radio to emulate the SBS. The wireless channel is established by srsLTE tool [82], which is used to create an LTE link between SRSU and VU. We create different values of uplink data-rate using *iperf* and measure the corresponding energy consumption on the Jetson TX2 board and the NI USRP B210 radio. The result of the consumed energy per second is reported in Fig. 2.5c. Note that due to hardware limitations, the maximum uplink datarate achievable over LTE by our experimental

setting is 6 Mbps. Therefore, we use curve fitting approach for the energy consumption model of the SBS at high data rate conditions. It can be observed that the energy consumption $P(r_{b,i})$ is linear to the uplink data rate $r_{b,i}$, that is, $P(r_{b,i}) = 0.14 r_{b,i}$, with the idle power = 4.5 W. Based on Eq. 2.7, we consider the following energy consumption model for E_B.

$$\begin{aligned} E_B &= 4.5\tau + 0.14 \sum_i r_{b,i} * \frac{\omega_{k',q,i}}{r_{b,i}} \\ &= 4.5\tau + 0.14 \sum_i \omega_{k',q,i} \end{aligned} \tag{2.9}$$

where $\omega_{k',q,i}$ is the data size per frame required to be transmitted corresponding to splitting point of subtasks for offloading.

2.5 Overall Approach and Problem Formulation

2.5.1 Task partitioning and offloading

From the above real-world system models, we can observe that to optimize a single VU's end-to-end delay, we can offload all of its subtasks to the more powerful VEC server. However, an inferior wireless channel quality can potentially increase the communication delay and thus increase the end-to-end-delay resulting in low QoS utility. Hence, partitioning a task at a point that reduces the data size is desirable, in order to reduce the communication delay. Moreover, when multiple VUs try to offload all their subtasks to one VEC server simultaneously, the resource constraint at the VEC server may increase the average computing delay and can potentially violate the energy constraint in Eq. 2.8. Therefore, we need an optimal offloading strategy that allows VUs to selectively offload part of their subtasks based on their transmission rate, local computing capacity, and current VEC server load, as well as energy constraint.

In this paper, we consider the following three partitioning and offloading strategies denoted by y_i, for a VU with the considered object detection application (1) y_i = 1: *Full Offloading*, (2) y_i = 2: *Partial Offloading*, and (3) y_i = 3: *Encoded Partial Offloading*. We also denote y_i = 0 as the *Local Only* strategy, where all the subtasks are executed at the VLC

node. The high level block diagrams of these strategies are shown in Fig. 2.6, where the blocks represent each subtask. Blue blocks indicate the subtask is executed locally and green blocks indicate the subtask is executed at the VEC server. Red dash arrow indicates where the wireless uplink data transmission between the VU and the VEC server happens.

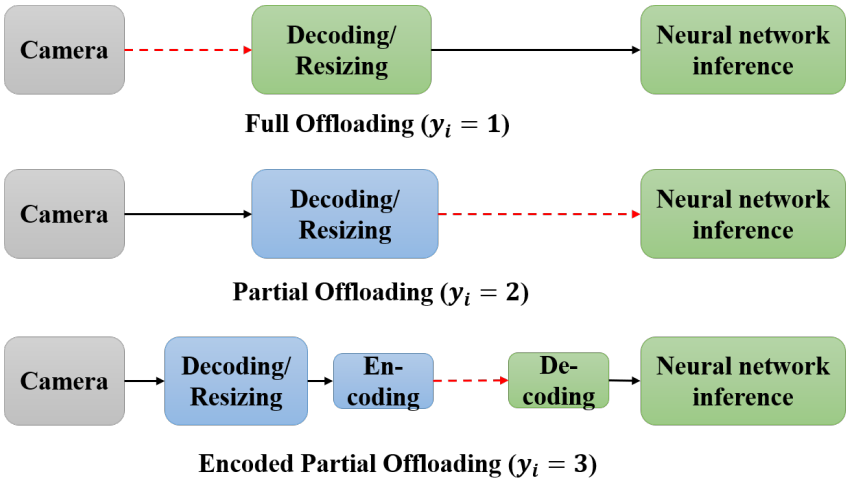

Figure 2.6. Possible task partitioning and offloading strategies in object detection application using SSD-MobileNetV2

For *Full Offloading*, strategy, VU will transmit the captured image (i.e. with size $\omega_{1,q,i}$) to the SRSU, and hence, offload both of the DR and neural network inference to the VEC server. On the other hand, for *Partial Offloading* strategy, VU will first execute DR subtask at VLC node, then offload the decoded as well as resized 2-dimensional input image features (i.e. with size $\omega_{2,q,i}$) to SRSU, and let the VEC server execute the neural network inference. However, note that the data size after decoding is several times larger than the encoded image, which is not feasible for transmission in real-time unless the transmission rate is very high. Therefore, in this paper, we propose another partial offloading strategy: *Encoded Partial Offloading*.

In *Encoded Partial Offloading* strategy, at VLC node, VU will encode the resized image feature again to a jpeg image with the same resolution as the smaller resized image (i.e. 300x300 pixels in the studied example) before transmission. Subsequently, the VEC server will decode the received image to the 2-dimensional image feature and then send to the neural network inference

Table 2.6. Encoded data size for transmission of *Encoded Partial Offloading* strategy with different compression levels

Compression level q	1	2	3	4
Encoding bitrate level	100%	75%	50%	25%
$\omega^c_{2,q,i}$ (KByte)	21	18	12.5	6.2

for object detection. Compared to *Partial Offloading* strategy, *Encoded Partial Offloading* strategy incurs overhead of execution of extra encoding at the VLC node and extra decoding at the VEC server, with the trade-off for a high gain in reduction of communication delay due to highly reduced data size. The data size of the 300x300 resized image feature after encoding is shown in Table 2.6, where $\omega^c_{2,q,i}$ is the encoded data size of $\omega_{2,q,i}$.

Impacts of offloading strategies to end-to-end delay

Previously we have separately modeled T_1, T_2 at the VLC node and the VEC server, and T_{tx} under different data rates and transmitted data sizes. In this section we model the end-to-end delay combining the offloading strategies and the above empirical models. While all of the offloading strategies (i.e. $y_i > 0$) offload the neural network inference to the VEC server, only *Full Offloading* strategy offloads the DR subtask. Therefore, we model T_1 for DR as a function of the number of *Full Offloading* users h and T_2 for the neural network inference as a function of the number of total offloading users n, where $h = \sum_{i:y_i=1} 1$ and $n = \sum_{i:y_i>0} 1$.

We use $T^l_1(c_{l,i})$ and $T^e_1(c_e, h)$ to denote DR delay on VLC node and VEC server, respectively, given h, VLC node configuration $c_{l,i}$ and VEC server configuration $c_e \in \mathcal{C}$. $\mathcal{C}$ is the set of available CPU-GPU configurations for the VEC server. Similarly, $T^l_2(c_{l,i})$ and $T^e_2(c_e, n)$ denote the inference delay, respectively, given n, VLC node configuration $c_{l,i}$ and VEC server configuration c_e. Therefore, the end-to-end delay of VU i using y_i offloading strategy can be

modeled as,

$$d_i(y_i,q_i,n,h,c_e,c_{l,i}) = \begin{cases} T_1^l(c_{l,i}) + T_2^l(c_{l,i}); & \text{if } y_i = 0 \\ T_1^e(c_e,h) + T_2^e(c_e,n) + \frac{\omega_{1,q_i,i}}{r_{i,b}}; & \text{if } y_i = 1 \\ T_1^l(c_{l,i}) + T_2^e(c_e,n) + \frac{\omega_{2,q_i,i}}{r_{i,b}}; & \text{if } y_i = 2 \\ T_1^l(c_{l,i}) + T_2^e(c_e,n) + T_3^l(c_{l,i}) + T_4^e + \frac{\omega^c_{2,q_i,i}}{r_{i,b}}; & \text{if } y_i = 3 \end{cases} \tag{2.10}$$

where $T_3^l(c_{l,i})$ is the encoding delay for a smaller resized 300x300 pixels jpeg image given the VLC node configuration $c_{l,i}$, and T_4^e is the decoding delay for the same resized image at the VEC server. Based on our observation, T_3 is 0.003s and 0.007s, respectively, for VLC configuration VLC_config1 and VLC_config2. We have measured that given the worst VEC server configuration, T_4^e is 0.003 s, which is very small compared to T_1 (i.e. decoding and resizing for 1080p jpeg image). Therefore, we set T_4^e to 0.003 s and ignore the impact of c_e to the value of T_4^e.

Similarily, with the above offloading strategies, based on Eq. 2.8 and 2.9, we model the empirical total energy consumption of SRSU E_R as,

$$\begin{aligned} &E_R(\omega',n,h,c_e) = E_S(n,h,c_e) + E_B(\omega') = \\ &\frac{\tau}{n}(h\mathcal{E}'_1(n,c_e) + (n-h)\mathcal{E}'_2(n,c_e)) + 4.5\tau + 0.14\omega' \end{aligned} \tag{2.11}$$

where ω' is the summation of the data size that needs to be transmitted depending on the decisions of y_i and q_i, $\forall i \in \mathcal{I}$. $\mathcal{E}'_1$ is the energy consumption of the VEC server shown in Fig. 2.5(a), and $\mathcal{E}'_2$ is the energy consumption shown in Fig. 2.5(b).

Since not all of the n VUs will offload both of the subtasks simultaneously, for the sake of simplicity, we assume the overall energy consumption is the interpolation of the corresponding energy consumption values when all of the n VUs offload both subtasks (i.e. $\mathcal{E}'_1(n,c_e)$) and when all of the n VUs offload just the neural network (i.e. $\mathcal{E}'_2(n,c_e)$). The above empirical

models are specifically for SSD-MobileNetV2-based object detection applications. For other types of applications, once the action space of y_i is defined and the delay, accuracy, and energy consumption models are established correspondingly, our work can be applied to those vehicular applications.

2.5.2 Overall Approach and Problem Formulation

We assume that at each time slot, each VU will send an offloading request for this application. The request will include information of the local computing capacity $c_{l,i}$, available compression levels $\mathcal{Q}$, and the subtask composition of $\mathcal{K}$. The SRSU will take the above information, along with the available solar energy E_t, bandwidth W, and VEC server configurations $\mathcal{C}$, and make optimal offloading decision y_i as well as compression level q_i for each VU i. We assume the SRSU already knows the delay and accuracy models like Table 2.4 and Fig. 2.4. The decisions will be sent to each VU by the SBS as the offloading instruction. In the meantime, SRSU will need to decide the VEC server's CPU-GPU configuration c_e for operation.

Fig. 2.7 depicts the whole process. In Fig. 2.7, blue arrow shows the flow of offloading requests from VUs to the SRSU, with the included information listed in blue boxes; green arrow shows the flow of SRSU's information, which is listed in green boxes including channel conditions, bandwidth, solar energy, and VEC server's computing availabilities; red arrows indicate the flow of decisions for the SRSU and VUs. The objective of this paper is to determine in real-time the VEC server's operating configuration c_e and the optimal offloading strategy y_i as well as the compression level q_i for each VU i to maximize the average QoS utility of all the VUs in $\mathcal{I}$ at any given time slot. The decision is made at the beginning of the time slot. The optimization problem for each time slot can, therefore, be formulated as,

$$\underset{y_i, q_i : i \in \mathcal{I}, c_e}{\text{maximize}} \quad \hat{QoS} \tag{2.12a}$$

$$\text{subject to} \quad E_R \leq E_t \tag{2.12b}$$

$$W_i = W_j, \forall i, j : y_i > 0, y_j > 0 \tag{2.12c}$$

$$\sum_{i : y_i > 0} W_i \leq W \tag{2.12d}$$

$$\sum_{i : y_i > 0} 1 \leq N_{VEC} \tag{2.12e}$$

where Constraint 2.12b is the energy consumption constraint of SRSU. Constraint 2.12c states that the utilized bandwidth is evenly distributed among the VUs who are offloading. Constraint 2.12d ensures that the overall utilized bandwidth does not exceed the available bandwidth of the system W. Constraint 2.12e shows that the total number of offloading VUs can not exceed the maximum capacity of the VEC server, which is also equivalent to the maximum number of application instance N_{VEC} that the VEC server can run simultaneously due to the computing capacity and computer memory limitations. For example, $N_{VEC} = 6$ for the Jetson TX2 board running SSD-MobileNetV2-based object detection applications.

Fig. 2.8 shows the time domain flow of the SRSU-assisted VEC system. For each VU, F_m represents the execution of the m^{th} frame of the application to process the m^{th} input camera image in the current time slot t, and the width of the F_m box shows its end-to-end delay. The small colored blocks in each F_m box are the subtasks for the application, where blue and green blocks indicate that the corresponding subtask is executed locally at the VLC or offloaded to the VEC, respectively. The orange boxes represent the execution of the offloading decision making. The offloading decision making for the next time slot takes the current states of VUs' request information and SRSU's resource capacities, then returns the optimal offloading decisions as well as compression levels at the beginning of the next time slot.

Since the partitioning and offloading decisions are made at the beginning of a time slot,

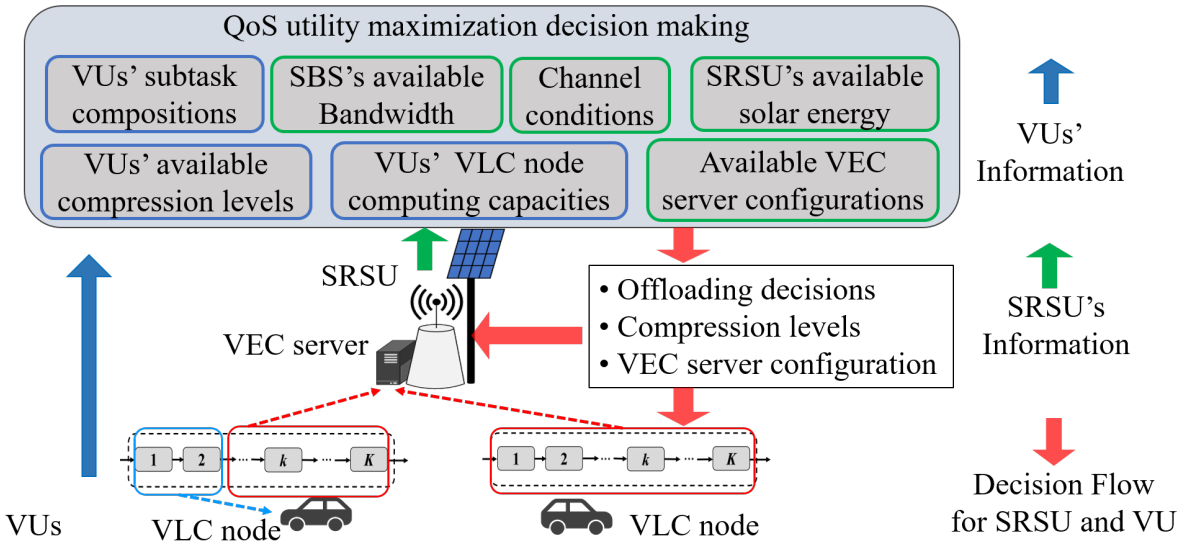

Figure 2.7. Overview of the SRSU-assisted VEC system, including offloading request and decision flows

the duration of the current time slot, τ, should be determined by the following equation,

$$\tau \geq \max\{\max_{i\in\mathcal{I}} d_i, T_{decision}\}, \tag{2.13}$$

so that no computation task will occupy any computing and communication resources when the next time slot begins. Note that for VU i, d_i is upper bounded by $d_i(0, q_{max}, 0, 0, 0, c_{l,i}) = \sum_{k=1}^{K} T_k^l(c_{l,i})$, which is the local execution delay ($y_i = 0$) in Eq. 2.10, where $c_{l,i}$ is the VLC configuration and k is the index of each subtask. $T_{decision}$ is the delay of making optimal partitioning and offloading decisions, which depends on the VEC computing capacity and complexity of the decision making. We will show with experimental result in latter section that τ is mostly bounded by the $T_{decision}$ of our proposed decision algorithm with reasonable size of VLC and VEC computing capacities, and can be small enough to ensure an unchanged data rate, $r_{b,i}$.

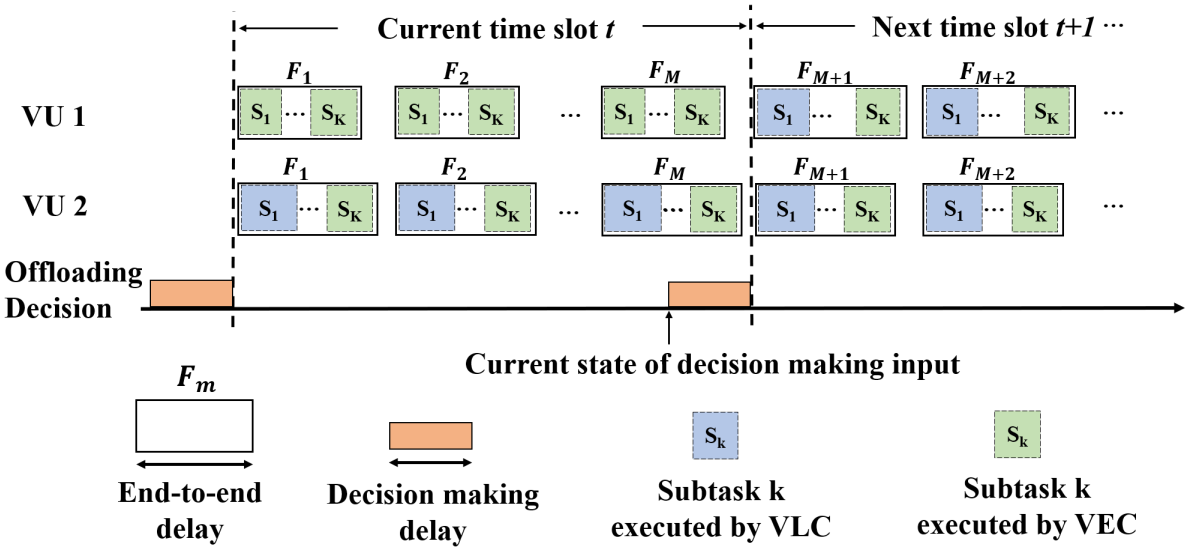

Figure 2.8. Overview of the time domain flow of both application execution frames and task partitioning as well as offloading decisions in the SRSU-assisted VEC system.

2.6 Solution Methodology

Note that in our problem formulation 2.12, the decision variables $y_i, q_i, \forall i \in \mathcal{I}$ and c_e are integers while $\hat{QoS}$ is a non-linear function of the above variables. Therefore, problem 2.12 is an NP-hard nonlinear integer programming problem [83]. The complexity of exhaustively listing all the possible values of y_i, q_i and c_e, and tracking for the solution which gives the maximum objective value is $O(Y^I Q^I)$. Where Y is the number of possible choices for partitioning and offloading (including *Local Only* strategy), Q is the number of compression levels for application level adaption of each VU, and I is the total number of VUs. If there are only a few VUs in the area, exhaustively search can provide optimal solution with a low time-complexity. However, the complexity of this problem grows exponentially with the number of VUs. Moreover, since we are considering vehicular users, the number of VUs changes over time, and it is very likely that there are tens of vehicles in the coverage area of an SRSU (e.g. during the peak hour of a highway). This leads to prohibitively expensive time-complexity for the exhaustive search approach. Therefore, in this work, we propose a dynamic programming-based heuristic algorithm to solve problem 2.12.

For a given instance of problem 2.12, and assuming $\sum_{i:y_i>0} 1 = N'$, $c_e = c'$, where both N' and c' are fixed integers, we consider a matrix f with dimension $I*N'*N'*N'*N'$. $f(i,n,h,u,v)$ represents the maximum average QoS utility achievable considering VU set $\{1,2,...,i\}$ and allowing n VUs offloading. On the other hand, h, u, and v are the numbers of the offloading VUs using *Full Offloading*, *Partial Offloading*, and *Encoded Partial Offloading* strategies, respectively. The core formula of this dynamic programming strategy is in Eq. 2.14,

$$f(i,n,h,u,v) = \begin{cases} 0; & \text{if } n \neq h+u+v \\ 0; & \text{if } i = 0 \\ 0; & \text{if } i,h,u, \text{ or } v < 0 \\ max(A_y, \quad \forall y \in \mathcal{Y}); & \text{otherwise} \end{cases} \tag{2.14}$$

where $\mathcal{Y}$ is the set of all the possible values of y_i, and A_y is defined in the following. For y = 0,

$$A_0 = P_i^l + f(i-1,n,h,u,v) \tag{2.15}$$

is the QoS utility when including VU i in the considered VU set while VU i is not offloading any subtask, where P_i^l is the QoS utility using VLC node of VU i. For $y > 0$,

$$A_y = \begin{cases} max_q \; P_{i,y,q} + f(i-1,n-1,h-\mathbb{1}_{y=1},u-\mathbb{1}_{y=2},v-\mathbb{1}_{y=3}); \\ \quad \text{if} \quad E_R(w'(i-1,n-1,h-\mathbb{1}_{y=1},u-\mathbb{1}_{y=2},v-\mathbb{1}_{y=3}) + \omega_{y,q,i},n,h,c') < E_t \\ 0; \text{otherwise} \end{cases} \tag{2.16}$$

is the QoS utility while including VU i using $y_i = y$ ($y = 1,2$, or 3 in the considered use case) with compression level q_i^*, which gives the maximum QoS utility $P_{i,y,q}$ among all the possible q. Note that $P_{i,y,q} = \alpha * d_n/d_i(y,q,n,h,c_e,c_{l,i}) + (1-\alpha)*a(q)$. $\mathbb{1}_{y=j}$ is an indicator function whose value is 1 if y = j, otherwise, its value is 0. The $E_R < E_t$ inequality is used to ensure the energy

Algorithm 5: Dynamic programming Algorithm for Fixed Offloading VU and System configuration (DAFOS)

Input: $N', c', \mathcal{I}, p_i^l, r_{b,i}, c_{l,i}, \forall i \in \mathcal{I}$
Output: $f', \hat{\mathbf{y}}, \hat{\mathbf{q}}$

$f, \psi, \pi \leftarrow zeros(I, N', N', N', N')$;
for $i \leftarrow 1$ **to** I **do**
  for $n \leftarrow 0$ **to** N' **do**
    for $h \leftarrow 0$ **to** N' **do**
      calculate
      $P_{i,y_i,q_i} = \alpha * d_n / d_i(y_i, q_i, n, h, c_e, c_{l,i}) + (1-\alpha) * a(q_i) \quad \forall y_i \in \mathcal{Y}, q_i \in \mathcal{Q}$;
      for $u \leftarrow 0$ **to** N' **do**
        for $v \leftarrow 0$ **to** N' **do**
          update $f(i,n,h,u,v)$ by Eq. 2.14 ;
          update $\psi(i,n,h,u,v)$, $\pi(i,n,h,u,v)$, and $w'(i,n,h,u,v)$;
$f' \leftarrow \max_{h,u,v} f(I, N', :, :, :)$;
$h^*, u^*, v^* \leftarrow \text{argmax}_{h,u,v} \quad f(I, N', :, :, :)$;
$\hat{N} \leftarrow N'$;
$\hat{\mathbf{y}}, \hat{\mathbf{q}} \leftarrow zeros(I)$;
for $i \leftarrow I$ **to** 1 **do**
  $\hat{\mathbf{y}}[i] \leftarrow \psi(i, N', h^*, u^*, v^*)$;
  $\hat{\mathbf{q}}[i] \leftarrow \pi(i, N', h^*, u^*, v^*)$;
  if $\hat{\mathbf{y}}[i] > 0$ **then**
    $\hat{N} \leftarrow \hat{N} - 1$;
    if $\hat{\mathbf{y}}[i] == 1$ **then**
      $u^* \leftarrow h^* - 1$;
    else if $\hat{\mathbf{y}}[i] == 2$ **then**
      $v^* \leftarrow u^* - 1$;
    else
      $z^* \leftarrow v^* - 1$;
return $f', \hat{\mathbf{y}}, \hat{\mathbf{q}}$;

constraint, where $w'(i-1, n-1, h-\mathbb{1}_{y=1}, u-\mathbb{1}_{y=2}, v-\mathbb{1}_{y=3})$ is the total data size required to be transmitted for VU 1 to $i-1$. $\omega_{y,q,i}$ is the data size required to be transmitted by VU i when using $y_i = y$ and $q_i = q$.

On the other hand, we use $\psi(i,n,h,u,v)$ and $\pi(i,n,h,u,v)$ to record the optimal offloading decision y_i^* and compression level q_i^* that correspond to the value of $f(i,n,h,u,v)$ for VU i. Matrices f, ψ, π are initialized as zero matrices. We then recursively calculate the elements in f for v from 0 to N', u from 0 to N', h from 0 to N', n from 0 to N', i from 0 to I, until all the elements in f are updated. The optimal cumulative QoS utility for VU set $\mathcal{I}$ considering $\sum_{i:y_i>0} 1 = N'$ and $c_e = c'$ is then the maximum elements among $f(I, N', :, :, :)$. We then calculate

Algorithm 6: System and Application aware Multiple User Offloading Algorithm (SAMOA)

Input: $\mathcal{I}, \mathcal{C}, N_{VEC}, \mathcal{Y}, \mathcal{Q}, r_{ib}, c_{l,i}, \forall i \in \mathcal{I}$
Output: L^*, p^*
1 $p^l \leftarrow zeros(N)$;
2 **for** *i in I* **do**
3 $\quad p_i^l \leftarrow \frac{\alpha * d_n}{d_i(i,0,q_{max},0,0,c_{l,i})} + (1-\alpha) * a(q_{max})$;
4 $\hat{\mathbf{P}} \leftarrow zeros(N_{VEC}, \mathcal{C})$;
5 $\hat{\mathbf{L}} \leftarrow list()$;
6 **for** $N' \leftarrow 1$ **to** N_{VEC} **do**
7 $\quad$ **for** c' *in* $\mathcal{C}$ **do**
8 $\quad\quad \hat{\mathbf{P}}[N', c'], \hat{\mathbf{y}}, \hat{\mathbf{q}} \leftarrow DAFOS(N', c', \mathcal{I}, p_i^l, r_{ib}, c_{l,i}, \forall i \in \mathcal{I})$;
9 $\quad\quad$ append $\{\hat{\mathbf{y}}, \hat{\mathbf{q}}\}$ in $\hat{\mathbf{L}}$
10 $N^*, c_e^* \leftarrow \underset{N', c'}{\text{argmax}}\ \hat{\mathbf{P}}[N', c']$;
11 $p^* \leftarrow \hat{\mathbf{P}}[N^*, c_e^*]$;
12 $L^* \leftarrow \hat{\mathbf{L}}[N^*, c_e^*]$;

the optimal offloading and compression level decisions for each VU iteratively from $i = I$ to $i = 1$ by using ψ and π. We list the steps for updating elements in f in Algorithm 5, which we name as Dynamic programming Algorithm for Fixed Offloading VU and System configuration (DAFOS). Steps 2 to 9 execute the core function of dynamic programming and steps 14 to 24 retrieve the recorded optimal offloading and compression level decisions in ψ and π.

Note that DAFOS returns the heuristic solution of problem 3.14 under the condition that $\sum_{i:y_i>0} 1 = N'$ and $c_e = c'$. To obtain the solution of problem 3.14, all the possible values of N' and c' need to be considered. Therefore, we propose the following System and Application aware Multiple User Offloading Algorithm (SAMOA), which executes DAFOS on different N' and c' and returns the maximum QoS utility, the corresponding offloading strategies as well as compression levels for each VU. The steps of SAMOA are listed in Algorithm 6. In SAMOA, steps 1 to 3 calculate the QoS utility of *Local Only* for each VU. We start to execute DAFOS on different N' and c' and pick the maximum possible optimal solution between steps 6 to 9. We use $\hat{\mathbf{P}}[N', c']$ to record the returned optimal QoS utility. We then append the corresponding offloading strategies and compression levels $\{\hat{\mathbf{y}}, \hat{\mathbf{q}}\}$ to $\hat{\mathbf{L}}[N', c']$. After all the possible sets of N' and c' are iterated, in steps 10 to 12, SAMOA will return the maximum elements in $\hat{\mathbf{P}}$ as the optimal QoS

utility and its corresponding offloading strategies and compression levels of each VU.

Note that for DAFOS, the variables n, h, u, v need to iterate N_{VEC} times and Eq. 3.16 requires iteration of all the Q compression levels and Y offloading decisions in the worst case. Therefore, the complexity of DAFOS is $O(I * N_{VEC}^4 * Q * Y)$. On the other hand, in SAMOA, DAFOS is the component that has the largest complexity and DAFOS is executed $N_{VEC} * C$ times, where C is the number of possible VEC server configurations. Therefore, the complexity of SAMOA is $O(I * N_{VEC}^5 * Q * C * Y)$. Since N_{VEC}, Q, and C are constant, the time complexity of SAMOA is $O(I)$, where I is total number of VUs. Hence, as validated with our experimental results reported in the next section, SAMOA can be executed in real-time for reasonable size of VU set $\mathcal{I}$.

2.7 Performance Evaluation

We first show how SAMOA performs under different resource conditions. Then, we present the online trace-driven simulation framework and demonstrate the performance comparison of SAMOA with existing approaches. Finally, we show how SAMOA can be applied to more dense VU scenarios.

2.7.1 SAMOA Performance Evaluation

Herein, we present and analyze how our algorithm decides the optimal decisions in a single time slot. In Fig. 2.9 (a), (b), (c), and (d), we show the evolution of offloading strategies y_i as well as compression level q_i of each VU determined by SAMOA in different resource availability and system parameter regions. The time slot duration τ is set to 1 second. For simplicity, we assume there are 5 identical VUs in $\mathcal{I}$, all of them have an VLC node with VLC_config2 configuration (i.e. listed in Table 2.5) and SNR value 50 dB. We consider scenarios with different bandwidth, energy, and VEC computation capacities as well as different trade-off factors α. We mimic the impact of computing load caused by other applications sharing the VEC server by reducing the CPU-GPU resources of the Jetson TX2 board. *VEC server*

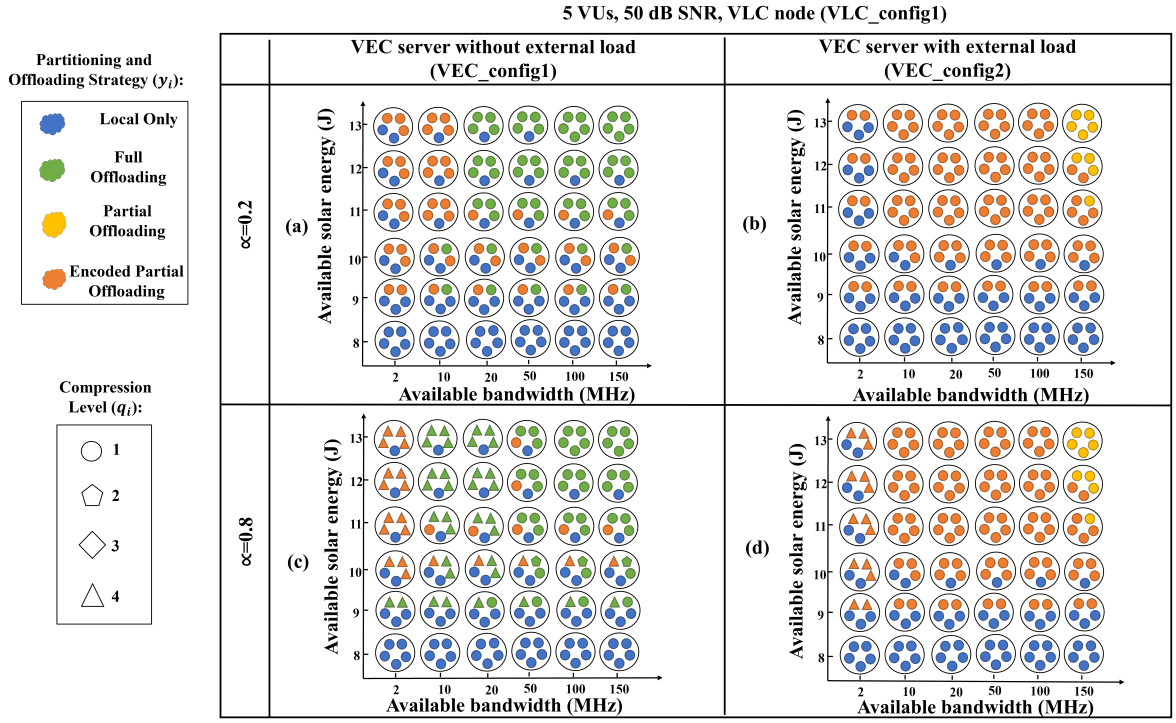

Figure 2.9. Partitioning and Offloading strategy with compression levels for individual VU under different resource availability

without external load (i.e. using VEC_config1 configuration) represents the condition when the VEC server is not busy computing other applications and *VEC server with external load* (i.e. using VEC_config2 configuration) means the VEC server is simultaneously executing other applications. On the other hand, vertically we vary the trade-off factor value α from 0.2 to 0.8. The color and shape of each circle show the y_i and q_i decisions, respectively, to each VU. Blue, yellow, red, and green colors represent *Local Only*, *Partial Offloading*, *Encoded Partial Offloading*, and *Full Offloading* strategies, respectively. Circle, pentagon, diamond, and triangle shapes, respectively, show the compression levels 1 to 4 (i.e. listed in Table 2.6).

Fig 2.9(a) considers the scenario where $\alpha = 0.2$ and *VEC without external load*, where y_i changes from *Local Only* to *Encoded Partial Offloading*, then to *Full Offloading* strategy when the available solar energy and bandwidth increases. It is because compared to *Full Offloading* strategy, (1) *Encoded Partial Offloading* strategy needs less bandwidth as it transmits the encoded image after resizing, (2) *Encoded Partial Offloading* strategy executes the DR subtask in VLC

node and transmits lesser number of bits, hence requires less energy consumption in VEC server. Therefore, in the regions where SRSU lacks of either bandwidth or solar energy, *Encoded Partial Offloading* outperforms *Full Offloading* strategy in terms of QoS utility.

On the other hand, with $\alpha = 0.2$ and when VEC server *has external load* (i.e. the Fig. 2.9(b)), the *Encoded Partial Offloading* strategy dominates when the available bandwidth is below 100 MHz and solar energy is below 13 J. After the bandwidth reaches 150 MHz, we can observe some VUs use *Partial Offloading* strategy. This is because *Partial Offloading* strategy transmits the resized image without encoding, while the reduction in computing delay dominates the growth of transmission delay only when the transmission rate is very high. Also, there is no *Full Offloading* strategy observed because the computing capacity at VEC server is low because of load. Thus offloading with VLC node executing DR subtask first can achieve higher average QoS utility.

In Fig. 2.9(c), we can observe the optimal decision involves different compression levels because of higher α which indicates more importance of delay sacrificing some accuracy. We can observe that VUs offload at the highest compression level (i.e. lowest image quality, the triangle shape) when both the bandwidth and solar energy are in lower availability. We also observe some VUs transmit at the compression level 2 (e.g. the pentagon shape at 10 J, 50 MHz) when the bandwidth is higher than 50 MHz and available solar energy is in medium region (i.e. 10 J). In Fig. 2.9(d), we can also observe that VUs offload at the highest compression level when the bandwidth availability is low. After the available bandwidth exceeds 10 MHz, VUs offload at the lowest compression level because the resulting transmission delay will be small enough such that high accuracy can be achieved.

For the sake of consistency in the granularity of dimensions of the labels in Fig. 2.9, we did not show the condition when our algorithm chooses compression level 3. Actually, in Fig 2.9(c), compression level 3 will be chosen at 10 J, 25 MHz with *Full Offloading* strategy for 3 VUs while the rest 2 VUs use *Local Only* strategy with compression level 1. Overall, Fig. 2.9 demonstrates how VEC server's computing capacity and different choices of α impact the

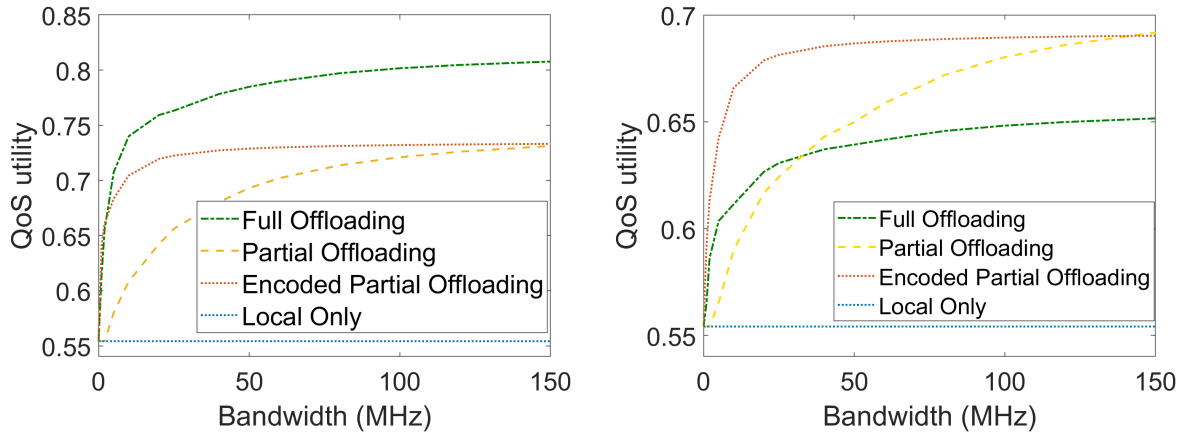

Figure 2.10. Impact of different offloading decisions on QoS utility under different bandwidth resource availability when available solar energy is 13 J for 1 second time slot and VEC server is (a) without external load and (b) with external load

optimal partitioning and offloading strategy for each VU under different resource availabilities. When the algorithm emphasizes more on optimizing the end-to-end delay, we observe some higher compression levels used by VUs for the trade-off between accuracy and end-to-end delay.

While Fig. 2.9 shows the optimal offloading decisions by SAMOA, Fig. 2.10 shows the resulting average QoS utility that drives the decision for the above 5 VUs. We show two scenarios (VEC with and without load) under various bandwidth conditions with 13 J of available solar energy and α=0.8, and show the QoS utilities for various offloading decisions. Note that, a VU can always use *Local Only* strategy, if other available strategies are not feasible in a parameter region. Fig. 2.10(a) shows the results for VEC server *without external load*. The red curve (i.e. *Encoded Partial Offloading*) tops the blue curve (*Full Offloading*) when bandwidth availability is low (<5MHz), while the green curve (i.e. *Full Offloading* strategy) dominates the others afterward. The observation matches the results in Fig. 2.9(c), where the *Full Offloading* strategy dominates at high solar energy and bandwidth regions. On the other hand, the result for VEC server *with external load* is shown in Fig. 2.10(b). We can observe the red curve dominates other curves until bandwidth reaches 150 MHz, where the yellow curve (i.e. *Partial Offloading* strategy) tops the red one. Also, the green curve is always lower than either red or yellow curves.

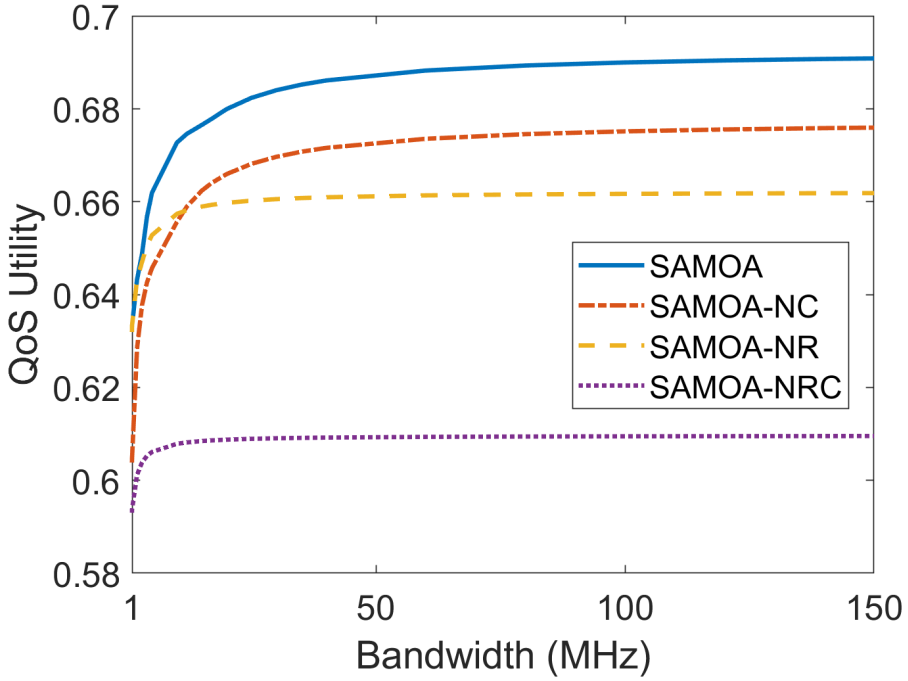

Figure 2.11. QoS utility comparison of SAMOA, SAMOA-NC, SAMOA-NR, SAMOA-NRC under varying bandwidth availabilities at solar energy 10 J and VEC without external load

The observation conforms with the results in Fig. 2.9(d).

Impact of system and application level adaption:

Next we present results to show the benefit of using system level as well as application level (i.e. compression levels) adaptions. Fig. 2.11 shows the results for the scenario when the solar energy is 10 J for a time slot with $\tau = 1s$ and VEC server does not have external load. In this figure, SAMOA-NC denotes the SAMOA algorithm with no additional compression (i.e. lowest fixed compression level 1), SAMOA-NR denotes the SAMOA algorithm with no reconfiguration (i.e., fixed VEC server configuration with highest possible CPU-GPU frequencies in VEC_config1), and SAMOA-NRC denotes the SAMOA algorithm with no additional compression and reconfiguration, i.e., fixed compression level as SAMOA-NC and the fixed VEC server configuration as SAMOA-NR.

Including the compression and reconfiguration, the gain in the performance of SAMOA is apparent. When bandwidth is above 60 MHz, the average QoS utility of SAMOA is 2%, 4%,

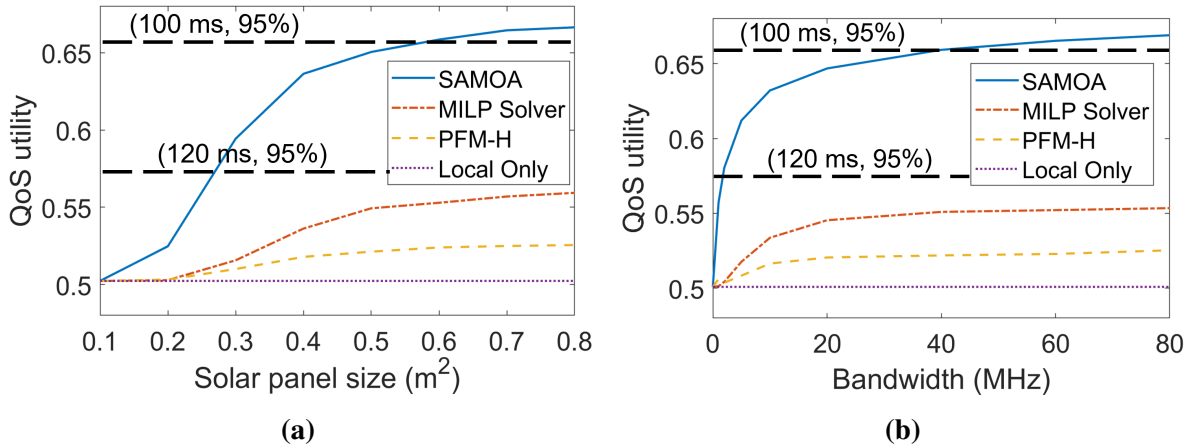

Figure 2.12. QoS utility of 4 algorithms under various scenarios of (a). solar panel size (b) bandwidth availability

and 13% higher than SAMOA-NR, SAMOA-NC, and SAMOA-NRC. The difference between SAMOA and SAMOA-NRC is $> 10\%$, shows the importance of joint system and application level adaptation to improve the QoS utility performance.

2.7.2 Real-world Trace Driven Simulation

Next, we present the online performance of SAMOA using a simulator we have developed [84], which allows creation of realistic trace driven movements, topology, location, and channel condition for each VU at every time slot. The tool simulates the vehicle's trace in a 1000x800 m^2 rectangular neighborhood in Brooklyn, New York City based on historical vehicular traffic data obtained from [11]. With the street topology and traces of vehicles, the tool generates the SNR values from each VU to the 20 SRSUs located in the area. The SNR is generated by assuming VU's transmit power, ρ_i, is 23 dbm and using B1 Manhattan grid layout [44] as the pathloss and slow fading, and the Nakagami-m distribution [45] as the fast fading for the uplink channel model.

At each time slot, we assume each VU is associated to the SRSU which corresponds to the highest signal strength. For the following experiment, we pick one of the SRSUs in this area to demonstrate the simulation result. We assume each VU will have 50% of probability to have a

VLC node with capacity VLC_config1 and 50% of probability to have a VLC node with capacity VLC_config2, which are listed in Table 2.5. On the other hand, at each time slot, we assume VEC server will have 50% of probability to be *without external load* (VEC_config1) and 50% to be *with external load* (VEC_config2), with the specific available CPU-GPU configurations as specified in Fig. 2.9.

Compared Algorithms

We compare the performance of SAMOA with two other relevant algorithms, MILP Solver [68] and PFH-M [70], which are the two closest approaches to SAMOA as they both allow task dependency aware partitioning and offloading with limited VEC communication and computing resources.

MILP Solver: In [68], the authors model the partitioning and offloading problem as a Mixed Integer Linear Programming (MILP) problem. They then propose to use existing standard MILP software packages to find the optimal solution and minimize the end-to-end delay of an DNN application.

PFM-H algorithm: In [70], the authors address the challenges of maximizing the throughput under limited edge computing and communication resources using optimal task partitioning and bandwidth allocation decisions. They formulate the problem to a variant of Knapsack Problem and propose to find the heuristic solution by using Performance Function Matrix based Heuristic (PFM-H) algorithm.

Although the above two approaches consider the constrained communication and computing capacities in VEC server, they do not consider energy constraint. Therefore, we impose an energy constraint check point after these approaches return their offloading and partitioning solution. If the resulting energy consumption violates the constraint, we ask all the VUs to execute their tasks locally. We also present the performance of the naive strategy, *Local Only*, which only allows VUs to execute their tasks locally using VLC nodes.

Trace driven online simulation result

In this experiment, we run the simulation for 1 hour, starting from 9 AM. The duration of each time slot is 1 second, namely, τ= 1s. Fig. 2.12 (a) and (b) demonstrate the average QoS utility over the total simulation time for all the 4 algorithms under different solar panel sizes and bandwidth, respectively. The average QoS utility of the total simulation time is defined as the average of the QoS Utility of every VU instance in every time slot during the total simulation time. In the simulated neighborhood area of Brooklyn, because vehicles are dense and vehicle speed is high, the end-to-end delay is very critical to driving experience. Therefore, we set α= 0.8, which make SAMOA emphasizes more on the end-to-end delay.

Impact of solar panel size:

In Fig. 2.12 (a), the x-axis shows the different solar panel sizes vary from 0.1 to 0.8 m^2. The bandwidth of the SRSU is 20 MHz and equally distributed among the offloading VUs. When the solar panel size is 0.5 m^2, it is shown that the average QoS utility of SAMOA is the best among all the algorithms and is 18.4%, 24.8%, and 29.5% better than MILP Solver, PFM-H, and *Local Only*, respectively. On the other hand, the dash lines in Fig. 2.12(a) shows the end-to-end delay and accuracy values corresponding to the specific average QoS utility values. Note that except SAMOA, none of the above algorithms can achieve the 120 ms end-to-end delay and 95% accuracy simultaneously. SAMOA achieves the average QoS utility of 120 ms end-to-end delay and 95% accuracy when solar panel size is around 0.3 m^2. Moreover, when the solar panel size is higher than 0.55 m^2, SAMOA delivers an average QoS utility of 100 ms end-to-end delay and 95% accuracy.

Impact of bandwidth availability:

In Fig. 2.12 (b), the x-axis shows the different available bandwidth varies from 0 to 80 MHz and the solar panel size of SRSU is 0.5 m^2. When the bandwidth is 40 MHz, the average QoS utility of SAMOA is the best among all the algorithms and is 16.6%, 26.3%, and 31.6% better than MILP Solver, PFM-H, and *Local Only*, respectively. On the other hand, except

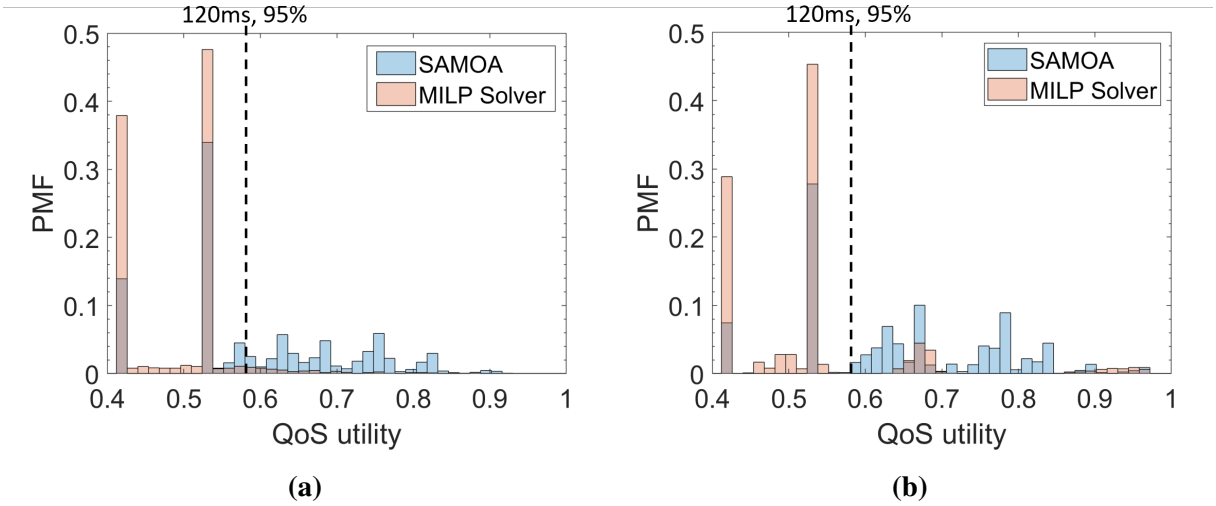

Figure 2.13. PMF of the QoS utility for each individual VU using SAMOA and MILP solver, with 20 MHz bandwidth and solar size equals (a). left, 0.3 m^2 (b) right, 0.8 m^2

SAMOA, none of the above algorithms can achieve the 120 ms end-to-end delay and 95% accuracy simultaneously. SAMOA achieves an average QoS utility of 120 ms end-to-end delay and 95% accuracy when the available bandwidth is around 2.5 MHz. Moreover, when available bandwidth is higher than 35 MHz, SAMOA achieves an average QoS utility of 100 ms end-to-end delay and 95% accuracy.

Empirical probability mass function (PMF) of the QoS:

In Fig. 2.13, we show the empirical probability mass function (PMF) of the individual QoS utility for the VUs. To clearly demonstrate the gap between SAMOA and others, we compare SAMOA with the second best algorithm, MILP Solver, in Fig. 2.13. In Fig. 2.13(a), solar panel size is 0.3 m^2 and bandwidth is 20 MHz. Even though the energy availability is low, 45% of the VU instances can still achieve the QoS utility corresponds to 120 ms end-to-end delay and 95% accuracy by using SAMOA algorithm while only 6% of VUs achieves the same QoS utility by using MILP Solver. When the solar panel size increases to 0.8 m^2, in Fig. 2.13(b), the VU instances that achieve the same QoS utility increases to 65% by using SAMOA algorithm, which is 3 times larger than using MILP Solver.

Impact of α values in delay and accuracy performance:

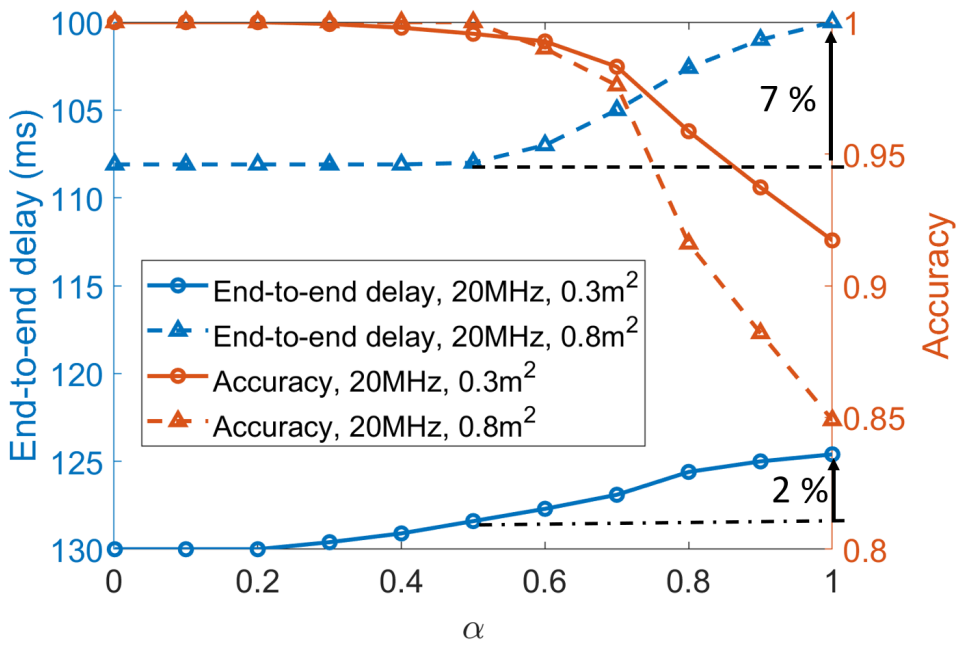

Figure 2.14. Impact of different α values on the end-to-end delay and accuracy performance

In Fig. 2.14, we show how the average end-to-end delay and accuracy will change when α value changes from 0 to 1. For consistency with Fig. 2.13, we also choose 20 MHz bandwidth with 0.3 m^2 and 0.8 m^2 solar panel size, respectively, for comparison. When α increases, the accuracy is reduced in exchange for the decreased end-to-end delay. On the other hand, while the accuracy starts to decrease at $\alpha = 0.5$ when solar panel size is 0.8 m^2, it starts decreasing earlier at $\alpha = 0.2$ when solar panel size is 0.3 m^2. Although larger solar panel size leads to a lesser tunable range for α values, the delay improvement is better. For example, the end-to-end delay reduces by 7% when α increases from 0.5 to 1 for 0.8 m^2 solar panel size while the delay reduces just by 2% for 0.3 m^2 solar panel size within the same range of α. With results like Fig. 2.14, the SRSU operators or service providers can jointly decide the optimal solar panel size and α value during the SRSU deployment based on the desired delay and accuracy performance.

2.7.3 Scalability of SRSU

Note that, in the empirical model and the experiment setup, the maximum available number of offloading VUs N_{VEC} is 6. Herein, we demonstrate the performance of SAMOA when both the capacity of SRSU and the number of served VUs are scaled up. We emulate the scaled up computing capacity of SRSU by adding additional Jetson TX2 boards to the SRSU. The bandwidth and energy availabilities are scaled up in terms of Hz and Joule, respectively. At each time slot, which has duration 1 second, for a given instance which has more than 6 VUs, we execute the following VU distribution algorithm before executing SAMOA. VU distribution algorithm will first calculate the required number of Jetson boards x,

$$x = min(\left\lceil \frac{I}{N_{VEC}} \right\rceil, \left\lfloor \frac{E_t}{10} \right\rfloor) \tag{2.17}$$

where I is the total number of VU, N_{VEC} is 6 in our scenario, and E_t is the current available energy. We use $\lfloor \frac{E_t}{10} \rfloor$ to ensure each active board has at least 10 J of energy for operation within the time slot. Then the algorithm will sort VUs by their SNR values and evenly distribute them by the sorted order into x groups. Finally, the distribution algorithm will assign each group to one Jetson board and execute SAMOA respectively for VUs in that group. For performance comparison, we use the same VU distribution algorithm for MILP Solver and PFM-H. Fig. 2.15 shows the numerical result of these three algorithms using the distribution algorithm under different values of I. We consider all the VUs are using VLC node configuration VLC_config2 and VEC servers (i.e. Jetson boards) *without external load*. The average QoS utility is calculated after 10 rounds of simulations, in which we randomly and uniformly generate the SNR values between 10 to 50 dB for each VU in set $\mathcal{I}$.

Fig. 2.15 shows that with the same available bandwidth and energy, SAMOA performs the best compared to the other two algorithms and *Local Only* even when the number of VUs is high. For example, when there are 20 VUs, SAMOA performs 17.1%, 24.4%, and 27.5% better than MILP Solver, PFM-H, and *Local Only* approaches, respectively, with 80 MHz bandwidth

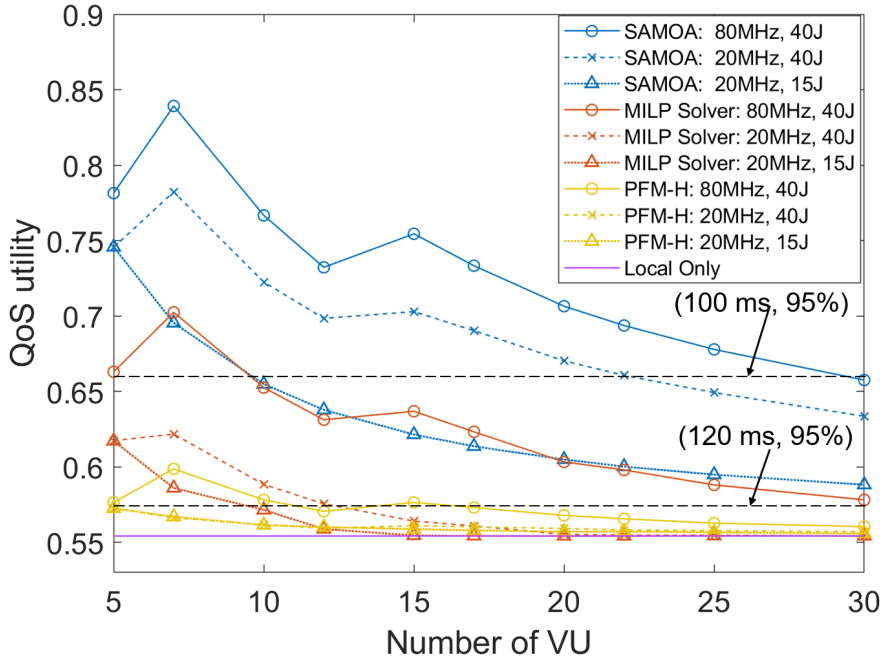

Figure 2.15. QoS utility performance of the 4 algorithms with distributive computing capacity expansion under different number of VUs

and 40 J solar energy. In low resource availability when there are 20 MHz bandwidth and 15 J solar energy, SAMOA's capability will be constrained, but still performs 9.1%, 8.6%, and 9.1% better than MILP Solver, PFM-H, and Local Only approaches, respectively, for 20 VUs.

Fig. 2.15 also shows that when the number of VUs exceeds 10, only SAMOA can achieve the average QoS utility of 100 ms end-to-end delay and 95% accuracy even when the available bandwidth and energy resources are high (i.e. 80 MHz and 40 J). SAMOA achieves the average QoS utility of 120 ms end-to-end delay and 95% accuracy at lower resource availability (20 MHz and 15 J) for up to 30 VUs. However, MILP Solver requires higher available resources to achieve the same average QoS utility for up to 30 VUs and PFM-H can only achieve the same performance for up to 17 VUs.

The results in Fig. 2.15 clearly demonstrate the advantage of SAMOA over other approaches. Moreover, the above trade-off analysis between QoS utility and different resource availability will enable the the service providers to identify the best SRSU configurations given

expected solar generation and VU density.

Run-time analysis:

To measure the run-time complexity of SAMOA, we implement SAMOA and the VU distribution algorithm using Python on the Nvidia Jetson TX2 board. Since the Nvidia Jetson TX2 platform allows parallel processing on multiple cores, we have developed an efficient implementation of SAMOA with parallel multi-core processing, as shown in Fig. 2.16, where we parallelly distribute and execute all the c DAFOS processes of a SAMOA algorithm to the M available CPU cores on the edge computing platform. Note that c is decided by the number of available CPU-GPU configurations. In our experimental setup, $c = 6$ and $M = 6$.

The resulting average execution time of SAMOA, $T_{decision}$ is 250 ms, allowing SAMOA-based task partitioning decision to be made as frequently as every 250 ms. Note that the end-to-end delays for local execution of the vehicular object detection application are 131 ms and 188 ms, respectively, using VLC_config1 and VLC_config2. Hence with reasonable size of VEC and VLC configurations, this experimental result shows that the duration of a time slot τ can be defined as small as 250 ms, which makes the assumption of the constant data rate within a time slot more realistic while ensuring the completion of all the vehicular computation tasks. Note that the application's end-to-end delay is independent of the execution time of SAMOA. SAMOA is executed before a time slot starts, and the resulting decision is used by each VU to partition and offload the application tasks for multiple subsequent executions of the application during the decision time slot, till the next execution of SAMOA and resulting change in partitioning decision.

2.8 Conclusion

In this paper, we propose a real-time system and application adaptive task partitioning and offloading algorithm, SAMOA, to support the computation intensive applications of the vehicles using solar-powered RSU. The algorithm jointly minimizes the end-to-end delay and

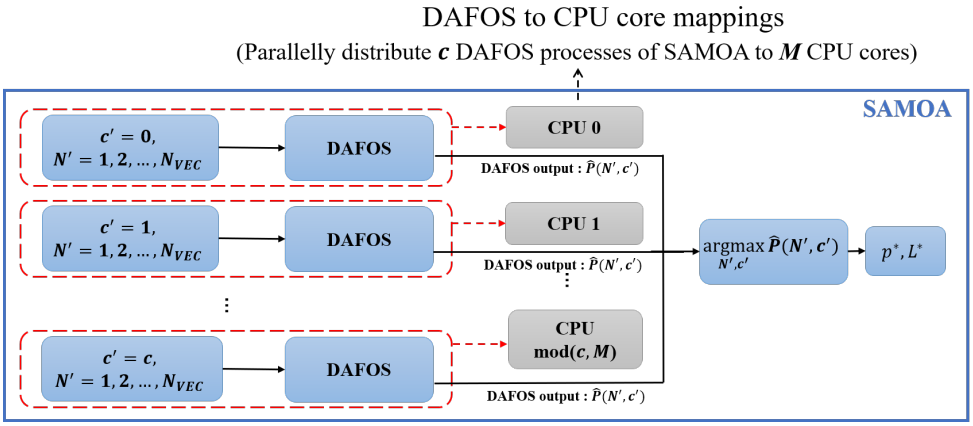

Figure 2.16. Multi-core parallel processing of SAMOA using Nvidia Jetson TX2

maximizes the object detection accuracy, which we jointly define as QoS utility, based on the communication bandwidth, computing, and energy resources availabilities at the SRSU, as well as the computing capacity at the VLC nodes.

We establish empirical models for the computation and communication capacities as well as energy consumption of SRSU. With the empirical model-based simulation, we show that SAMOA significantly maximizes the average QoS utility compared to existing techniques under various resource availability and VU density. As dense deployment of RSUs takes place in our cities and neighborhoods in the next several years, our research results will help service providers and city planners to adopt solar energy based RSUs to avoid additional impact on carbon footprint. Moreover, they will be able to use SAMOA and the simulation and analysis tools we provide to identify adequate SRSU designs, with appropriate computing, communication and solar capacities, for the expected vehicular traffic load and desired delay-accuracy performance.

In the next chapter, we present how the RSU-assisted VEC system can facilitate a more complicated and advanced vehicular application, the multi-vehicle perception fusion application, which involves multiple vehicles' individual vehicular perception tasks and additional fusion tasks at the end. We identify specific challenges and address them with a two-step approach that determines bandwidth allocation, as well as task partitioning and scheduling in real time.

Chapter 2, in full, is a reprint of the material as it appears in IEEE Transactions on

Vehicular Technology 2021, Yu-Jen Ku, Sabur Baidya, and Sujit Dey. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Uncertainty-aware Task Offloading for Multi-vehicle Perception Fusion over Vehicular Edge Computing

3.1 Introduction

This chapter presents our study on enabling an advanced and complex vehicular application, the multi-vehicle perception fusion application, with low execution delay by using an Road-Side Unit (RSU)-assisted Vehicular Edge Computing (VEC) system. The rapid advancement of Internet of Things (IoT) technologies has enabled the emerging connected and autonomous vehicles with the capabilities of multi-sensor data acquisition, processing, and connectivity to support advanced vehicular applications. As a result, these vehicles are now smarter and can coordinate with other vehicles, pedestrians, and road side infrastructures over vehicle-to-everything (V2X) [85] communications, creating an Internet-of-Vehicles (IoV) [86] paradigm.

Nowadays, Advanced Driver Assistance Systems (ADAS) [87] involve multiple vehicular applications, e.g., object detection, localization, navigation, and tracking, which are highly based on complex Artificial Intelligence (AI) driven algorithms. One of the important vehicular applications in the IoV is the perception of the vehicular environment in terms of obtaining knowledge of surrounding objects. A good perception of the vehicular environment helps in

driving assistance, avoiding collisions and accidents, and reducing traffic congestion. However, the individual vehicular perception may at times miss visual or contextual information due to limited field of view, occlusions, bad light, or bad weather conditions. Fusion of perceptions from multiple vehicles can improve the perception of the vehicular environment as a whole, and provide more precise and complete assistance to the involved vehicles.

The above multi-vehicle perception fusion application needs AI-based algorithms for both object detection on each Vehicle User (VU) and object matching across multiple VUs to identify distinct objects. Therefore, performing this perception fusion on some VUs may not be feasible due to insufficient computing capabilities, or may be inefficient in terms of fusion, as choosing a VU for the fusion application can be complex and nontrivial. In order to support the aforementioned computations, a more powerful computing server is required apart from the on-board computing units of the VUs. A viable solution is to use VEC server, which is collocated with a RSU, as VEC server is usually more powerful than the vehicle's on-board Vehicular Local Computing (VLC) unit. Moreover, VEC server is located at the vehicular network edge with just one-hop wireless communication away from the VUs, and thus doesn't incur routing or queuing delays as in the cloud computing over the backbone network. Additionally, VEC server being a centralized unit can efficiently perform the fusion of perception data from individual vehicles. Fig. 3.1 shows the schematic diagram of the edge-based multi-vehicle perception fusion system, with a Macro Base Station (MBS) acting as a controlling entity.

However, offloading the whole multi-vehicle perception fusion application to the VEC server may not be always feasible due to inferior wireless channel conditions and the limited computing capacity of the VEC server. Partitioning and offloading the application, thus, may be necessary. In this work, we consider a fine-grained dynamic offloading scenario where the above application can be partitioned into multiple tasks. For those tasks that are originated from the VUs (e.g. object detection), they can be executed locally at the VLC servers or offloaded and executed at the VEC server. To efficiently utilize the computing resources of both the VLC and the VEC servers, making partitioning and offloading decisions is challenging and involves the

considerations of different wireless channel conditions between each VU and the RSU, different computing capacities of the VLC and the VEC servers, and the dependency among the tasks. Moreover, the decisions need to be determined in real-time due to the highly dynamic wireless channel conditions and network pattern of the vehicular network.

With a different objective and system scenario, in one of our another works [88], we proposed a real-time task partitioning and offloading algorithm for the offloading of AI-based object detection applications to minimize the end-to-end execution delay while maximizing the detection accuracy performance. However, the proposed algorithm can only be applied to tasks with sequential dependency, and the available bandwidth is assumed to be evenly distributed between all the VUs. The tasks of the multi-vehicle perception fusion application considered in this study, on the contrary, may have both sequential as well as parallel dependencies. Moreover, note that the task composition of a multi-vehicle perception fusion application is dependent on the number of detected objects, which is the output of the object detection task. Now, the object detection task itself is one of the tasks in this application. In this situation, partitioning and offloading decisions need to be made while the algorithm is uncertain of the complete task dependency graph. Finally, to utilize the wireless communication resources more efficiently, we also actively determine the bandwidth allocated to each VU for task partitioning and offloading, according to real-time channel conditions.

To minimize the end-to-end delay of a multi-vehicle perception fusion application while addressing the above challenges, in this work, we propose FOrecasting and unceRtainty-aware Multi-vehicle task Offloading and Scheduling Algorithm (FORMOSA). FORMOSA is running at the MBS's Mobile Edge Computing (MEC) server and determines in real-time the optimal task partitioning and offloading strategy, the task execution schedule (i.e. the start and end time of the execution of each task), and the bandwidth resource allocation. To address the uncertainty of the task dependency graph, FORMOSA adopts a two-step approach. The first step enforces a dynamic programming-based heuristic algorithm that determines the above decisions over an estimated static task dependency graph. The second step is a heuristic task offloading and

scheduling refinement algorithm, which reacts in ultra-low latency after the actual graph is known.

The main contributions of this work are summarized below.

(a) To the best of our knowledge, this is the first work of joint task partitioning, offloading, and scheduling with bandwidth coordination to minimize the end-to-end fusion delay for edge-based multi-vehicle perception fusion applications.

(b) We identify the uncertainty of the task dependency graph to the multi-vehicle perception fusion applications as the major challenge during offloading decision-making, and address this challenge by task graph prediction and proposing a two-step algorithm.

(c) Given a task dependency graph with both sequential and parallel dependencies, we propose a dynamic programming based heuristic algorithm for task partitioning and offloading, task scheduling, and bandwidth allocation to minimize the application completion time.

(d) We propose a two-phase task offloading and scheduling adaptation algorithm to accommodate any immediate change of the task dependency graph in real time during the execution of the multi-vehicle perception fusion application.

3.2 Related Work

Information sharing in an IoV network can be a challenging topic, [89] and [90] analyze the challenges for information sharing in a city-scale Vehicular Sensing Network (VSN). They optimally select the optimal information source to maximize the network capacity and minimize communication costs. Additionally, in [89], a gaming-based methodology is further proposed to ensure that each participating vehicle shares the correct information with the network. However, these works only address the data transmission challenges. The corresponding challenges in vehicular application computing are not considered.

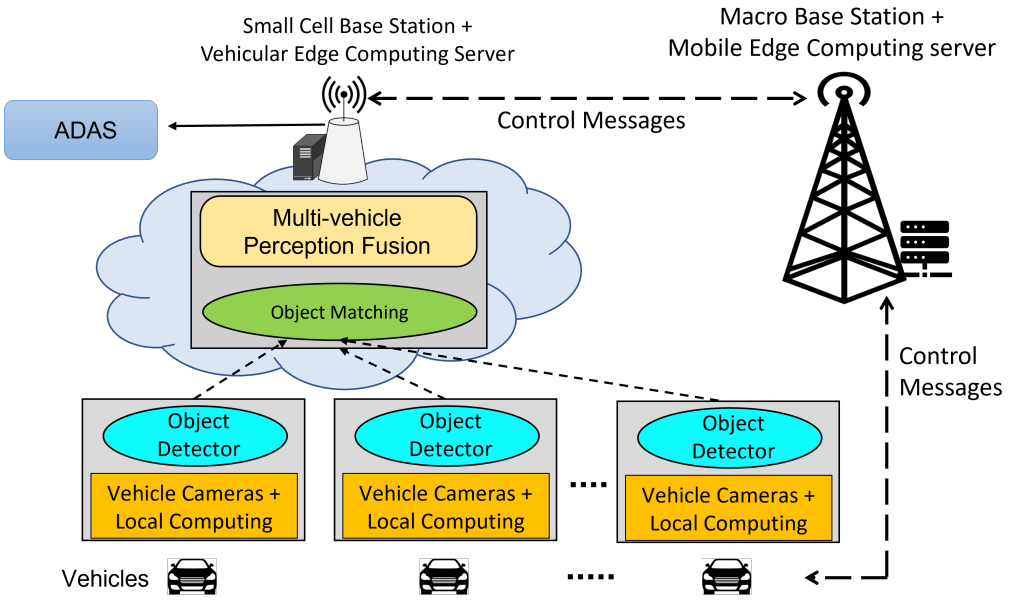

Figure 3.1. A building block overview of the edge-based multi-vehicle perception fusion system with participating IoV devices

Minimizing the application's computation delay for the vehicular edge computing system by using task offloading strategies has been studied in [58, 60, 61]. However, these works consider only the utilization of edge computing resources. In this work, both the VUs and RSU are equipped with computing units (i.e. VLC and VEC servers), and hence, are capable of executing tasks for vehicular applications. Since the computing capacity at VLC and VEC servers are limited, how to wisely utilize their computing resources for computing vehicular applications is a challenging problem. [63–65, 91] study task offloading strategies that are aware of both the VLC and VEC servers' capacities. [63] and [64] proposed joint task partitioning and offloading as well as communication resource allocation methods to minimize the cost for utilizing the communication and computation resources, under the application's delay constraints. [65] minimizes both the application's computation delay and the utilized VEC server's computing resources. [91] proposes a particle swarm optimization-based partial computation offloading approach to maximize the application reliability under task latency constraints. However, these studies assume the tasks of the considered applications can be arbitrarily partitioned and executed in parallel on different servers, ignoring the dependency between each task.

The authors in [66–69], progressively, consider the task dependency while designing the task partitioning and offloading strategies. These studies focus on Deep Neural Network (DNN)-based applications. To minimize computation delay and communication overhead, [66] proposed to partition the convolutional layers of the DNN network into several tasks which, because of the data dependency, can be executed in parallel on different edge servers. [67] proposed a similar idea to jointly minimize the utilized edge server memory and communication resources. On the other hand, [68] and [69] partition the DNN network sequentially for offloading to minimize the DNN network's execution delay [68] or to minimize the cost for utilizing communication and computing resources [69]. They allow the VLC server to stop the execution of a DNN early and offload the rest of the convolutional layers to the VEC server. The edge server can decide whether to execute the rest of the layers or to adopt the current output with an object detection performance trade-off. Although these task partitioning methods consider task dependency during decision-making, they require the tasks to be fully partitioned either in parallel or in sequential. Multi-vehicle perception fusion application considered in this work, however, involves tasks that have both parallel and sequential dependencies. Therefore, the decision-making complexity and the possible partitioning and offloading decisions for all the tasks are much more sophisticated than the above scenarios.

[92] and [93] aim at facilitating task partitioning and offloading under joint parallel and sequential task dependencies. [92] proposed a bucket algorithm that determines the set of tasks for offloading to minimize the maximum task completion time among all vehicles. [93] focuses on minimizing the average application completion time and average energy consumption for all the vehicles by planning the task offloading schedule with a three-phase algorithm. These works rely only on VEC servers to execute the tasks, and the rest of the tasks are queued in each vehicle. [94–99], on the other hand, consider task partitioning and offloading between both VLC and VEC servers while the tasks have both parallel and sequential dependencies. They proposed to assign priority to each task to address the offloading challenge under joint parallel and sequential task dependencies. For example, in [94–96], the priority assigned to a

task depends on its computation complexity, communication cost, and the maximum priority value among all of its immediate successor tasks. Tasks are scheduled and offloaded following the descending priority order. They then propose the task offloading and scheduling algorithms to minimize the maximum task completion time [94, 95], or to minimize the overall reliability of execution of all the tasks for an application under the assumption that each server has a failure rate for task execution [95]. The authors in [97, 98] assign to each task a priority based on its computational complexity and determine the maximum priority among all its immediate successor tasks. Following the descending priority order, Reinforcement Learning algorithms are then proposed to determine the offloading decision for each task to minimize the average application completion time [97, 98] and the average energy consumption [97] of all the vehicles. [99] proposed to categorize tasks into high, medium, and low priority groups based on the dependency of the tasks that are currently being executed. Separate algorithms are proposed for each priority group to determine the task offloading and scheduling plans for minimizing the maximum task completion time. However, these studies require the complete task graph of the vehicular applications to be available for the algorithms during the decision-making phase. Contrary to these works, in this study, we consider a scenario where the information of the task dependency graph is not complete when the task partitioning, offloading, and scheduling decisions are made. Moreover, these studies either assume the bandwidth allocated to each VU is fixed or equally divided. Hence cannot leverage the potential delay reduction for data transmission by actively and optimally allocating bandwidth resources to each VU. The authors in [100] jointly determine the task partitioning and offloading, as well as bandwidth allocation decisions while considering both the parallel and sequential task dependencies. They focus on data streaming applications and aim at maximizing the data streaming throughput by optimizing the maximum delay of the slowest task and the slowest data transmission link. However, in our considered scenario of multi-vehicle fusion, the application completion time involves the accumulation of both the execution delay of all the tasks and the transmission delay of their input data. Moreover, similar to the aforementioned studies, the proposed approach cannot be applied

(a) Camera scene from vehicle 1 (b) Camera scene from vehicle 2 (c) Camera scene from vehicle 3 (d) Camera scene from vehicle 4

Figure 3.2. Two cases of multi-vehicle perception fusion from real-world vehicular images

to task partitioning and offloading decision-making with incomplete knowledge of the whole task graph.

To the best of our knowledge, this is the first work that not only jointly determines the optimal task partitioning, offloading, and scheduling along with bandwidth allocations for the edge-based multi-vehicle perception fusion applications, but also addresses the challenges of minimizing the application completion time under the uncertainty of the task dependency graph.

3.3 Edge-based Multi-vehicle Perception Fusion

In this section, we briefly introduce the background of edge-based multi-vehicle perception fusion and the concept of object matching. Perception fusion combines the perception of individual sources in a given region and fuses them together to give an enhanced view of the environment. This enhanced perception is more powerful as it covers missing detections, occluded objects, and out-of-view objects from individual perceptions. The beneficiaries from this perception fusion are not only the smart vehicles that are the individual data sources, but all road users including the legacy vehicles, bicyclists, and pedestrians as well. In this work, we focus on camera-based multi-vehicle perception fusion.

3.3.1 Fusion at the Edge

Perception fusion involves fusion tasks to combine the pre-computed data from multiple sources. If fusion tasks are executed at each individual source site, it requires high volume camera data sharing among all the vehicles, creating extremely huge data transmission overhead.

Moreover, all individual vehicles may not have enough computation capability to perform the fusion tasks along with their individual tasks. Therefore, in this work, we consider that the fusion tasks are executed at the edge. The edge computing server has a much higher computational capacity than individual vehicles, and each source only needs to share the camera data once to the edge server. Furthermore, vehicle dynamics do not change the computing paradigm.

3.3.2 Object Matching

A single vehicle can miss detections of some objects' presence due to false negatives, or not being able to detect objects out of its field-of-view. Augmenting views from other sources can include objects not detected by a vehicle, but it can also have some duplicate detections of the same objects, although from different angular viewpoints. Fig. 3.2 shows two such scenarios. Fig. 3.2(a) and (b) are images taken at the same time, facing the same intersection, from vehicles 1 and 2, respectively. In the left image pair, Fig. 3.2(a) sees 4 vehicles, but only 3 are in the view of Fig. 3.2(b); hence the additional detection can be shared with the vehicle 2. Similarly, Fig. 3.2(c) and (d) are images taken at the same time, facing the same intersection, from vehicles 3 and 4, respectively. The object detection result in Fig. 3.2(c) encounters a false negative due to occlusion from pedestrians (i.e. presence of the white car is blocked by the pedestrians). However, vehicle 4 (i.e. Fig. 3.2(d)) can clearly see the white car and can share the corresponding detection with vehicle 3. To facilitate correct and efficient detection information sharing, object matching across vehicles is required. Object matching aims to determine which object's information is missing for a vehicle so that correct information can be provided. The matching result (i.e., fusion result) is a list of distinct objects the involved vehicles detect. Determining this list is essential, as duplicate detections can cause misleading information and distrust for the vehicles receiving duplicate information. Meanwhile, drivers might be misinformed due to missed detections. Matching objects from different viewpoints is challenging; moreover, the cameras are at different distances from the objects, and thus the size of detected objects will vary across cameras. In one of our previous studies [101], we used a deep learning approach for object matching which learns the

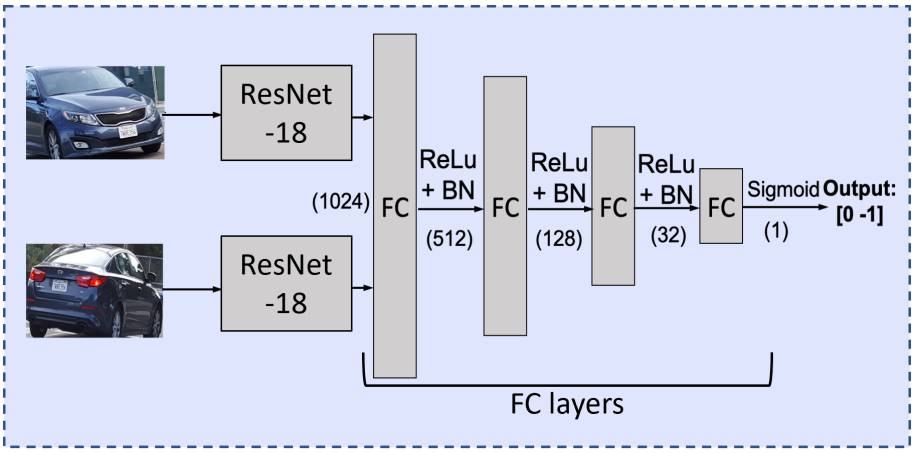

Figure 3.3. Object matching machine learning model

correspondence of the same objects from different views and thus can find distinct objects for perception fusion.

For the object matching learning model, we use ResNet-18 [102] pre-trained on ImageNet [103] as a feature extractor with an additional Fully Connected (FC) layer added to the output to act as the object matching classifier as shown in the Fig. 3.3. To preserve the feature set learned from the large ImageNet dataset, we freeze the ResNet-18 layers during training and only adjust the weights of the FC layer. We collected real-world images from an IoV environment and created our own dataset of 13,781 pairs of images from various viewpoints, including both vehicle-to-vehicle camera views as well as street camera-to-vehicle camera views. The dataset is used for model training, and the corresponding performances are listed in [101].

3.3.3 Perception fusion Performance

To determine the gain of perception fusion, Fig. 3.4 shows an example of the cumulative number of objects detected on a temporal scale of 3 seconds with camera framerate of 6 fps. In this experiment, we used the same two cameras settings as Fig. 3.2(a) and (b), one of which produced n detections per frame and the other which produced m detections per frame. We

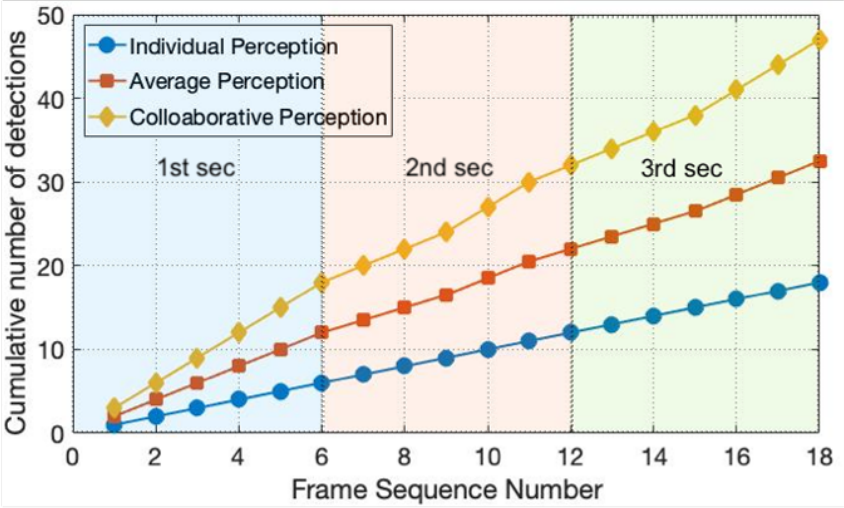

Figure 3.4. Temporal performance of different perception schemes of the two vehicles over 3 seconds

compare the performance of one individual camera's detections with respect to the average detections, i.e. $(m+n)/2$, and perception fusion performance (i.e., collaborative perception), which is in terms of the number of detections of distinct objects. The plot shows that the perception fusion can detect more distinct objects than the average perceptions of the individual camera sources, and can sometimes experience a significant gain when an individual camera is under occlusion or false negative situations. The percentage gain w.r.t. the average perception for the entire image pair dataset is 38.2%.

Based on the proposed fusion application for perception fusion, in the next section, we are going to introduce the system model of the edge-based multi-vehicle perception fusion system considered in Fig. 3.1.

3.4 Systems Overview

In this section, we introduce an overview of the edge-based multi-vehicle perception fusion system and the corresponding challenges. The fusion application modeled in this section includes both the object matching tasks and the object detection tasks for each vehicle. For ease of reference, we list the key notations in Table 3.1.

Table 3.1. Summary of key notations and abbreviations

Notation	Description
$r_{b,i}$	uplink transmission rate from VU i to SRSU b
$\mathcal{I}$	current VU set
b	the SRSU for study
i	the index of VU
v	index of tasks, can also be represented as $V_{i,k,n}$
$D(k)$	data size of each input for task type k
x_v	offloading decision of task v
B_i	bandwidth allocated to VU i
S_v	the scheduled start time of task v
F_v	the scheduled finish time of task v
c_i^l	CPU-GPU configuration of VLC node of VU i
c^e	CPU-GPU configuration of the VEC server
T_v^e	the execution delay of task v at VEC server
T_v^l	the execution delay of task v at VLC server
Ψ_v^s	the immediate successor set of task v
Ψ_v^p	the immediate predecessor set of task v
O_i	the random variable for the number of detected objects at VU i
o_i	the actual number of detected objects at VU i
$\hat{o}_i$	the predicted number of detected objects at VU i
t_i	the time when the outcome of O_i is available
$\mathcal{G}$	the fusion application task graph
$\mathcal{V}$	the set of nodes (i.e. tasks) in $\mathcal{G}$
$\mathcal{E}$	the set of edges (i.e. task dependencies) in $\mathcal{G}$

3.4.1 Network Model:

We consider that the edge-based multi-vehicle perception fusion system consists of one RSU b and multiple VUs, whose operations are controlled by a MBS. The RSU is equipped with a radio unit Small Base Station (SBS) and a computation module VEC server. Each VU is equipped with an On-Board Unit (OBU) radio unit as well as a computation module VLC server, and is capable of providing the vehicular data for fusion. We assume both VEC and VLC servers can only execute one task at a time. We divide the operation time of the system into multiple time slots. We assume the following modelings and discussions are within a single time slot t. Therefore, we simplify superscript t for each variable. For each time slot, there exists a set of VUs $\mathcal{I} = \{1, 2, ..., I\}$ in the coverage area of the RSU b. The set $\mathcal{I}$ and the channel condition of

each VU in $\mathcal{I}$ will vary in different time slots due to the mobility of the vehicles.

3.4.2 Communication Model:

The Signal-to-Noise Ratio (SNR) of uplink transmission from VU i to the SBS of RSU b is denoted as $\eta_{b,i} = \frac{\rho_i * g_{b,i}}{N_0}$, where ρ_i is the transmit power of VU i, $g_{b,i}$ is the uplink channel gain, and N_0 is the noise level. We assume Orthogonal Frequency-Division Multiplexing (OFDM) technology is used for uplink transmission, hence, the interference from other VUs is negligible and the uplink transmission rate from VU i to RSU b can be represented as,

$$r_{b,i} = W * B_i * log_2(1 + \eta_{b,i}) \tag{3.1}$$

where B_i is the number of the Smallest Bandwidth Element (BE) (e.g. the Physical Resource Block (PRB)s in Long-Term Evolution (LTE) system) that allocated to VU i and W is the bandwidth of each BE. Note that the overall bandwidth available for the SBS are limited, in other words,

$$\sum_{i=1}^{I} W * B_i \leq W_{max} \tag{3.2}$$

where W_{max} is the maximum available bandwidth for the uplink transmission.

The inter-cell interference is assumed to be mitigated by Inter-Cell Interference Coordination (ICIC) technologies, e.g. Fractional Frequency Reuse (FFR) [73]. Note that the multi-vehicle perception fusion application's output data size is very small compared to its input data size. For example, the application considered in this study uses the captured images as input, whose data sizes are usually more than several KBytes. However, its output, the list of detected distinct objects, can be stored in less than 1 KBytes of memory. Moreover, due to the higher transmission power, the downlink data rate is usually higher than the uplink data rate [47]. Therefore, we ignore the impact of downlink data transmission in our study.

Note that the uplink channel gain $g_{b,i}$, and hence $r_{b,i}$ will change rapidly across different

time slots due to the mobility of VU. In fact, in Section 3.6, we assume the channel gain is generated for each time slot by the current VU's position. We use B1 Manhattan grid layout [44] as the pathloss and slow fading models, and the Nakagami-m distribution [45] as the fast fading model, which have been widely used by the industry [46, 48] and are shown to be sufficient to model vehicular communication channel [45]. Therefore, the optimal bandwidth allocation B_i will be different for each time slot.

3.4.3 Multi-vehicle Perception Fusion Task Model:

We assume that at each time slot, every VU provides an image frame captured by its onboard camera for fusion. Each image is assumed to be a 1080p RGB image. Each image is first decoded, resized, and encoded into a 2-dimensional 300 by 300 matrix input and forwarded to an object detection application. Note that the data size of the decoded 1080p RGB image is very large, typically around 6 MB. Therefore, we combine decode and resize as one task in our task model. In this study, we use SSD-MobileNetV2 [56] for object detection because of its lightweight computations. An image of the bounding area of each detected object is then sent to a ResNet-18 DNN network, which is the feature extractor of the object matching model. Each detected object needs an exclusive ResNet-18 task. Its output feature, along with the output feature of another ResNet-18 for another detected object from a different VU will be provided to an FC layer, where final object matching classification decision is made. Note that each pair of objects, where these two objects are from two different VUs, requires an FC layer for the object matching classification decision.

The task graph of the whole multi-vehicle perception fusion application considered in this study is then shown in Fig. 3.5. $V_{i,k,n}$ denotes the n-th type k task that is created by device i, where $i \in \{\mathcal{I} \cup 0\}$ and $i = 0$ denotes the RSU. k is the task type. For example, in this study, $k = 1, 2, 3, 4, 5$ represent Decoding and Resizing, Encoding, SSD-MobileNetV2, ResNet-18, and FC layer tasks, respectively. For consistency, we assume task type $1 \geq k \geq 4$ are created by devices $i > 0$ (i.e. VU), and task type $k = 5$ is created by devices $i = 0$ (i.e. RSU). $1 \geq n \geq N(i,k)$,

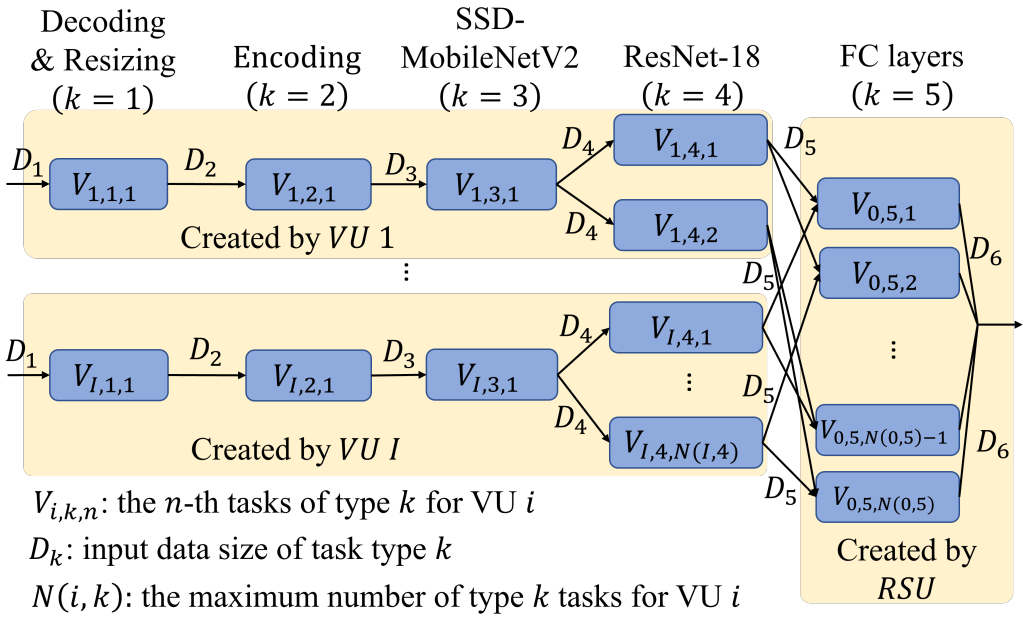

Figure 3.5. Task composition of the edge-based multi-vehicle perception fusion application

Table 3.2. The data size of each input for different task types

Task type k	1	2	3	4	5
Data size (KBytes)	27	270	6.2	0.9	4

where $N(i,k)$ is the maximum number of the k-th type of tasks created by device i. Without loss of generality, we assume the data size of each input for the task $V_{i,k,n}$ only depends on k and is the same across different VUs. We denote the data size of each input for task $V_{i,k,n}$ as $D(k)$. We have listed the input data size for all the task types considered in this work in Table 3.2.

For each VU i, $i > 0$, the number of ResNet-18 tasks $N(i,4)$ depends on o_i, the number of objects detected in its captured image (i.e. one of the outputs of its SSD-MobileNetV2). Consequently, for RSU, the number of FC layers $N(0,5)$ depends on the value of o_i from each VU $i, i > 0$. Note that the values of o_i, $\forall i > 0$ are not determined at the beginning of the current time slot t_0. Therefore, mathematically, we model the tasks shown in Fig. 3.5 as a task graph $\mathcal{G}(O_1, O_2, ..., O_I) = (\mathcal{V}(O_1, O_2, ..., O_I), \mathcal{E}(O_1, O_2, ..., O_I))$, $O_i, i \in \mathcal{I}$ is the random variable of the number of detected objects in VU i's image. $\mathcal{V}$ is the set of nodes representing the tasks, that is $\mathcal{V} = \{V_{i,k,n} | \forall i, \forall k, \forall n\}$. $\mathcal{E}$ is the set of edges representing the dependency between these

tasks. The value of O_i is determined at time t_i, where $t_i > t_0$ and is equal to the time when SSD-MobileNetV2 of VU i (i.e. $V_{i,3,1}$) is completed. Therefore, it is challenging to determine the optimal bandwidth allocation and task offloading as well as scheduling decisions at t_0 based on incomplete knowledge of the task graph $\mathcal{G}$. On the other hand, if the above decisions are made at t_i, the offloading and scheduling of any tasks before $V_{i,3,1}$ that were already executed will not be optimal.

We assume that if a task $v \in \mathcal{V}$ is created by a VU (i.e. $v = V_{i,k,n}$ and $i > 0$), it can be either executed locally at the VLC server or be offloaded and executed at the VEC server. On the other hand, for a task $v = V_{i,k,n}$ such that $i = 0$, it can only be executed at the VEC server. We use x_v to represent the offloading decision of task v, the action space of x_v for each task v is defined as following,

$$x_v \in \begin{cases} \{i, 0\} & \text{if } v = V_{i,k,n} \text{ and } i > 0, \\ \{0\} & \text{if } v = V_{i,k,n} \text{ and } i = 0. \end{cases} \tag{3.3}$$

Furthermore, once a task v is offloaded and executed at the edge, its successor tasks will be executed at the edge. That is,

$$x_v \leq \min_{v' \in \Psi_v^s} x'_v \tag{3.4}$$

where we define Ψ_v^p and Ψ_v^s as the sets of immediate predecessors and successors for each task $v \in V$, respectively. That is, $v' \in \Psi_v^p$ if the directed edge $(v', v) \in \mathcal{E}$ and $v' \in \Psi_v^s$ if the directed edge $(v, v') \in \mathcal{E}$.

3.4.4 Task Execution Delay

In this work, we choose to empirically model the execution delay for each task $v \in \mathcal{V}$ when using the VEC and VLC servers for execution. Since both VEC and VLC servers can only execute one task at a time, the task execution delay depends only on the computing capacity of VEC and VLC servers. For each task $v = V_{i,k,n}$, we denote $T_v^e(c^e)$ as the task execution delay

Table 3.3. The measured execution delay for each task type under different server configurations (unit: ms)

Task type k \ Configuration index	1	2	3	4	5	6
1	123	61	33.3	19	16.5	15.5
2	11.4	5.4	3.5	2	1.7	1.58
3	51	18	11	5.1	5.9	4.3
4	16.9	6	3.6	1.7	1.6	1.5
5	2.25	1.05	0.65	0.38	0.32	0.3

Table 3.4. The CPU, GPU, and EMC configurations for each chosen server configuration index (unit: MHz)

Configuration index	1	2	3	4	5	6
CPU frequency	345	652	960	1570	1880	2300
GPU frequency	112	318	522	1120	1230	1400
EMC frequency	200	665	665	2300	2300	2300

when task v is executed at the VEC server which has computing capacity c^e and $T_v^l(c_i^l)$ as the task execution delay when task v is executed at the VLC server which has computing capacity c_i^l. Table 3.3 shows an example of the empirical task execution delay that we measured by implementing each type of tasks on a commercial edge computing device, Nvidia Jetson Xavier [104]. We also list the corresponding computing capacity in Table 3.4, where the computing capacity is modeled in terms of the CPU, GPU, and EMC (External Memory Controller) frequencies which we extracted from Nvidia Jetson Xavier board [105].

3.4.5 Task Start and Finish Time

We denote S_v and F_v as the start and finish time for task v. The start time S_v indicates the time when task v starts to be executed at the computing unit defined by x_v and the finish time F_v represents the time when the execution of task v finishes and its output data is available at the computing unit. In the following, we discuss the start and finish time of task v in two modes, the local computing mode ($x_v > 0$) and edge offloading mode ($x_v = 0$):

Local computing mode ($x_v > 0$)

If task $v = V_{i,k,n}$ is assigned to be executed at the VLC server, its finish time F_v can be defined as,

$$F_v = S_v + T_v^l(c_i^l) \tag{3.5}$$

where $T_v^l(c_i^l)$ is the execution delay by using VU i's VLC server to execute task v. Furthermore, its start time S_v can be defined as the following,

$$S_v \geq \max\{F_{l,v}^{server}, F_{l,v}^{pred}\} \tag{3.6}$$

where $F_{l,v}^{server}$ represents the earliest available time at VU i's VLC server that can start to execute task v and $F_{l,v}^{pred}$ is the time when all of the task v's immediate predecessor tasks are executed. We model $F_{l,v}^{server}$ and $F_{l,v}^{pred}$ by using the following equations:

$$F_{l,v}^{server} = \max_{v' \text{ s.t. } S_{v'} < S_v \text{ and } x_{v'} = x_v} F_{v'} \tag{3.7}$$

and,

$$F_{l,v}^{pred} = \max_{v' \in \Psi_v^p} F_{v'} \tag{3.8}$$

Edge offloading mode ($x_v = 0$)

If task v is assigned to be executed at the VEC server, its finish time F_v can be defined as,

$$F_v = S_v + T_v^e(c^e) \tag{3.9}$$

where $T_v^e(c^e)$ is the execution delay by using the VEC server to execute task v. However, before the VEC server starts to execute task v, the following requirements need to be fulfilled, 1) the VEC server is not executing any task, 2) the immediate predecessor tasks of task v are completed, and 3) the output of the immediate predecessor tasks of task v are transmitted to the VEC server if these tasks are executed locally. Therefore, S_v should satisfy the following equation,

$$S_v \geq \max\{F_{e,v}^{server}, F_{e,v}^{pred}\} \tag{3.10}$$

and $F_{e,v}^{server}$ denotes the earliest possible time when the VEC server is available, which can be defined as,

$$F_{e,v}^{server} = \max_{v' \text{ s.t. } S_{v'} < S_v \text{ and } x_{v'}=0} F_{v'} \tag{3.11}$$

On the other hand, $F_{e,v}^{pred}$ denotes the time when all the immediate predecessor tasks of task v are executed and their output are transmitted to the VEC server, which can be expressed as,

$$F_{e,v}^{pred} = \max_{v'=V_{i',k',n'} \in \Psi_v^p} F_{v'} + \mathbb{1}_{x_{v'}>0}(x_{v'}) \frac{D(k')}{r_{i',b}} \tag{3.12}$$

where $\mathbb{1}_{x_{v'}>0}(x_{v'})$ is the indicator function such that,

$$\mathbb{1}_{x_{v'}>0}(x_{v'}) := \begin{cases} 1 & \text{if } x_{v'} > 0\,, \\ 0 & \text{if } x_{v'} = 0\,. \end{cases} \tag{3.13}$$

Based on the above discussion, the task execution scheduling S_v and F_v, and task offloading decision x_v need to be jointly determined while satisfying both parallel and sequential task dependencies (e.g., Fig. 3.5).

3.4.6 End-to-end Fusion Delay and Problem Formulation

Herein, we consider the performance metric for multi-vehicle perception fusion in terms of the end-to-end fusion delay, which is the time taken for all the tasks to be completed. Therefore, the end-to-end fusion delay can be defined as the maximum finish time among all the tasks in $\mathcal{V}$. Take Fig. 3.5 as an example, the end-to-end fusion delay is defined as the finish time of the last FC layer task that is being executed.

Given each VLC server's capacity and the channel conditions between RSU to each VU, the objective of this paper is to determine the optimal task partitioning and offloading strategy x_v, the task scheduling plan S_v and F_v, and the bandwidth resource allocation B_i, to minimize the end-to-end fusion delay for a multi-vehicle perception fusion application. Since VUs are moving, and hence the VU set and each VU's uplink channel conditions are changing, the above decisions need to be updated for every time slot to guarantee that they are still optimal. At the

beginning of each time slot, the decisions have to be made in real-time based on incomplete knowledge of the actual task graphs, as shown in Section 3.4.3, while satisfying both parallel and sequential task dependencies in Eq. 3.6-3.12. For each time slot, the optimization problem can be formulated as,

$$\underset{x_v, S_v, F_v \forall v \in \mathcal{V}, B_i \forall i \in \mathcal{I}}{\text{minimize}} \quad \underset{v \in \mathcal{V}}{max}\, F_v \tag{3.14a}$$

$$\text{subject to } (3.2) - (3.6), (3.9), (3.10) \tag{3.14b}$$

3.5 Solution Methodology

We first consider a rather restricted and simple formulation where the task graph $\mathcal{G}$ is fixed before t_0, that is, the outcomes of random variables of the number of detected objects of each VU, $o_1, o_2, ..., o_I$, are known before t_0 (i.e. $t_i \leq t_0 \forall i \in \mathcal{I}$). Note that in our problem formulation 3.14, given the fix task graph $\mathcal{G}$, the decision variables $x_v, S_v, F_v \forall v \in \mathcal{V}$ and $B_i \forall i \in \mathcal{I}$ are a mixture of integers and continuous values, and the objective of problem 3.14 is a non-linear function of the above variables. Therefore, problem 3.14 is an NP-hard nonlinear integer programming problem [83]. The complexity of exhaustively searching for the optimal solution is extremely high for this restricted problem formulation. Even by assuming S_v and F_v can be optimally determined within linear time, the complexity of exhaustively searching for all possible values of $x_v \ \forall v \in \mathcal{V}$ and $B_i \ \forall i \in \mathcal{I}$ is $O(2^{|\mathcal{V}|} B^I)$, where I is the total number of VUs and $|\mathcal{V}|$ is the overall number of tasks in $\mathcal{G}$. Note that $|\mathcal{V}|$ grows exponentially with the number of VUs, and the number of objects detected by each VU. This leads to prohibitively expensive time complexity.

However, the problem considered in this study is more complicated than the above restricted formulation as the task graph $\mathcal{G}(O_1, O_2, ..., O_I)$ varies over time. $\mathcal{G}(O_1, O_2, ..., O_I)$ varies at $t_i, i \in \mathcal{I}$, which is the time when the object detection task of VU i is completed. That is, $t_i = F_v$ such that $v = V_{i,3,1}$.

Although the scheduling decisions for the object detection tasks do not affect the outcome

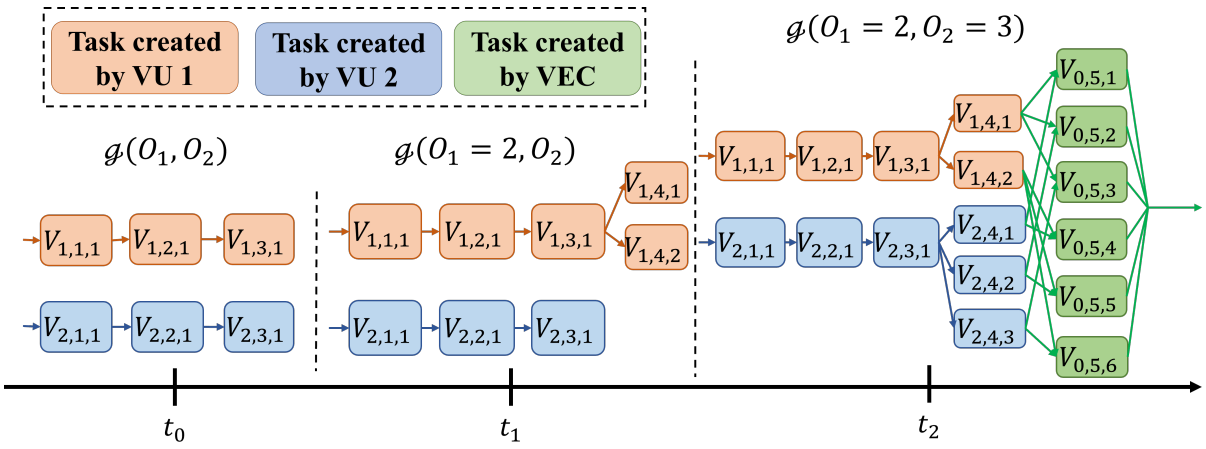

Figure 3.6. The variance of knowledge of task graph by time according to the outcome of random variables of the number of detected objects

values of $O_1, O_2, ..., O_I$, they will affect the time when the exact composition of the task graph is revealed to the decision-making algorithm. Before all the object detection tasks are completed, the decision-making algorithm can only be based on incomplete knowledge of task graph $\mathcal{G}$, leading to non-optimal end-to-end fusion delay.

Fig. 3.6 demonstrates an example of the variance of the task graph across time by assuming VU 1 detects 2 objects and VU 2 detects 3 objects. At t_1, the greedy offloading and scheduling decision will offload one of the two ResNet-18 tasks to the VEC server and execute the remaining one locally. As the red outline orange boxes show in Fig. 3.7(a). Then, at t_2, the three ResNet-18 tasks of VU 2 will be scheduled at the VEC server because their predecessor task (i.e. $V_{2,3,1}$) has been offloaded. However, if the decisions made at t_1 are aware of the existence of the three ResNet-18 tasks of VU 2, both the VU 1's ResNet-18 tasks can be scheduled to local execution and hence reduce the end-to-end fusion delay, as shown by Fig. 3.7(b). Therefore, the scheduling and offloading decisions made under task uncertainty are easily non-optimal if they are greedily determined. Moreover, it is not practical to wait until t_2 to make those decisions, as the waiting time will impact the end-to-end fusion delay, especially when the number of VU increases.

Therefore, in this work, we propose a real-time predictive two-step based algorithm,

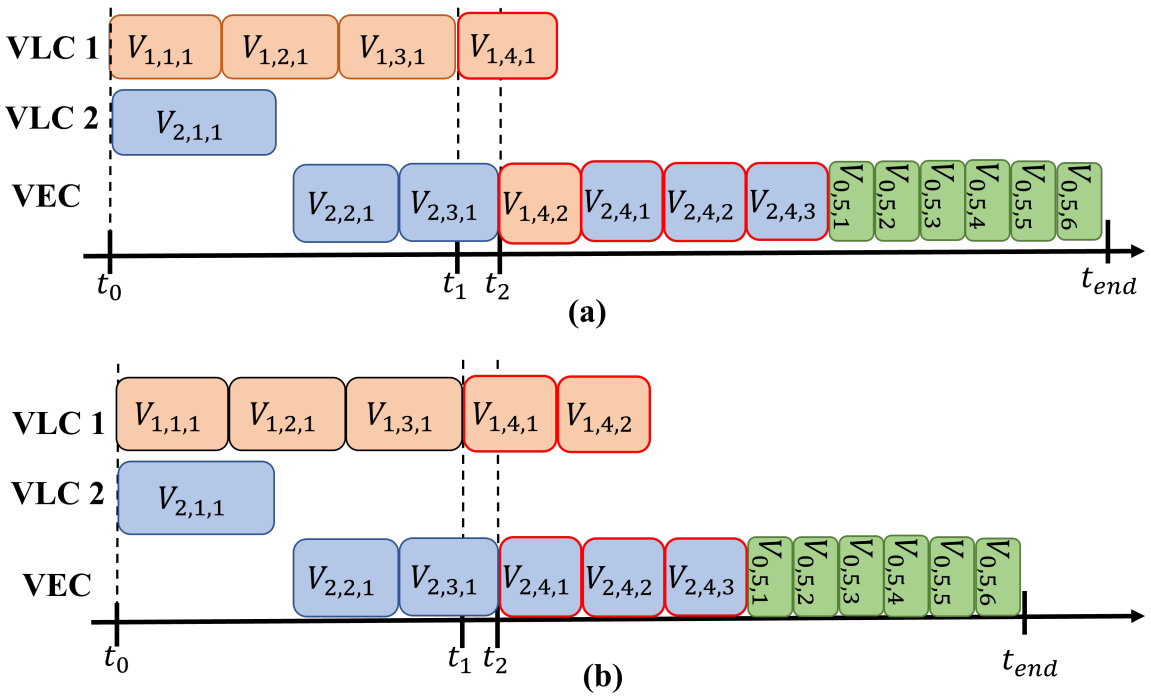

Figure 3.7. An offloading and scheduling timeline example with decisions made (a) without the complete knowledge of $\mathcal{G}$, and (b) with the complete knowledge of $\mathcal{G}$

FOrecasting and unceRtainty-aware Multi-source task Offloading and Scheduling Algorithm (FORMOSA) to solve problem 3.14. The overview of the FORMOSA algorithm is shown in Fig. 3.8. First, Fig. 3.8(a) shows the FORMOSA procedures before a new cycle of multi-vehicle perception fusion application (i.e., before time slot t_0 begins). Each VU will share its VLC capacity and the predicted number of detected objects with MBS (i.e., step ① in Fig. 3.8). In the meantime, RSU will share with MBS the measured uplink channel conditions of each VU. The MEC at MBS will produce a fixed graph $\mathcal{G}'$ based on the predicted number of detected objects of each VU (step ②). Then, we propose a low complexity Dynamic Programming-based heuristic algorithm, namely, *Dynamic programming-based static Multi-source task Offloading and Scheduling Algorithm* (DMOSA), which jointly determines the task partitioning and offloading strategy, the task scheduling plan, and the bandwidth resource allocation to minimize the end-to-end fusion delay of the predicted graph $\mathcal{G}'$ (step ③). Note that DMOSA is executed at MEC server. Hence, it can be processed while the previous instance of

multi-vehicle perception fusion application for the previous time slot is still in execution.

Second, Fig. 3.8(b) demonstrates the FORMOSA procedures when t_0 begins. The determined task partitioning and scheduling plan, and the bandwidth resource allocation will be sent to VUs and RSU (step ④), where the tasks of the multi-vehicle perception fusion application will be executed according to the received task partitioning and scheduling plan (step ⑤). Note that the FORMOSA decision-making is processed and made at MEC server, but the tasks of a multi-vehicle perception fusion application are executed at VUs and RSU. In Fig. 3.8(b)'s example, VU 1's task $V_{1,4,1}$ is scheduled to be executed at VEC. Due to the limited space, we didn't show the examples for VU 2 and VU 3, but the corresponding tasks will be partitioned and scheduled in the meantime.

Finally, in Fig. 3.8(c), we show the FORMOSA procedures whenever an object detection task of a VU is completed and the actual number of detected objects is known. If the predicted number of detected objects is not aligned with the actual number of detected objects, then $\mathcal{G}'$ is mispredicted from $\mathcal{G}$. The actual number of detected objects will be sent to MEC, where the task partitioning and scheduling plan is adjusted accordingly (step ⑥). We propose a real-time offloading and scheduling re-arrangement algorithm, namely *Rectifying Misprediction dynamic Offloading and Scheduling Algorithm* (RMOSA), to make the adjustment (step ⑦). The adjusted task partitioning and scheduling plan will then be sent to VU and RSU (step ⑧), where the task execution will resume based on the new partitioning and scheduling plan (step ⑨). Note that the procedures in Fig. 3.8(c) will take place once for every VU, as long as the object number mis-prediction happens. The whole signaling process of FORMOSA in the temporal domain is described in Fig. 3.9, where an example of two VUs is shown, the color of each signaling procedure and the step numbers align with the signaling color in Fig. 3.8.

Note that the task graph prediction and DMOSA are executed before the time slot begins (i.e. t_0), they can be processed simultaneously with the previous instance of multi-vehicle perception fusion application as they are executed at the MEC server. Therefore, when t_0 begins, the tasks for the current instance of multi-vehicle perception fusion application can be executed

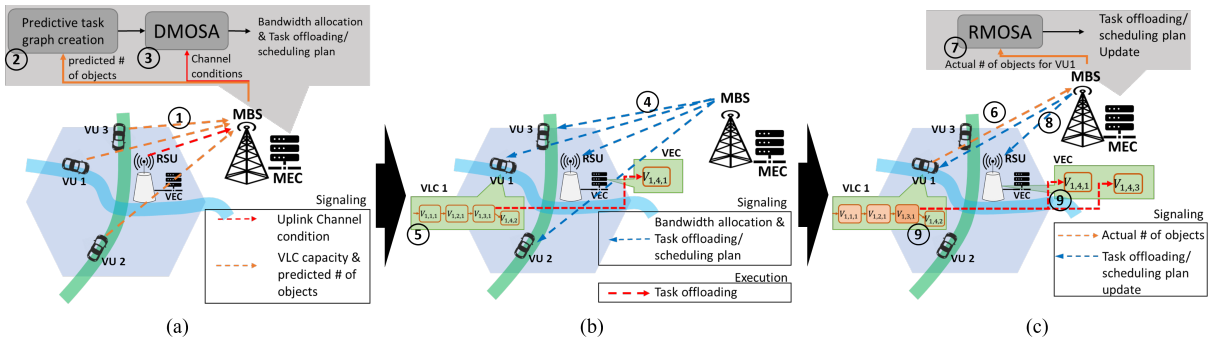

Figure 3.8. Overview of FORMOSA with phases (a) DMOSA determines task partitioning/scheduling plan and bandwidth resource allocation, based on predicted task graph, and (b)VUs and RSU execute the task partitioning/scheduling plan, and (c) RMOSA updates task partitioning/scheduling plan when an object detection task of a VU is completed. (DMOSA and RMOSA are executed at the MEC server, while the multi-vehicle perception fusion application is executed at the VEC and VLC servers)

immediately by VEC and VLC servers by following the DMOSA's task offloading and scheduling decisions. In the simulation result section, we will show that the time complexity of task graph prediction and DMOSA is low enough, such that they can be executed before the time slot begins by assuming that the channel condition will not change during the algorithm execution, and hence, will not affect the optimal solution.

3.5.1 Signaling Cost

Note that the signaling delays induced by FORMOSA procedures are negligible. For example, in Fig. 3.8(a), the VLC capacity and predicted number of objects can be represented by two integers. Uplink channel conditions can be represented by 16-bit floating point values. The corresponding signaling cost is 4 Bytes per VU. In Fig. 3.8(b), there is no additional signaling cost for bandwidth allocation as it is already implemented in existing standards. For each task, its offloading decision is an 8-bit integer and its scheduling plan can be represented by a 16-bit floating point value (either the start time or finish time, or even just the execution order needs to be shared). Even by assuming there are a thousand tasks, that is only 3 KB of downlink data for signaling. Based on our study in [106], these data can be transmitted within 15 ms by using the commercial Cellular-V2X technology, with the lowest possible transmission datarate (i.e.,

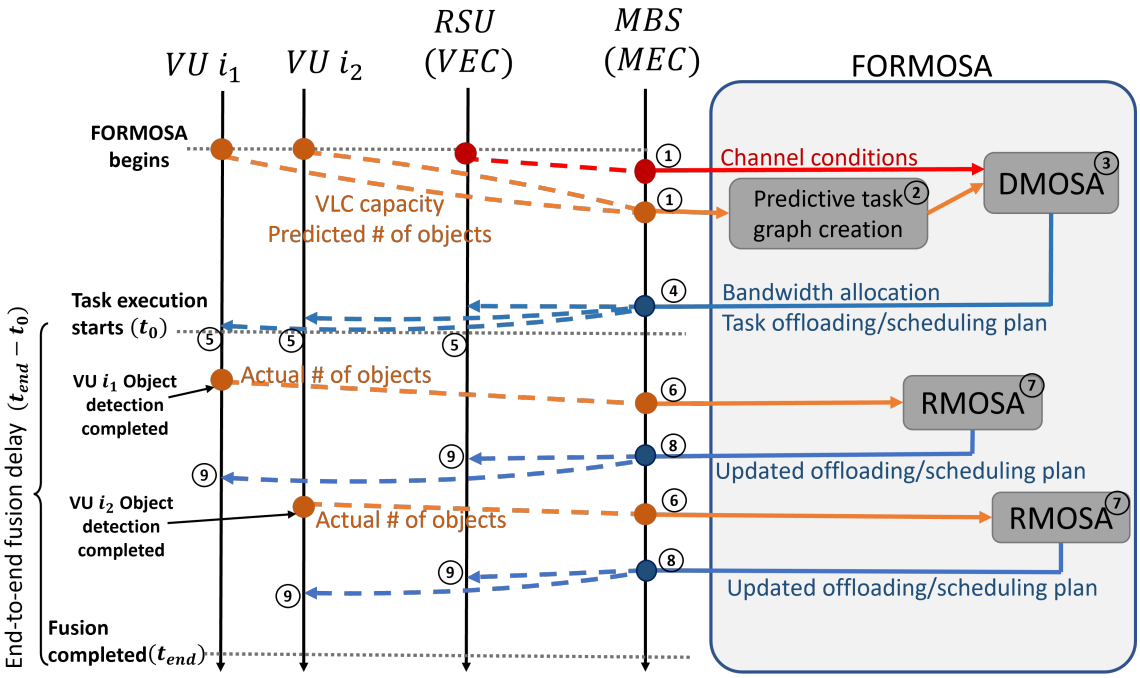

Figure 3.9. Overview of the temporal domain signaling of FORMOSA.

broadcasting with the lowest Modulation Coding Scheme (MCS)). Given that the used unicast downlink datarate is usually higher and the channel capacity between RSU and MBS can be even higher (with a consistent channel), the corresponding signaling delay can be negligible. The signaling delay for procedures in Fig. 3.8(c) is similar to Fig. 3.8(a) and Fig. 3.8(b), therefore, the corresponding discussion is skipped.

3.5.2 Task Graph Prediction

First, we propose to predict the outcome of $(O_1, O_2, ..., O_I)$ to form a predictive task graph for the current time slot. Note that O_i represents the number of detected objects in the captured camera image of VU i. Because vehicle camera is continuously capturing images and by nature, the objects are continuously moving across multiple consecutive images. Therefore, the outcome of O_i for the current time slot can be predicted based on the o_i history from the previous time slots. In this work, we predict the value of O_i as $\hat{o}_i$ by averaging the number of detected objects from the previous 5-time slots.

3.5.3 Dynamic programming-based static Multi-source task Offloading and Scheduling Algorithm (DMOSA)

Once the predicted graph $\mathcal{G}'$ is formulated, we then apply DMOSA algorithm, which is shown in Algorithm 7. DMOSA uses dynamic programming-based heuristic technique to solve problem 3.14 given that $O_i = \hat{o}_i \; \forall i \in \mathcal{I}$.

Task sorting

Firstly, for a given predicted dependency graph $\mathcal{G}'$, DMOSA sorts all the tasks in $\mathcal{G}'$ to derive a task sequence which will be used as the task order when determining the task offloading, scheduling and bandwidth allocation decisions. Based on the discussion in [94], given that $v = V_{i,k,n}$, we first assign to each task v a rank value by the following equation,

$$rank(v) = \frac{T_v^l + T_v^e}{2} + \frac{D(k) * \max(|\psi_v^p|, 1)}{2 * r_{i,b}} + \max_{v' \in \psi_v^s} rank(v'). \tag{3.15}$$

where the first and the second terms are the expected task execution and the input data transmission delay, respectively, by assuming this task has 50% probability to be executed at VLC and VEC servers. The third term is the maximum rank among all this task's immediate successors. Then, we sort all the tasks in $\mathcal{G}'$ by their rank values in descending order. Because of the third term in Eq. 3.15, all the predecessors of a task v will be ordered before v. We denote the sorted task sequence as $\hat{\mathcal{V}}$.

Dynamic programming

First, to control the complexity of DMOSA, we control the bandwidth allocation granularity (i.e. the size of the Smallest Bandwidth Element (BE)). We denote $\hat{B}$ as the maximum number of the BE that can be allocated to a VU, that is, $\hat{B}$ = floor(W_{max}/W), where W is the bandwidth of each BE and W_{max} is the maximum available bandwidth for uplink data transmission. For example, if the bandwidth allocation granularity is 2 PRB (i.e. W = 2*180kHz) for an 18 MHz data channel, then $\hat{B} = 50$.

For a given instance of problem 3.14 with the predicted graph $\mathcal{G}'$ and sorted task sequence as $\hat{\mathcal{V}}$, we consider a matrix $\mathcal{F}$ with dimension $|\hat{\mathcal{V}}| * \hat{B}$, where $|\hat{\mathcal{V}}|$ is the number of all the tasks. $\mathcal{F}(u,\beta)$ represents the minimum of the maximum task completion time among tasks with order 1 to u in $\hat{\mathcal{V}}$ using exactly β amount of BE. The core formula of this dynamic programming strategy is in Eq. 3.16 by assuming $u = V_{i,k,n}$,

$$\mathcal{F}(u,\beta) = \begin{cases} 0; & \text{if } u \leq 0 \\ 0; & \text{if } \beta \leq 0 \\ min(A_{edge}, A_{local}); & \text{otherwise} \end{cases} \tag{3.16}$$

where A_{local} and A_{edge} are the achievable minimal of the maximum task completion time with u and β settings under the condition that task $u = V_{i,k,n}$ is executed at the VLC and VEC servers, respectively. A_{local} can be determined by Eq. 3.17,

$$A_{local} = \max\{T_u^l + L(u-1,\beta,i), \mathcal{F}(u-1,\beta)\} \tag{3.17}$$

where $L(u-1,\beta,i)$ represents the earliest available time of vehicle i's VLC server, considering the task offloading and scheduling plans are determined for the first to the $(u-1)$-th tasks in $\hat{\mathcal{V}}$ while using β BEs. Therefore, $T_u^l + L(u-1,\beta,i)$ is the maximum task completion time for all the tasks among task order 1 to u that are scheduled to be executed at vehicle i's VLC server. Since there may be other VLC or VEC servers that have larger task completion time than vehicle i's VLC server, we determine the value of A_{local} by choosing the maximum between $T_u^l + L(u-1,\beta,i)$ and $\mathcal{F}(u-1,\beta)$, where the latter term records the maximum task completion time among all the servers, except vehicle i's VLC server.

On the other hand, A_{edge} can be calculated by the following equation for assigning task u to VEC server and using bandwidth resource β to serve all the first to u-th tasks,

$$
\begin{aligned}
&A_{edge} = \\
&\min_{b' \leq \beta} \{ \max[T_u^e + \max(L(u-1, \beta - b', 0), L(u-1, \beta - b', i) \\
&\quad + \frac{\sum_{j \in \psi_u^p} (1 - X(u-1, \beta - b', j)) * D(k)}{r_{i,b} * b'}), \mathcal{F}(u-1, \beta - b')] \}
\end{aligned}
\tag{3.18}
$$

Firstly, for A_{edge}, since task $u = V_{i,k,n}$ is assigned to VEC server, it can only be executed after i) when the VEC server completes executing any tasks between the first to the $(u-1)$th tasks, and ii) when all the task u's inputs are transmitted or available at the VEC server. These two requirements are involved in the last *max* function in Eq. 3.18. For the first requirement, we denote $L(u-1, \beta - b', 0)$ as the earliest available time of VEC server after the first to the $(u-1)$-th tasks are executed based on the corresponding offloading plan and $\beta - b'$ BEs are used. Similarly, we denote the same for vehicle i's VLC server as $L(u-1, \beta - b', i)$. If any of the task u's predecessors is executed at the VLC server, then the delay for transmitting its output to the VEC server needs to be considered, as the second term in the last *max* function shows.

Secondly, since there may be other VLC servers that have larger task completion time than the VEC server, we use the first *max* function in Eq. 3.18 to record the maximum among them. Eventually, the value of A_{edge} is chosen after all the possible number of BEs $0 < b' \leq \beta$ are evaluated. We choose the optimal b'^* as the number of BEs that are allocated to vehicle i (i.e., VU i) for task u's offloading.

The steps of DMOSA is shown in Algorithm 7. Steps 1 and 2 are for task sorting. Steps 3 to 18 are the dynamic programming process, where $\mathcal{W}$ records the number of BEs that are being allocated to each VU and is calculated as the summation of BEs that are assigned to any tasks of VU i (i.e. $\forall 1 \leq v \leq u$ s.t. $v = V_{i,k,n}$ for any possible k, n). On the other hand, we use matrix X to record the corresponding task offloading strategy for all the tasks in $\mathcal{V}'$ and we use matrix P to indicate whether this task can only be executed at the VEC server (i.e. at least one of its predecessor tasks is offloaded to the VEC server). Note that in steps 10, 12, 14,

the "$X \leftarrow (condition)?Y : Z$" statement means that if $condition$ is true, X will be assigned with the value of Y, otherwise, X will be assigned with the value of Z. DMOSA then recursively calculates the elements in $\mathcal{F}$ for task v from 1 to $|\hat{\mathcal{V}}|$ and bandwidth elements b' from 1 to $B-I$, where I is the total number of vehicles. Note that we reserve 1 BE for each vehicle to guarantee their minimum uplink transmission capabilities. Finally, based on the task partitioning and offloading results and available bandwidth resources for each VU, in steps 22 to 34, DMOSA heuristically determines for each task u the start and finish time, S_u and F_u, following the order in the sorted task sequence $\mathcal{V}'$.

3.5.4 Rectifying Misprediction dynamic Offloading and Scheduling Algorithm (RMOSA)

Because $\mathcal{G}'$ and the decisions of DMOSA are determined based on the prediction $O_i = \hat{o}_i$, the prediction deviation needs be accommodated once a SSD-MobileNetV2 task of a vehicle (i.e. $v = V_{i,3,1}$ for VU i) has been executed. We then propose the *Rectifying Misprediction dynamic Offloading and Scheduling Algorithm* (RMOSA) algorithm to adjust the current task offloading and scheduling decisions in real time. The steps of RMOSA are listed in Algorithm 8. If the predicted value is less than the actual number of detected objects for VU i (i.e. $\hat{o}_i < o_i$), RMOSA will execute the $underprediction()$ function to add the extra tasks, the new ResNet-18s and the following FC layers, that are created for the additional detected objects to the current task offloading and scheduling plan. On the contrary, if $\hat{o}_i > o_i$, RMOSA will execute the $overprediction()$ function to remove the tasks, the redundant ResNet-18s and their following FC layers, that are created previously in $\mathcal{G}'$ for the objects that do not exist, from the current task offloading and scheduling plan. The new task offloading and scheduling plan is then sent to RSU and VUs, as the lower part of Fig. 3.9 shows.

Algorithm 7: Dynamic-programming-based static Multi-source task-Offloading & Scheduling Algorithm (DMOSA)

Input: $\mathcal{G}', \mathcal{V}', T_v^e, T_v^l \forall v \in \mathcal{V}, r_{b,i}, c_i^l \forall i \in \mathcal{I}, c^e, D(k) \forall k$
Output: $x_v, S_v, F_v \forall v \in \mathcal{V}, B_i \forall i \in \mathcal{I}$

calculate rank for$\mathcal{V}$ by Eq. 3.15;
$\hat{\mathcal{V}} \leftarrow$ sort all tasks in $\mathcal{V}$ by rank ;
initialize $\mathcal{F}, L, W, X, P$;
for $j \leftarrow 1$ **to** $B - I$ **do**
  for $u = V_{i,k,n} \in \hat{\mathcal{V}}$ **do**
    $A_{local}, A_{edge} \leftarrow \infty$;
    if $P(u-1, j, u) = 0$ **then**
      calculateA_{local} by Eq. 3.17
    calculateA_{edge} by Eq. 3.18, record Q^* be the minimum of the first term in Eq. 3.18
    and b'^* is the optimal b' ;
    $\hat{b}' \leftarrow (A_{local} \leq A_{edge})?0 : b'^*$;
    $X(u, j, :) \leftarrow X(u-1, j-\hat{b}', :)$;
    $X(u, j, u) \leftarrow (A_{local} \leq A_{edge})?i : 0$;
    $L(u, j, :) \leftarrow L(u-1, j-\hat{b}', :)$;
    $L(u, j, i) \leftarrow (A_{local} \leq A_{edge})?L(u-1, j, i) + T_v^l(c_i^l) : Q^*$;
    $W(u, j, :) \leftarrow W(u-1, j-\hat{b}', :)$;
    $W(u, j, u) \leftarrow W(u, j, u) + \hat{b}'$;
    $P(u, j, :) \leftarrow P(u-1, j-\hat{b}', :)$;
    If $A_{local} \leq A_{edge}$: $P(u, j, q) \leftarrow 1 \forall q \in \psi_u^s$
$x_v \leftarrow X(|\hat{\mathcal{V}}|, B-I, v) \forall v \in \hat{\mathcal{V}}$;
$b_i \leftarrow W(|\hat{\mathcal{V}}|, B-I, i) \forall i \in \mathcal{I}$;
$L' \leftarrow zeros(I+1), T_F \leftarrow zeros(I+1)$;
for $u = V_{i,k,n} \in \hat{\mathcal{V}}$ **do**
  if $x_u > 0$ **then**
    $S_u = L'(x_u)$, $F_u = S_u + T_u^l(c_u^l)$;
    $L'(x_u) = F_u$;
  else
    $d_{ready} = 0$;
    for $q = V_{i',k',n'} \in \psi_u^p$ **do**
      if $x_q > 0$ **then**
        $T_F(k') \leftarrow \frac{Dk}{r_{i,b} b_i} + max(T_F(k'), F_q)$;
        $d_{ready} = max(d_{ready}, T_F(k'))$;
        $x_q \leftarrow 0$;
    $S_u = max(L'(x_u), d_{ready})$, $F_u = S_u + T_u^e(c^e)$;
    $L'(x_u) = F_u$;
return $x_v, S_v, F_v \forall v \in \mathcal{V}, B_i \forall i \in \mathcal{I}$;

Algorithm 8: Rectifying Misprediction dynamic Offloading and Scheduling Algorithm (RMOSA)

Input: $x_v, S_v, F_v \forall v \in \mathcal{V}, B_i \forall i \in \mathcal{I}, O_i = o_i$
Output: $x_v, S_v, F_v \forall v \in \mathcal{V}$
1 **if** $o_i > \hat{o}_i$ **then**
2 $\quad x_v, S_v, F_v \forall v \in \mathcal{V} \leftarrow$ underprediction();
3 **if** $o_i < \hat{o}_i$ **then**
4 $\quad x_v, S_v, F_v \forall v \in \mathcal{V} \leftarrow$ overprediction();
5 return $x_v, S_v, F_v \forall v \in \mathcal{V}$

Underprediction

The steps of $underprediction()$ function are listed in Algorithm 9. First, we denote the set of timelines on the VEC server that has not been scheduled to execute any tasks as Θ, which is defined as the free schedule slot set. Θ is in the (Θ_s, Θ_f) format with $\theta_s \in \Theta_s$ represents the start time of the free schedule slot θ and $\theta_f \in \Theta_f$ represents the end time of the free schedule slot θ. Note that we will remove any free schedule slot θ in Θ which has $\theta_f < S_{v_m}$, where $v_m = V_{i,3,1}$, as these schedule slots are no longer available for task scheduling.

For each additional object j detected by VU i, we create a new ResNet-18 task $V_{i,4,\hat{o}_i+j}$ in step 5. Then, we first check if there is any free schedule slot θ^* which has enough duration that $V_{i,4,\hat{o}_i+j}$ can be scheduled for execution in steps 6 and 7. Scheduling $V_{i,4,\hat{o}_i+j}$ to any schedule slot in Θ will not increase end-to-end fusion delay. Note that end-to-end fusion delay is defined as the maximum finish time of all the tasks. Since there exists a task v_k which has a larger start time than $S_{V_{i,4,\hat{o}_i+j}}$ (i.e., the beginning of this free schedule slot θ_s^*), based on lemma 3.5.1, $F_{v_k} > S_{v_k} > F_{V_{i,4,\hat{o}_i+j}}$. End-to-end fusion delay must be greater or equal to F_{v_k}, and hence will not be impacted.

Lemma 3.5.1. *Given two tasks v_a and v_b running on the same server, if $S_{v_a} > S_{v_b}$, then $S_{v_a} \geq F_{v_b}$. Where S_{v_i} is the scheduled start time of v_i, and F_{v_i} is the scheduled finish time of v_i.*

Proof. Since we assume a server can only run one task at the same time, if $S_{v_a} < F_{v_b}$ when $S_{v_a} > S_{v_b}$, then the server needs to execute v_a and v_b simultaneously from time S_{v_a} to F_{v_b}, contradicts the assumption above. □

Algorithm 9: Underprediction

Input: $x_v, S_v, F_v \forall v \in \mathcal{V}, B_i \forall i \in \mathcal{I}, \hat{o}_i, O_i = o_i, \Theta = (\Theta_s, \Theta_f)$
Output: $x_v, S_v, F_v \forall v \in \mathcal{V}$

1 let $v_m = V_{i,3,1}$;
2 $LF \leftarrow \max_{v \text{ s.t. } x_v = i} F_v$;
3 $EF \leftarrow \max(F_{v_m} + \frac{D(4)}{r_{i,b} * B_i}, \max_{v \text{ s.t. } x_v = 0} F_v)$;
4 **for** $j \leftarrow 1$ **to** $o_i - \hat{o}_i$ **do**
5 create a task $v' = V_{i,4,\hat{o}_i + j}$;
6 **if** $\exists\ \theta^* \in \Theta$ *s.t.* $\theta^*_f - \theta^*_s > T^e_{v'}$ **then**
7 $x_{v'} \leftarrow 0$, $S_{v'} \leftarrow \theta^*_s$, $F_{v'} \leftarrow \theta^*_s + T^e_{v'}$, $\theta^*_s \leftarrow F_{v'}$;
8 **else**
9 **if** x_{v_m} *is* 0 **then**
10 $EF \leftarrow EF + T^e_{v'}$;
11 **if** $EF - max_{v \text{ s.t. } x_v = 0} F_v > \varepsilon$ **then**
12 $\Theta = \Theta \cup (\max_{v \text{ s.t. } x_v = 0} F_v, EF)$
13 $x_{v'} \leftarrow 0$, $S_{v'} \leftarrow EF - T^e_{v'}$, $F_{v'} \leftarrow EF$;
14 **else**
15 **if** $LF + T^l_{v'} < EF + T^e_{v'}$ **then**
16 $x_{v'} \leftarrow i$, $S_{v'} \leftarrow LF$, $F_{v'} \leftarrow LF + T^l_{v'}$;
17 $LF \leftarrow LF + T^l_{v'}$;
18 **else**
19 $EF \leftarrow EF + T^e_{v'}$;
20 **if** $EF - max_{v \text{ s.t. } x_v = 0} F_v > \varepsilon$ **then**
21 $\Theta = \Theta \cup (\max_{v \text{ s.t. } x_v = 0} F_v, EF)$
22 $x_{v'} \leftarrow 0$, $S_{v'} \leftarrow EF - T^e_{v'}$, $F_{v'} \leftarrow EF$;
23 $\mathcal{V} \leftarrow \{\mathcal{V} \cup v'\}$, $\mathcal{E} \leftarrow \{\mathcal{E} \cup (v_m, v')\}$ s.t. $v_m = V_{i,3,1}$;
24 $EF \leftarrow \max_{v \text{ s.t. } x_v = 0} F_v$;
25 First return: $x_v, S_v, F_v\ \forall v \in \mathcal{V}$;
26 **for** $j \leftarrow 1$ **to** $o_i - \hat{o}_i$ **do**
27 **for** $i' \in \mathcal{I} \setminus i$ **do**
28 **for** $o \leftarrow 1$ *to* o_i **do**
29 create a task $v' = V_{0,5,(o + \max_{v = V_{0,5,n} \in \mathcal{V}} n)}$;
30 $x_{v'} \leftarrow 0$, $S_{v'} \leftarrow EF$, $F_{v'} \leftarrow EF + T^e_{v'}$;
31 $EF \leftarrow EF + T^e_{v'}$;
32 $\mathcal{V} \leftarrow \{\mathcal{V} \cup v'\}$, $\mathcal{E} \leftarrow \{\mathcal{E} \cup (v_m, v')\}$ s.t. $v_m = V_{i,3,1}$;
33 Second return: $x_v, S_v, F_v\ \forall$ new created $v = V_{0,5,n}$;

If such free schedule slot in Θ doesn't exists, the schedule and offloading solution of $V_{i,4,\hat{o}_i+j}$ depends on where its predecessor v_m is executed. If v_m is offloaded to the VEC server, in steps 10-13, we will schedule $V_{i,4,\hat{o}_i+j}$ to the end of VEC server's task schedule. If v_m is executed locally, in steps 15-22, we will compare the expected finish time of $V_{i,4,\hat{o}_i+j}$ at VLC and VEC servers and assign it to the optimal server and update Θ correspondingly. That is,

$$x_{V_{i,4,\hat{o}_i+j}} = \begin{cases} i & \text{if } LF + T^l_{V_{i,4,\hat{o}_i+j}} < EF + T^e_{V_{i,4,\hat{o}_i+j}} \text{ ,} \\ 0 & \text{otherwise .} \end{cases} \tag{3.19}$$

where $LF = \max_{v \text{ s.t. } x_v=i} F_v$ is the current end time of the VLC server i's task schedule and $EF = \max(F_{v_m} + \frac{D(4)}{r_{i,b} * B_i}, \max_{v \text{ s.t. } x_v=0} F_v))$ is the maximum among the current end time of the VEC server's task schedule and the earliest time that the input of this Resnet-18 $V_{i,4,\hat{o}_i+j}$ (i.e., the image of the detected object) is available at the VEC server.

After all the new ResNet-18 tasks are scheduled, in steps 26 to 32, Algorithm 9 adds extra FC layers for the pairs of each extra detected object and each existing detected object of other VUs. These new FC layers are scheduled to the end of the VEC server's current task schedule.

Note that all the servers will stop executing after a v_m (i.e.$V_{i,3,1}$) is completed, until receiving the new task offloading and scheduling plan. Hence, the time complexity of RMOSA will impact the end-to-end fusion delay. To minimize the end-to-end fusion delay, we propose to separate the last step of $underprediction()$ into two phases. The first phase is steps 4-24 and returns the new task offloading and scheduling plan for the current tasks and the new created ResNet-18s. While the first phase returns the new plan and the servers resume task execution, MEC server calculates the second phase and returns the task scheduling plan for the new created FC layer tasks.

Overprediction

The steps of $overprediction()$ are listed in Algorithm 10. We denote Ω as the set of tasks that are going to be removed from $\mathcal{G}'$ and the current task scheduling plan. To optimally choose $\hat{o}_i - o_i$ ResNet-18 tasks for VU i from $\mathcal{G}'$ for removing, we first consider the following two lemmas,

Lemma 3.5.2. *Given a task graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a task offloading and scheduling plan $x_v, S_v, F_v \forall v \in \mathcal{V}$ for a multi-vehicle perception fusion application, the schedule plan for an object feature extraction task (i.e., Resnet-18) v_a and v_a's successor tasks (i.e., FC layers) can be*

exchanged with the schedule plan for another object feature extraction task v_b and v_b's successor tasks when v_a and v_b share the same predecessor object detection task (i.e., SSD-MobileNetV2).

Proof. Note that v_a and v_b are object feature extraction tasks, they have only one predecessor task, as shown in Fig. 3.5. If v_a and v_b share the same predecessor object detection task, then $S_{v_a} \geq F_{v_m}$ and $S_{v_b} \geq F_{v_m}$, where v_m is the common predecessor of v_a and v_b. Therefore, we can exchange values of S_{v_a} (and F_{v_a}) with S_{v_b} (and F_{v_b}) without breaking the task dependency to v_a and v_b's predecessor task. On the other hand, each v_a's successor task v_a^s has two predecessors, v_a and an object feature extraction task v_c (e.g. Resnet-18) from another VU j, as shown in Fig. 3.5. Therefore, $S_{v_a^s} \geq F_{v_a}$ and $S_{v_a^s} \geq F_{v_c}$. On the other hand, there exists one of v_b's successor tasks v_b^s that shares the same v_c as its predecessor. Thus, we have $S_{v_b^s} \geq F_{v_b}$ and $S_{v_b^s} \geq F_{v_c}$. Under the condition that F_{v_a} is exchanged with F_{v_b}, we can swap the value of $S_{v_a^s}$ and $S_{v_b^s}$ while satisfying the above two equations. Therefore, v_a and v_a's successor tasks' schedule plan can be exchanged with v_b and v_b's successor tasks' schedule plan. □

Lemma 3.5.3. *Assuming tasks v_a and v_b are the same type and scheduled to be executed on the same server and $F_{v_a} < F_{v_b}$, if a task v_d can be rescheduled to the schedule slot (S_{v_a}, F_{v_a}) that is originally scheduled to execute task v_a, then v_d can be reassigned to slot (S_{v_b}, F_{v_b}), too.*

Proof. If a task v_d can be rescheduled to the schedule slot (S_{v_a}, F_{v_a}) that is originally scheduled to execute task v_a, then all v_d's predecessors must be completed before F_{v_a} and v_d's inputs must be ready before F_{v_a}. Note that $S_{v_b} \geq F_{v_a}$ because of $S_{v_a} < S_{v_b}$ and lemma 3.5.1. It guarantees that all v_d's predecessors and inputs are completed and ready, respectively, before S_{v_b}. Since v_a and v_b are of the same type, the schedule slot (S_{v_b}, F_{v_b}) has the same duration as (S_{v_b}, F_{v_b}). Therefore, v_d can be rescheduled to the schedule slot (S_{v_b}, F_{v_b}). □

According to lemma 3.5.2, removing any ResNet-18 task of VU i can result in the same free schedule slots θs, by swapping another ResNet-18 task of VU i to the schedule slots of the removed ResNet-18 task. On the other hand, based on lemma 3.5.3, removing the task which has

a larger finish time than the other same type of task creates more flexibility for rescheduling the remaining tasks. Therefore, in Algorithm 10, we will first iteratively pick the ResNet-18 tasks which have the maximum finish time, along with their successor FC layer tasks, for removing.

Then we append this ResNet-18 task and its immediate successor FC layers to Ω. Once all the extra ResNet-18 tasks and the corresponding FC layers are added to Ω in steps 3 to 8, we remove them from $\mathcal{G}'$ and the current task offloading and scheduling plan, and update the free schedule slot set Θ accordingly in steps 9 to 11. Finally, we examine all the remaining tasks on whether their start and end time can be shifted forward according to the new Θ in steps 13 to 26. Similar to $underprediction()$ function, we divide $overprediction()$ into two phases. For the first phase, steps 13 to 19, we will only update the schedule of tasks whose start time is within ω duration after F_{v_m}. ω is carefully chosen so that the first phase will update task scheduling for only a limited set of tasks. While the new decisions of task schedule is transmitted to the VLC and VEC servers, which then resume the execution of tasks, RMOSA will calculate for the remaining tasks the new schedule in steps 20 to 26 for the second phase.

3.5.5 Complexity Analysis

Note that in DMOSA, the variables j and u need to iterate $B-I$ and $|\mathcal{V}|$ times for the for loop begins from steps 4 to 18, respectively. Within the for loop, step 9 will check at most $B-I$ possible bandwidth allocations. Therefore, the complexity of DMOSA is $O((B-I)^2 * |\mathcal{V}|)$. To further reduce the complexity of DMOSA, note that FC layers will always be executed at the VEC server in the considered edge-based multi-vehicle perception fusion system. Therefore, in step 5, we can iterate variable u for tasks in $|\mathcal{V}|$ without iterating u for all the tasks $u = V_{i,k,n}$ where $k = 5$.

The complexity of RMOSA depends on the complexity of $underprediction()$ and $overprediction()$. In the first phase of $underprediction()$, the j variable iterates from 1 to $o_i - \hat{o}_i$ and step 6 takes $|\Theta|$ times in worst case. Note that $|\Theta|$ is limited by $|\mathcal{V}|$ and o_i is very small compared to $|\mathcal{V}|$. Therefore, first phase complexity is $O(|\mathcal{V}|)$. For the second phase,

Algorithm 10: Overprediction

Input: $x_v, S_v, F_v \forall v \in \mathcal{V}, B_i \forall i \in \mathcal{I}, \hat{o}_i, O_i = o_i, \Theta = (\Theta_s, \Theta_f)$
Output: $x_v, S_v, F_v \forall v \in \mathcal{V}$

1 let $v_m = V_{i,3,1}$;
2 $\Omega \leftarrow$ list() ;
3 **while** $\hat{o}_i > o_i$ **do**
4 $\quad o_i \leftarrow o_i + 1$;
5 $\quad v^* \leftarrow \max_{v=V(i,4,n) \forall n} F_v$;
6 $\quad$ **if** x_{v^*} *is* 0 **then**
7 $\quad\quad \Theta \leftarrow \Theta \cup (S_{v^*}, F_{v^*})$;
8 $\quad \Omega \leftarrow \Omega \cup \{v^*\} \cup \psi^s_{v^*}$;
9 $\quad \Theta \leftarrow \Theta \cup \{(S_{v'}, F_{v'}) \forall v' \in \psi^s_{v^*}\}$;
10 $\mathcal{V} \leftarrow \mathcal{V} \setminus \Omega$;
11 $\mathcal{E} \leftarrow \mathcal{E} \setminus$ all the edges correspond to Ω;
12 sort Θ by Θ_s in ascending order ;
13 **for** $v' \in \mathcal{V}$ *s.t.* x'_v *is* 0 *and* $S'_v - S_{v_m} \leq \omega$ **do**
14 $\quad$ **for** $\theta \in \Theta$ **do**
15 $\quad\quad$ **if** $\theta_s > S_{v'}$ **then**
16 $\quad\quad\quad$ break;
17 $\quad\quad$ **if** $\theta_f - \theta_s > T^e_{v'}$ **then**
18 $\quad\quad\quad S_{v'} \leftarrow \theta_s, F_{v'} \leftarrow \theta_s + T^e_{v'}, \theta_s \leftarrow F_{v'}$;
19 First return: $x_{v'}, S_{v'}, F_{v'} \forall v' \in \mathcal{V}$ s.t. $S_{v'} - S_{v_m} \leq \omega$;
20 **for** $v' \in \mathcal{V}$ *s.t.* x'_v *is* 0 *and* $S'_v > S_{v_m} + \omega$ **do**
21 $\quad$ **for** $\theta \in \Theta$ **do**
22 $\quad\quad$ **if** $\theta_s > S_{v'}$ **then**
23 $\quad\quad\quad$ break;
24 $\quad\quad$ **if** $\theta_f - \theta_s > T^e_{v'}$ **then**
25 $\quad\quad\quad S_{v'} \leftarrow \theta_s, F_{v'} \leftarrow \theta_s + T^e_{v'}, \theta_s \leftarrow F_{v'}$;
26 Second return: $x_{v'}, S_{v'}, F_{v'} \forall v' \in \mathcal{V}$ s.t. $S_{v'} > S_{v_m} + \omega$

assuming O_N is the maximum number of detected objects of a single VU, its complexity is defined by the for loop in steps 24 to 30, which is $O(O_N^2 * I)$ in worst case.

On the other hand, the complexity of the first phase of $overprediction()$ includes the while loop in steps 3 to 9 and for loop in steps 13 to 18. Because step 5 takes $O(1)$ if F_v are sorted for $v = V_{i,4,n} \forall n$, the while loop iterates $o_i - \hat{o}_i$ times. Also, the ω parameter in step 13 limits the for loop in steps 13 to 18 to constant iterations. Therefore, its first phase complexity is $O(|\mathcal{V}|)$ because of the for loop complexity in step 14. Similarly, the second phase complexity is $O(|\mathcal{V}|^2)$.

3.6 Performance Evaluation

In this section, we first show the performance evaluation of DMOSA and RMOSA separately. Then we demonstrate the performance comparison of FORMOSA with existing approaches using real-world trace-driven simulations.

3.6.1 DMOSA and RMOSA Performance Evaluation

Herein, we first analyze the performance of DMOSA under different bandwidth allocation granularities. Then we demonstrate the gain of using RMOSA to adjust the task scheduling plan. Both DMOSA and RMOSA are implemented by using Python. In both experiments, we assume each vehicle has 50% probability to possess a configuration index 1 (defined in TABLE 3.3) VLC server and another 50% probability for a configuration index 2 VLC server. The configuration index for VEC server is 6. DMOSA and RMOSA are executed on a MEC server, whose computing capacity is emulated by a computing server that has 20 CPU cores (Intel 12-th Gen i7 cores) with 2.1 GHz CPU frequency and 16 GB RAM. The SNR, $\eta_{b,i}$, is randomly generated by $\mathcal{U}(0,40)$ dB for each VU i. o_i is randomly generated by $\mathcal{U}(1,8)$ and $o_i - \hat{o}_i$ is randomly generated by by $\mathcal{U}(-4,4)$. The total bandwidth is 20 MHz. The experiment results are generated after 500 times of simulation.

Fig. 3.10a demonstrates the end-to-end fusion delay performance of DMOSA under multiple number of VUs scenarios by using different bandwidth allocation granularities. For example, the blue curve shows the average end-to-end fusion delay performance of DMOSA when BE, the Smallest Bandwidth Element, is 2 PRBs. While smaller granularity leads to lower end-to-end fusion delay because of a more fine-grained bandwidth resource allocation, the corresponding time complexity for DMOSA to search for the optimal solution is higher. As Fig. 3.10b shows, DMOSA takes more than 400 ms to calculate the solution with 2 PRBs bandwidth allocation granularity. Note that in Fig. 3.9, DMOSA calculates for the task offloading, scheduling, and bandwidth allocation decisions before the beginning of the execution of the

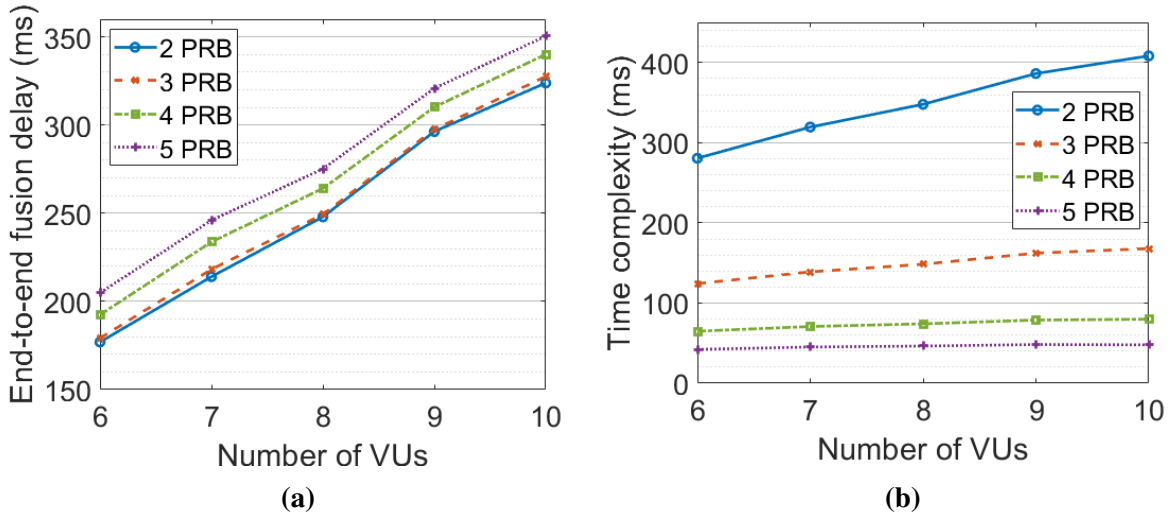

Figure 3.10. The performance of DMOSA by using different bandwidth allocation granularities on (a). end-to-end fusion delay, and (b). algorithm time complexity

multi-vehicle perception fusion application. To guarantee that DMOSA's decision is optimal, the channel conditions should remain the same during the execution of DMOSA. In one of our previous work [88], we have shown that it is practical to assume the channel condition will not change for a vehicular wireless channel within a 250 ms time slot. Therefore, in the rest of the experiments, we will operate DMOSA with 3 PRBs bandwidth allocation granularity (i.e. BE is 3 PRBs), which takes less than 200 ms to compute the decisions for 10 VUs as shown in Fig. 3.10b.

Fig. 3.11a demonstrates the benefit of using RMOSA to minimize the impact of misprediction of the number of detected objects for each VU on the end-to-end fusion delay. Without RMOSA, the end-to-end fusion delay values with just DMOSA are 19%, 13%, and 8% larger than the performances with FORMOSA when the number of VUs are 6, 8, and 10, respectively. Fig. 3.11b shows that by enabling two-phase response in RMOSA, the first response time of RMOSA can be reduced to less than 1 ms. That means, all the servers need to wait less than 1 ms until they receive the new task offloading and scheduling decisions once a VU completes its SSD-MobileNetV2 task.

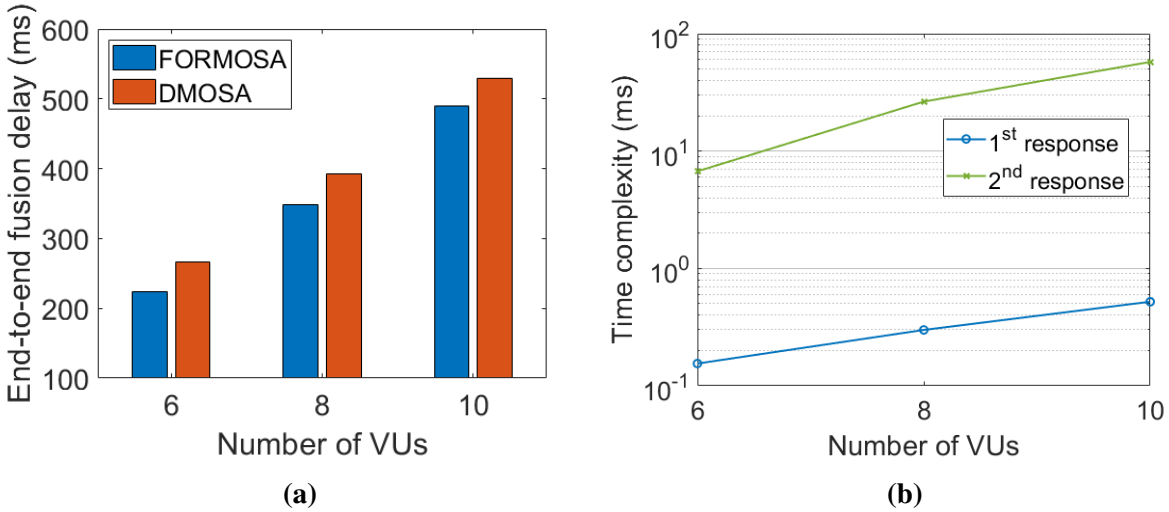

Figure 3.11. (a) The performance comparison between FORMOSA and DMOSA (i.e. with and without RMOSA), and (b). RMOSA's first and second response time complexity

3.6.2 FORMOSA Performance Evaluation

Next, we evaluate the performance of FORMOSA by comparing it with three other relevant algorithms, CEFO [94], PCSO [99], and Last Step Edge Offloading (LSEO).

CEFO algorithm: In [94], the Centralized Earliest-Finish time Offloading (CEFO) algorithm first assigns a computing and communication cost-related priority value to each task. Tasks are sorted by the priority values and are greedily determined to be offloaded to the server which provides the earliest possible finish time, following the sorted task order.

PCSO algorithm: Instead of assigning priority values to each task, the authors in [99] propose dynamic Priority-based Computation Scheduling and Offloading (PCSO) algorithm and classify tasks into high, medium, and low priority groups based on the current tasks that are being executed and their dependencies. They then propose a knapsack-based algorithm and a depth as well as data size-based heuristic algorithm to determine task offloading and scheduling strategies for the high and medium priority groups, respectively.

LSEO algorithm: The Last Step Edge Offloading (LSEO) algorithm is created in this study to provide comparisons with the scenario without task offloading. Under this algorithm, tasks that are created by VUs (i.e. $k \leq 4$) will be executed locally and only the FC layers for fusion are

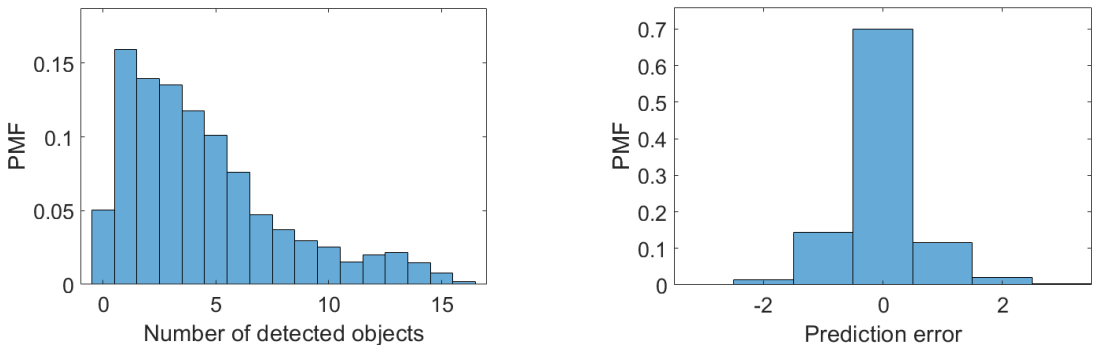

Figure 3.12. The probability mass function of (a) the number of detected objects in each frame over the dataset traces, and (b) prediction error

executed at the edge.

To be more realistic, we use the object detection and tracking trace data from the KITTI dataset [107, 108] to generate the value of o_i. Also, we use the method proposed in Section 3.5.2 to predict the number of detected objects on each trace. Fig. 3.12a shows the probability mass function (PMF) of the number of objects detected in each frame over the 21 traces and 8008 frames in the above object detection and tracking trace dataset. Fig. 3.12b shows the corresponding prediction error, namely $o_i - \hat{o}_i$, in terms of the number of objects. Therefore, we randomly generate the value of o_i and $\hat{o}_i$ by using the distributions shown in Fig. 3.12. VLC server configuration and SNR assumptions are the same as Section 3.6.1. FORMOSA will take $\hat{o}_i$, and SNR of all the VUs as input, then generate the task graph and the corresponding task execution schedule. The schedule is generated considering each VLC and VEC's capacity and VUs' uplink data rate. The schedule for tasks before SSD-MobileNetV2 is fixed for each VU. But the remaining tasks' schedule will be adjusted once for each VU, by RMOSA with the actual number of objects o_i. We will first adjust the VU with the earliest SSD-MobileNetV2 finish time and vice versa. The execution delay of RMOSA first response is recorded and will be added to the resulting end-to-end fusion delay. Finally, the end-to-end delay is calculated as the finish time of the last FC layer plus the RMOSA execution delay.

Impact of VU density : We first compare FORMOSA with the other three algorithms under

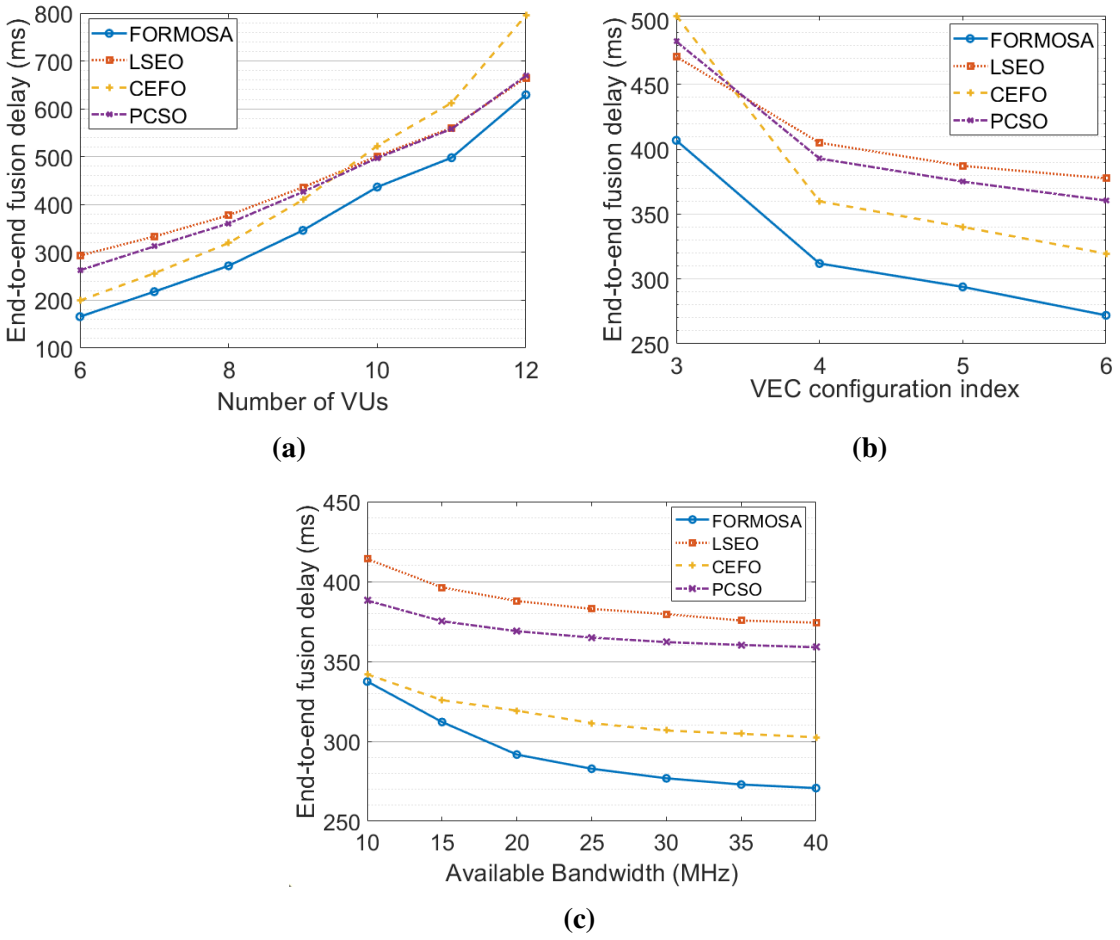

Figure 3.13. End-to-end fusion delay under different scenarios of (a) VU densities, (b) VEC server capacities, and (c) available bandwidth resources

different VU densities in Fig. 3.13a. In this experiment, the VEC configuration index is set to 6 and bandwidth is 40 MHz. The number of VUs in the network are controlled from 6 to 12 and we average the end-to-end fusion delay over 500 times of simulation for each VU density case. Over all the tested cases, FORMOSA outperforms other algorithms. For example, the end-to-end fusion delay by using FORMOSA is 15%, 24%, and 28% lower than the delay by using CEFO, PCSO, and LSEO, respectively, when there are 8 VUs. Moreover, when VU density increases, the performance gap between CEFO and FORMOSA escalates from 47.6 ms(15%) to 115 ms(19%) when the number of VUs grows from 8 to 11. This is because when CEFO determines an offload decision for a task, it greedily compares the task completion time of this

task at VEC and VLC servers and doesn't consider the impact of the successor tasks to the end-to-end fusion delay. When the VEC's capacity is higher than the VLC server, CEFO tends to offload tasks to the edge. However, this will put higher burden on the VEC server and piles up the end-to-end fusion delay when VU increases. Fig. 3.13a also shows that FORMOSA can keep the average end-to-end fusion delay to less than 500 ms when the number of VUs is no more than 10. In Section 3.6.3, we will demonstrate how FORMOSA can support larger numbers of VUs in a practical scenario.

Impact of VEC computing capacity: In this experiment, we vary the VEC configuration index from 3 to 6 and keep the number of VU and bandwidth to 8 and 40 MHz, respectively. The results are shown in Fig. 3.13b. FORMOSA outperforms other algorithms across all the tested VEC configuration indices. For example, the end-to-end fusion delay by using FORMOSA is 19%, 16%, and 14% lower than the delay by using CEFO, PCSO, and LSEO, respectively, when the VEC configuration index is 3. While CEFO has the closest delay performance to FORMOSA when VEC configuration index ≥ 4, it performs the worst when the VEC configuration index is 3. This matches the observation in the previous experiment about the disadvantage of CEFO when the VEC server has higher burden.

Impact of available bandwidth: In Fig. 3.13c, we compare FORMOSA's end-to-end fusion delay performance with the other three algorithms under different bandwidth availabilities. The X-axis shows the different available bandwidth from 10 MHz to 40 MHz. The configuration index for VEC server is 6 and the VU density is 8. It shows that FORMOSA utilizes the increased available bandwidth resources more efficiently. The gap between FORMOSA and the second best algorithm, CEFO, increases from 1.3% at 10 MHz to 11.7% at 40 MHz. Moreover, FORMOSA can achieve less than 300 ms end-to-end delay performance when the available bandwidth is > 20 MHz while CEFO cannot achieve the same level of performance.

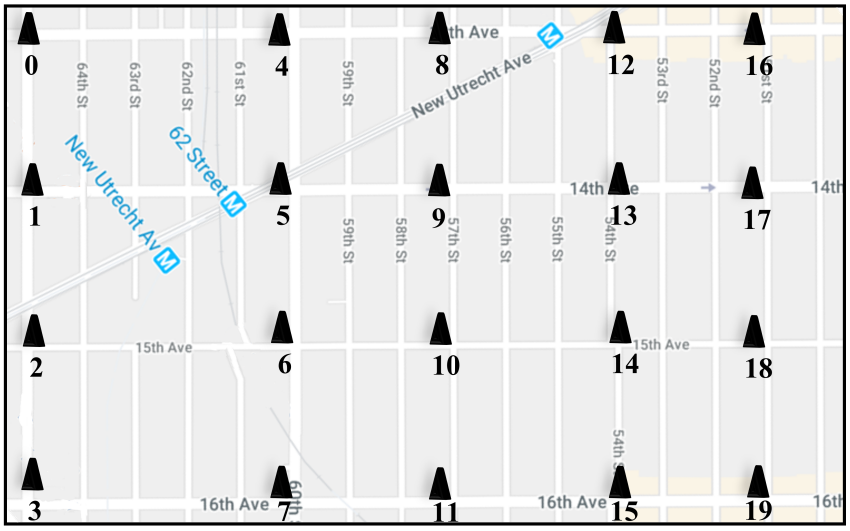

Figure 3.14. Real-trace driven simulation layout in Brooklyn, New York City

3.6.3 Real-trace driven simulation

In this section, we demonstrate the performance of FORMOSA for a real-world traffic scenario. In our previous work [84], we have developed a traffic simulator, which allows the creation of realistic trace-driven movements, topology, location, and channel conditions for each VU at every time slot. The simulator generates the VU's trace in a 1000x800 m^2 rectangular neighborhood in Brooklyn, New York City, using historical vehicular traffic data collected from [11]. It then generates the channel conditions between each VU and the 20 RSUs located in the area based on street topology and the real-time location of vehicles. VU's transmit power is 23 dBm. The channel condition is generated by using B1 Manhattan grid layout [44] as the pathloss and slow fading models, and the Nakagami-m distribution [45] as the fast fading model. At each time slot, VUs are associated with the strongest signal RSU. The geographical layout of the simulated neighborhood and RSU locations are displayed in Fig. 3.14.

Fig. 3.15 shows the simulated history of the number of associated VUs for 4 of the busiest RSUs from 6:00 to 22:00. These 4 RSUs have the highest average number of VUs from 6:00 to 22:00 in the neighborhood. To demonstrate the performance of FORMOSA over a large number of VUs, for the following experiment, we pick RSU 1 (the busiest RSU) and its associated VUs at

each time slot from 14:00 to 20:00 (i.e., the region between the two black dash lines in Fig. 3.15) for performance evaluation. Note that based on Fig. 3.13a, we can observe that FORMOSA can support up to 10 VUs if we want to guarantee less than 500 ms of end-to-end fusion delay. We believe the above limitation is mainly due to the VEC server computing capacity constraint. Based on Fig. 3.15, the number of VUs for RSU 1 can be close to 20. Therefore, we expand the VEC server computing capacity by adding two more Nvidia Jetson Xavier boards to the VEC server. At each time slot, we will execute the following VU distribution mechanism before executing FORMOSA. The mechanism will first sort VUs by the value $\frac{\hat{o}_i}{r_{b,i}}$, where $\hat{o}_i$ and $r_{b,i}$ are the predicted number of detected objects and uplink data rate (assuming $B_i = 1$) for VU i, respectively. Then it will evenly distribute VUs by the sorted order into 3 groups and assign each group to a VEC server (i.e., one Jetson Xavier board). FORMOSA is then executed separately for each group. The available bandwidths are separated evenly to each group.

After the multi-vehicle perception fusion is complete for each group, extra object matching tasks will be executed to match the resulting detected distinct objects across different groups. Therefore, the corresponding object matching task execution delays are added to the maximum end-to-end fusion delay among all the groups. The complete end-to-end fusion delay is then estimated as,

$$\max_{v \in \mathcal{V}} F_v = (\max_{v' \in \mathcal{V}_g, 1 \leq g \leq G} F_{v'}) + \frac{1}{G} \sum_{g=1}^{G} \sum_{q=g+1}^{G} o_g * o_q \tag{3.20}$$

where G is the number of VU groups (i.e., 3), o_g and o_q are the number of distinct objects detected by VUs in groups g and q, respectively. $\mathcal{V}_g$ is the set of all the multi-vehicle perception fusion tasks for group g. Fig. 3.16 shows the complete end-to-end fusion delay performance of the compared 4 algorithms for associated VUs of RSU 1 between 14:00 to 20:00. Each data point in the figure is the average over all the simulated time slot in a 10-minute period. The performance of FORMOSA is the best among all the algorithms. The average end-to-end fusion delay of LSEO, CEFO, and PCSO over the simulated period are 23.1%, 11.3%, and 18.2% worse than FORMOSA. Moreover, the gap between FORMOSA and the second-best algorithm, CEFO,

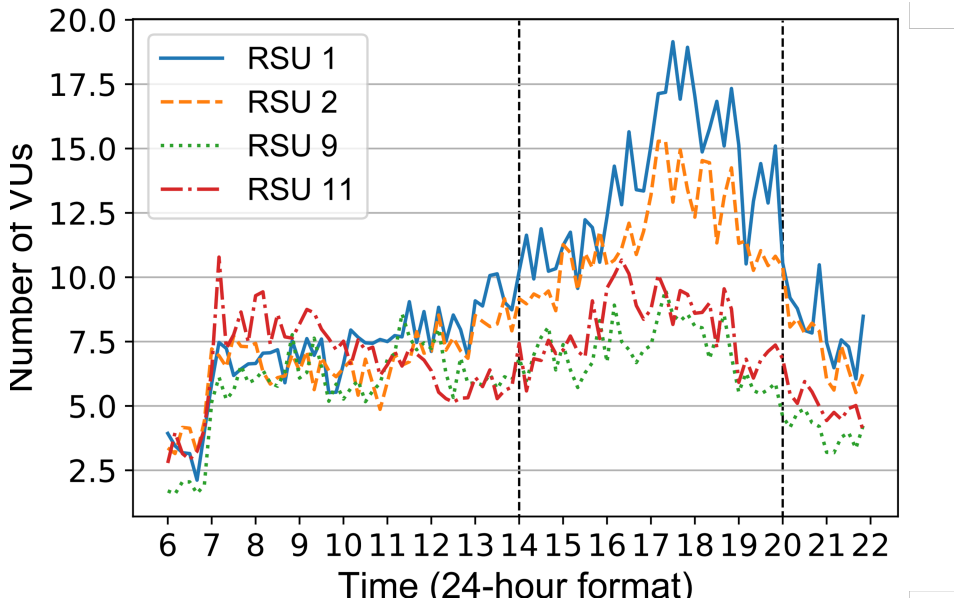

Figure 3.15. History of the number of associated VUs of the busiest 4 RSUs from 6:00 to 22:00

will further increase to over 13% when the number of VUs peaks between 17:00 and 18:00. Note that the number of VEC servers is not limited to 3. Based on the traffic load pattern of an RSU, our algorithm allows the service provider to support a higher number of vehicles, by choosing a higher VEC server capacity (e.g., using multiple VEC servers).

3.7 Conclusion

In this paper, we propose a real-time joint task partitioning, offloading, and scheduling as well as bandwidth allocation algorithm, FORMOSA, for an edge-based multi-vehicle perception fusion system. FORMOSA minimizes the end-to-end fusion application delay of the perception fusion application while addressing joint parallel and sequential task dependencies, varying uplink channel conditions, as well as uncertain task graph challenges. With real-world trace-driven simulation results, we demonstrate that FORMOSA significantly reduces the end-to-end fusion application delay compared to existing algorithms under various resource availability and VU density scenarios. Nowadays, as the vehicular applications are becoming more and more compute-intensive and latency-sensitive, our research facilitates the network service providers

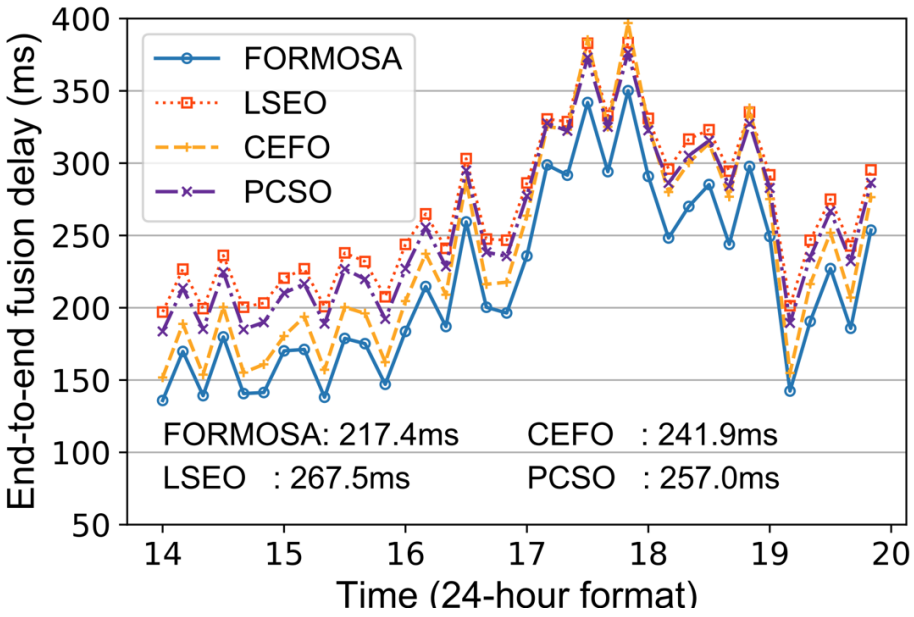

Figure 3.16. End-to-end fusion delay for RSU 1 from 14:00 to 20:00

and vehicular application designers an additional dimension of using RSU to accelerate the execution of vehicular applications. Moreover, with the real-world trace-driven simulation results provided, they will be able to determine the adequate communication and computing capacity of RSUs for the expected end-to-end fusion delay performance and VU density.

Chapter 3, in full, has been submitted for publication of the material as it may appear in IEEE Transactions on Vehicular Technology 2023, Yu-Jen Ku, Sabur Baidya, and Sujit Dey. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Conclusion

This thesis has presented several methodologies to address existing and new challenges in enabling delay-sensitive modern vehicular applications by using resource-limited RSU-assisted VEC systems. The presented methodologies address the challenges of using edge computing resources to assist vehicular applications and the challenges of communication, computing, and energy resource allocation for the RSU.

Chapter 1 has presented a two-phase approach that jointly optimizes solar energy usage and storage, user association, and RSU's computing and communication resource allocation for Solar-powered RSU-assisted VEC systems. The approach minimizes service disruption of the offloaded vehicular applications under the intermittent solar power supply.

Chapter 2 has presented an algorithm that further reduces the vehicular application's execution delay by using computing resources from both the VEC server and the VLC units. Furthermore, the algorithm is able to adjust the VEC server's platform configuration and application's detection accuracy performance based on the available resources at the RSU. We have also implemented an object detection vehicular application on a low-power edge computing platform and established empirical models for the RSU-assisted VEC systems running vehicular applications. Analysis based on the empirical models shows the ability of the proposed algorithm to minimize the application's end-to-end delay while maximizing its detection accuracy.

Chapter 3 has presented a real-time mechanism to minimize the end-to-end delay of a

more comprehensive vehicular application, the multi-vehicle perception fusion application, using the RSU-assisted VEC system. The mechanism jointly determines the optimal RSU's computing and communication resource allocation, as well as task partitioning and scheduling strategies, which are adaptive to the dynamic task composition of the application. Numerical results with real-world traffic data simulation show that the proposed approach significantly reduces the end-to-end delay of the fusion application compared to existing techniques.

In the future, we would like to extend our research in the following directions. Firstly, for the solar-powered RSU-assisted VEC system, when the generated solar power is low due to weather conditions, vehicular applications will always have the risk of experiencing high QoS loss or high execution delay, even with our proposed approaches. Hence, we plan to investigate including other RE sources (e.g., wind energy) to ensure diversity of energy supply and, more surely, minimize the QoS loss and execution delay.

Secondly, we plan to investigate the feasibility of using solar or other renewable energy power sources to support the RSU-assisted VEC system for the multi-vehicle perception fusion application. Renewable energy-powered RSUs are more sustainable and easier to deploy compared to grid-powered RSUs. However, both the RSU's communication and computing resources will be constrained by the limited energy supply. Therefore, the corresponding optimal resource allocation to minimize the application's end-to-end delay will be more challenging.

Finally, the multi-vehicle perception fusion application supported by the RSU-assisted VEC system, we would like to study a content-aware task offloading scenario for vehicular perception fusions based on the information that each detected object contains e.g. image, moving direction, object type, and how such information can benefit the perception of the current or future vehicular environments.

Bibliography

[1] S. Olariu and M. C. Weigle, *Vehicular networks: from theory to practice*. Chapman and Hall/CRC, 2009.

[2] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4377–4387, 2019.

[3] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. ZHANG, "Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36–44, 2017.

[4] M. H. Alsharif, J. Kim, and J. H. Kim, "Green and sustainable cellular base stations: An overview and future research directions," *Energies*, vol. 10, no. 5, 2017.

[5] M. Z. Shakir, K. A. Qaraqe, H. Tabassum, M.-S. Alouini, E. Serpedin, and M. A. Imran, "Green heterogeneous small-cell networks: toward reducing the co2 emissions of mobile communications industry using uplink power adaptation," *IEEE Communications Magazine*, vol. 51, no. 6, pp. 52–61, 2013.

[6] X. Ge, S. Tu, G. Mao, C.-X. Wang, and T. Han, "5g ultra-dense cellular networks," *IEEE Wireless Communications*, vol. 23, no. 1, pp. 72–79, 2016.

[7] A. Mohamad Aris and B. Shabani, "Sustainable power supply solutions for off-grid base stations," *Energies*, vol. 8, no. 10, pp. 10 904–10 941, 2015.

[8] Y.-J. Ku, P.-H. Chiang, and S. Dey, "Quality of service optimization for vehicular edge computing with solar-powered road side units," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, 2018, pp. 1–10.

[9] P.-H. Chiang, R. Guruprasad, and S. Dey, "Renewable energy-aware video download in cellular networks," in *2015 IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2015, pp. 1622–1627.

[10] P.-H. Chiang, S. P. V. Chiluvuri, S. Dey, and T. Q. Nguyen, "Forecasting of solar photovoltaic system power generation using wavelet decomposition and bias-compensated random forest," in *2017 Ninth Annual IEEE Green Technologies Conference (GreenTech)*, 2017, pp. 260–266.

[11] Department of Transportation, New York, NY, USA, *NYS Traffic Data Viewer*, (Accessed: Mar. 14, 2023). [Online]. Available: https://www.dot.ny.gov/tdv

[12] L. X. Cai, Y. Liu, T. H. Luan, X. S. Shen, J. W. Mark, and H. V. Poor, "Sustainability analysis and resource management for wireless mesh networks with renewable energy supplies," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 2, pp. 345–355, 2014.

[13] F. Parzysz, M. Di Renzo, and C. Verikoukis, "Power-availability-aware cell association for energy-harvesting small-cell base stations," *IEEE Transactions on Wireless Communications*, vol. 16, no. 4, pp. 2409–2422, 2017.

[14] V. Chamola, B. Krishnamachari, and B. Sikdar, "Green energy and delay aware downlink power control and user association for off-grid solar-powered base stations," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2622–2633, 2018.

[15] C. Li, A. Qouneh, and T. Li, "iswitch: Coordinating and optimizing renewable energy powered server clusters," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 512–523, 2012.

[16] Í. Goiri, K. Le, M. E. Haque, R. Beauchea, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenslot: scheduling energy consumption in green datacenters," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–11.

[17] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[18] J. Xu and S. Ren, "Online learning for offloading and autoscaling in renewable-powered mobile edge computing," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–6.

[19] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 3, pp. 361–373, 2017.

[20] L. Chen, J. Xu, and S. Zhou, "Computation peer offloading in mobile edge computing with energy budgets," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[21] H. Wu, L. Chen, C. Shen, W. Wen, and J. Xu, "Online geographical load balancing for energy-harvesting mobile edge computing," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[22] F. Guo, L. Ma, H. Zhang, H. Ji, and X. Li, "Joint load management and resource allocation in the energy harvesting powered small cell networks with mobile edge computing,"

in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2018, pp. 299–304.

[23] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[24] T. Yang, Z. Zheng, H. Liang, R. Deng, N. Cheng, and X. Shen, "Green energy and content-aware data transmissions in maritime wireless communication networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 751–762, 2015.

[25] W. S. Atoui, W. Ajib, and M. Boukadoum, "Offline and online scheduling algorithms for energy harvesting rsus in vanets," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 6370–6382, 2018.

[26] W. S. Atoui, M. A. Salahuddin, W. Ajib, and M. Boukadoum, "Scheduling energy harvesting roadside units in vehicular ad hoc networks," in *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*. IEEE, 2016, pp. 1–5.

[27] X. Mao, A. Maaref, and K. H. Teo, "Adaptive soft frequency reuse for inter-cell interference coordination in sc-fdma based 3gpp lte uplinks," in *IEEE GLOBECOM 2008-2008 IEEE Global Telecommunications Conference*. IEEE, 2008, pp. 1–6.

[28] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE communications surveys & tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[29] M. Alasti, B. Neekzad, J. Hui, and R. Vannithamby, "Quality of service in wimax and lte networks [topics in wireless communications]," *IEEE Communications Magazine*, vol. 48, no. 5, pp. 104–111, 2010.

[30] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2019.

[31] N. T. Ti and L. B. Le, "Computation offloading leveraging computing resources from edge cloud and mobile peers," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.

[32] Z. Xu, C. Yang, G. Y. Li, S. Zhang, Y. Chen, and S. Xu, "Energy-efficient configuration of spatial and frequency resources in mimo-ofdma systems," *IEEE Transactions on Communications*, vol. 61, no. 2, pp. 564–575, 2013.

[33] B. Wu, Y. Cui, D. Xiao, and C. Zhang, "Prediction of energy consumption time series using neural networks combined with exogenous series," in *2015 11th International Conference on Natural Computation (ICNC)*. IEEE, 2015, pp. 37–41.

[34] T. Han and N. Ansari, "On optimizing green energy utilization for cellular networks with hybrid energy supplies," *IEEE Transactions on Wireless Communications*, vol. 12, no. 8, pp. 3872–3882, 2013.

[35] X. Sun and N. Ansari, "Green cloudlet network: A distributed green mobile cloud network," *IEEE network*, vol. 31, no. 1, pp. 64–70, 2017.

[36] B. Gavish and H. Pirkul, "Algorithms for the multi-resource generalized assignment problem," *Management science*, vol. 37, no. 6, pp. 695–713, 1991.

[37] M. Yagiura, S. Iwasaki, T. Ibaraki, and F. Glover, "A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem," *Discrete Optimization*, vol. 1, no. 1, pp. 87–98, 2004.

[38] S. Haddadi and H. Ouzia, "An effective lagrangian heuristic for the generalized assignment problem," *INFOR: Information Systems and Operational Research*, vol. 39, no. 4, pp. 351–356, 2001.

[39] C. J. Geyer, "Markov chain monte carlo maximum likelihood," in *Computing Science and Statistics, Proceedings of the 23rd Symposium on the Interface*, 1991, pp. 151–163.

[40] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[41] M. Grant and S. Boyd, *CVX: Matlab software for disciplined convex programming, ver. 2.0 beta,*, Sep. 2013. [Online]. Available: http://cvxr.com/cvx

[42] Google, *New Utrecht Ave, Brooklyn*, (Accessed: Dec. 10, 2019.). [Online]. Available: https://www.google.com/maps/place/New+Utrecht+Ave,+Brooklyn,+NY/@40.6259898,-73.9983422,17z

[43] K. Alexandris, N. Nikaein, R. Knopp, and C. Bonnet, "Analyzing x2 handover in lte/lte-a," in *2016 14th international symposium on modeling and optimization in mobile, ad hoc, and wireless networks (WiOpt)*. IEEE, 2016, pp. 1–7.

[44] P. Kyosti, "Winner ii channel models," *IST, Tech. Rep. IST-4-027756 WINNER II D1. 1.2 V1. 2*, 2007.

[45] L. Cheng *et al.*, "Mobile vehicle-to-vehicle narrow-band channel measurement and characterization of the 5.9 ghz dedicated short range communication (dsrc) frequency band," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 8, pp. 1501–1516, 2007.

[46] T. 3rd Generation Partnership Project, "Study on lte-based v2x services," *3GPP-REF-36885, rel. 14, 2016. [Online]. Available: http://www.3gpp.org/. Accessed: Jan. 19,*, 2018.

[47] ——, "Small cell enhancements for e-utra and e-utran - physical layer aspects,," *3GPP-REF-36872, rel. 12, 2013. [Online]. Available: http://www.3gpp.org/. Accessed: Jan. 7,*, 2018.

[48] C.-L. Huang *et al.*, "Intervehicle transmission rate control for cooperative active safety system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 3, pp. 645–658, 2010.

[49] T. 3rd Generation Partnership Project, "Etura; base station (bs) radio transmission and reception,," *3GPP-REF-36104, rel. 12, 2016. [Online]. Available: http://www.3gpp.org/. Accessed: Apr. 10,*, 2019.

[50] Z. Wang and R. Stirling-Gallacher, "Frequency reuse scheme for cellular ofdm systems," *Electronics Letters*, vol. 38, no. 8, pp. 387–388, 2002.

[51] Benchoff, Brian, *Benchmarking the Raspberry Pi 2*, Feb. 5, 2015. (Accessed Apr. 11, 2019.). [Online]. Available: https://hackaday.com/2015/02/05/benchmarking-the-raspberry-pi-2/

[52] W. Wu, M. Yagiura, and T. Ibaraki, "Generalized assignment problem," *Handbook of Approximation Algorithms and Metaheuristics, Second Edition*, pp. 713–736, 2018.

[53] G.-S. Initiative and Deloitte, "Digital with purpose: Delivering a smarter 2030," *2019. [Online]. Available: https://gesi.org/research/gesi-digital-with-purpose-full-report. Accessed: Nov. 16,*, 2020.

[54] Y. Ku and S. Dey, "Sustainable vehicular edge computing using local and solar-powered roadside unit resources," in *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, 2019, pp. 1–7.

[55] S. Baidya, Y.-J. Ku, H. Zhao, J. Zhao, and S. Dey, "Vehicular and edge computing for emerging connected and autonomous vehicle applications," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE, 2020, pp. 1–6.

[56] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Salt Lake City, UT, USA, 2018, pp. 4510–4520.

[57] Nvidia, "Nvidia jetson tx2 series system-on-moduledatasheet v1.2, 2014," , [Online] https://developer.nvidia.com/embedded/jetson-tx2, Accessed: Apr. 9, 2020.

[58] G. Qiao, S. Leng, K. Zhang, and Y. He, "Collaborative task offloading in vehicular edge multi-access networks," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 48–54, 2018.

[59] Y. Zhou *et al.*, "Distributed scheduling for time-critical tasks in a two-layer vehicular fog computing architecture," in *IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2020, pp. 1–7.

[60] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 2092–2104, 2019.

[61] B. Gu and Z. Zhou, "Task offloading in vehicular mobile edge computing: A matching-theoretic framework," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 100–106, 2019.

[62] M. Li, J. Gao, L. Zhao, and X. Shen, "Deep reinforcement learning for collaborative edge computing in vehicular networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 4, pp. 1122–1135, 2020.

[63] C. Yang, Y. Liu, X. Chen, W. Zhong, and S. Xie, "Efficient mobility-aware task offloading for vehicular edge computing networks," *IEEE Access*, vol. 7, pp. 26 652–26 664, 2019.

[64] S. Mao, S. Leng, and Y. Zhang, "Joint communication and computation resource optimization for noma-assisted mobile edge computing," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. Shanghai, China: IEEE, 2019, pp. 1–6.

[65] L. P. Qian, B. Shi, Y. Wu, B. Sun, and D. H. K. Tsang, "Noma-enabled mobile edge computing for internet of things via joint communication and computation resource allocations," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 718–733, 2019.

[66] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn: Local distributed mobile computing system for deep neural network," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. Lausanne, Switzerland: IEEE, 2017, pp. 1396–1401.

[67] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[68] M. Gao, W. Cui, D. Gao, R. Shen, J. Li, and Y. Zhou, "Deep neural network task partitioning and offloading for mobile edge computing," in *2019 IEEE Global Communications Conference (GLOBECOM)*. Waikoloa, HI, USA: IEEE, 2019, pp. 1–6.

[69] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, Atlanta, GA, USA, 2017, pp. 328–339.

[70] L. Yang, J. Cao, Z. Wang, and W. Wu, "Network aware multi-user computation partitioning in mobile edge clouds," in *2017 46th International Conference on Parallel Processing (ICPP)*, 2017, pp. 302–311.

[71] Y. Dai *et al.*, "Joint computation offloading and user association in multi-task mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 12, pp. 12 313–12 325, 2018.

[72] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 726–738, 2019.

[73] N. Saquib, E. Hossain, and D. I. Kim, "Fractional frequency reuse for interference management in lte-advanced hetnets," *IEEE Wireless Communications*, vol. 20, no. 2, pp. 113–122, 2013.

[74] S. Chen *et al.*, "Vehicle-to-Everything (v2x) Services Supported by LTE-Based Systems and 5G," *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 70–76, 2017.

[75] K. Bansal, K. Rungta, S. Zhu, and D. Bharadia, "Pointillism: Accurate 3d bounding box estimation with multi-radars," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, ser. SenSys '20, 2020, p. 340–353.

[76] D.-H. Kim, "Lane detection method with impulse radio ultra-wideband radar and metal lane reflectors," *Sensors*, vol. 20, no. 1, 2020.

[77] M. M. William *et al.*, "Traffic signs detection and recognition system using deep learning," in *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*, 2019, pp. 160–166.

[78] X.-Y. Ye *et al.*, "A two-stage real-time yolov2-based road marking detector with lightweight spatial transformation-invariant classification," *Image and Vision Computing*, vol. 102, p. 103978, 2020.

[79] J. Zheng, K. Okamoto, and P. Tsiotras, "Vision-based autonomous vehicle control using the two-point visual driver control model," *arXiv preprint arXiv:1910.04862*, 2019.

[80] S.-C. Lin *et al.*, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 751–766.

[81] Y.-J. Ku, S. Sapra, S. Baidya, and S. Dey, "State of energy prediction in renewable energy-driven mobile edge computing using cnn-lstm networks," in *IEEE Green Energy and Smart Systems Conference (IGESSC 2020)*. Long Beach, CA, USA: IEEE, 2020, pp. 1–6.

[82] I. Gomez-Miguelez *et al.*, "srslte: an open-source platform for lte evolution and experimentation," in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, 2016, pp. 25–32.

[83] R. Hemmecke, M. Köppe, J. Lee, and R. Weismantel, "Nonlinear integer programming," in *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 561–618.

[84] Y. J. Ku, P. H. Chiang, and S. Dey, "Real-time qos optimization for vehicular edge computing with off-grid roadside units," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 11 975–11 991, 2020.

[85] E. Uhlemann, "Initial steps toward a cellular vehicle-to-everything standard [connected vehicles]," *IEEE Vehicular Technology Magazine*, vol. 12, no. 1, pp. 14–19, 2017.

[86] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 2014, pp. 241–246.

[87] A. Shaout, D. Colella, and S. Awad, "Advanced driver assistance systems - past, present and future," in *2011 Seventh International Computer Engineering Conference (ICENCO'2011)*, 2011, pp. 72–82.

[88] Y.-J. Ku, S. Baidya, and S. Dey, "Adaptive computation partitioning and offloading in real-time sustainable vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 13 221–13 237, 2021.

[89] J. Wang, C. Jiang, K. Zhang, T. Q. S. Quek, Y. Ren, and L. Hanzo, "Vehicular sensing networks in a smart city: Principles, technologies and applications," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 122–132, 2018.

[90] J. Wang, C. Jiang, Z. Han, Y. Ren, and L. Hanzo, "Internet of vehicles: Sensing-aided transportation information collection and diffusion," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 3813–3825, 2018.

[91] X. Hou, Z. Ren, J. Wang, W. Cheng, Y. Ren, K.-C. Chen, and H. Zhang, "Reliable computation offloading for edge-computing-enabled software-defined iov," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7097–7111, 2020.

[92] M. Wang, T. Ma, T. Wu, C. Chang, F. Yang, and H. Wang, "Dependency-aware dynamic task scheduling in mobile-edge computing," in *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, 2020, pp. 785–790.

[93] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, and F. Yang, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4961–4971, 2020.

[94] C. Shu, Z. Zhao, Y. Han, G. Min, and H. Duan, "Multi-user offloading for edge computing networks: A dependency-aware and latency-optimal approach," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 1678–1689, 2020.

[95] C. Shu, Z. Zhao, Y. Han, and G. Min, "Dependency-aware and latency-optimal computation offloading for multi-user edge computing networks," in *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2019, pp. 1–9.

[96] Y. Shang, J. Li, and X. Wu, "Dag-based task scheduling in mobile edge computing," in *2020 7th International Conference on Information Science and Control Engineering (ICISCE)*, 2020, pp. 426–431.

[97] H. Liu, H. Zhao, L. Geng, Y. Wang, and W. Feng, "A distributed dependency-aware offloading scheme for vehicular edge computing based on policy gradient," in *2021 8th IEEE CSCloud/2021 7th IEEE EdgeCom*, 2021, pp. 176–181.

[98] H. Liu, H. Zhao, L. Geng, and W. Feng, "A policy gradient based offloading scheme with dependency guarantees for vehicular networks," in *2020 IEEE Globecom Workshops (GC Wkshps)*, 2020, pp. 1–6.

[99] R. Chai, M. Li, T. Yang, and Q. Chen, "Dynamic priority-based computation scheduling and offloading for interdependent tasks: Leveraging parallel transmission and execution," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 10, pp. 10 970–10 985, 2021.

[100] L. Yang, J. Cao, Z. Wang, and W. Wu, "Network aware mobile edge computation partitioning in multi-user environments," *IEEE Transactions on Services Computing*, vol. 14, no. 5, pp. 1478–1491, 2021.

[101] S. Thornton and S. Dey, "Machine learning techniques for vehicle matching with non-overlapping visual features," in *2020 IEEE 3rd Connected and Automated Vehicles Symposium (CAVS)*, 2020, pp. 1–6.

[102] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[103] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[104] Nvidia, "Jetson agx xavier series modules and developer kit," , [Online] https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/, Accessed: Jun. 28, 2022.

[105] ——, "Nvidia jetson developer guide," , [Online] https://docs.nvidia.com/jetson/l4t/, Accessed: Mar. 7, 2023.

[106] Y.-J. Ku, B. Flowers, S. Thornton, S. Baidya, and S. Dey, "Adaptive c-v2x sidelink communications for vehicular applications beyond safety messages," in *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, 2022, pp. 1–6.

[107] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[108] J. Luiten, A. Osep, P. Dendorfer, P. Torr, A. Geiger, L. Leal-Taixe, and B. Leibe, "Hota: A higher order metric for evaluating multi-object tracking," *International Journal of Computer Vision (IJCV)*, 2020.