Computation Offloading and Task Scheduling on Network Edge

by

Mushu Li

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2021

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Yuguang Michael Fang Distinguished Professor Department of Electrical and Computer Engineering University of Florida
Supervisor(s):	Xuemin (Sherman) Shen University Professor Department of Electrical and Computer Engineering University of Waterloo
Internal Member:	Kshirasagar Naik Professor Department of Electrical and Computer Engineering University of Waterloo
Internal Member:	Xiaodong Lin Adjunct Associate Professor Department of Electrical and Computer Engineering University of Waterloo
Internal-External Member:	Yaoliang Yu Assistant Professor School of Computer Science University of Waterloo

Author's Declaration

Abstract

The Fifth-Generation (5G) networks facilitate the evolution of communication systems and accelerate a revolution in the Information Technology (IT) field. In the 5G era, wireless networks are anticipated to provide connectivity for billions of Mobile User Devices (MUDs) around the world and to support a variety of innovative use cases, such as autonomous driving, ubiquitous Internet of Things (IoT), and Internet of Vehicles (IoV). The novel use cases, however, usually incorporate compute-intensive applications, which generate enormous computing service demands with diverse and stringent service requirements. In particular, autonomous driving calls for prompt data processing for the safety-related applications, IoT nodes deployed in remote areas need energy-efficient computing given limited on-board energy, and vehicles require low-latency computing for IoV applications in a highly dynamic network.

To support the emerging computing service demands, Mobile Edge Computing (MEC), as a cutting-edge technology in 5G, utilizes computing resources on network edge to provide computing services for MUDs within a radio access network. The primary benefits of MEC can be elaborated from two perspectives. From the perspective of MUDs, MEC enables low-latency and energy-efficient computing by allowing MUDs to offload their computation tasks to proximal edge servers, which are installed in access points such as cellular base stations, Road-Side Units (RSUs), and Unmanned Aerial Vehicles (UAVs). On the other hand, from the perspective of network operators, MEC allows a large amount of computing data to be processed on network edge, thereby alleviating backhaul congestion. MEC is a promising technology to support computing demands for the novel 5G applications within the RAN. The interesting issue is to maximize the computation capability of network edge to meet the diverse service requirements arising from the applications in dynamic network environments. However, the main technical challenges are: 1) how an edge server schedules its limited computing resources to optimize the Quality-of-Experience (QoE) in autonomous driving; 2) how the computation loads are balanced between the edge server and IoT nodes in computation loads to enable energy-efficient computing service provisioning; and 3) how multiple edge servers coordinate their computing resources to enable seamless and reliable computing services for high-mobility vehicles in IoV.

In this thesis, we develop efficient computing resource management strategies for MEC, including computation offloading and task scheduling, to address the above three technical challenges. First, we study computation task scheduling to support real-time applications, such as localization and obstacle avoidance, for autonomous driving. In our considered scenario, autonomous vehicles periodically sense the environment, offload sensor data to an edge server for processing, and receive computing results from the edge server. Due to mobility and computing latency, a vehicle travels a certain distance between the instant of offloading its sensor data and the instant of receiving the computing result. Our objective is to design a scheduling scheme for the edge server to minimize the above traveled distance of vehicles. The idea is to determine the processing order according to the individual vehicle mobility and computation capability of the edge server. We formulate a Restless Multi-Armed Bandit (RMAB) problem, design a Whittle index-based stochastic scheduling scheme, and determine the index using a Deep Reinforcement Learning (DRL) method. The proposed scheduling scheme can avoid the time-consuming policy exploration common in DRL scheduling approaches and makes effectual decisions with low complexity. Extensive simulation results demonstrate that, with the proposed index-based scheme, the edge server can deliver computing results to the vehicles promptly while adapting to time-variant vehicle mobility. Second, we study energy-efficient computation offloading and task scheduling for an edge server while provisioning computing services for IoT nodes in remote areas. In the considered scenario, a UAV is equipped with computing resources and plays the role of an aerial edge server to collect and process the computation tasks offloaded by ground MUDs. Given the service requirements of MUDs, we aim to maximize UAV energy efficiency by jointly optimizing the UAV trajectory, the user transmit power, and computation task scheduling. The resulting optimization problem corresponds to nonconvex fractional programming, and the Dinkelbach algorithm and the Successive Convex Approximation (SCA) technique are adopted to solve it. Furthermore, we decompose the problem into multiple subproblems for distributed and parallel problem solving. To cope with the case when the knowledge of user mobility is limited, we apply a spatial distribution estimation technique to predict the location of ground users so that the proposed approach can still be valid. Simulation results demonstrate the effectiveness of the proposed approach to maximize the energy efficiency of the UAV. Third, we study collaboration among multiple edge servers in computation offloading and task scheduling to support computing services in IoV. In the considered scenario, vehicles traverse the coverage of edge servers and offload their tasks to their proximal edge servers. We develop a collaborative edge computing framework to reduce computing service latency and alleviate computing service interruption due to the high mobility of vehicles: 1) a Task Partition and Scheduling Algorithm (TPSA) is proposed to schedule the execution order of the tasks offloaded to the edge servers given a computation offloading strategy; and 2) an artificial intelligence-based collaborative computing approach is developed to determine the task offloading, computing, and result delivery policy for vehicles. Specifically, the offloading and computing problem is formulated as a Markov decision process. A DRL technique, i.e., deep deterministic policy gradient, is adopted to find the optimal solution in a complex urban transportation network. With the developed framework, the service cost, which includes computing service latency and service failure penalty, can be minimized via the optimal computation task scheduling and edge server selection. Simulation results show that the proposed AI-based collaborative computing approach can adapt to a highly dynamic environment with outstanding performance.

In summary, we investigate computing resource management to optimize QoE of MUDs in the coverage of an edge server, to improve energy efficiency for an aerial edge server while provisioning computing services, and to coordinate computing resources among edge servers for supporting MUDs with high mobility. The proposed approaches and theoretical results contribute to computing resource management for MEC in 5G and beyond.

Acknowledgements

Pursuing the Ph.D. degree at the University of Waterloo was one of the biggest and best decisions I've ever made. It would not have been possible for me to complete my Ph.D. without the support and assistance of many people. I would like to take this opportunity to give my sincere thanks to them.

First of all, my most sincere and deepest thanks go to my supervisor, Professor Xuemin (Sherman) Shen, who has provided me with continuous support, patient guidance, and endless help during my Ph.D. study at the University of Waterloo. His insightful advice, positive philosophy, and continuous encouragement have broadened my academic horizons and motivated me to strive for excellence. His dedication, empressement, and enthusiasm towards his work, his students, and his family really inspire me to be a better researcher and person. I feel very fortunate and privileged to be his student.

I would like to thank Professor Kshirasagar Naik, Professor Xiaodong Lin, Professor Yaoliang Yu, and Professor Yuguang Michael Fang for serving on my thesis examination committee. Their insightful comments and valuable suggestions have significantly improved the quality of my thesis.

I would like to express my gratitude to Professor Weihua Zhuang for all her great support in my Ph.D. study and life. Her insightful thoughts and valuable instructions have greatly improved my research skills and helped me develop a rigorous academic attitude. Also, I would like to thank Prof. Lian Zhao at Ryerson University. Her inspiration and support influenced my decision to pursue a Ph.D. and begin my research career.

I received invaluable friendships, help, and support from former and current BBCR members during the past four years. I would like to express my sincere thank to all BBCR members. I would like to especially thank Prof. Jie Gao, Prof. Nan Chen, Prof. Nan Cheng, Prof. Ning Zhang, Prof. Qiang Ye, Dr. Huaqing Wu, Dr. Wen Wu, Dr. Qihao Li, Dr. Weisen Shi, Dr. Junling Li, Dr. Haixia Peng, Prof. Peng Yang, Dr. Jiayin Chen, Conghao Zhou, Mingcheng He, and Yingying Pei for their great help in both my research and life. I am also grateful for all the time spent with Prof. Jianbing Ni, Dr. Dongxiao Liu, Dr. Kaige Qu, Dr. Omar Alhussein, Si Yan, Liang Xue, and many other current and former BBCR members. Thanks to all of you for your great support and precious friendship.

Finally, I would like to thank my parents deeply for their deep love and endless support during my Ph.D. study.

Dedication

To my beloved parents, Nuan Xu and Xiaodong Li.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

3GPP	3rd Generation Partnership Project
5G	Fifth-Generation
AC	Actor-Critic
ADMM	Alternating Direction Method of Multipliers
AI	Artificial Intelligence
AoR	Age-of-Computing-Results
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DFVS	Dynamic Frequency and Voltage Scaling
DQN	Deep Q Network
DRL	Deep Reinforcement Learning
EAoR	Expected Age-of-Computing-Results
eMBB	Enhanced Mobile Broadband
ETSI	European Telecommunications Standards Institute
FMC	Follow Me Cloud
IT	Information Technology

IoT Internet of Things

IoV Internet of Vehicle

LoS Line-of-Sight

LSTM Long Short-Term Memory

MAB Multi-Armed Bandit

MCC Mobile Cloud Computing

MDP Markov Decision Process

MEC Mobile Edge Computing

mMTC Massive Machine-Type Communications

QoE Quality of Experience

QoS Quality of Service

MUDs Mobile User Devices

RAN Radio Access Network

RMAB Restless Multi-Armed Bandit

RSUs Roadside Units

RTT Round Trip Time

SCA Successive Convex Approximation

SOC Second Order Cone

SSA Scheduling Scheme for Asynchronous Offloading

TPSA Task Partition and Scheduling Algorithm

UAVs Unmanned Aerial Vehicles

URLLC	Ultra-Reliable and Low Latency Communication
V2I	Vehicle-to-Infrastructure
V2X	Vehicle-to-Everything
VANETs	Vehicular Ad Hoc Networks
VMs	Virtual Machines
VR	Virtual Reality

Chapter 1

Introduction

As mobile applications continue to develop, wireless networks are evolving rapidly, resulting in the convergence of novel technologies and new network architectures. Prior to Fifth-Generation (5G), wireless networks were designed to support communication services, such as voice and multimedia services. The increase in data rate and the number of Mobile User Devices (MUDs) promote the development of compute-intensive mobile applications, while bringing extensive computing demands to wireless networks. Before 5G, the computing tasks generated by MUDs can be either processed locally or offloaded to a cloud server located on the Internet. However, emerging applications in 5G and beyond, such as Internet of Things (IoT) and Artificial Intelligence (AI), further motivate the diversification of mobile applications, and the corresponding computing demands are becoming pervasive in wireless networks [1, 2]. Conventional computing solutions, i.e., cloud computing or computing on HMDs locally, cannot offer sufficient computation capability to support the ever-growing computing demands, which poses new challenges that lie beyond communication capacity improvement in 5G and beyond. To address the challenge, a new computing paradigm, Mobile Edge Computing (MEC) has been proposed. In this chapter, we provide an overview of MEC, discuss the role of MEC in 5G and beyond, and elaborate on how the MEC technology satisfies the increasing computing service demands. We then present three key research problems investigated in this thesis.

1.1 Overview of Mobile Edge Computing

A tremendous technological development in mobile applications results in escalating demands for the computing resource in wireless networks. Because local MUDs have limited computing and energy resources, they would suffer from high latency and energy consumption when processing computing demands on local devices. As a result, networks continue to improve mobile computation capability in order to meet increasing computing demands. For the last decade, Mobile Cloud Computing (MCC) has provided centralized computing solutions to MUDs by deploying cloud servers over the Internet. At this stage, wireless networks are aimed at supporting data services to MUDs for computation offloading and providing access to computing resources at the cloud server, collaborating with backbone IP networks and cellular core networks [3]. As computing demands in network continuously increase, the limitations of MCC have emerged. First, the novel mobile applications or use cases mandate stringent computing requirements. For example, Virtual Reality (VR) streaming requires ultra-low latency on video processing to satisfy motion-to-photon latency ($<$ 20ms) requirement. Ideally, a Round Trip Time (RTT) on the order of 2 ms between the computing server and MUDs is recommended to satisfy the requirement [4], while the RTT of MCC is around 50 ms [5]. Second, the computing tasks generated by mobile applications become complicated and compute-intensive, which further increases computation loads on the cloud server. As a result of the high reliance on cloud computing, extensive computing data would enter core networks and further aggravate traffic congestion in backbone networks. Therefore, it is necessary to innovate the paradigm of computing for mobile applications to cope with the above challenges.

To address the challenges posed by MCC, a new computing paradigm, i.e., MEC, has been proposed as a key technology for 5G [6, 7]. MEC enhances the computation capability on network edge by deploying the computing resources, i.e., storage and processing capacity, to edge nodes at the Radio Access Network (RAN). An edge node, equipped with computation capability, can serve as a computing server, i.e., the edge server, to process the computing tasks generated by MUDs or other network entities at the RAN within its communication range. The main purpose of introducing MEC is to move the computing resources to network edge closer to MUDs. As such, MEC inherits the benefits of MCC in terms of computation offloading to provide fast and energy-efficient computing services for resource-constrained MUDs. Moreover, compared with MCC, MEC offers much lower latency on computation offloading by processing computing tasks near MUDs, usually in one hop, thereby enabling low-latency and real-time computing services [8]. In addition, MEC processes the computing data on network edge without entering the core network, thereby alleviating congestion on the backbone and further optimizing network traffic flow.

The ultra-dense deployment of edge nodes in 5G endows the concept of MEC with significant value, which is not limited to low-latency computing services. First, as the number of edge nodes increases, MEC is a low cost and flexible approach to enhance the computing capability in the network, compared with establishing additional data centers and Virtual

Machines (VMs) in MCC. With the development of aerial communications, such as Unmanned Aerial Vehicles (UAVs) assisted networks, MEC enables ubiquitous computing services for MUDs, in particular for MUDs in remote areas. In addition, taking the advantage of the proximity to MUDs, edge servers can perform specific types of computation functions based on network context information, such as locations, and implement specific computing resource management policies corresponding to network characteristics of the MUDs, such as traffic patterns and computation loads. By optimizing computing service deployment, MEC can effectively improve the performance of computing services and hence the Quality of Experience (QoE) among MUDs. For example, in vehicular networks, edge servers on Roadside Units (RSUs) can manage their computing resources according to real-time vehicle traffic flow and deploy computing functions for vehicle-related services. Besides the above advantages, MEC is also featured by security and privacy protection and reliability due to the distributed computing architecture [9].

Across academia and industry, MEC has been gaining significant attention due to its advantages. Since MEC is standardized by European Telecommunications Standards Institute (ETSI) in 2014 [6], the implementation of MEC has been accelerated. Telecommunication companies, such as Huawei, Ericsson, Microsoft, etc., have focused on MEC's commercialization and established MEC products for diverse applications, including IoT applications [10, 11]. In the research community, the 3rd Generation Partnership Project (3GPP) has conducted extensive studies on enabling MEC in 5G [12] and has included MEC as a crucial element in 5G architecture [13]. Overall, MEC is a crucial component in the 5G architecture which prompts wireless networks to achieve digital and intelligent transformation. In this thesis, we will focus on resource management in MEC to support emerging services in 5G and beyond.

1.2 MEC in 5G and Beyond

5G networks aim to support three major service scenarios: Enhanced Mobile Broadband (eMBB), Ultra-Reliable and Low Latency Communication (URLLC), and Massive Machine-Type Communications (mMTC). These service scenarios raise diverse service requirements. Although the service scenarios of next-generation networks beyond 5G have not been determined yet, as a preliminary consensus, the service scenarios in sixth-generation networks would be the combinations of eMBB, URLLC, and mMTC [14,15]. To support these service scenarios, MEC can be a promising solution which facilitates flexible and powerful computation capability in the network with a low-cost manner. In this thesis, we focus on three use cases emerging in 5G: autonomous driving, IoT in remote areas, and

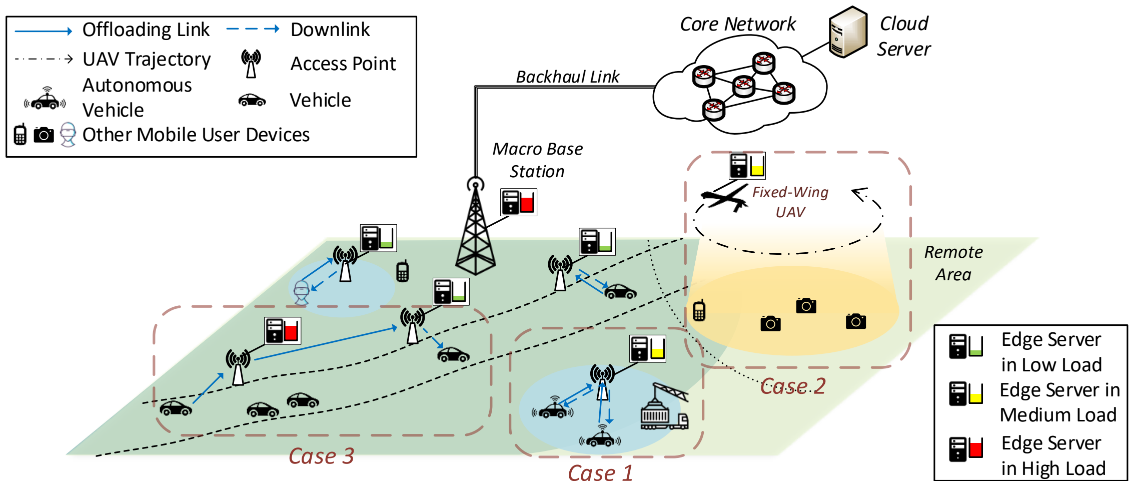


Figure 1.1: An overview of MEC-empowered networks.

vehicles, to demonstrate the necessity of MEC in future networks.

- **MEC for autonomous driving**: In autonomous driving, autonomous vehicles are equipped with advanced sensors, such as cameras, lidar, radar, etc., which continuously generate sensor data for safety-related applications, such as localization and obstacle avoidance [16]. The amount of sensor data to be processed can be huge, up to 10TB and 20TB per hour per vehicle [17]. Meanwhile, to process the sensor data, machine learning techniques are usually adopted, such as the Yolo algorithm in object detection [18]. Executing such compute-intensive tasks can be too demanding for in-vehicle processing, particularly for real-time applications. Therefore, MEC can be adopted to process the sensor data generated by autonomous vehicles with low latency, thereby supporting real-time autonomous driving applications. In addition, by MEC, an edge server can aggregate sensor data provided by multiple autonomous vehicles and compute perception results synthetically. In this way, perception accuracy and decision-making efficiency can be further improved by data fusion. The computing scenario is shown as Case 1 in Fig. 1.1.

- **MEC for IoT in remote areas**: In the 5G era, billions of IoT nodes are envisioned to be deployed worldwide for diverse sensing tasks. A substantial portion of the IoT nodes operate in remote or challenging areas, such as forests, deserts, mountains, or underwater locations [19], and have computing applications to be performed, such as long pipeline infrastructures control [20] and underwater infrastructures monitoring [21]. Such IoT nodes located in remote areas usually have limited onboard energy and cannot reach terrestrial network infrastructure. Thanks to the development of ubiquitous communications in 5G, UAVs can be equipped with computation capabilities to form an on-premises computing platform for the IoT nodes. By offloading computation loads to a UAV-mounted cloudlet, IoT nodes can reduce their energy consumption, thus extending their onboard battery life. The computing scenario is shown as Case 2 in Fig. 1.1. In this case, MEC provides an energy-efficient and flexible computing solution for the IoT nodes.

- **MEC for Internet of Vehicle (IoV)**: Conventional Vehicular Ad Hoc Networks (VANETs) focus on vehicle-to-vehicle or vehicle-to-RSU communications to share safety-related information. As the vehicular networks continue to evolve, onboard applications are not limited to safety message exchange and have become more advanced and diversified, such as traffic control and optimization [22], onboard entertainments [23], argument reality services [24], etc. Supporting the diverse IoV applications calls for high computation capability in vehicular networks. In MEC, edge servers can be deployed at RSUs or other access points adjacent to roads to provide low-latency computing services to vehicles. Edge servers can update location-based information (e.g., real-time traffic reports, high-definition maps) provided to vehicles based on road conditions and vehicle traffic flow, thus providing highly flexible computing services. Additionally, edge servers can support onboard entertainments, such as video streaming and gaming, by offering sufficient computing resources to vehicles. The computing scenario is shown as Case 3 in Fig. 1.1.

The applications of MEC in 5G and beyond are not limited to the above three use cases. For example, MEC is envisioned to support video processing for mobile VR video streaming, which requires ultra-low latency on video processing and delivery [25]. Moreover, with the widespread adoption of AI-empowered mobile applications, it is foreseeable that MEC will be a critical technology to cope with the AI applications that pervade the entire network in the future [26].

1.3 Computing Service Provisioning on Network Edge

MEC aims to support computing services for MUDs while satisfying their service requirements, such as low latency or low energy consumption, which calls for appropriate strategies on resource management. Resource management strategies in MEC consists of three fundamental parts: computation offloading, task scheduling, and mobility management.

Resource allocation policies in these three parts are highly coupled and jointly determine computing service performance. The details of the three parts can be summarized as follows:

- **Computation offloading:** MUDs offload their computation loads to edge servers via wireless links to obtain computing resources. There are three major decisions in computation offloading. The first decision is to determine the portion of computation loads to be offloaded. A computing task generated by an MUD can be divided into several sub-tasks with different computation loads, i.e., task partitions, and the subtasks can be offloaded to various edge servers or computed locally. An optimal task partition ratio can improve the effectiveness of computation offloading, in which the computation loads are assigned to different network entities based on their computation capability and communication link quality. Second, if an MUD lies within a coverage area reachable to multiple edge servers, a proper edge server should be selected to execute the offloaded computation loads. Last, communication resources should be properly allocated for computation offloading to enable prompt computation offloading. The corresponding decision variables include transmit power, spectrum bandwidth for computation offloading, and computing resources.

- **Task scheduling:** After an edge server receives computing tasks offloaded by MUDs, the server needs to determine the execution order of the tasks. The tasks scheduled to be executed earlier will have a relatively shorter computing latency, while the tasks scheduled to be executed later would experience a longer computing latency. In addition to computing latency, the scheduling policy also affects the energy consumption of edge servers. For Dynamic Frequency and Voltage Scaling (DFVS) based CPU architecture, the energy consumption for computing can increase cubically with computation loads [9]. Different scheduling policies would result in different computing intensities over time, thereby affecting computing energy efficiency.

- **Mobility management:** Different from MCC, the service coverage of an edge server is limited, depending on the communication coverage of the corresponding edge node. Thus, an edge server needs to determine when and where to migrate the computation loads offloaded by MUDs if the MUDs travel out of the communication coverage of an edge server. In addition, mobility is also an intrinsic trait of MEC for some applications, such as localization of vehicles [9]. The location and movement trajectory of MUDs provide additional information to edge servers for offering intelligent computing services to the MUDs. For example, in argument reality (AR) assisted museum tours, different videos can be delivered to MUDs according to their locations to achieve an immersive experience; in autonomous vehicles, the speed and location of vehicles are useful information to edge servers to perceive road conditions and execute safety-related applications. Therefore, how to manage the computing resources to support such real-time applications according to the movement of MUDs would be a significant technical issue in MEC.

In this thesis, we will focus on resource management in the context of above three parts. We will identify the challenges in computation offloading, task scheduling, and mobility management, in different use cases, i.e., autonomous driving, IoT in remote areas, and IoV, and address the challenges by proposing effectual edge computing strategies.

1.4 Research Motivations and Contributions

1.4.1 Challenges of MEC in 5G and Beyond

Although MEC technology provides a possible solution to low-latency and energy-efficient computing in wireless networks, implementing MEC in the network still faces the following challenges:

1) **High mobility of MUDs:** The challenges brought by mobility are two-fold. First, communication connectivity between edge servers and MUDs with high mobility, such as vehicles, is intermittent, while completing computing tasks may take some time, especially when the tasks have longer computing sessions, such as VR video streaming. Without proper mobility management strategies, it is difficult to deliver the computing results back to MUDs if the MUDs travel out of the communication coverage of the edge server to which the computation loads are offloaded. The interruption in computing would result in service discontinuity and degrades computing service quality. However, developing an effective mobility management strategy can be a challenge due to high network dynamics. For example, user's viewpoint movement would result in different videos to be processed and delivered from edge servers, thereby introducing computing demand dynamics. Second, as aforementioned, in some location-based applications, the movement of MUDs could be an important factor in resource management for MEC. However, it is challenging to devise an optimal resource allocation strategy with low overhead for a large number of MUDs with heterogeneous and dynamic mobility patterns. Additionally, measuring the impact of MUD movement on QoE in location-based applications, such as localization in autonomous driving, is another challenge to be addressed in resource management.

2) **Complex decision making:** In MEC, computation offloading, task scheduling, and mobility management are highly coupled with each other. Specifically, an edge server can schedule computing tasks only if the tasks are offloaded from MUDs; additionally, the computing latency at the edge server affects the offloading decisions of MUDs and the frequency of service migrations. The three types of policies, i.e., computation offloading, task scheduling, and mobility management, determine computing service performance jointly, thus increasing the difficulty in finding the optimal resource management strategy for MEC. Furthermore, when UAVs are utilized as an edge server for terrestrial MUDs, the dimensions of decision variables can be further enlarged. The settings of UAVs, such as flying trajectory, speed, and accelerations, would affect the transmit power and data rate of MUDs in computation offloading and should be jointly optimized in the resource management strategies, which further complicating MEC implementation in wireless networks.

3) **Network Constraints:** While edge servers are much more resourceful than MUDs, resources on edge servers are still limited when there are a lager number of connected MUDs. First, the computing units at an edge server are limited, while computing service demand is time-varying. An optimal computing task scheduling policy is required to support real-time computing services and maximize the QoE of all connected MUDs, while the heterogeneity of the computing tasks, such as computation loads and properties of associated MUDs, makes the optimization process non-trivial. Second, computing is energy-consuming. In particular, different from the terrestrial edge servers, it can be difficult for a UAV-mounted cloudlet to provide enduring computing services due to its limited onboard energy. With the energy constraints in both the UAV-mounted cloudlet and MUDs, how to design an energy-efficient resource management strategy is another challenge.

1.4.2 Approaches and Contributions

In this thesis, we aim to develop efficient computing resource management strategies to address the challenges in MEC, in which both computation offloading and task scheduling policies are designed to increase computation capability and boost computing service performance at network edge. We focus on three research issues to address the aforementioned challenges.

1) We investigate how an edge server schedules its computing resource to optimize the QoE of MUDs by taking their mobility dynamics into account. To be specific, we

study computing task scheduling to support real-time applications for autonomous driving. In the considered scenario, autonomous vehicles sense the driving environment periodically and offload the sensor data to an edge server. Due to the mobility of vehicles and non-negligible computing latency, vehicles cannot possess computing results that accurately match the current driving conditions. We calculate the mismatch by evaluating the vehicle traveled distance since the last data offloading before receiving the latest result delivery from the edge server. Our objective is to minimize the above traveled distance of vehicles by scheduling the computing resource at the edge server. To achieve the objective, we formulate the scheduling problem into a Restless Multi-Armed Bandit (RMAB) problem and propose a Whittle index-based stochastic scheduling scheme. To cope with the dynamics of vehicular mobility, we develop a novel Deep Reinforcement Learning (DRL) method to determine the indexes for vehicles. The proposed learning-based scheme can support task scheduling for a large number of MUDs with low computing and communication overheads. Simulations show that the proposed task scheduling scheme enables the edge server to deliver computing results promptly while adapting to the time-variant vehicular mobility in real time.

2) We investigate how an edge server provides computing services in an energy-efficient manner, given energy and computing constraints. A UAV plays the role of an edge server to collect and preforming computing tasks offloaded by IoT nodes in remote areas. We aim to maximize the energy efficiency of the UAV-mounted cloudlet by jointly optimizing UAV trajectory planning, computation offloading and task scheduling, given the energy and computing constraints at MUDs and the UAV. The corresponding optimization problem is non-convex fractional programming. We first adopt the Dinkelbach algorithm and the successive convex approximation (SCA) technique to transform the problem into a solvable form. To cope with the complexity of the problem, we further decompose the optimization problem into multiple subproblems to be solved in a distributed and parallel manner. With problem decomposition, both MUDs and the UAV can achieve optimal resource allocation results cooperatively with less local information sharing, such as the trajectory of the UAV. Furthermore, we extend the proposed computing resource management strategy to accommodate the scenario in which the knowledge of the mobility of MUDs is limited. A spatial distribution estimation technique is adopted to predict the location of ground users so that our proposed method can also be implemented given device computing demand distribution. Simulation results demonstrate that the proposed resource management strategy can maximize the computation capability of the UAV-mounted cloudlet with lower energy consumption.

3) We investigate how edge servers cope with the high mobility of MUDs to provide seamless computing services by jointly design computation offloading, task scheduling, and mobility management policies. We propose a computing collaboration framework to provide reliable low-latency computing for IoV applications. In the framework, multiple edge servers are selected, based on the mobility of the vehicles, to process the offloaded tasks cooperatively. In addition, the proper edge servers are proactively chosen to deliver the computing results back to the vehicles to avoid service interruption. Under this framework, we propose a novel task offloading and computing approach that reduces the overall computing service latency and improves service reliability. In the first step, we propose a task partition and scheduling algorithm (TPSA) that schedules the computing tasks offloaded to the edge servers. The algorithm achieves a near-optimal task scheduling solution for the non-convex integer problem with low time complexity. Furthermore, we adopt a DRL approach to determine the computation offloading policy, which selects the edge servers to receive tasks offloaded by vehicles, compute the tasks, and deliver the results of the tasks back to vehicles. The proposed model-free approach yields an optimal offloading policy while adapting network dynamics in a complex urban transportation network. With the proposed DRL-assisted computation offloading and task scheduling algorithm, the service cost, which includes the computing latency and service failure penalty, can be minimized through intelligent collaboration among edge servers. The simulation results demonstrate the effectiveness of the proposed collaborative computing approach in reducing service costs under highly dynamic environments.

1.5 Thesis Outline

The remainder of the thesis is organized as follows: In Chapter 2, we provide a comprehensive review on the state-of-the-art computing resource management strategies for MEC. In Chapter 3, we design an adaptive computing task scheduling scheme to support real-time computing services for autonomous driving with MEC. In Chapter 4, we develop an energy-efficient computing solution for a UAV-mounted cloudlet given network constraints. The proposed optimization approach jointly optimizes the UAV trajectory, computation offloading, and task scheduling in a scalable manner. In Chapter 5, we investigate computing resource coordination among multiple edge servers to cope with the high mobility of MUDs and propose an AI-based collaborative computing solution in MEC. Finally, we conclude the thesis and discuss future research works in Chapter 6.

Chapter 2

Background and Literature Review

This chapter presents the background of MEC and survey state-of-the-art computing resource management strategies for MEC in three aspects: computation offloading, task scheduling, and mobility management.

2.1 Mobile Edge Computing

As the number of MUDs is growing exponentially in the network, the centralized cloud computing architecture gradually shows its disadvantages on providing on-demand computing services due to the severe communication and computing latency. Meanwhile, the enormous and frequent data exchanges between MUDs and remote cloud servers could cause backhaul and core network congestion, thus degrading computing service performance. To push the traffic towards the network edge, the concept of MEC was presented in [6]. MEC defines a new platform to provide Information Technology (IT) and computing service within the radio access network near MUDs. Taking the advantage of ultra-dense edge node deployment, a vast amount of computing resources on network edge can be utilized to reduce transmission and computing time. The main advantages of MEC are summarized as follows:

1) *Low Latency*: The computing service delay consists of the two main components: communication delay and computing delay. Compared to MCC, MEC can provide low latency computing services for users in multi-fold. First, the MEC server is located close to the user device (typically within 1km), while the distance between the

device and the MCC data center is from tens of kilometers to that cross continents [9]. MEC has the advantage to analyze and process the offloaded data with short propagation delay in communication. Furthermore, MEC isolates the offloading process from the core network, and the one-hop communication between the edge server and MUDs can greatly cut the communication latency. Last, although the remote cloud has ample computing resources, resources have to be shared among a great number of users, such that the computing delay of MCC may higher than the delay of MEC when the computation loads are high in the cloud server.

2) *Mobile Energy Saving*: Although most MUDs have a certain computation capability to process computing tasks locally, the on-board energy of MUDs is too strained to execute compute-intensive applications. MEC provides a promising approach to save on-board energy of MUDs by allowing computation offloading for them.

3) *Context Awareness*: Edge servers can leverage the proximity to MUDs to gather real-time information and features of users, such as locations and MUD behaviors. The information is able to help edge servers to make smart and user-centric decisions for computation offloading and deliver context-aware mobile computing services.

Moreover, the advantages of MEC also include low computing service deployment costs [6]. Therefore, MEC is suitable to support emerging mobile applications which require real-time and location-aware computing. The comparison between MCC and MEC is summarized in Table 2.1 [9] [28]. Note that fog computing has an overlapping concept with MEC. Both aim to push the computing processes closer to the user end. However, in terms of the computing system structure, fog computing is a centralized computing paradigm that processes the offloaded tasks at the local area network level. In contrast, the computing server is physically close to MUDs in MEC. The intelligence, communication capability, and processing power are pushed to the radio access network distributively [28]. We only focus on MEC in this thesis.

2.2 Resource Management for MEC

Low-latency and energy-efficient computing requires effective resource management in MEC. In this section, we present comprehensive reviews on resource management from three aspects: computation offloading, task scheduling, and mobility management. Due to the coupling between computation offloading and scheduling, both communication and computing resources should be jointly considered.

Table 2.1: Comparisons of MCC and MEC systems

	MCC	MEC
Sever hardware	Large-scale data centers with a large number of highly-capable servers	Small data center with moderated resource
Server location	Installed at dedicated buildings	Co-located with wireless gateways, WiFi routers, LTE base stations, and other access points
Distance to MUDs	Long (may across continents)	Short (tens to hundreds of meters)
Backhaul usage	Frequent use	Infrequent use
Deployment	Centralized	Decentralized
Network latency	$\geq$ 100 ms [27]	10 - 20 ms
Computational power	Sufficient	Limited
Applications	Delay-tolerant and compute-intensive applications, e.g., online social networking, and mobile health	Latency-critical and compute-intensive applications, e.g., augmented reality, and self-driving.

2.2.1 Computation Offloading

In computation offloading, the first major research issue is to determine the portion of computation loads to be offloaded to edge servers. There are two types of computation offloading: binary offloading and partial offloading [29]. On the one hand, in binary offloading, the whole computing task of an MUD should be handled by one agent, i.e. the task is either run on a local device or offloaded to an edge server. A binary offloading decision should be made to identify whether MUDs should offload their computation loads. This usually results in a mixed-integer optimization problem. In [30], Mao *et al.* utilize an online Lyapunov-based method to determine binary offloading decisions for multiple MUDs given the constrained onboard energy of MUDs and the limited computing resources of edge servers. Wang *et al.* solve the corresponding binary offloading problem iteratively and obtain a near-optimal solution in [31]. On the other hand, for some computing tasks, the dependencies among subtasks are weak, which allows the tasks to be subdivided. A computing task generated by a MUD can be divided into several sub-tasks that can be executed by different network entities, i.e., local MUDs, edge servers, and the cloud server. As the development of microservices architecture [32], such task partition becomes widely adopted in edge and cloud computing, which facilitates partial offloading to MEC. In partial offloading, a partition ratio between the offloaded bits and the local computing bits is aimed to be optimized, with the objective of minimizing computing latency by parallelizing computing between the edge server and MUDs. In [33], Kuang *et al.* propose partial offloading scheduling and power allocation algorithms to jointly minimize the overall computing delay according to the characteristics of the computation loads. The offloading ratio optimization is also investigated in [34–36] considering different computing service requirements.

The second major research issue in computation offloading is to decide which edge servers to offload. In heterogeneous wireless networks, the computation offloading problem can be difficult due to the overlapped communication coverage among edge servers. In [37], a computation offloading method for a heterogeneous wireless network is proposed for MCC, in which the heterogeneity of edge nodes mainly provides the diversity of communication links in computation offloading. In MEC, since edge nodes are endowed with computation capability, both communication and computing resources on edge servers should be analyzed to determine the association between MUDs and edge servers. In [38], Cheng *et al.* investigate MUD-edge server associations to jointly minimize the computing delay, user energy consumption, and the server computing cost under a space-air-ground integrated network. A model-free approach is proposed in the work to find an optimal offloading solution in a complex network environment. In [39], Liu *et al.* consider the communication link quality and server computation capability when selecting edge servers for MUDs. In [40, 41], Rodrigues *et al.* investigate transmit power control and service migration policy to balance the computation load among edge servers and reduce the overall computing delay accordingly. Moreover, the mobility of MUDs affects the time duration of association between MUDs and edge servers. In [42], Saleem *et al.* investigates how to dynamically changing the MUD-server association for a MUD according to its movement trajectory. In [43], Sun *et al.* adopt an online learning algorithm, *i.e.*, multi-armed bandit, to determine the computing and communication association among vehicles according to the moving trajectory of vehicles.

Furthermore, in computation offloading, communication resources should be allocated accordingly to enable prompt and energy-efficient computing. Since the input data size of computing services is usually large than the conventional services, computation offloading would result in non-negligible offloading latency. The latency is jointly determined by transmit power, bandwidth, and channel fading, In [44], Peng *et al.* develop a learning-based approach to allocate communication bandwidth and computing units to facilitate the computation offloading of autonomous vehicles. In [45], Liu *et al.* investigate millimeter-wave cellular networks in conjunction with MEC to maximize the throughput in VR video delivery thereby improving QoE of VR users. As communication resource allocation depends on MUD-server associations and offloaded computation loads, a comprehensive computation offloading policy should evaluate all three of these aspects, i.e., task partition or binary offloading decisions, the association between MUDs and edge servers and resource allocation for offloading computing tasks.

2.2.2 Task Scheduling

The edge server needs to schedule the execution order of the computing tasks after computing tasks have been offloaded. The tasks scheduled to be executed first would have a shorter processing time; otherwise, they would experience queuing delay for acquiring computing resources at the edge server. The works [38,39,46] apply a first-in-first-out computing strategy to accommodate the computing tasks from multiple MUDs with limited computing units on the edge server. Meanwhile, altering the order of task execution would help to substantially improve the QoE of MUDs across the network.

Firstly, computing task execution is energy-consuming. The energy consumption may increase cubically as computing intensity increases, and, therefore, the unbalanced computation loads result in inefficient energy consumption. In [47,48], the works aim to balance the computation loads executed over time to reduce computing consumption in a UAV-mounted cloudlet. Furthermore, considering energy harvesting devices, the works [49–51] schedule computing tasks to avoid energy outages based on the status of energy buffers in MUDs.

Secondly, MUDs with different characteristics may have different sensitivity on computing service performance, such as computing delay and energy consumption. Therefore, the computing tasks can be scheduled according to the characteristics of MUDs, thereby improving the QoE of MUDs for the whole system. Considering different computation loads in computing tasks offloaded by MUDs, in [33,52–54], task scheduling, *i.e.*, ordering the task execution sequences, is evaluated to maximize QoE of all MUDs. In those works, the task scheduling problem is formulated into a mixed-integer programming problem, and heuristic algorithms are proposed to obtain near-optimal solutions efficiently. In [55], an edge server schedules the offloaded computing tasks according to the deadline of the tasks offloaded by MUDs and the location of the MUDs, and the work proposes heuristic algorithms to schedule computing tasks with different network scenarios.

2.2.3 Mobility Management

The above state-of-the-art works mainly focus on the computing resource management for an edge server with a fixed association between MUDs and edge servers. However, for MUDs with high mobility, such as vehicles, the associations are time-varying due to the limited communication coverage of an edge server, which necessitates the collaboration of multiple edge servers to execute the computing tasks for an MUD. Collaboration can be categorized into two types. First, a task can be partitioned into multiple subtasks and distributed to different edge servers, which reduces computing time by parallelizing task processing, thereby preventing service interruptions. Second, by using VM migration, the tasks can be migrated to other edge servers according to the moving trajectory of the MUDs, i.e., service migration.

In the aspect of collaboration among edge servers, the works [56, 57] study cooperative computing among multiple cloud service providers. They considered the resources in cloud service providers as a resource pool, and the common objective of the works is to maximize the global computing service capacity. The works [58, 59] investigate the cooperation among edge servers. In [60], Laredo *et al.* consider computing resource sharing among edge servers to minimize the energy consumption for the whole system, in which load balancing is the primary motivation to formulate the cooperation of computing. Moreover, to address the mobility, the works [61] and [62] consider that MUDs, i.e., vehicles, divide and offload the computing tasks to multiple servers according to the predicted traveling traces. Vehicle-to-vehicle communication is used to disseminate the computing result if the edge server cannot connect with the vehicle at the end of a computing session. Overall, the cooperation among edge servers significantly improves the computing service experience. However, the communication cost is also raised due to the communication overhead during the collaboration among servers. The trade-off between the extra communication cost and computing performance improvement should be considered in the cooperative computing policy design.

Furthermore, service migration aims to deal with the interruption of services caused by vehicles' departures from a server's communication range. According to a MUD's moving trajectory, ongoing computing services may be moved to another edge server that will associate with the MUD in the future. Migration decisions are made according to a variety of factors from the environment, including the communication link quality, computation capability, and user mobility. The works [63, 64] first propose service migration schemes among federated cloud data centers, i.e., Follow Me Cloud (FMC), which also can be extended to the MEC system. In these works, the ongoing service can be migrated to another selected server as the corresponding MUD moves out. Based on the random walk model, the Markov Decision Process (MDP) for determining the migration policy is developed in [65]. Compared to FMC, the MDP method can provide a proactive decision on whether a service should be migrated according to the mobility of MUDs. Similar proactive service migration strategies are also investigated in [66] and [67]. Overall, computing service migration provides a practical policy for implementing MEC in a highly dynamic environment. However, it increases the complexity of resource management. An effective migration policy needs to consider the channel condition, server computation capability, MUD locations, migration overhead dynamically, etc. [9]. One possible approach is to generate a proactive service migration policy using AI technology, which may achieve good performance by learning the users' mobility and channel condition.

2.3 Summary and Discussions

In this chapter, we have surveyed the existing literature for resource management for MEC from three aspects: computation offloading, task scheduling, and mobility management. Through this literature review, we identify the limitations of current studies and develop efficient resource management strategies in MEC.

Firstly, for the real-time safety-related computing applications in autonomous driving, vehicles may require frequent computation offloading and timely delivery of computing results [68,69]. In such a scenario, a customized computing scheduling scheme is important yet has not been investigated. There is a need for deeper analysis on how mobility and computing scheduling impacts the performance of autonomous driving.

Secondly, most existing resource management strategies only consider resource allocation in MEC with fixed edge infrastructures. To provide on-demand services for remote IoTs, this thesis studies edge computing supported by a UAV-mounted cloudlet, which introduces dynamic channel conditions and mechanical operation constraints. Moreover, the coupling of policies among UAV trajectory, computation offloading, and task scheduling make the problem complex and hard to solve in a scalable manner.

Thirdly, in terms of mobility management, both service migration and cooperation strategies have their limitations when avoiding service discontinuity. For service migration, frequent VM migration results in service interruptions and increases computing time. For service cooperation, a high offloading overhead would occur if the computing tasks are divided at too many sub-tasks and executed into different edge servers. The limitations motivate us to develop an effective approach to achieving service reliability improvement while adapting MUD movement with low overhead.

Chapter 3

Computation Task Scheduling for Autonomous Driving

In this chapter, we investigate computing task scheduling to support real-time applications in autonomous driving. Taking into account the characteristics of the real-time applications, we introduce a novel performance metric to evaluate the QoE of computing services considering vehicle mobility, i.e., Age-of-Computing-Results (AoR). A lower AoR means that the vehicle receives accurate and timely computing results based upon its location. We first propose an index-based task scheduling scheme to optimize the QoE of MUDs, i.e., minimize the average AoR of all vehicles, given limited computing resources at an edge server. Furthermore, to address the dynamics of vehicle mobility, we develop a DRL algorithm that helps the index-based task scheduling scheme to make effective and adaptive scheduling decisions with low complexity. Specifically, Section 3.1 introduces the background and motivation of the work, in which the contribution of the work is addressed. Section 3.2 describes the system model of the considered scenarios. Section 3.3 introduces the proposed index-based scheduling scheme for the synchronous computation offloading scenario. The learning approach that matches the index policy is presented in Section 3.4. Section 3.5 introduces the scheduling scheme for the asynchronous scenario. Simulation results are presented in Section 3.6.

3.1 Background and Motivation

Safety is one of the main focuses in autonomous driving related industries. Various techniques have been adopted to improve safety, e.g., embedding advanced sensors in vehicles and developing precise perception using the sensor data. Such safety measures usually yield a large amount of sensor data to be processed with low latency [70–72], which could be too demanding for local in-vehicle processing given the usually limited onboard computation capability. To facilitate the safety measures, MEC has emerged as an approach to provide additional computing power with low processing delay [73]. In such an approach, vehicles can offload their sensor data to a proximal edge server, such as a base station or a roadside unit, for fast data processing.

Despite the potential of MEC in enabling low-latency computation offloading for autonomous driving, many challenges exist for MEC to support real-time safety-related computing services. In most safety applications, such as localization and obstacle avoidance, vehicles may need to offload sensor data to the edge server and require computing results [68, 69]. Because of the mobility and computing delay, a vehicle would have traveled some distance between the instant of offloading sensor data to the edge server and the instant of receiving the computing result from the edge server. Evidently, it is important to reduce the above traveled distance as much as possible, considering that the computing result may involve the vehicle's position at the instant of sensor data collection. Otherwise, the computing result may no longer be accurate for real-time applications. Consider a vehicle traveling at 50 km/h as an example. If the vehicle receives the computing result 2 seconds after sensor data offloading, the gap in the distance between the real-time location and the location at the instant of offloading would be 28 m. This gap is larger than the reaction distance at 50 km/h, which is 21 m [74]. The challenge in reducing the traveled distance is that the edge server may have limited computation units, which must be shared among all vehicles in its proximity. As a result, the delay from the sensor data offloading to the result delivery may increase with the number of vehicles, and so does the traveled distance. Without proper scheduling, the delay can become excessive [75].

There are extensive existing works on computing resource scheduling in MEC [52, 54]. Generally, the main objective of the proposed schemes is to minimize the computing latency. However, most of the works focus on myopic computing service scheduling, which considers the existing computation load at the edge server but not future computing service demands. As vehicles may collect sensor data and request computing service from time to time, developing a long-term proactive scheduling scheme is important. An effectual scheduling scheme should provide timely and frequent result delivery for vehicles so that the computing results can reflect their real-time status, e.g., position, as much as possible.

Another important requirement for effectual scheduling is the capability to adapt to the dynamics of vehicular mobility. Different vehicles may travel at different speeds, which can result in different service delay tolerance for real-time applications. Consider obstacle avoidance as an example. The edge server should schedule high-speed vehicles with high priority to minimize their traveled distance, and low-speed vehicles may have low priority. In addition, the speed of a vehicle may change over time, which may further complicate long-term scheduling as such change is not known in advance. There are existing works on resource allocation in MEC considering the mobility dynamics, e.g., the offloading bandwidth assignment in [44,76], computing resource allocation in [43,77], and cooperative computing in [62]. However, mobility has a more significant impact on the performance in our considered problem since the traveled distance, which we aim to minimize, is dependent on vehicle mobility.

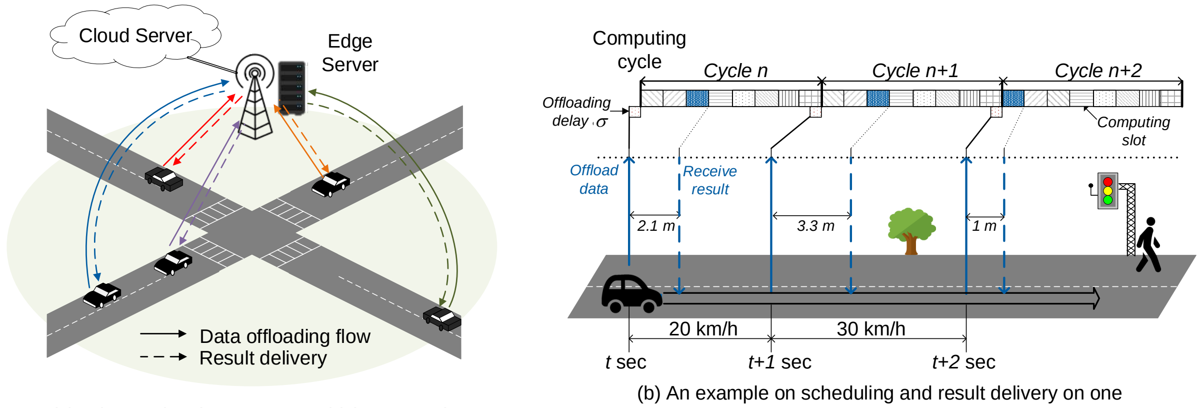
In this chapter, we design a computing resource scheduling scheme at the edge server for real-time applications in autonomous driving, considering the vehicle mobility dynamics. Vehicles periodically offload sensor data, referred to as observations, to the edge server for processing, while the edge server determines the processing order considering its computation capability and the unknown vehicle mobility dynamics. We define the AoR of a vehicle as its traveled distance since the last data offloading before receiving the latest result delivery from an edge server. Our objective is to minimize the expected AoR of vehicles to deliver computing results timely. We formulate the long-term scheduling problem as a RMAB problem and propose a Whittle index-based scheduling scheme. Two offloading scenarios are investigated in the thesis: *synchronous*, in which all computing requests arrive simultaneously, and *asynchronous*, in which the computing requests arrive arbitrarily.

The main contributions of this work are as follows:

1) We design a novel computing resource scheduling scheme for the edge server to support autonomous driving, targeting at minimizing the AoR while considering the vehicle mobility dynamics. The scheduling scheme can support a large number of computing tasks with low complexity.

2) We obtain the Whittle index of the formulated RMAB problem in closed form and prove the indexability of the scheduling problem. The index can reveal the value of scheduling each computing request and guide the computing policies in both the synchronous and the asynchronous scenarios.

3) We exploit the DRL method to estimate the unknown mobility dynamics of vehicles in the future according to their mobility dynamics in the past. The learning process does not rely on the scheduling decisions and can be pre-trained either at the edge server or the vehicles.



(a) Edge-assisted autonomous driving network

(b) An example on scheduling and result delivery on one processor at an edge server

Figure 3.1: Edge-assisted autonomous driving model.

3.2 System Model

In this section, we present the system model of the considered problem.

3.2.1 Network Model

Consider a vehicular network for autonomous driving shown in Fig. 3.1(a). Vehicles sense the surrounding environment periodically (e.g., for localization, obstacle detection, and object tracking), using sensors such as light detection and ranging and cameras. The sensor data is referred to as observations, which can be offloaded to the closest edge node (e.g., roadside unit or base station) for processing. The sensing process occurs periodically for each vehicle, and each period is referred to as a sensing cycle. The processing of an observation offloaded to an edge node in a sensing cycle is referred to as a computing task. In the sequel, we focus on a particular edge server.

An edge server is located in the edge node and may have one or more processors. The server determines the execution order and processes the received computing tasks periodically at each processor, and each period is referred to as a computing cycle. We assume that each processor handles a specific type of computing tasks, which has the same processing time. For brevity, we focus on the scheduling of computing tasks in one processor, as the extension to multiple processors is straightforward.

The time length of a computing cycle is T, which is divided into K computing slots. The length of a slot is the processing time for a computing task, denoted by ω. The length of a computing cycle depends on vehicles' AoR requirements and their sensing frequency. A shorter computing cycle can lead to a lower AoR since the likelihood of scheduling a vehicle in a certain period can be increased; however, too short computing cycle raises the vehicles' sensing frequency, which may result in redundant task offloading and high communication resource consumption. The objective of scheduling scheme on the edge server is to allocate computing tasks to the slots in each computing cycle. The edge node is connected to a cloud server through a backhaul link. As a controller, the cloud server determines computing cycle length T and coordinates the computing resources among all edge nodes.

If a task from a particular vehicle and a particular sensing cycle is not scheduled until the next task of the vehicle is generated and to be offloaded to the edge server, it becomes outdated and discarded, and the server schedules the newly sensed task corresponding to a subsequent and up-to-date observation from the same vehicle. After processing the computing task, the edge server delivers the computing result back to the corresponding vehicle. Therefore, the computing schedule also affects the order of result delivery. Since we consider real-time applications and the cycle length is in seconds, the probability that the vehicle travels out of the communication range of the edge server is neglected. For the case that the vehicle travels out of the communication range during a computing cycle, multiple edge nodes can cooperatively deliver the result back to the vehicle, which has been discussed in our previous work [78]. We ignore the transmission delay on result delivery since the downlink transmission time is neglectable compared to the computing time [44, 79].

Due to the mobility and computing latency, the traveled distance during offloading and edge computing is nonzero for any vehicle and any task. An example of computing scheduling and result delivery is shown in Fig. 3.1(b), where σ represents the offloading time of a task.[1] A vehicle offloads the computing tasks once the task is scheduled in the corresponding cycle. The speed of the vehicle remains constant within each sensing cycle but may vary in different cycles. Because of the changing speed of the vehicle, its traveled distance in two cycles can be different. This is true even if the vehicle is assigned to the same slot in two different cycles. As shown in the figure, the traveled distances of the vehicle in cycle n and $n + 1$ are different although the vehicle is assigned to the third slot of both the computing cycles. In the sequel, we focus on minimizing the expected traveled distance from the position implied by the newest received computing results, *i.e.*, minimizing AoR.

[1]As all tasks handled by the same processor is of the same type, we assume that the offloading time is constant and the same for all vehicles.

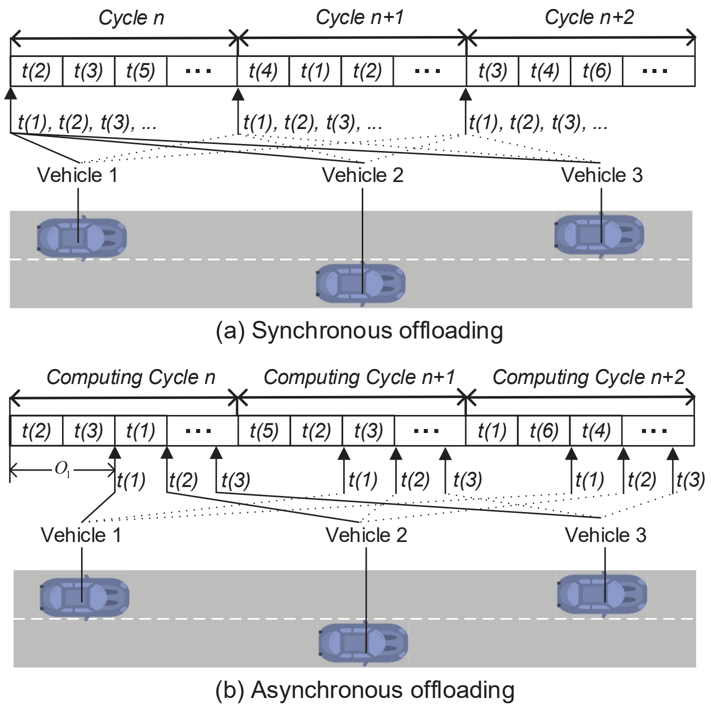


Figure 3.2: An illustration of the scheduling scenarios, where $t(i)$ represents the computing task from vehicle i.

3.2.2 Computing Scheduling Scenarios

Based on the task arrival pattern of vehicles in each computing cycle, two computing scheduling scenarios are analyzed in this work:

- **Synchronous offloading**: In this scenario, all vehicles have the same sensing cycle. Vehicles offload their sensor data to the edge server at the beginning of a computing cycle if the vehicles are scheduled in the corresponding sensing cycle. In addition, the computing cycle at the edge server is also the same in length as the sensing cycle. This is illustrated in Fig. 3.2(a). The edge server decides the processing order of the tasks received at the beginning of the cycle.

- **Asynchronous offloading**: In this scenario, vehicles offload their observations to the edge server at arbitrary instants. This is illustrated in Fig. 3.2(b). The time duration from the beginning of the computing cycle to the offloading time of vehicle i is denoted by O_i. For simplicity, we assume that a computing cycle at the edge

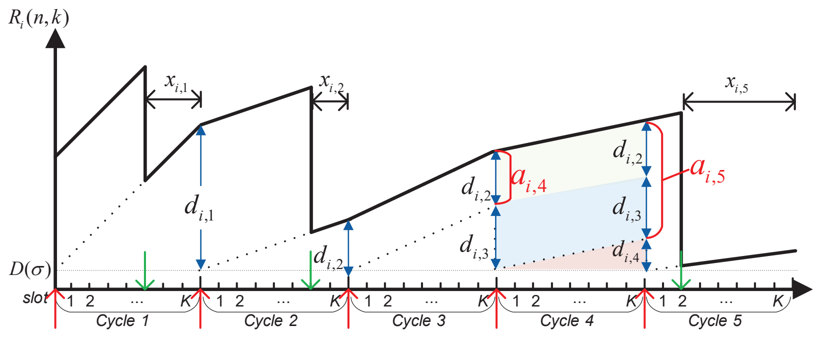


Figure 3.3: The evolution of AoR of vehicle i in five computing cycles, where red arrows represent the task offloading completion instant of a vehicle, and green arrows represent result delivery instant.

server is the same in length as the sensing cycle in this scenario as well. However, the proposed scheme can be readily extended even if the assumption does not hold. Since the sensing and computing cycles are not aligned, a vehicle may have two tasks from adjacent sensing cycles scheduled in the same computing cycle, one to be offloaded in the previous computing cycle and the other to be offloaded in the current computing cycle.

In the remainder of the chapter, we mainly focus on the synchronous offloading. In Section 3.5, we extend the scheduling scheme to the scenario of asynchronous offloading.

3.2.3 Age of Computing Results

The evolution of AoR of a vehicle in the synchronous offloading scenario is shown in Fig. 3.3. The real-time AoR for vehicle i in cycle n and computing slot k is denoted by $R_i(n,k)$. Let $d_{i,n}$ denote the traveled distance of the vehicle in cycle n. As the example shown in Fig. 3.3, vehicle i is scheduled in the cycles 1, 2, and 5. In these scheduled cycles, when the task computing is completed, the AoR drops to the level of the dashed lines, which represent the traveled distance from the starting of this cycle. Otherwise, if the task is not scheduled in the corresponding cycle, the AoR accumulates due to adding the traveled distance within this cycle.

The real-time AoR at the beginning of the n-th cycle, $R_i(n,0)$, can be written as

$$R_i(n,0) = a_{i,n} + d_{i,n-1} + D(\sigma), \tag{3.1}$$

where $d_{i,n-1}$ is the traveled distance in the previous cycle $n-1$, $D(\sigma)$ is the traveled distance in the offloading time, and the non-negative variable $a_{i,n}$ is a carryover component contributed to AoR. It can be written as

$$a_{i,n+1} = \begin{cases} a_{i,n} + d_{i,n-1}, & \text{if } i \in \{\mathcal{V}_n \backslash \mathcal{S}_n\}, \\ 0, & \text{if } i \in \mathcal{S}_n, \end{cases} \tag{3.2}$$

where $\mathcal{V}_n$ denotes the set of the vehicles that offloaded their observations in computing cycle n. The sets $\mathcal{S}_n$ and $\{\mathcal{V}_n \backslash \mathcal{S}_n\}$ denote the set of vehicles that are scheduled and not scheduled in cycle n, respectively. Equation (3.2) shows that if the computing is scheduled in the n-th cycle, the carryover component will be reset as zero; otherwise, the carryover component accumulates due to adding the vehicle traveled distance in the previous cycle. In the example shown in Fig. 3.3, $a_{i,2}$ and $a_{i,3}$ are zeros since the tasks are scheduled and processed in cycles 1 and 2, respectively. However, the AoR starts accumulating since cycle 3. Thus, $a_{i,4} = d_{i,2}$, and $a_{i,5} = d_{i,2} + d_{i,3}$. Furthermore, the number of time slots from the task finishing instant to the end of the cycle is denoted as $x_{i,n}$, where $x_{i,n} = 0$ if $i \in \{\mathcal{V}_n \backslash \mathcal{S}_n\}$ and $x_{i,n} \geq 0$ otherwise.

The time average AoR of vehicle i in this process can be represented by the area under age $R_i(n,k)$ in the age evolution graph, as shown in Fig. 3.3, normalized by the overall time length. The area under age $R_i(n,k)$ in cycle n is denoted by $Q_{i,n}$ and can be represented as follows:

$$Q_{i,n} = (a_{i,n} + d_{i,n-1})(T - x_{i,n}\omega) + \frac{1}{2} d_{i,n} T + D(\sigma)T. \tag{3.3}$$

The Expected Age-of-Computing-Results (EAoR) for all vehicles in the communication range of the edge server is:

$$\begin{aligned} E_R &= \lim_{N\to\infty} \frac{1}{TN} \sum_{n=1}^{N} \frac{1}{|\mathcal{V}_n|} \sum_{i\in\mathcal{V}_n} Q_{i,n}, \\ &= \lim_{N\to\infty} \frac{1}{TN} \sum_{n=1}^{N} \frac{1}{|\mathcal{V}_n|} \sum_{i\in\mathcal{V}_n} (a_{i,n} + d_{i,n-1})(T - x_{i,n}\omega) + \Lambda_{i,n}, \end{aligned} \tag{3.4}$$

where E_R denotes the value of EAoR, and $\Lambda_{i,n}$ represents the constant term of the AoR for vehicle i in cycle n, where

$$\Lambda_{i,n} = \frac{1}{2} d_{i,n} T + D(\sigma)T.$$

Neglecting the constant component in (3.4), the objective function for minimizing EAoR can be represented as follows:

$$\begin{aligned}&\min_{\pi\in\Pi}\mathbb{E}_{\pi}\Big[\frac{1}{TN}\sum_{n=1}^{N}\frac{1}{|\mathcal{V}_n|}\sum_{i\in\mathcal{V}_n}(a_{i,n}+d_{i,n-1})(T-x_{i,n}\omega)\Big], \qquad (3.5)\\ &\text{s.t.} |\mathcal{S}_n|\leq K, \forall n,\\ &\quad 0\leq x_{i,n}<K, x_{i,n}\in\mathbb{Z}, \forall i,n,\end{aligned}$$

where π and Π represent the optimal policy and the feasible policy set, respectively. Since the length of a computing cycle is much shorter than the time for a vehicle leaving the service coverage of an edge server, we apply the long-term policy for vehicle $i \in \mathcal{V}_n$. To solve the scheduling problem, we first determine which tasks should be scheduled in the cycle when $K < |\mathcal{V}_n|$, *i.e.*, finding $\mathcal{S}_n$, based on the maximum AoR by setting the computing slot at the very end of the corresponding computing cycle ($x_{i,n} = 0$). The problem is a long-term optimization problem since the age of a vehicle will accumulate if the tasks are not scheduled properly. After the tasks to be processed in the cycle are selected, we determine when to process the selected tasks, *i.e.*, finding $x_{i,n}$, which only affects the instantaneous age in the corresponding cycle.

3.3 Restless Multi-armed Bandit Formulation and Index Policy

To solve the optimization problem in (3.5), a myopic policy is to schedule the vehicle with the highest AoR in each computing cycle. As proved in [80], in a symmetric network, in which all vehicles always have the same mobility, such policy follows a round-robin pattern and attains the minimum AoR. However, the optimality of the myopic policy would be lost if vehicles have diversified and time-variant mobility. Therefore, considering the mobility dynamics, we reformulate the AoR minimization problem into an RMAB problem. We then propose an index-based scheduling scheme, which assigns an index to each vehicle to be scheduled for measuring the value to activate an arm at a particular state. A Whittle's index policy, which is an optimal policy to a relaxation of the RMAB problem [81, 82], is utilized to schedule the processing order for the synchronous offloading scenario.

3.3.1 Restless Multi-armed Bandit Formulation

Different from classic Multi-Armed Bandit (MAB), RMAB considers a generalized bandit process, in which the states of arms can evolve over time even when the arms are not activated. Moreover, instead of selecting only one arm in a decision step in MAB, RMAB can activate multiple arms in a step. In our case, a cycle is a decision step and can schedule K tasks in each decision step. To determine which tasks to schedule in a computing cycle, *i.e.*, the set $\mathcal{S}_n$, we assume $x_{i,n} = 0$ at first, in which the maximum EAoR is minimized in this step.

The state of a vehicle consists of two parts: the current carryover component, *i.e.*, $a_{i,n}$ and the past vehicle mobility profile. We use the traveled distance of the vehicle for each cycle in past W cycles, *i.e.*, $\delta_{i,n} = \{d_{i,n-W}, \dots, d_{i,n-1}\}, \forall i \in |\mathcal{V}_n|$, as the past mobility profile to predict vehicle future mobility. Denote $\delta_{i,n}^{-1}$ as the vehicle traveled distance in past $W-1$ cycles, *i.e.*, $\delta_{i,n}^{-1} = \{d_{i,n-W+1}, \dots, d_{i,n-1}\}$. Let a binary variable $u_{i,n}$ indicate the action of the RMAB problem for vehicle i in cycle n. If the task is scheduled in the cycle, *i.e.*, $i \in \mathcal{S}_n$, the arm i is activated, and $u_{i,n} = 1$. Otherwise, the arm i is not activated, and $u_{i,n} = 0$.

According to the evolution of the carryover component $a_{i,n}$ in (3.4), when action $u_{i,n} = 1$, the state transition probability of vehicle i can be obtained as follows:

$$P(a_{i,n+1} = 0, \{\delta_{i,n}^{-1}; d_{i,n}\} | a_{i,n}, \delta_{i,n}) = p(d_{i,n} | \delta_{i,n}), \tag{3.6}$$

where $p(d_{i,n}|\delta_{i,n})$ is the probability that the vehicle travels $d_{i,n}$ distance in cycle n, given past mobility profile $\delta_{i,n}$. On the other hand, when action $u_{i,n} = 0$, the state transition probability of vehicle i can be obtained as follows

$$P(a_{i,n+1} = a_{i,n} + d_{i,n-1}, \{\delta_{i,n}^{-1}; d_{i,n}\} | a_{i,n}, \delta_{i,n}) = p(d_{i,n} | \delta_{i,n}). \tag{3.7}$$

The Whittle index policy introduces a service charge whenever an arm makes active action. Let $C(\delta_{i,n})$ represent the service charge for vehicle i given the mobility profile $\delta_{i,n}$. The objective of introducing the service charge is to make the passive and active actions equal in a long-term cost [81]. Although there is no instantaneous cost when the arm is passive in a step, it would result in an accumulation of age and a service charge in the future.

3.3.2 Indexability and Index Policy

In the Whittle index policy, a service charge is the Lagrange multiplier of the RMAB constraint. The original RMAB problem is relaxed and decoupled into subproblems for individual arms such that the service charge for each arm can be evaluated separately. The subproblem for each arm is referred to as decoupled model, and the service charge obtained by the decoupled model only depends on the characteristic of the arm itself [81]. Although problem decomposition would relax the original RMAB problem, a near-optimal solution can be obtained by such decoupled model [80, 81]. However, the service charge cannot be regarded as the Whittle index unless the considered RMAB problem is indexable.

Denote the set of all policies that schedule vehicle i by $\Pi(i)$. To determine whether the task should be scheduled, we minimize the upper bound performance of the AoR achieved by the selection. Decoupling the objective function (3.5) for vehicle i, the new objective function for minimizing the AoR for this individual vehicle is given by

$$\begin{aligned}&\min_{\pi(i)\in\Pi(i)} \mathbb{E}_{\pi(i)}\Big[J_i\Big] \\ &\text{s.t. } J_i = \frac{1}{TN}\sum_{n=1}^{N}(a_{i,n}+d_{i,n-1})T + C(\delta_{i,n})u_{i,n},\end{aligned} \tag{3.8}$$

To simplify the notation, we neglect the vehicle index i when we evaluate the problem of the decouple model in (3.8).

Let $J_n(a_n,\delta_n)$ be the value function representing the minimum EAoR for problem (3.8) when the state is (a_n,δ_n), which can be formulated to the following dynamic programming:

$$J_n(a_n,\delta_n) = (a_n+d_{n-1})T + \min_{u_n\in\{0,1\}}\Big\{C(\delta_n)u_n + \sum_{d_n} J_{n+1}(a_{n+1},\{\delta_n^{-1};d_n\})p(d_n|\delta_n)\Big\}. \tag{3.9}$$

According to [83], the value function of the optimal policy in a finite-horizon average cost minimization problem can be represented as

$$\begin{aligned}J^*(a_n,\delta_n) &\approx h(a_n,\delta_n) + \lambda TN + o(N) \\ &= (a_n+d_{n-1})T + \min_{u_n\in\{0,1\}}\Big\{C(\delta_n)u_n + \sum_{d_n} p(d_n|\delta_n) \\ &\quad \times \big[h(a_{n+1},\{\delta_n^{-1};d_n\}) + \lambda T(N-1) + o(N)\big]\Big\},\end{aligned} \tag{3.10}$$

where λ is the optimal average cost normalized by the number of computing slots, $h(a_n, \delta_n)$ is the differential cost function representing the cost incurred when the state transits from (a_n, δ_n) to $(0, \delta_{n+1})$ for the first time, and $o(N)$ is the cost caused by not completing the execution at the end of the time horizon. All states communicate with one another, which implies that $h(0, \delta_n) = 0, \forall n$. Thus, we have differential cost function $h(a_n, \delta_n)$ as follows:

$$h(a_n, \delta_n) = (a_n + d_{n-1})T - \lambda T + \min_{u_n \in \{0,1\}} \Big\{ C(\delta_n)u_n + \sum_{d_n} p(d_n|\delta_n) h(a_{n+1}, \{\delta_n^{-1}; d_n\}) \Big\}. \tag{3.11}$$

By solving the Bellman equation (3.11), the optimal service charge $C(a, \delta)$ for vehicle i can be obtained, given Proposition 1 below.

Proposition 1. *The service charge $C(a, \delta)$ for a vehicle in state (a, δ_n) is*

$$C(a, \delta_n) = \Big\{ 2a + d_{n-1} + \mathbb{E}\big[\sum_{t=0}^{\infty} (a - D_{n+t}) \mathbf{1}(D_{n+t} \leq a) | \delta_n \big] \Big\} T, \tag{3.12}$$

where $D_{n+t} = \sum_{x=0}^{t} d_{n+x}$, and function $\mathbf{1}(x)$ indicates whether x is true or not, i.e., x is true if $\mathbf{1}(x) = 1$, and x is false if $\mathbf{1}(x) = 0$.

Proof. To obtain the service charge, similar to the approach in [80], we assume a threshold policy at first. The threshold is denoted by A. The arm is passive when the carryover component $0 \leq a_n \leq A$ and active when $a_n > A$.

According to the threshold policy, $u_n = 1$ when $a_n > A$. The following condition has to be satisfied:

$$C(\delta_n) > \sum_{d_n} h(a_n + d_{n-1}, \{\delta_n^{-1}; d_n\}) p(d_n|\delta_n), \text{ when } a_n > A. \tag{3.13}$$

Let j^+ be a positive number. The differential cost in state $(A + j^+, \delta_n)$ for the case $a_n > A$ can be simplified as

$$h(A + j^+, \delta_n) = C(\delta_n) + (A + j^+ + d_{n-1} - \lambda)T. \tag{3.14}$$

The equation shows that $h(A + j^+, \delta_n)$ monotonically increases with j^+.

On the other hand, $u_n = 0$ when $0 \leq a_n \leq A$. Assume that the vehicle speed precision level is high, and existing a state (A, δ_n) has a service charge as follows:

$$C(A, \delta_n) = \sum_{d_n} h(A + d_{n-1}, \{\delta_n^{-1}; d_n\})p(d_n|\delta_n). \tag{3.15}$$

Let j^- be a negative number that greater than $-A$. Then, the differential cost in state $(A + j^-, \delta_n)$ for the case $0 \leq a_n \leq A$ is

$$h(A + j^-, \delta_n) = (A + j^- + d_{n-1} - \lambda)T + \sum_{d_n} h(A + j^- + d_{n-1}, \{\delta_n^{-1}; d_n\})p(d_n|\delta_n). \tag{3.16}$$

Similarly, equation (3.16) shows that $h(A+j^-, \delta_n)$ monotonically increases with j^-. Given the threshold policy, the term of $h(A + j^- + d_{n-1}, \{\delta_n^{-1}; d_n\})$ can be determined according to the value of the term of $(A + j^- + d_{n-1})$. The differential cost in (3.16) in could be further derived to (3.17), where $B_{n+t} = \sum_{x=-1}^{t} d_{n+x}$.

$$\begin{aligned} h(A + j^-, \delta_n) = {} & (A + j^- + d_{n-1} - \lambda)T + P(d_{n-1} \leq -j^-|\delta_n) \sum_{d_n} \Big[(A + j^+ + d_{n-1} + d_n - \lambda)T \\ & + \sum_{d_{n+1}} h(A + j^- + d_{n-1} + d_n, \{\delta_n^{-2}; d_n; d_{n+1}\})p(d_{n+1}|\{\delta_n^{-1}; d_n\})\Big]p(d_n|\delta_n) \\ & + P(d_{n-1} > -j^-|\delta_n) \sum_{d_n} \Big[C(\{\delta_n^{-1}; d_n\}) + (A + j^- + d_{n-1} + d_n - \lambda)T\Big]p(d_n|\delta_n). \end{aligned} \tag{3.17}$$

In a similar manner, we then evaluate the differential cost for the next states recursively, which could be summarized as (3.18).

$$\begin{aligned} h(A + j^-, \delta_n) = {} & 2(A + j^- + d_{n-1} - \lambda)T + \mathbb{E}[d_n|\delta_n]T + (A + j^- - \lambda)T \sum_{t=-1}^{\infty} P(B_{n+t} \leq -j^-|\delta_n) \\ & + \mathbb{E}[\sum_{t=-1}^{\infty} B_{n+t+2}\mathbf{1}(B_{n+t} \leq -j^-)|\delta_n]T + \mathbb{E}[\sum_{t=0}^{\infty} C(\hat{\delta}_{n+t+2})\mathbf{1}(B_{n+t} > -j^-, B_{n+t-1} \leq -j^-)|\delta_n] \\ & + \sum_{d_n} C(\{\delta_n^{-1}; d_n\})p(d_n|\delta_n)P(d_{n-1} > -j^-|\delta_n), \text{ where } \hat{\delta}_{n+t} = \{\delta_n^{-t}; d_n; \ldots; d_{n+t-1}\}. \end{aligned} \tag{3.18}$$

Next, to simplify the complex form in (3.18), the term of $h(A+d_{n-1}, \{\delta_n^{-1}; d_n\})p(d_n|\delta_n)$

in (3.15) is expanded according to the threshold policy, we have

$$C(\delta_n) = (A + d_{n-1} - \lambda)T + \mathbb{E}[d_n|\delta_n]T + \sum_{d_n} C(\{\delta_n^{-1}; d_n\}) \\ \times p(d_n|\delta_n)[P(d_{n-1} > -j^-) + P(d_{n-1} \leq -j^-)]. \tag{3.19}$$

Then, we recursively apply (3.19) to represent service charge $C(\delta_n)$. Finally, we obtain the relation in (3.20).

$$C(\delta_n) = (A + j^- + d_{n-1} - \lambda)T + \mathbb{E}[d_n|\delta_n]T + \mathbb{E}[\sum_{t=0}^{\infty} C(\hat{\delta}_{n+t+2})\mathbf{1}(B_{n+t} > -j^-, \tag{3.20} \\ B_{n+t-1} \leq -j^-)|\delta_n] + \sum_{d_n} C(\{\delta_n^{-1}; d_n\})p(d_n|\delta_n)P(d_{n-1} > -j^-|\delta_n) \\ + (A - \lambda)T[\sum_{t=-1}^{\infty} P(B_{n+t} \leq -j^-|\delta_n)] + \mathbb{E}[\sum_{t=-1}^{\infty} (d_{n+t+2} + d_{n+t+1})\mathbf{1}(B_{n+t} \leq -j^-)|\delta_n]T.$$

Combining (3.18) and (3.20) together, the differential cost on state $(A + j^-, \delta_n)$ is obtained as follows:

$$h(A + j^-, \delta_n) = (A + d_{n-1} - \lambda)T + 2j^- + C(\delta_n) + \mathbb{E}[\sum_{t=-1}^{\infty} B_{n+t}\mathbf{1}(B_{n+t} \leq -j^-)|\delta_n]T \\ + j^- T P(B_{n+t} \leq -j^-|\delta_n). \tag{3.21}$$

Utilizing the property of $h(0, \delta_n) = 0$ and letting $j^- = -A$ in (3.21), the service charge $C(\delta_n)$ is:

$$C(\delta_n) = (A + \lambda - d_{n-1})T + \mathbb{E}[\sum_{t=-1}^{\infty} (A - B_{n+t})\mathbf{1}(B_{n+t} \leq A)|\delta_n]T. \tag{3.22}$$

According to equation (3.22), we further obtain the service charge for state $\{\delta_n^{-1}; d_n\}$, where

$$C(\{\delta_n^{-1}; d_n\}) = (A + \lambda - d_n)T + \mathbb{E}[\sum_{t=-1}^{\infty} (A - B_{n+t+1} + d_{n-1}) \times \mathbf{1}(B_{n+t+1} - d_{n-1} \leq A)|\delta_n]T. \tag{3.23}$$

Substituting (3.23) into (3.19) yields:

$$C(A,\delta_n) = (2A + d_{n-1})T + \mathbb{E}[\sum_{t=-1}^{\infty}(A - B_{n+t+1} + d_{n-1}) \times \mathbf{1}(B_{n+t+1} - d_{n-1} \le A)|\delta_n]T$$
$$= (2A + d_{n-1})T + \mathbb{E}[\sum_{t=0}^{\infty}(A - D_{n+t})\mathbf{1}(D_{n+t} \le A)|\delta_n]T. \tag{3.24}$$

From (3.14) and (3.21), we see that the differential cost $h(a,\delta_n)$ increases with carryover component a. Therefore, the solution of the Bellmen equation (3.11) follows a threshold policy. To evaluate the service charge of state (a,δ), we substitute A to a, and the corresponding service charge is presented in equation (3.24). □

The purpose of the service charge is to measure the value of activating an arm. A high service charge indicates that the cost of making passive action on the arm is high. If the RMAB problem is indexable, the service charge can be regarded as the Whittle index, and the Whittle index policy is to schedule the tasks with K highest service charges in a cycle. We denote set $\mathcal{P}(C)$ as the set of state (a,δ) for which the arm is passive according to the service charge $C(a,\delta)$. If the problem is indexable, the following condition should be satisfied:

Definition 1. *If an arm is indexable, set $\mathcal{P}(C)$ of the corresponding single-armed bandit process increases monotonically from the empty set $\emptyset$ to the whole state space as charge C increases from $-\infty$ to $+\infty$ [81, 84].*

The condition on indexability indicates that the optimal action of an arm can never switch from passive action to active action with an increase of C [84]. The RMAB problem is indexable only if all arms are indexable. According to the above definition, we prove the indexability of the considered RMAB problem.

Theorem 1. *In computing cycle n, vehicle i is indexable, and the Whittle index of vehicle i is identical with the service charge of vehicle i, where*

$$C_i(a_{i,n},\delta_{i,n}) = \Big\{2a_{i,n} + d_{i,n-1} + \mathbb{E}\big[\sum_{t=0}^{\infty}(a_{i,n} - D_{i,n+t}) \times \mathbf{1}(D_{i,n+t} \le a_{i,n})|\delta_{i,n}\big]\Big\}T. \tag{3.25}$$

Proof. Since both $a_{i,n}$ and $d_{i,n}$ are non-negative, there is no such state to make the service charge as a negative number. Therefore, $\mathcal{P}(C)$ is an empty set when $C_i = -\infty$. Moreover, when $\delta_{i,n}$ is fixed, C_i increases monotonically with $a_{i,n}$. An arm in state $(a_{i,n},\delta_{i,n})$ will not switch from passive to active action with the increase of C_i. If C_i is $+\infty$, $a_{i,n}$ can only be $+\infty$ since $d_{i,n} < +\infty$. In such a case, $\mathcal{P}(C)$ is the whole state set. Therefore, the RMAB problem is indexable, and the Whittle index for the arm is the corresponding service charge. $\square$

As shown in the Whittle index, as carryover component $a_{i,n}$ and traveled distance in the past cycle $d_{i,n-1}$ increase, the value for activating the arm also increases, which is similar compared to the myopic policy. Furthermore, given the same $a_{i,n}$ and $d_{i,n-1}$, a vehicle with high mobility in the subsequent cycles may have a lower index compared to the one with low mobility. This is because, if the vehicle with high mobility is scheduled, the age of the result would increase fast in the future, and the delivered computing results would be outdated quickly. Given the limited computing resource, the edge server will select the one that is most efficient to reduce the overall AoR in the long term.

According to the closed-form Whittle index in (3.25), we can schedule tasks in each cycle according to the Whittle index policy. The index implies the value of scheduling a vehicle. From a long-term perspective, scheduling the vehicle with a higher index value leads to a lower AoR of the network. Recall the model for the synchronous offloading scenario. The edge server can collect the Whittle indexes of vehicles and schedule K vehicles with the highest index values at the beginning of each computing cycle. After selecting tasks, we allocate the computing slot for those tasks. As mentioned above, the slot allocation when set $\mathcal{S}_n$ is given only affects the instantaneous AoR within a cycle. To obtain $x_{i,n}$, we formulate the following optimization problem:

$$\min_{\{x_{i,n}, i\in\mathcal{V}_n\}} \sum_{i\in\mathcal{V}_n} (a_{i,n} + d_{i,n-1})(T - x_{i,n}\omega). \tag{3.26}$$

As $(a_{i,n} + d_{i,n-1})$ is constant, the optimal solution of the linear programming (3.26) is to allocate the slot with the highest $x_{i,n}$ to the vehicle with the highest carryover component $a_{i,n} + d_{i,n-1}$. The policy in this step is sorting those tasks according to $(a_{i,n} + d_{i,n-1})$ in a decreasing order and scheduling them in sequence.

Although we have obtained the closed-form of the Whittle index and found the Whittle index policy for the synchronous scenario, the future vehicle mobility is unknown. Next, we develop a DRL-assisted method to learn the vehicle mobility in the future from the vehicle mobility profile in the past in Section 3.4.

3.4 DRL-assisted Scheduling

We denote the term related to the future mobility information in the Whittle index by $V(a_{i,n}, \delta_{i,n})$, where

$$V(a_{i,n}, \delta_{i,n}) = \mathbb{E}\big[\sum_{t=0}^{\infty}(a_{i,n} - D_{i,n+t})\mathbf{1}(D_{i,n+t} \leq a_{i,n})|\delta_{i,n}\big]. \tag{3.27}$$

We refer to $V(a_{i,n}, \delta_{i,n})$ as the value function for state $(a_{i,n}, \delta_{i,n})$. Given the same carryover component $a_{i,n}$, the value function for state $(a_{i,n}, \{\delta_{i,n}^{-1}; d_{i,n}\})$ can be obtained by

$$V(a_{i,n}, \{\delta_{i,n}^{-1}; d_{i,n}\}) = \mathbb{E}\big[\sum_{t=1}^{\infty}(a_{i,n} - D_{i,n+t} + d_{i,n}) \times \mathbf{1}(D_{i,n+t} - d_{i,n} \leq a_{i,n})|\{\delta_{i,n}^{-1}; d_{i,n}\}\big]. \tag{3.28}$$

We then observe that the value function $V(a_{i,n}, \delta_{i,n})$ can be represented in an alternative way as

$$V(a_{i,n}, \delta_{i,n}) = \mathbb{E}\big[(a_{i,n} - d_{i,n})\mathbf{1}(d_{i,n} \leq a_{i,n}) + V(a_{i,n} - d_{i,n}, \{\delta_{i,n}^{-1}; d_{i,n}\})|\delta_n\big]. \tag{3.29}$$

Given a known traveled distance $d_{i,n}$, equation (3.29) is a Bellman equation and can be solved by dynamic programming. Equation (3.29) also shows that $V(0, \delta_{i,n}) = 0$. To adapt the real-time vehicle mobility, we use a DRL method to approximate $V(a_{i,n}, \delta_{i,n})$ iteratively, which gets the vehicle mobility from the environment, *i.e.*, the transportation network, and updates the value function based on the current estimation.

Different from other reinforcement learning methods like Deep Q Network (DQN), the considered learning process only focuses on the state transition and the value function approximation. The state for the DRL-learning problem is $s_{i,n} = (a_{i,n}, \delta_{i,n})$. The reward for each decision step is:

$$r_{i,n} = (a_{i,n} - d_{i,n})\mathbf{1}(d_{i,n} \leq a_{i,n}),$$

which can be observed from the vehicle mobility in the environment. Instead of observing the vehicle state of the next decision step in traditional reinforcement learning methods, the next state is known if $d_{i,n}$ is obtained. According to (3.29), the next state $s'_{i,n}$ is:

$$s'_{i,n} = (a_{i,n} - d_{i,n}, \{\delta_{i,n}^{-1}; d_{i,n}\}).$$

Furthermore, since the state space of the considered problem is large, we adopt deep neural

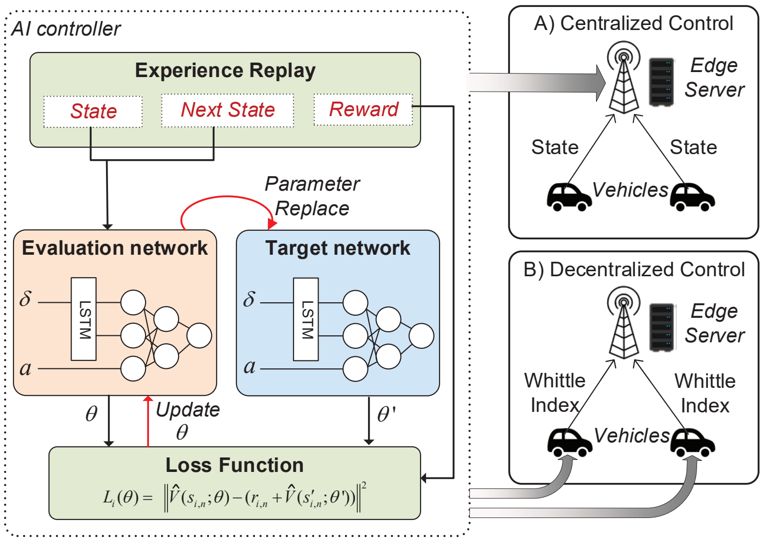


Figure 3.4: Deep reinforcement learning structure.

networks to learn the value function $V(a_{i,n}, \delta_{i,n})$. Inspired by DQN, we use two neural networks, *i.e.*, the evaluation network and the target network, to learn the value function. As shown in Fig. 3.4, the evaluation network is trained in an online manner according to a loss function, which is shown as follows:

$$L_i(\theta) = \|\hat{V}(s_{i,n};\theta) - (r_{i,n} + \hat{V}(s'_{i,n};\theta'))\|_2^2, \tag{3.30}$$

where $\hat{V}(s;\theta)$ represents the estimated value function obtained by weights θ in the evaluation network, and $\hat{V}(s;\theta')$ represents the estimated value function obtained by weights θ' in the target network. The weights in the target network are periodically replaced by the weights in the evaluation network.

The algorithm of the proposed DRL-assisted index policy is shown in Algorithm 1. At the beginning of each computing cycle, the Whittle indexes of all vehicles are calculated using (3.25). The edge server gathers the indexes and schedules the K vehicles with the highest index values according to the Whittle index policy, which is presented in Line 5 of Algorithm 1. To improve the learning efficiency, in experience replay, we select the state-reward tuples in the memory by weighted sampling in Line 8. Via a positive parameter η, the state with non-zero $a_{i,n}$ has a higher probability to be selected. As shown in Line 13, a soft parameter replacement scheme for the target network is adopted, where τ is a number less than one. Furthermore, we add an Long Short-Term Memory (LSTM) layer in the neural network to analyze the time correlation of the vehicle traveling mobility profile in the state. As illustrated in Fig. 3.4, there are two options to implement the proposed policy:

- Centralized control: An AI controller is deployed at the edge server. Vehicles report their current state, and the edge server stores all the state-reward tuples obtained in this decision step in the memory. The neural network is trained by the aggregated mobility profiles reported by the vehicles. In this case, the edge server can learn the mobility feature of a group of vehicles.
- Decentralized control: At each vehicle, an AI controller is deployed to learn the vehicle driving behavior. Since the Whittle index can be generated independently by each vehicle, vehicles only need to update their Whittle indexes to the server for scheduling. Compared to reporting the state in each cycle in the centralized control, the communication overhead is much lower via the decentralized control.

Either option can be selected, depending on the characteristics of the communication and transportation network scenarios. For example, in an urban area, the centralized method would be preferred since the geometric features in traffic profiles can be learned by the edge server. On the other hand, in areas with smooth traffic flow or limited communication resources, the decentralized control would be preferred to reduce communication overhead.

3.5 Scheduling Scheme for Asynchronous Offloading

In this section, the index-based scheduling scheme is extended from a synchronous scenario to an asynchronous scenario, in which vehicles sense and offload their observation to the edge server at arbitrary instants. As mentioned in the system model, all vehicles have the same sensing cycle, while the proposed scheme can be extended to adapt the case with different sensing cycle lengths.

Compared to the synchronous scenario, in which a vehicle only has one task to be scheduled in a computing cycle, up to two tasks offloaded by a vehicle can be scheduled in a computing cycle in the asynchronous scenario. For the edge server, vehicle i with offset O_i divides a computing cycle into two parts. In the set of the first several computing slots, denoted by $\mathcal{T}_i^1 = \{1, \dots, \lceil O_i/\omega \rceil\}$, the edge server can schedule a task that is generated in the previous computing cycle by vehicle i but not scheduled yet. Such task is denoted by task $n^{(1)}$. In the set of the rest of computing slots in the cycle, denoted by $\mathcal{T}_i^2 =$

Algorithm 1 Learning-assisted Whittle index policy

```
1: Initialize the weights of the evaluation and target networks ($\theta$ and $\theta'$), respectively.
2: Initialize the experience replay buffer.
3: for each cycle $n$ do
   % Schedule tasks with the Whittle index policy.
4:     Obtain the initial Whittle index for vehicle $i$ using (3.25) corresponding to the
       estimated value function, i.e., $\hat{V}(a_{i,n}, \delta_{i,n})$.
5:     Edge server gathers the Whittle indexes from all vehicles and schedules the $K$
       vehicles with the highest index values.
   % Train the deep neural network.
6:     Observe the traveled distance of vehicles, i.e., $\{d_{i,n}, \forall i\}$.
7:     Store state-reward tuple $(s_{i,n}, s'_{i,n}, r_{i,n})$ in the experience replay buffer. Delete the
       oldest transition set if the buffer is full.
8:     Sample a mini-batch of $N$ samples using a weighted sampling, where the weight for
       a state with carryover component $a$ is $w = \mathbf{1}(a > 0) + \eta$, and $\eta > 0$.
9:     for state $s_0$ with $a = 0$ in the mini-batch do
10:        $\hat{V}(s_0; \theta') = 0$.
11:    end for
12:    Update the weights in the evaluation network by minimizing the loss function (3.30).
13:    Update the target network: $\theta' = \tau(1 - \tau)\theta + \tau\theta'$.
14: end for
```

$\{\lceil O_i/\omega \rceil + 1, \ldots, K\}$, the edge server can schedule a task that is generated by vehicle i in the current computing cycle. Such task is denoted by task $n^{(2)}$. Therefore, three indexes can be calculated for this vehicle: The first index is the service charge for scheduling task $n^{(1)}$ in $\mathcal{T}_i^1$, which is denoted by $C_{i,n}^1$. The value of $C_{i,n}^1$ inherits the index of the task in the previous cycle. The second and third indexes are for scheduling task $n^{(2)}$ in $\mathcal{T}_i^2$. If task $n^{(1)}$ is not scheduled, carryover component $a_{i,n}$ will increase as given in (3.2). The corresponding index for scheduling task $n^{(2)}$ is denoted by $C_{i,n}^2$. Otherwise, if task $n^{(1)}$ is scheduled, carryover component $a_{i,n}$ will be reset as zero. The corresponding index for scheduling task $n^{(2)}$ is denoted by $C_{i,n}^3$.

Extending from the Whittle index policy for the synchronous scenario, we propose the Scheduling Scheme for Asynchronous Offloading (SSA) algorithm, which is shown in Algorithm 2. Instead of evaluating the Whittle index for all vehicles at the beginning of a cycle in the synchronous scenario, the index values are compared in each of computing slots in the asynchronous scenario. The idea behind Algorithm 2 is similar to the Whittle index policy, in which the server schedules the vehicle with the highest Whittle index for each computing slot. At the beginning of the algorithm, the initial Whittle indexes of tasks $n^{(1)}$ and $n^{(2)}$ are obtained via Algorithm 1 without considering asynchronous offloading. We use two indicator vectors $I_{i,n}^{(1)}$ and $I_{i,n}^{(2)}$ to represent whether vehicle i is scheduled in part $\mathcal{T}_i^1$ and part $\mathcal{T}_i^2$ of the computing cycle, respectively. If the vehicle is scheduled in part $\mathcal{T}_i^1$, $I_{i,n}^{(1)} = 0$; otherwise, $I_{i,n}^{(1)} = 1$. Similarly, if the vehicle is scheduled in part $\mathcal{T}_i^2$, $I_{i,n}^{(2)} = 0$; otherwise, $I_{i,n}^{(2)} = 1$. In Lines 6 to 13, the vehicle with the highest index is scheduled for each slot iteratively. As shown in Lines 8 and 9 in Algorithm 2, we define variables $F_{i,n}^{(1)}$ and $F_{i,n}^{(2)}$ to represent the initial index for $n^{(1)}$ and $n^{(2)}$ of vehicle i in a computing cycle. To avoid scheduling a task twice in different computing cycles, the values of $F_{i,n}^{(1)}$ and $F_{i,n}^{(2)}$ will be -1 if the corresponding task is scheduled. Furthermore, the index for task $n^{(2)}$ will change if the computing policy for task $n^{(1)}$ changes. The approximated Whittle index for the vehicle in slot k is defined as follows:

$$C_{i,k} = F_{i,n}^{(1)}\mathbf{1}(k \in \mathcal{T}_i^1) + F_{i,n}^{(2)}\mathbf{1}(k \in \mathcal{T}_i^2).$$

The vehicle with the highest approximated index is selected to be scheduled in the slot as presented in Line 9. Lines 10 to 12 update the indicator according to the policy made by Line 9. Since the task allocated to a slot with a small number has a short queuing time, the algorithm allocates the tasks to slots 1 to K in a cycle consecutively, and the computing slot with a small number has priority in selecting a task. Since the changed policy made in the first part of a computing cycle, *i.e.*, $\mathcal{T}_i^1$, for vehicle i results in a different approximated index in the second part of the computing cycle, *i.e.*, $\mathcal{T}_i^2$, the indicators may change as the policy changes. The SSA algorithm iteratively updates the computing policy until the indicator vectors no longer change or the iteration number reaches e_{max}

The time complexity of the algorithm primarily depends on the number of iterations of the outer loop in Algorithm 2, which checks the convergence of the indicator vectors. In each iteration, the time complexity is the same as the selective sorting algorithm if the time complexity on solving the Whittle indexes using equation (3.25) is neglected. Since a vehicle has two indexes during a computing cycle, the time complexity of comparing index values at all vehicles is $2K|\mathcal{V}_n|$. Furthermore, since the indicators may change when the policy changes, the convergence of the algorithm may not be guaranteed. The algorithm will be terminated after e_{max} iterations if the algorithm cannot converge. Thus, the time complexity of Algorithm 2 is $O(2K|\mathcal{V}_n|e_{max})$ in the worst case.

Algorithm 2 Scheduling scheme for asynchronous offloading (SSA)

```
1: Initialize indicator vectors $I^{(1)}_{i,n} = I^{(2)}_{i,n-1}$, and $I^{(2)}_{i,n} = 0$, $\forall i$.
2: Set the indicator vectors $J^{(1)}_{i,n} = I^{(1)}_{i,n}$, and $J^{(2)}_{i,n} = I^{(2)}_{i,n}$, $\forall i$.
3: Obtain the initial Whittle indexes, $C^1_{i,n}$, $C^2_{i,n}$, and $C^3_{i,n}$, using Algorithm 1.
4: Initialize counter $e = 1$.
5: Initialize assignment vector $\mathbf{W} = \mathbf{0}$
6: for $k = \{1, \ldots, K\}$ do
7:     $F^{(1)}_{i,n} = C^1_{i,n} J^{(1)}_{i,n} - (1 - J^{(1)}_{i,n})$, $\forall i$.
8:     $F^{(2)}_{i,n} = [C^2_{i,n} J^{(1)}_{i,n} + C^3_{i,n}(1 - J^{(1)}_{i,n})] J^{(2)}_{i,n} - (1 - J^{(2)}_{i,n})$, $\forall i$.
9:     For slot $k$, $i^* = \text{argmax}_i C_{i,k}$
10:    If $C_{i,k} < 0$: $W_k = \emptyset$.
11:    Else if $\mathbf{1}(k \in \mathcal{T}^1_{i^*}) == 1$: $J^{(1)}_{i^*,n} = 0$, $W_k = i^*$.
12:    Else if $\mathbf{1}(k \in \mathcal{T}^1_{i^*}) == 0$: $J^{(2)}_{i^*,n} = 0$, $W_k = i^*$.
13: end for
14: if $J^{(1)}_{i,n} == I^{(1)}_{i,n}, J^{(2)}_{i,n} == I^{(2)}_{i,n}$, $\forall i$ then
15:     $C^1_{i,n+1} = [C^2_{i,n} J^{(1)}_{i,n} + C^3_{i,n}(1 - J^{(1)}_{i,n})] J^{(2)}_{i,n} - (1 - J^{(2)}_{i,n})$.
16:     Assignment is completed.
17: else
18:     $I^{(1)}_{i,n} = J^{(1)}_{i,n}, I^{(2)}_{i,n} = J^{(2)}_{i,n}$, $\forall i$.
19:     $e = e + 1$.
20:     If $e < e_{\max}$: Return to Line 5
21:     Else: Return to Line 15
22: end if
```

3.6 Simulation Results

In this section, we present the numerical results of Whittle index policy that is found by the proposed scheduling scheme and compare the performance with two other benchmarks: *highest-AoR-first* and *round-robin* scheduling policies. The highest-AoR-first policy is a myopic policy that always schedules the task with the highest AoR first, and the round-robin computing policy schedules vehicles in a circular order. We also compare the performance of the proposed scheme with a *SSA-only* scheme. In the SSA-only scheme, we schedule the vehicles by the proposed scheduling scheme shown in Algorithm 2. Different from the proposed scheme, the SSA-only scheme uses vehicles' AoR to guide the task scheduling rather than the Whittle index. We simulate our computing scheduling scheme

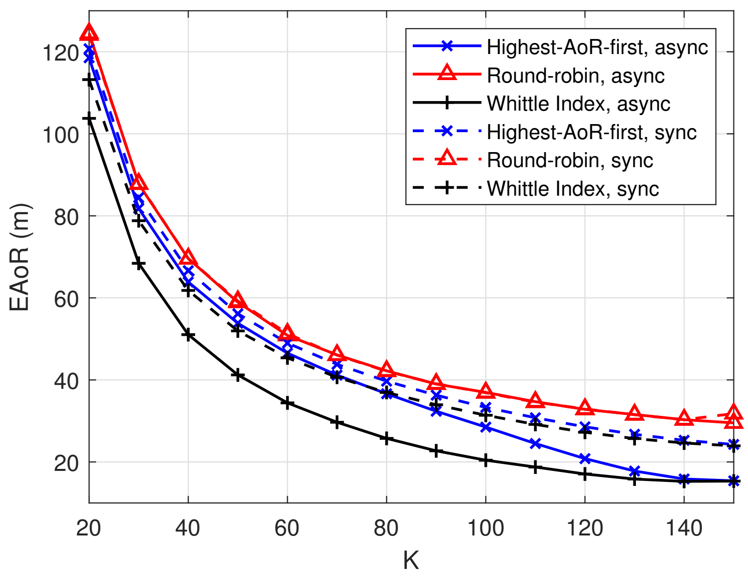


Figure 3.5: EAoR versus the number of computing slots in a cycle with mobility profile P1.

Table 3.1: Vehicle speed distribution in a computing cycle (km/h)

	Group 1		Group 2		Group 3		Group 4		Group 5	
	μ	ξ	μ	ξ	μ	ξ	μ	ξ	μ	ξ
P1	20	20	40	20	60	20	80	40	60	40
P2	20	0	40	0	60	0	80	0	60	0

and the learning approach based on a real-world vehicle mobility dataset.[2] The centralized learning model is considered in the simulation.

3.6.1 Numerical Results

In this subsection, we generate vehicle mobility profiles and simulate the proposed index-based scheduling scheme. In this numerical example, the edge server schedules the vehicles' tasks according to the vehicle mobility profiles. We consider a total of 150 vehicles traveling under the service coverage of the edge server. According to the different speed distributions,

[2]The data came from the Didi Gaia Data Opening Plan: `https://gaia.didichuxing.com`.

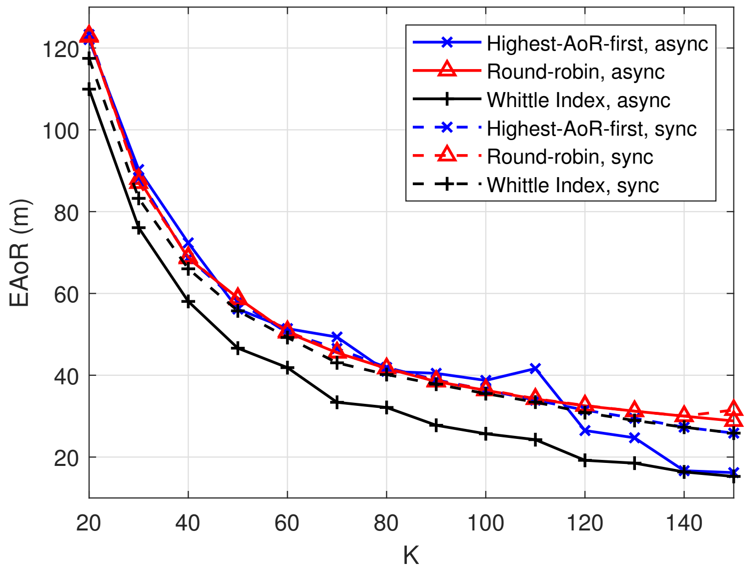


Figure 3.6: EAoR versus the number of computing slots in a cycle with mobility profile P2.

they are divided into five groups with the same number of 30 vehicles in a group. We consider two mobility profiles, *i.e.* P1 and P2, which is summarized in Table 3.1. The speed of vehicles in P1 follows normal distributions, where μ and ξ represent mean and standard deviation, respectively. In terms of profile P2, all vehicles travel at constant speeds. For both mobility profiles, we evaluate the performance of EAoR for both synchronous and asynchronous scenarios, where the offset of vehicles in the asynchronous scenario is selected randomly from the interval $[0, T)$. The length of a computing cycle T is 2 seconds. The policies are used in 500 cycles to obtain the average AoR for all vehicles. We demonstrate the performance of the computing policies versus the number of slots in each cycle, *i.e.*, K. The term *sync* in the legend represents the synchronous offloading scenario, and *async* represents the asynchronous offloading scenario.

The performance of EAoR with mobility profile P1 is shown in Fig. 3.5 with the number of slots in a cycle, *i.e.*, K, changing from 20 to 150. In both the synchronous and asynchronous scenarios, our proposed scheme has the lowest AoR among all scheduling policies. In the asynchronous scenario, when K is 80, the proposed scheme can reduce the EAoR by up to 40% and 30% compared to the round-robin and highest-AoR-first policies, respectively. In the synchronous scenario, vehicles in the Whittle index policy experience

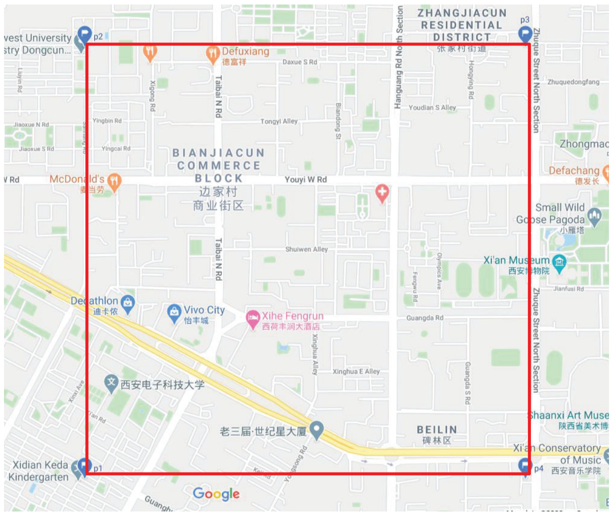


Figure 3.7: A snap shot of the simulation region.

a higher AoR compared to the asynchronous scenario, and the performance gaps between the proposed scheme and the other two policies are smaller. This is because all requests are arrived at the same time in the synchronous scenario. Some tasks are inevitably being allocated to the computing slots at the end of the computing cycle, which will increase the AoR significantly. Although the same could also happen using the highest-AoR-first policy, when the computing slots are insufficient, the tasks with the highest AoR are likely about to outdated. Processing the outdated tasks rather than the newly arrived tasks results in a high EAoR when K is small. Therefore, the AoR of the highest-AoR-first policy with the asynchronous offloading is not reduced significantly compared to that with the synchronous offloading, especially when computing slots are insufficient.

The performance of EAoR with mobility profile P2 is shown in Fig. 3.6, in which all vehicles travel at constant speed. In such a case, the major advantage of the Whittle index policy, which schedules the tasks according to the mobility dynamics, has less influence on the scheduling performance. Therefore, the Whittle index policy is close in performance with the other two policies in the synchronous scenario. Moreover, as proved in [80], when all vehicles have the same speed at the same time, the highest-AoR-first policy is equivalent to the round-robin policy and provides the optimal scheduling solution. Since vehicles have similar mobility profiles, the three policies would all achieve near-optimal performance. On the other hand, for the asynchronous scenario, the Whittle-index policy has better performance since it always schedules the tasks considering the age.

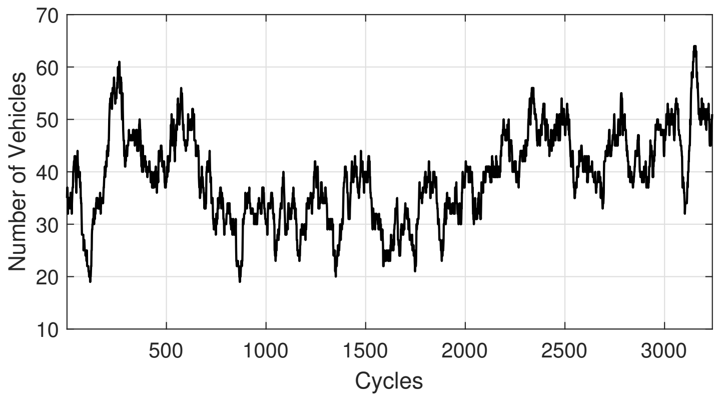


Figure 3.8: The number of vehicles in the simulation time window.

Table 3.2: Neural network structure in DRL

Layers	Number of neurons	Activation function
LSTM	40	relu
Fully connected 1	120 (50% dropout)	relu
Fully connected 2	60	relu
Fully connected 3	30	relu
Fully connected 4	15	elu
Fully connected 5	1	elu

3.6.2 Simulation with a Real Dataset

In this subsection, we simulate the proposed index-based scheduling scheme and the learning approach based on the taxi driving trace data collected by Didi Gaia Data Opening Plan in Xi'an, China. We investigate the vehicle trajectories from 16:30 to 18:20 on Oct. 1st, 2016 in a 2km $\times$ 2km area as shown in Fig. 3.7. The attributes of a set of data include timestamp, vehicle ID, and vehicle location (longitude and latitude). The length of a computing cycle is 2 seconds. Thus, there are 3300 computing cycles in the simulation. We select the vehicles which travel in the 2km $\times$ 2km area consistently in the prediction window W, where W is 80 in our simulation. Under such condition, there are 126376 sets of data to be analyzed, and the number of vehicles counted in each computing cycle is shown in Fig. 3.8. The settings of the DRL algorithm are summarized as follows: The learning rate is 0.001, the soft replacement parameter τ is 0.8, the memory can store 800 state-reward tuples, and the batch size is 48. The structure of the neural network is

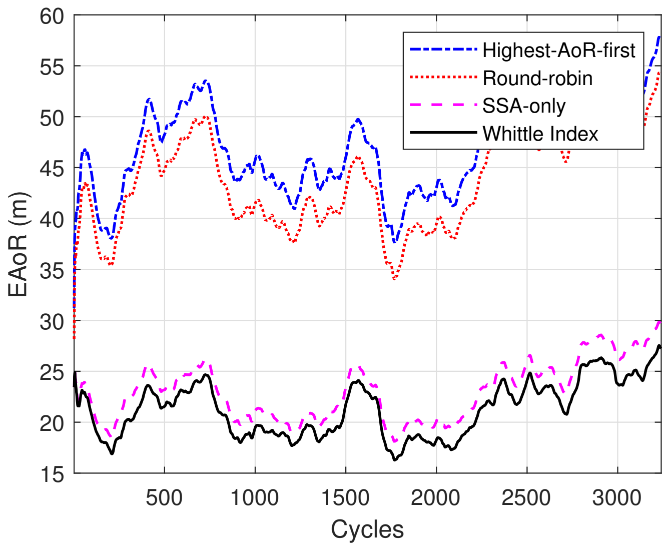


Figure 3.9: EAoR when the number of computing slots in each cycle, *i.e.*, K, is 10.

Table 3.3: Average algorithm running time (in second)

Number of slots	Highest-AoR-first	Round-robin	SSA-only	Whittle Index
$K = 10$	0.0009	0.0003	0.0321	0.1176
$K = 20$	0.0022	0.0005	0.0626	0.1550

presented in Table 3.1. We apply a moving average to measure the EAoR value for the vehicles in an online manner, where the window size is 200 cycles.

The EAoR performances in the simulation time window are shown in Figs. 3.9 and 3.10, where the number of computing slots in a cycle is 10 and 20, respectively. For both scenarios, the proposed scheme reduces the AoR significantly. Although the computing resource is limited, the proposed scheme maintains the AoR around 20m and 10m when K is 10 and 20, respectively. Moreover, compared to the SSA-only scheme which schedules the tasks without considering future mobility, our proposed scheme can adapt to the time-variant vehicle mobility in real time and further reduce the EAoR. As the number of slots increases, the performance gap between the proposed scheme and SSA-only scheme also increases since the influence of future mobility on AoR performance is significant when computing slots are insufficient.

The average running time of the proposed and benchmark schemes is presented in

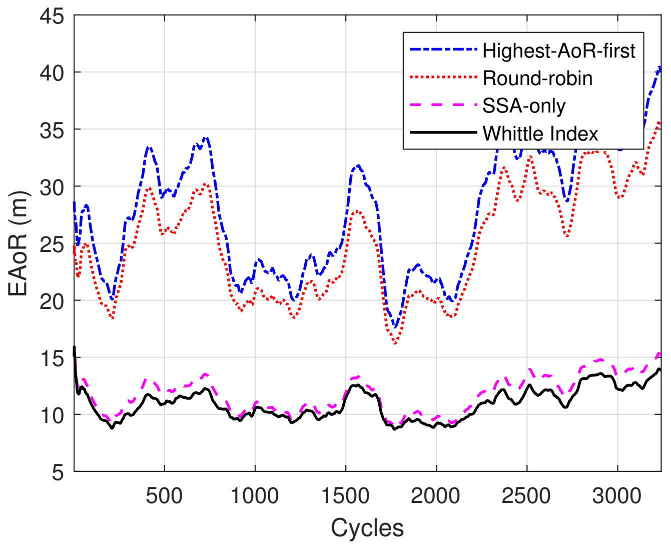


Figure 3.10: EAoR when the number of computing slots in each cycle, *i.e.*, K, is 20.

Table 3.4: Efficiency on performance improvement compared with the round-robin scheme ($\times 10^3$)

Number of slots	Highest-AoR-first	SSA-only	Whittle Index
$K = 10$	-6.11	0.92	0.27
$K = 20$	-5.03	0.41	0.12

Table 3.3. The simulation is performed on a machine with Intel i7-9750H CPU. Since the proposed index-based scheduling scheme obtains the Whittle indexes from the neural network and attains scheduling policy iteratively, the running time is the highest compared to other benchmarks. However, the time consumption for running the proposed scheme is still on the millisecond level and can be acceptable to real-time applications. Moreover, the time consumed on the neural network dominates the overall running time of the proposed scheme, which can be seen from the time difference between SSA-only and the proposed scheme. Such running time can be further decreased with the advancement of GPUs. We further evaluate and compare the efficiency on performance improvement among the scheduling schemes in Table 3.4. The efficiency is defined as the ratio of EAoR deduction and running time increment. As the round-robin scheme has the shortest running time, we use its performance as the baseline to evaluate the efficiency of other scheduling schemes.

The efficiency is calculated by the following equation:

$$\eta_s^{EPI} = \frac{\text{Average EAoR}_s - \text{Average EAoR}_{round-robin}}{\text{Running time}_s - \text{Runing time}_{round-robin}}, \tag{3.31}$$

where η_s^{EPI} is the efficiency on performance improvement for scheduling scheme s, and Average EAoR$_s$ and Running time$_s$ are the average EAoR and running time achieved by scheduling scheme s, respectively. The highest-AoR-first scheme has worse performance than the round-robin scheme and, thus, has a negative value of efficiency. Moreover, the SSA-only scheme has the highest efficiency compared to other schemes since it eliminates the time-consuming learning process involved in calculating the Whittle index. Therefore, when the accuracy of computing results is not a critical requirement in a system, the SSA-only scheme can be used to achieve a lower AoR with higher efficiency, compared with the proposed learning-based scheduling scheme. When the user has a strict requirement for result accuracy, the proposed learning scheme can be used to obtain an accurate Whittle index and reduce AoR.

3.7 Summary

In this chapter, we have proposed a proactive index-based scheduling scheme based on predicted future vehicle mobility dynamics for the edge server to process real-time computing tasks offloaded by autonomous vehicles. We have adopted a novel DRL approach to estimate the mobility of the vehicles in the future and assist the evaluation of scheduling indexes. As a result, the proposed scheduling scheme can adapt to real-time vehicle mobility and support safety-related computing services with low computing complexity and communication overhead.

Chapter 4

Energy-Efficient Resource Management for UAV-assisted MEC

In this chapter, we investigate energy-efficient computation offloading and task scheduling for computing service provisioning at an edge server. A UAV-mounted cloudlet plays the role as an edge server to collect and execute computing tasks offloaded from IoT nodes in a scenario of IoT in remote areas. The aim of this work is to optimize UAV trajectory, computation offloading, and task scheduling in order to maximize the energy efficiency of the UAV given both the UAV and IoT node resource constraints. Optimization approaches are developed to achieve an optimal resource management strategy for the corresponding non-convex fractional problem. Specifically, Section 4.1 introduces the background and motivation of the work, in which the contribution of the work is addressed. The system model is provided in Section 4.2. Problem formulation and the corresponding approach are presented in Sections 4.3 and 4.4, respectively. The extended implementation of the proposed approach is provided in Section 4.5. Extensive simulation results are provided in Section 4.6.

4.1 Background and Motivation

Driven by the visions of IoT and 5G communications, MEC is considered as an emerging paradigm that leverages the computing resource and storage space deployed at network edges to perform latency-critical and compute-intensive tasks for mobile users [9]. The computing tasks generated by mobile users can be offloaded to the nearby edge server, such as macro/small cell base station and Wi-Fi access point, to reduce computing delay and computing energy cost at mobile devices. Moreover, by pushing the traffic, computation, and network functions to the network edges, mobile users can enjoy low task offloading time with less backhaul usage [85].

Specifically, in IoT era, MEC is considered as a key enabling technology to support the computing services for billions of IoT nodes to be deployed [86, 87]. Since the most of IoT nodes are power-constrained and have limited computation capability, they can offload their computing tasks to network edges to extend their battery life and improve the computing efficiency. However, many IoT nodes are operating in unattended or challenging areas, such as forests, deserts, mountains, or underwater locations [19], to execute some compute-intensive applications, including long pipeline infrastructures monitoring and control [20], underwater infrastructures monitoring [21], and military operations [88]. In these scenarios, the terrestrial communication infrastructures are distributed sparsely and cannot provide reliable communications for the nodes. Therefore, in this work, we utilize UAVs to provide ubiquitous communication and computing supports for IoT nodes. Equipped with computing resources, UAV-mounted cloudlet can collect and process the computing tasks of ground IoT nodes that cannot connect to the terrestrial edges. As UAVs are fully controllable and operate at a high altitude, they can be dispatched to the designated places for providing efficient on-demand communication and computing services to IoT nodes in a rapid and flexible manner [89–92].

Despite the advantages of UAV-assisted MEC, there are several challenges in network deployment and operation. Firstly, the onboard energy of a UAV is usually limited. To improve the user experience on the computing service, UAVs should maximize their energy efficiency by optimizing their computing ability in the limited service time. Secondly, planning an energy-aware UAV trajectory is another challenge in UAV-assisted networks. The UAV is required to move to collect the offloaded data from sparsely distributed users for the best channel quality, while a significant portion of UAV energy consumption stems from mechanical actions during flying. Thirdly, the computation load allocation cannot be neglected even though the computing energy in UAV-mounted cloudlet is relatively small compared to its mechanical energy. In the state-of-art MEC server architecture, the DFVS technique is adopted. The computing energy for a unit time is growing cubically as the allocated computation load increases [9]. Without proper allocation, the computing energy consumption could blow up, or the offloaded tasks cannot be finished in time. More importantly, UAV trajectory design, computation load allocation, and communication resource management are coupled in the MEC system [93, 94], which makes the system even more complex. To the best of our knowledge, the joint optimization of UAV trajectory, computation load allocation, and communication resource management considering energy efficiency has not been investigated in the UAV-assisted MEC system.

To address the above challenges, we consider an energy constrained UAV-assisted MEC system in this chapter. IoT nodes as ground users can access and partially offload their computing tasks to the UAV-mounted cloudlet according to their service requirements. The UAV flies according to a designed trajectory to collect the offloading data, process computing tasks, and send computing results back to the nodes. For each data collection and task execution cycle, we optimize the energy efficiency of the UAV, which is defined as the ratio of the overall offloaded computing data to UAV energy consumption in the cycle, by jointly optimizing the UAV trajectory and resource allocation in communication and computing aspects. The main contributions of the work are summarized as follows.

1) We develop a model for energy-efficient UAV trajectory design and resource allocation in the MEC system. The model incorporates computing service improvement and energy consumption minimization in a UAV-mounted cloudlet. The communication and computing resources are allocated subject to the user communication energy budget, computation capability, and the mechanical operation constraints of the UAV.

2) We exploit the Successive Convex Approximation (SCA) technique and Dinkelbach algorithm to transform the non-convex fractional programming problem into a solvable form. In order to improve scalability, we further decompose the optimization problem by the Alternating Direction Method of Multipliers (ADMM) technique. UAV and ground users solve the optimization problem cooperatively in a distributed manner. By our approach, both users and UAV can obtain the optimal resource allocation results iteratively without sharing local information.

3) We further consider the scenario with limited knowledge of node mobility. A spatial distribution estimation technique, Gaussian kernel density estimation, is applied to predict the location of ground users. Based on the predicted location information, our proposed strategy can determine an energy-efficient UAV trajectory when the user mobility and offloading requests are ambiguous at the beginning of each optimization cycle.

4.2 System Model

In this section, we present the system model of the considered problem.

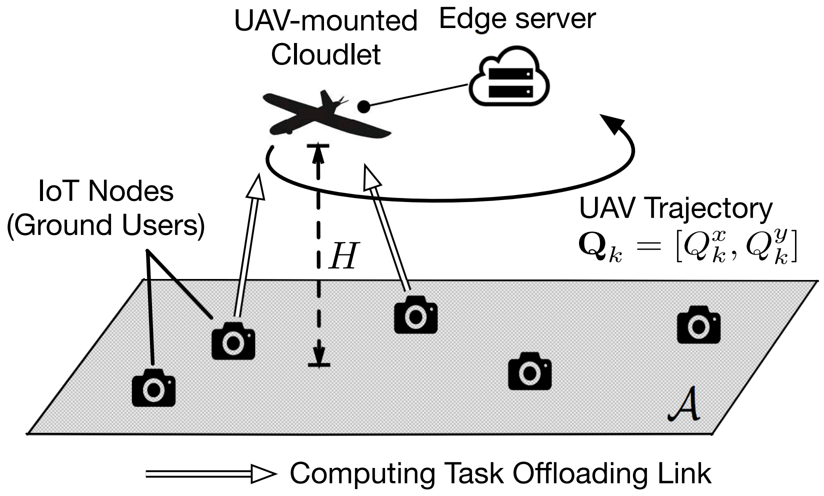

UAV-mounted Cloudlet
Edge server
IoT Nodes (Ground Users)
UAV Trajectory $\mathbf{Q}_k = [Q_k^x, Q_k^y]$
H
$\mathcal{A}$
Computing Task Offloading Link

Figure 4.1: UAV-assisted MEC system model.

4.2.1 Network Model

The UAV-assisted MEC system is shown in Fig. 4.2.1, in which a single UAV-mounted cloudlet is deployed to offer edge computing service for ground users in area $\mathcal{A}$. The UAV periodically collects and processes the computing tasks offloaded from ground users. Each user processes the rest of the computing tasks locally if the task cannot be fully collected by the UAV. Define the computing cycle as a duration of T seconds. Each cycle contains K discrete time slots with equal length. Denote the set of time slots in the cycle by $\mathcal{K}$. Thus, the time length for a slot is T/K, which is denoted by Δ.

At the beginning of each cycle, ground users with computing tasks in area $\mathcal{A}$ send offloading requests to the UAV-mounted cloudlet. Denote the set of those ground users by $\mathcal{I}$, where $\mathcal{I} = \{1, \ldots, N\}$. Assume the ground users in $\mathcal{I}$ can connect to the UAV for all time slots in the cycle. In this work, the UAV and the users cooperatively determine the offloading and resource allocation strategy for this cycle, including the UAV moving trajectory, the transmit power of ground users, and computation load allocation for UAV-mounted cloudlet. Assume that the computation loads on solving the optimization problem are negligible compared to the computation loads of the offloaded tasks. During the cycle, UAV flies over the ground users and offers the computing service according to the designed trajectory and resource allocation strategy. By the end of the cycle, UAV returns to a predetermined final position.

4.2.2 Communication Model

The quality of communication links between the UAV and ground users is dependent on their location. To represent their locations, we construct a 3D Cartesian coordinate system. For IoT node i, the horizontal coordinate at time k is denoted by $\mathbf{q}_{i,k} = [q^x_{i,k}, q^y_{i,k}]$. Assume that nodes know their trajectory for the upcoming cycle, *i.e.*, $\{\mathbf{q}_{i,k}, \forall k\}$. For the UAV, the horizontal coordinate at time k is denoted by $\mathbf{Q}_k = [Q^x_k, Q^y_k]$. The UAV moves at a fixed altitude H. The UAV trajectory plan, as an optimization variable, consists of UAV positions in the whole cycle, *i.e.*, $\mathbf{Q} = [\mathbf{Q}_1; \dots; \mathbf{Q}_K]$. The average UAV velocity in slot k is given by

$$\mathbf{v}_k(\mathbf{Q}) = \frac{\mathbf{Q}_k - \mathbf{Q}_{k-1}}{\Delta}, \forall k. \tag{4.1}$$

The average acceleration in slot k is given by

$$\mathbf{a}_k(\mathbf{Q}) = \frac{\mathbf{v}_k(\mathbf{Q}) - \mathbf{v}_{k-1}(\mathbf{Q})}{\Delta}, \forall k. \tag{4.2}$$

The magnitudes of velocity and acceleration are constrained by the maximum speed and acceleration magnitude, which are denoted by $v_{\max}$ and $a_{\max}$, respectively.

It is assumed that the doppler frequency shift in the communication can be compensated at the receiver. The channel quality depends on the distance between the UAV and users. Due to the high probability of Line-of-Sight (LoS) links in UAV communication [95], we assume that the channel gain follows a free-space path loss model. The channel gain for user i in slot k is denoted by $h_{i,k}$, where

$$h_{i,k}(\mathbf{Q}_k) = \frac{g_0}{\|\mathbf{Q}_k - \mathbf{q}_{i,k}\|_2^2 + H^2}, \tag{4.3}$$

where $\|\cdot\|_2$ is the notation representing the L2 norm. The parameter g_0 denotes the received power at the reference distance (e.g., $d = 1$ m) between the transmitter and the receiver. We consider two channel access schemes: i) orthogonal access, in which the bandwidth is divided into N sub-channels each occupied by one user; and ii) non-orthogonal access, in which the frequency bandwidth is shared among users. Denote the channel bandwidth for the uplink by B. The amount of data that can be offloaded by user i in slot k is

$$R_{i,k}(\delta_{i,k}, \mathbf{Q}_k) = \frac{B\Delta}{N} \log\big[1 + \frac{\delta_{i,k} h_{i,k}(\mathbf{Q}_k) P}{\sigma^2 (B/N)}\big], \tag{4.4}$$

under the orthogonal access model, and,

$$R_{i,k}(\boldsymbol{\delta}_k,\mathbf{Q}_k)=B\Delta\log\big[1+\frac{\delta_{i,k}h_{i,k}(\mathbf{Q}_k)P}{\sigma^2 B+\sum_{j\neq i}\delta_{j,k}h_{j,k}(\mathbf{Q}_k)P}\big], \tag{4.5}$$

under the non-orthogonal channel model. The parameter P and σ^2 denote the maximum transmit power of ground users and the power spectral density of channel noise, respectively. The variable $\delta_{i,k} \in [0,1]$ represents the portion of the maximum power that is allocated to user i within time slot k, which is a part of the offloading strategy. The symbol $\boldsymbol{\delta}_k$ denotes the vector of $\delta_{i,k}$ for all $i \in \mathcal{I}$ in slot k. The noise power in the transmission is represented by n_0, where $n_0 = \sigma^2 B/N$ for the orthogonal channel access model, and $n_0 = \sigma^2 B$ for the non-orthogonal channel access model. In non-orthogonal model, users share the same channel to offload their tasks. The communication power allocated for a user will interfere the data rate of other users.

4.2.3 Computation Model

Due to the limited battery and the computation capability of the UAV, only a part of tasks can be offloaded and executed in the UAV-mounted cloudlet. Full granularity in task partition is considered, where the task-input data can be arbitrarily divided for local and remote executions [66,79,96]. Accordingly, a portion of the computing tasks are offloaded to the cloudlet while the rest are executed by the ground users locally. Users upload the input data for their offloaded tasks, and the UAV processes the corresponding computation loads of those tasks. Assume that the computation load can be executed once the input data is received, and the computing data amount is equal to the input data amount of tasks [96]. A task partition technique is considered, where the partition of the computation input bits are utilized to measure the division between the offloaded computation load and local computation load. The overall input data size for computing tasks of user i is denoted by I_i. We set the threshold $\check{I}_i$ as the minimum input data amount required to be offloaded to the cloudlet for user i, where $\check{I}_i \leq I_i$. The threshold represents the part of computing tasks having to be conducted in the cloudlet. Thus, the overall offloaded bits of user i is constrained as follows:

$$\check{I}_i \leq \sum_{k\in\mathcal{K}} R_{i,k}(\boldsymbol{\delta}_k, \mathbf{Q}_k) \leq I_i, \forall i. \tag{4.6}$$

Under the scenario that the threshold is satisfied, if user's tasks cannot be fully offloaded, the rest of the tasks are processed by IoT nodes locally.

After users upload the input data, the UAV will save the received data to a buffer with enough capacity for further processing. The UAV processes the received data according to the workload allocation results. Let the variable $W_{i,k}$ denote the amount of data, which is from user i's offloaded task, to be processed in slot k. The UAV can only compute the task which is offloaded and received, and all offloaded tasks should be executed by the end of the cycle. Therefore, the following computing constraints are given:

$$\sum_{t=1}^{k} R_{i,t}(\boldsymbol{\delta}_k, \mathbf{Q}_k) \geq \sum_{t=1}^{k} W_{i,t}, \forall k \tag{4.7a}$$

$$\sum_{t=1}^{K} R_{i,t}(\boldsymbol{\delta}_k, \mathbf{Q}_k) = \sum_{t=1}^{K} W_{i,t}. \tag{4.7b}$$

In addition, for the local computing, the CPU-cycle frequency of the IoT node i is fixed as f_i^M. For the UAV-mounted cloudlet, we consider the CPU featured by DFVS technique. The CPU-cycle frequency can step-up or step-down according to the computing workload and is bounded by the maximum CPU-cycle frequency f_{max}^U. As given in [9,66], the CPU-cycle frequency for the cloudlet can be calculated by

$$f_k^U(\mathbf{W}_k) = \frac{\sum_i \chi_i W_{i,k}}{\Delta} \leq f_{max}^U, \forall k, \tag{4.8}$$

where $f_k^U(\mathbf{W}_k)$ represents the CPU-cycle frequency in time slot k, and χ_i denotes the number of computation cycles needed to execute 1 bit of data.

4.2.4 Energy Consumption Model

We elaborate on the energy consumption of the UAV-mounted cloudlet and ground users in this subsection. Specifically, the energy consumed by the ground user includes the communication energy for offloading computing loads and the computing energy for processing residual computing loads. In addition, the UAV-mounted cloudlets consume propulsion energy for collecting the computing loads offloaded by users and computing energy for process the loads. Although downlink transmission from the UAV to users exists in our system, this part of energy consumption is negligible for two reasons: 1) The communication energy is too small compared to the UAV propulsion and computing energy. 2) The output computing results usually have much less data amount compared to the input data amount [97].

Energy Consumption at Nodes

The main energy consumption of nodes is the energy cost from communication and local computing. Firstly, the communication energy for user i offloading tasks in slot k can be formulated as

$$S_{i,k}(\delta_{i,k}) = \delta_{i,k} P \Delta. \tag{4.9}$$

The overall offloading communication energy of user i is bounded by E_i^T, *i.e.*,

$$\sum_k S_{i,k}(\delta_{i,k}) \leq E_i^T, \forall i. \tag{4.10}$$

Therefore, the energy consumption of a user on communication can be reduced if the UAV is closer. On the other hand, for the computing energy consumption, we consider that the lower bound of offloaded bits $\check{I}_i$ guarantees the local computing energy under the user's computing energy requirement, *i.e.*,

$$E_i^M = \kappa \chi_i (I_i - \check{I}_i)(f_i^M)^2 \leq \hat{E}_i^M, \tag{4.11}$$

where E_i^M is the maximum computing energy that could be reached by threshold $\check{I}_i$, and $\hat{E}_i^M$ is the parameter representing the constraint of the computing energy consumption. The computing energy model is adopted from [9, 98]. Parameters f_i^M and κ represent the fixed CPU-cycle frequency of user i and a constant related to the hardware architecture, respectively.

Energy Consumption at UAV-mounted Cloudlet

The main energy consumption at the UAV-mounted cloudlet consists of the energy cost from mechanical operation and computing. We adopt the refined UAV propulsion energy consumption model for fixed-wing UAV following [95] [1]. The propulsion energy consumption in slot k relates to the instantaneous UAV acceleration and velocity, which is given by

$$E_k^F(\mathbf{Q}) = \gamma_1 \|\mathbf{v}_k(\mathbf{Q})\|_2^3 + \frac{\gamma_2}{\|\mathbf{v}_k(\mathbf{Q})\|_2}(1 + \frac{\|a_k(\mathbf{Q})\|_2^2}{g^2}), \tag{4.12}$$

where g denotes the gravitational acceleration. γ_1 and γ_2 are fixed parameters related to the time duration of a time slot, the aircraft's weight, wing area, air density, etc. The value

[1] We deploy the fixed-wing UAV in the proposed system as an example. The proposed approach also can be adapted to the system with a quad-rotor UAV, where only the mechanical energy consumption model is different.

of parameters is given in [95, 96]. The computing energy for executing tasks from user i in time slot k is expressed as

$$E_{i,k}^{C,U}(\mathbf{W}_k) = \kappa \chi_i W_{i,k} \left(f_k^U(\mathbf{W}_k)\right)^2. \tag{4.13}$$

4.3 Problem Formulation

In this work, the main objective is to maximize the energy efficiency of the UAV-mounted cloudlet subject to user offloading constraints, UAV computation capabilities, and the mechanical constraints of the UAV. The energy efficiency of the UAV is defined as the ratio between the overall offloaded data and the energy consumption of the UAV in a cycle. The energy efficiency maximization problem is formulated as follows.

$$\begin{aligned}
\max_{\boldsymbol{\delta},\mathbf{W},\mathbf{Q}} \quad & \eta = \frac{\sum_{i\in\mathcal{I}}\sum_{k\in\mathcal{K}} R_{i,k}(\boldsymbol{\delta}_k, \mathbf{Q}_k)}{\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}} E_{i,k}^{C,U}(\mathbf{W}_k) + \sum_{k\in\mathcal{K}} E_k^F(\mathbf{Q})} && (4.14a)\\
\text{s.t.} \quad & \|\mathbf{v}_k(\mathbf{Q})\|_2 \le v_{max}, \forall k, && (4.14b)\\
& \|\mathbf{a}_k(\mathbf{Q})\|_2 \le a_{max}, \forall k, && (4.14c)\\
& \mathbf{Q}_K = \mathbf{Q}_f, \mathbf{v}_K(\mathbf{Q}) = \mathbf{v}_0, && (4.14d)\\
& 0 \le \delta_{i,k} \le 1, && (4.14e)\\
& (4.6), (4.7a), (4.7b), (4.8), (4.10).
\end{aligned}$$

The term $\mathbf{Q}_f$ represents the designated final position of the UAV, and $\mathbf{v}_0$ represents the initial velocity at the beginning of the cycle. The constraints can be categorized into three types: 1) user Quality of Service (QoS) constraints, including (4.6), (4.10), and (4.14e); 2) UAV computing ability constraints, including (4.7a), (4.7b), and (4.8); 3) UAV mechanical constraints, including (4.14b), (4.14c), and (4.14d). The optimization problem is a non-linear fractional programming. In addition, due to the interference among users in the non-orthogonal channel and the propulsion energy consumption for the fixed-wing UAV, both functions $R_{i,k}(\boldsymbol{\delta}_k, \mathbf{Q}_k)$ and $E_k^F(\mathbf{Q})$ are non-convex. Therefore, solving optimization problem (4.14) is challenging. To search the global optimizer of a non-convex problem is often slow and may not be feasible. In the following section, we will propose an approach to find a local optima efficiently.

4.4 Proposed Optimization Approach

In this section, an optimization approach is introduced to find a solution of problem (4.14). Firstly, an inner convex approximation method is applied to approximate the non-convex functions $R_{i,k}(\boldsymbol{\delta}_k, \mathbf{Q}_k)$ and $E_k^F(\mathbf{Q})$ by solvable convex functions. The SCA-based algorithm is adopted to achieve the local optimizer of the original problem. After the approximated convex functions are built, the fraction programming, which is in the inner loop of the SCA-based algorithm, is handled by the Dinkelbach algorithm. Moreover, in order to improve scalability, the problem is further decomposed into several sub-problems via ADMM technique, in which the power allocation is solved by users in a distributed manner, while the computation load allocation and UAV trajectory planning are determined by UAV itself. The details are presented in following subsections.

4.4.1 Successive Convex Approximation

Problem (4.14) is a non-convex problem due to $R_{i,k}(\boldsymbol{\delta}_k, \mathbf{Q}_k)$ and $E_k^F(\mathbf{Q})$. To construct an approximation that is solvable, we first introduce several auxiliary variables, $\{\xi_{i,k}, \omega_k, l_{i,k}, A_k, \check{R}_{i,k}, \hat{E}_{i,k}^F\}$. For the orthogonal channel access scheme, the new optimization problem is shown as follows:

$$\begin{aligned}
\max_{\mathcal{V}} \quad & \check{\eta}(\mathcal{V}) = \frac{\sum_{i\in\mathcal{I}}\sum_{k\in\mathcal{K}} \check{R}_{i,k}}{\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}} E_{i,k}^{C,U}(\mathbf{W}_k) + \sum_{k\in\mathcal{K}} \hat{E}_k^F} && \text{(4.15a)}\\
\text{s.t.} \quad & \check{R}_{i,k} \le \frac{B\Delta}{N}\log(1+\xi_{i,k}), \forall i,k && \text{(4.15b)}\\
& \xi_{i,k} l_{i,k} \le \delta_{i,k} P, \forall i,k && \text{(4.15c)}\\
& \frac{(\|\mathbf{Q}_k - \mathbf{q}_{i,k}\|_2^2 + H^2)n_0}{g_0} \le l_{i,k}, \forall i,k && \text{(4.15d)}\\
& \hat{E}_k^F \ge \gamma_1 \|\mathbf{v}_k(\mathbf{Q})\|_2^3 + \gamma_2 A_k, \forall k && \text{(4.15e)}\\
& \omega_k^2 \le \|\mathbf{v}_k(\mathbf{Q})\|_2^2, \forall k && \text{(4.15f)}\\
& \omega_k A_k \ge 1 + \frac{\|a_k(\mathbf{Q})\|_2^2}{g^2}, \forall k && \text{(4.15g)}\\
& \check{I}_i \le \sum_{k\in\mathcal{K}} \check{R}_{i,k} \le I_i, \forall i, && \text{(4.15h)}\\
& (4.6), (4.7a), (4.7b), (4.8), (4.10), (4.14b) - (4.14e).
\end{aligned}$$

Set $\mathcal{V}$ represents the union set of the primary and auxiliary optimization variables, where $\mathcal{V} = \{\boldsymbol{\delta}, \mathbf{W}, \mathbf{Q}, \boldsymbol{\xi}, \boldsymbol{\omega}, \mathbf{l}, \mathbf{A}, \check{\mathbf{R}}, \hat{\mathbf{E}}^F\}$. For the non-orthogonal channel model, constraint (4.15b) is replaced by the following constraint:

$$\check{R}_{i,k} \leq B\Delta\big[\log(1 + \sum_{i\in\mathcal{I}} \xi_{i,k}) - \log(1 + \sum_{j\in\mathcal{I}/\{i\}} \xi_{j,k})\big], \forall i, k. \tag{4.16}$$

Lemma 2. *Problem (4.15) is an equivalent form of problem (4.14).*

Proof. See Appendix A. □

Problem (4.15) includes four non-convex constraints, which are (4.15c), (4.15f), (4.15g), and (4.16). We approximate those non-convex constraints by their first order Taylor expansions and adopt the successive convex optimization technique to solve the problem. New auxiliary variables, $\{\xi_{i,k}^t, l_{i,k}^t, \omega_k^t, A_k^t, \mathbf{v}_k, z_{i,k}^t\}$, are introduced to represent the corresponding estimated optimizers at the previous iteration of optimization, *i.e.*, iteration t. The SCA-based algorithm iterates until the estimated solution reaches to a local optimizer. Constraint (4.15c) can be approximated as follows:

$$\|\xi_{i,k} + l_{i,k}, \xi_{i,k}^t - l_{i,k}^t, x_{i,k} - 1\|_2 \leq x_{i,k} + 1, \tag{4.17}$$

where

$$x_{i,k} = \delta_{i,k}P - \frac{(\xi_{i,k}^t - l_{i,k}^t)(\xi_{i,k} - l_{i,k})}{2}.$$

Constraint (4.15f) can be approximated as follows:

$$\omega_k^2 \leq \|\mathbf{v}_k^t\|_2^2 + 2(\mathbf{v}_k^t)^T(\mathbf{v}_k(\mathbf{Q}) - \mathbf{v}_k^t). \tag{4.18}$$

Constraint (4.15g) can be approximated as follows:

$$\|\omega_k - A_k, \omega_k^t + A_k^t, y_k - 1, 2, \frac{2a_k(\mathbf{Q}_k)}{g}\|_2 \leq y_k + 1, \tag{4.19}$$

where

$$y_k = \frac{(\omega_k^t + A_k^t)(\omega_k + A_k)}{2}.$$

Constraint (4.16) can be approximated as follows:

$$\check{R}_{i,k} \leq \frac{B\Delta}{N}\big[\log(1 + \xi_{i,k} + e_{i,k}) - \log(1 + e_{i,k}^t) - \frac{e_{i,k} - e_{i,k}^t}{\ln 2(1 + e_{i,k}^t)}\big], \tag{4.20}$$

where $e_{i,k} = \sum_{j\in\mathcal{I}/\{i\}} \xi_{i,k}$.

Lemma 3. *Non-convex constraints (4.15c), (4.15f), (4.15g), and (4.16) can be approximated by the convex forms in (4.17)-(4.20). The solution of the approximated problem is a local maximizer of problem (4.14), which provides the lower bound of the maximum energy efficiency that can be achieved.*

Proof. See Appendix B. □

Based on Lemma 2 and Lemma 3, the SCA-based algorithm is summarized by Algorithm 3. The term $\check{\eta}(\mathcal{V};\mathbb{A}^t)$ represents the energy efficiency $\check{\eta}(\mathcal{V})$ in (4.15) with the given value in auxiliary variable set $\mathbb{A}^t$. Note that the approximated problem inside the loop (Steps 3 and 4 in Algorithm 1) is a fractional programming problem and still non-convex. We will provide the optimal solution of the approximated problem in the remainder of the section. The convergence of SCA has been proven in [99], and the algorithm will stop after finite iterations if the local optimizer exists.

Algorithm 3 SCA-based Algorithm for Solving Problem (4.15)

```
1: Initialize the auxiliary variables $\mathbb{A}^0 = \{\xi^0_{i,k}, \omega^0_k, l^0_{i,k}, A^0_k, \check{R}^0_{i,k}, \hat{E}^{F,0}_{i,k}\}$ and loop index $t = 0$.
2: repeat
3:     Solve the approximated problem (4.21) for given $\mathbb{A}^t$, and denote the optimal solution
       for auxiliary variables by $\mathbb{A}^{t+1}$:
```

$$\begin{aligned} \max_{\mathcal{V}} \quad & \check{\eta}(\mathcal{V};\mathbb{A}^t) && (4.21)\\ \text{s.t.} \quad & (4.6), (4.7a), (4.7b), (4.8), (4.10), (4.14b)-(4.14e), \\ & (4.15d), (4.15e), (4.15h), (4.17)-(4.18), \\ & (4.15b), \text{ in the case of orthogonal channel,} \\ & (4.20), \text{ in the case of non-orthogonal channel.} \end{aligned}$$

```
4:     Update $t = t + 1$.
5: until The difference of the solutions between two adjacent iterations, i.e., $\|\mathbb{A}^{t+1} - \mathbb{A}^t\|$,
   is below a threshold $\theta_1$.
```

4.4.2 Dinkelbach Algorithm

Problem (4.21) is a fraction programming problem. We can adopt the Dinkelbach algorithm to achieve the optimal solution. The objective function (4.21) can be rewritten as the following parametric programming form:

$$F^t(\alpha) = \max_{\mathcal{V}} \Big\{ \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \check{R}_{i,k} - \alpha \big[\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} E_{i,k}^{C,U}(\mathbf{W}_k) + \sum_{k\in\mathcal{K}} \hat{E}_k^F \big] | \mathcal{V} \in \mathcal{F}^t \Big\}, \tag{4.22}$$

where $\mathcal{F}^t$ represents the feasible set of problem (4.21) at the t-th iteration in Algorithm 1. The function $F^t(\alpha)$ is a monotonic decreasing function of α. Let the term α^* denote the solution of $F^t(\alpha^*) = 0$. Due to the monotone decreasing property of $F^t(\alpha)$, $F^t(\alpha^*) = 0$ if and only if α^* is equal to the optimal result of problem (4.21), *i.e.*, $\alpha^* = \check{\eta}(\mathcal{V}^*; \mathbb{A}^t)$ [100]. The algorithm for solving problem (4.21) is shown in Algorithm 4.

Algorithm 4 Dinkelbach Algorithm for Solving Problem (4.21)

```
1: Initialize $\alpha^0 = 0$ if $t = 0$, $\alpha^0 = \alpha^*$ in loop $t-1$ if $t \geq 0$, and the loop index $m = 0$.
2: repeat
3:     Solve problem (4.22) for given $\alpha^m$, and denote the solution for the problem by $\mathcal{V}_d^m$.
4:     Update the Dinkelbach auxiliary variable $\alpha^{m+1} = \check{\eta}(\mathcal{V}_d^m; \mathbb{A}^t)$.
5:     $m = m + 1$.
6: until $F^t(\alpha^{m+1}) \leq \theta_2$.
```

Due to the nature of the SCA-based algorithm and Dinkelbach algorithm, we can further cut the iteration times based on the following Lemma.

Lemma 4. *Denote the optimal Dinkelbach parameter α^* for two consecutive SCA iterations by $\alpha^*(t-1)$ and $\alpha^*(t)$. We have $\alpha^*(t-1) \leq \alpha^*(t)$, and $F^t(\alpha^*(t-1)) \geq F^t(\alpha^*(t)) = 0$.*

Proof. Denote the optimization results and the corresponding Dinkelbach parameter at iteration $t-1$ by $\mathcal{V}^*(t-1)$ and $\alpha^*(t-1)$, respectively. From Dinkelbach algorithm, we have $\alpha^*(t-1) = \check{\eta}^*(\mathcal{V}^*(t-1); \mathbb{A}^{t-1}) \leq \check{\eta}^*(\mathcal{V}^*)$. As shown in Lemma 3, the approximated function provides the global lower bound of the original optimization function, and the results have to be inside the feasible set of the approximate optimization function for the next iteration. Thus, $\check{\eta}^*(\mathcal{V}^*(t-1); \mathbb{A}^{t-1}) \leq \check{\eta}(\mathcal{V}^*(t-1); \mathbb{A}^t) \leq \check{\eta}^*(\mathcal{V}^*(t); \mathbb{A}^t)$. Therefore, $\alpha^*(t-1) \leq \alpha^*(t)$. Moreover, due to the monotonically decreasing nature of $F(\alpha)$, $F^t(\alpha^*(t-1)) \geq F^t(\alpha^*(t)) = 0$. □

Given Lemma 4, the initial point in iteration t, *i.e.*, $\alpha^0(t)$, in Algorithm 4 can be set at $\alpha^*(t-1)$ rather than 0 so that the computing efficiency of the optimization algorithm can be further improved.

4.4.3 Sub-problem Decomposition by ADMM

By now, the UAV computing energy efficiency maximization problem has been transformed into a solvable form. However, solving problem (4.22) is time-consuming due to multiple Second Order Cone (SOC) constraints and requires the local information exchange between the UAV and users. Therefore, we propose a distributed solution, in which users maximize their offloaded computing tasks in parallel while the UAV aims to minimize its energy consumption. The original problem is decomposed into several sub-problems without losing optimality, and the UAV and users solve the optimization problem cooperatively. Local information, such as the mobility of users and the propulsion energy consumption function of the UAV, is not required to be shared among users and the UAV.

We adopt ADMM technique to decompose problem (4.22) [101]. The optimization solution is achieved in an iterative manner. Firstly, we introduce an auxiliary variable, $\mathbb{G}$, which is solved by users:

$$\mathbb{G} = \begin{bmatrix} \ddot{\mathbf{Q}}_{1,1} & \dots & \ddot{\mathbf{Q}}_{N,1} & W_{1,1} & \dots & W_{N,1} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \ddot{\mathbf{Q}}_{1,K} & \dots & \ddot{\mathbf{Q}}_{N,K} & W_{1,K} & \dots & W_{N,K} \end{bmatrix}^T$$

where $\ddot{\mathbf{Q}}_{i,k}$ denotes the UAV location in time slot k expected by user i. Each user solves a part of the matrix $\mathbb{G}_i = [\ddot{\mathbf{Q}}_{i,1}, W_{i,1}; \dots; \ddot{\mathbf{Q}}_{i,K}, W_{i,K}]$, and updates it to the UAV. Then, the UAV generates its trajectory, $\mathbf{Q}$, and overall computation load allocation according to the uploaded matrix $\mathbb{G}$. Denote the overall amount of computation load processed in slot k at UAV by V_k, where $\mathbf{V} = [V_1; \dots; V_K]$. The results determined by the UAV are summarized by matrix $\mathbb{H}$, where $\mathbb{H} = [\mathbb{I}_{(N\times 1)}\mathbf{Q}; \mathbf{V}]$. $\mathbb{I}_{(N\times 1)}$ is a vector where all N entries are 1. By the end of the ADMM algorithm, the expected UAV trajectories should be unified and follow the flying constraints. The computation load should be allocated under the UAV computation capability. Thus, in the final optimal solution, the following constraint should be satisfied:

$$\mathbb{P}^T\mathbb{G} = \mathbb{H}, \tag{4.23}$$

where

$$\mathbb{P} = \begin{bmatrix} \mathbb{I}_{(N\times N)} & \mathbf{0}_{(N\times 1)} \\ \mathbf{0}_{(N\times N)} & \boldsymbol{\chi} \end{bmatrix}$$

The vector $\boldsymbol{\chi}$ represents the computation intensity for users' tasks, where $\boldsymbol{\chi} = [\chi_1; \ldots; \chi_N]$. The sub-matrices $\mathbb{I}_{(N\times N)}$ and $\mathbf{0}_{(N\times N)}$ denote N-by-N identity matrix and zero matrix, respectively.

In addition, for the non-orthogonal channel model, we introduce another auxiliary variable, $e_{i,k}$, which denotes the summation of $\xi_{j,k}$ in all other users except user i. This variable is used to decouple the correlated $\xi_{j,k}$ in (4.16) to facilitate the independent optimization process at each user. At the end of the optimization, $e_{i,k}$ should be equal to $\sum_{j\in\mathcal{I}/\{i\}} \xi_{j,k}$. For simplicity of presentation, we transform this constraint as follows:

$$\frac{1}{N}(e_{i,k} + \xi_{i,k}) = \bar{\xi}_k, \tag{4.24}$$

where $\bar{\xi}_k$ is the mean of $\{\xi_{1,k}, \ldots, \xi_{N,k}\}$. Then, the augmented Lagrangian function is formulated as follows:

$$\begin{aligned}\Gamma(\mathcal{V}_{\mathrm{A}}) = &-\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}} \check{R}_{i,k} + \alpha\big[\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}} E_{i,k}^{C,U}(\mathbf{W}) + \sum_{k\in\mathcal{K}} \hat{E}_k^F\big] \\ &+\mathrm{Tr}\big\{\mathbf{U}_1^T(\mathbb{P}^T\mathbb{G} - \mathbb{H})\big\} + \frac{\rho_1}{2}\|\mathbb{P}^T\mathbb{G} - \mathbb{H}\|_F^2 \\ &+\varpi\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}\Big\{U_{2,i,k}[\frac{1}{N}(e_{i,k} + \xi_{i,k}) - \bar{\xi}_k] \\ &+\frac{\rho_2}{2}[\frac{1}{N}(e_{i,k} + \xi_{i,k}) - \bar{\xi}_k]^2\Big\},\end{aligned} \tag{4.25}$$

where $\|\cdot\|_F$ is the notation representing the Frobenius norm. Set $\mathcal{V}_{\mathrm{A}}$ represents variables $\{\mathcal{V}, \mathbb{G}, \mathbb{H}, \mathbf{U}_1, \mathbf{U}_2\}$. Variables $\mathbf{U}_1 \in \mathbf{R}^{(N+1)\times K}$ and $\mathbf{U}_2 \in \mathbf{R}^{N\times K}$ are Lagrangian multipliers for the two auxiliary constraints, (4.23) and (4.24), respectively. The parameter ϖ indicates the channel model. $\varpi = 1$ denotes the case of the non-orthogonal channel access scheme, and $\varpi = 0$ denotes the case of the orthogonal channel access scheme. Two parameters, ρ_1 and ρ_2, are penalty parameters.

Problem (4.22) can be separated into two sub-problems. The sub-problem solved in user i is organized as follows:

$$
\begin{aligned}
\min_{\mathcal{V}_1}\ & -\sum_{k\in\mathcal{K}}\check{R}_{i,k}+\mathrm{Tr}\big\{(\mathbf{U}_{1,i}^{n-1})^T\mathbb{P}_i^T\mathbb{G}_i\big\}+\frac{\rho_1}{2}\|\mathbb{P}_i^T\mathbb{G}_i-\mathbb{J}_i^{n-1}\|_F^2\\
&+\varpi\Big\{\frac{U_{2,i,k}^{n-1}(e_{i,k})}{N}+\frac{\rho_2}{2}(\frac{e_{i,k}-e_{i,k}^{n-1}}{N}+\theta_{i,k}^{n-1})^2\\
&+\sum_{j\in\mathcal{I}/\{i\}}[-\frac{U_{2,j,k}^{n-1}\xi_{i,k}}{N}+\frac{\rho_2}{2}(\frac{\theta_{j,k}^{n-1}}{N-1}+\frac{\xi_{i,k}^{n-1}-\xi_{i,k}}{N})^2]\Big\} && (4.26a)\\
\text{s.t.}\ & \frac{(\|\ddot{\mathbf{Q}}_{i.k}-\mathbf{q}_{i,k}\|_2^2+H^2)n_0}{g_0}\le l_{i,k},\forall i,k && (4.26b)\\
& (4.7a),(4.7b),(4.10),(4.14e),(4.15h),(4.17),\\
& (4.15b),\text{ if }\varpi=0,\\
& (4.20),\text{ if }\varpi=1,
\end{aligned}
$$

and the sub-problem solved in the UAV is organized as follows:

$$
\begin{aligned}
\min_{\mathcal{V}_2}\ & \alpha\Big[\sum_{k\in\mathcal{K}}\frac{\kappa V_k^3}{\Delta^2}+\sum_{k\in\mathcal{K}}\hat{E}_k^F\Big]-\mathrm{Tr}\big\{(\mathbf{U}_1^n)^T\mathbb{H}\big\}+\frac{\rho_1}{2}\|\mathbb{P}^T\mathbb{G}^n-\mathbb{H}\|_F^2 && (4.27a)\\
\text{s.t.}\ & \frac{V_k}{\Delta}\le f_{max}^U,\forall k, && (4.27b)\\
& (4.14b),(4.14c),(4.15e),(4.19),(4.18).
\end{aligned}
$$

The term $(x)^{n-1}$ represents the variable x obtained in iteration $n-1$. The Lagrangian multipliers $\mathbf{U}_1$ and $\mathbf{U}_2$ are updated at each iteration as follows:

$$
\begin{aligned}
\mathbf{U}_1^n &= \mathbf{U}_1^{n-1}+\rho_1(\mathbb{P}^T\mathbb{G}^n-\mathbb{H})^n && (4.28a)\\
\mathbf{U}_{2,i,k}^n &= \mathbf{U}_{2,i,k}^{n-1}+\rho_2\theta_{i,k}^n, && (4.28b)
\end{aligned}
$$

where $\theta_{i,k}^n$ is

$$
\theta_{i,k}^n=\frac{1}{N}(e_{i,k}^n+\xi_{i,k}^n)-\bar{\xi}_k^n. \tag{4.29}
$$

$\theta_{i,k}$ represents the difference between the user expected interference and the real interference. The value of penalty parameters ρ_1 and ρ_2 influence the convergence rate of the proposed distributed optimization approach. The relation between the penalty parameters and convergence rate for the ADMM algorithm has been investigated in [102, 103]. At iteration n, problem (4.26) is solved by each user individually. The optimization variable set $\mathcal{V}_1$ includes $\{\delta_{i,k}, W_{i,k}, \ddot{\mathbf{Q}}_{i,k}, \xi_{i,k}, \mathbf{l}, \check{\mathbf{R}}, e_{i,k}\}$ for all $k \in \mathcal{K}$. To decompose the auxiliary constraint (4.23) for each user i, we introduce sub-matrices $\mathbb{P}_i$, $\mathbb{H}_i$, and $\mathbf{U}_{1,i}$, which are defined as follows: The parameter matrix $\mathbb{P}_i$ is the sub-matrix sliced from $\mathbb{P}$, where $\mathbb{P}_i = \mathbf{diag}\{1, \chi_i\}$. The matrix $\mathbb{J}_i$ is obtained by the information from the UAV, where $\mathbb{J}_i^n = [\mathbf{Q}^n; \mathbf{V}^n/N + \chi_i {W_i}^n - \sum_{j\in\mathcal{I}} \chi_j \mathbf{W}_j^n/N]$. The sub-matrix $\mathbf{U}_{1,i}$ is sliced from the dual variable, where $\mathbf{U}_{1,i} = [\mathbf{U}_1(i,:); \mathbf{U}_1(N+1,:)]$. The detailed decomposition process is omitted due to the space limit. Subsequently, problem (4.27) is solved by the UAV. The optimization variable set $\mathcal{V}_2$ includes $\{\mathbf{Q}, \boldsymbol{\omega}, \mathbf{A}, \hat{\mathbf{E}}^F\}$.

Lemma 5. *If the initial value of $\{\mathbf{e}^0, \boldsymbol{\xi}^0, \mathbf{U}_1^0, \mathbf{U}_2^0\}$ is shared and unified among all users and the UAV, only information from the UAV required for computing the sub-problem on the user side at each iteration is $\{\mathbb{J}_i^{n-1}, \boldsymbol{\theta}^{n-1}\}$.*

Proof. If the initial value is unified among the UAV and users, the dual variables are not required to be shared and can be computed locally by the UAV and users. For computing the dual variable $\mathbf{U}_{1,i}$ at n, the following knowledge is required: the updated global value $\mathbb{J}_i^{n-1}$, the historical value for the local information $\mathbb{G}_i^{n-1}$, and the historical value of the dual variable $\mathbf{U}_{1,i}^{n-1}$. Therefore, if $\mathbf{U}_{1,i}^0$ is identical to all users and the UAV, $\mathbf{U}_{1,i}^n$ can be synchronized according to the historical value and the value from the global variable. Similarly, $\mathbf{U}_2$ can be updated by users if the initial value is known. □

Given the condition in Lemma 4, the distributed algorithm is given in Algorithm 5. In each optimization iteration, user side computes and share matrix $\mathbb{G}$ to the UAV, and UAV computes and shares the matrix $\mathbb{J}$ to users. Meanwhile, when $\varpi = 1$, excepting contributing matrix $\mathbb{G}_i$, user i needs the information $e_{j,k}$ and $\xi_{j,k}$ from other users $j \in \mathcal{I}/\{i\}$ to evaluate the interference.

By the problem decomposition, at the user side, each user only aims to maximize its own offloading data given the UAV trajectory computed by the UAV-mounted cloudlet and the interference environment in the previous iteration. At the UAV-mounted cloudlet side, the UAV aims to minimize energy consumption under the users' expected UAV trajectories to collect enough workload. The trade-off between the received offloaded tasks and the energy consumption is controlled by the parameter α which is updated out of the ADMM algorithm loop. Meanwhile, the corresponding variables and constraints are split into two groups. This introduces three main advantages. Firstly, local variables and parameters, such as user location and user offloading constraints, are not required to be uploaded to the UAV. Similarly, UAV's mechanical parameters and settings are not required to be

Algorithm 5 ADMM Algorithm for Solving Problem (4.22)

```
1: Initialize variables $\{\mathbf{e}^0, \boldsymbol{\xi}^0, \boldsymbol{\theta}^0, \mathbb{H}^0, \mathbb{G}^0\}$ and dual variables $\{\mathbf{U}_1^0, \mathbf{U}_2^0\}$. Loop index $n = 0$.
2: repeat
3:     For each user i:
4:     If $\varpi = 0$: Wait until receive updated $\mathbb{J}_i^{n-1}$.
5:     If $\varpi = 1$: Wait until receive updated $\{\mathbb{J}_i^{n-1}, \boldsymbol{\theta}^{n-1}\}$.
6:     Calculate the dual variable $\mathbf{U}_{1,i}^{n-1} = \mathbf{U}_{1,i}^{n-2} + \rho_1(\mathbb{P}_i^T \mathbb{G}_i^{n-1} - \mathbb{J}_i^{n-1})$.
7:     Calculate the dual variable $\mathbf{U}_2$ for all $i \in \mathcal{I}$ by (4.28b).
8:     Solve problem (4.26).
9:     If $\varpi = 0$: Send $\mathbb{G}_i^n$ to the cloudlet.
10:    If $\varpi = 1$: Send $\{\mathbb{G}_i^n, \mathbf{e}_i^n, \boldsymbol{\xi}_i^n\}$ to the cloudlet.
11:    For the UAV-mounted cloudlet:
12:    Gather information from users to form matrix $\mathbb{G}^n$.
13:    Solve problem (4.27), and update $\mathbb{H}^n$.
14:    Update dual variable $\mathbf{U}_1^n$ by (4.28a)
15:    If $\varpi = 1$: Update variables $\theta_{i,k}^n \forall i, k$ by (4.29), and send the variables to users.
16:    $n = n + 1$.
17: until $|\Gamma^n(\mathcal{V}, \mathbb{G}, \mathbf{V}, \mathbf{U}_1, \mathbf{U}_2) - \Gamma^{n-1}(\mathcal{V}, \mathbb{G}, \mathbf{V}, \mathbf{U}_1, \mathbf{U}_2)| \leq \theta_3$.
```

shared to users for offloading optimization. Secondly, less configuration is required when the UAV is replaced. Thirdly, the main computation load in solving the problem is from the SOC programming. The SOC constraints are now decomposed and solved by users in parallel such that the computing efficiency can be improved. For ADMM algorithm, in the orthogonal channel model, there are two main distributed blocks: the user side and the UAV side. The convergence of ADMM is guaranteed when the number of blocks is no more than two. In the non-orthogonal channel model, since each user is required to compute the interference variable $e_{i,k}$ parallelly, convergence is not always guaranteed. Proximal Jacobian ADMM can be adopted to ensure the convergence, in which the proximal term $\frac{\tau}{2}||x_i - x_i^k||^2$ is further combined in the primal problem of the current algorithm [103].

4.4.4 Convergence and Complexity Analysis

The convergence for the three loops in Algorithms 3 to 5 is guaranteed. For the SCA-based algorithm, if the problem is feasible and the initial values of the approximate variables are in the feasible set of the original optimization problem (4.14), the algorithm convergence is ensured [99]. Moreover, the Dinkelbach algorithm can achieve the optimal α^* with a super-linear rate.

The computation complexity of the problem is dominated by the SOC programming [93, 104]. Suppose that Algorithm 5 runs $L_1 \times L_2$ iterations, where the SCA algorithm loop repeats L_1 times, and the loop for the Dinkelbach algorithm repeats L_2 times. The problem before decomposition, *i.e.*, problem (4.22), has KN SOC constraints in 4 dimensions, K SOC constraints in 7 dimensions, and KN SOC constraints in 2 dimensions, where $6KN + 4K$ variables participates in those constraints. The overall complexity can be $L_1 L_2 O\big(\sqrt{2KN + K}(6KN + 4K)(20KN + 49K + (6KN + 4K)^2)\big)$. After ADMM decomposition, for the sub-problem on the user side, there are K SOC constraints in 4 dimensions and K SOC constraints in 2 dimensions. Thus, the computation complexity is $L_1 L_2 O(1/\theta_3) O\big(\sqrt{2K}(5K)(20K + (5K)^2)\big)$ for each user. On the UAV side, the sub-problem contains K SOC constraints in 7 dimensions. The complexity is $L_1 L_2 O(1/\theta_3) O\big(\sqrt{K}(2K)(49K + (2K)^2)\big)$.

4.5 Proactive Trajectory Design Based on Spatial Distribution Estimation

So far, we have introduced the trajectory design and resource allocation for the scenario that all computation load information and user location are known. However, some IoT nodes are mobile [105], and knowing their future positions during the upcoming computation cycle can be difficult. Moreover, users needs to send the offloading requests at the beginning of the cycle. It means that the user may buffer the computing task until a new cycle begins, which introduces extra delay for waiting to send the request. Thus, the maximum queue delay may reach to T seconds. To deal with the above issues, in this subsection, we introduce an approach to estimate the spatial distribution of user locations in a cycle. The mobility of users is predicted by an unsupervised learning tool, kernel density estimation method [106], and the computation load of each user is considered in a stochastic model correspondingly. The UAV trajectory is optimized via the estimated knowledge about ground users. Thus, UAV can collect the offloaded tasks of users without requesting in advance.

To estimate the location of users, each user need to report its current location periodically. The sampled location of user i is represented by q_i. We use the sampled location to estimate the spatial distribution of users for the cycle, where the probability density function for the user at (x, y) is denoted as $f(x, y)$.

In order to compute $f(x, y)$, consider a small region R which is a rectangle area with side length of h_x and h_y, *i.e.*, Parzen window. To count the number of users falling within the region, we define the following function to indicate if user i is in the area:

$$C(q_i^x, q_i^y; R) = \begin{cases} 1, \text{if } \max\{\frac{||q_i^x - x||}{h_x}, \frac{||q_i^y - y||}{h_y}\} \leq \frac{1}{2} \\ 0, \text{otherwise}, \end{cases} \tag{4.30}$$

where (x, y) is the central point of the area. Thus, for a large N, the general expression for non-parametric density estimation is [106]

$$f(x, y) = \frac{1}{N h_x h_y} \sum_{i \in \mathcal{I}} C(q_i^x, q_i^y; R). \tag{4.31}$$

To establish continuous estimation function, a smooth Gaussian kernel is applied, where

$$\hat{f}(x, y) = \frac{1}{N\sqrt{h_x h_y}} \sum_{i \in \mathcal{I}} \frac{1}{2\pi} e^{-[\frac{(q_i^x - x)^2}{2h_x} + \frac{(q_i^y - y)^2}{2h_y}]}. \tag{4.32}$$

The term $\hat{f}(x, y)$ is the distribution of Gaussian kernel estimation. In (4.32), h_x and h_y represent the bandwidth of the Gaussian kernel rather than the side length of the Parzen window. To improve the estimation quality, the proper bandwidth, h_x and h_y, needs to be selected to minimize the error between the estimated density and the true density. In this work, the maximum likelihood cross-validation method [106] is adopted to determine the bandwidth h_x and h_y. The optimal bandwidth is

$$[h_x^*, h_y^*] = \text{argmax}\{\frac{1}{N} \sum_{i \in \mathcal{I}} \log \hat{f}_{-i}(q_i^x, q_i^y)\}, \tag{4.33}$$

where $\hat{f}_{-i}(q_i^x, q_i^y)$ is the estimated distribution in which user i is left out of the estimation.

In order to apply the estimated distribution into our proposed approach, we divide the working area of the UAV $\mathcal{A}$ into $G \times G$ sub-areas. For each sub-area $\mathcal{A}_i$, there is a virtual user located at the center of the area. The virtual user carries all the computing tasks in the sub-area. It is assumed that the distribution of the task input data size and user spatial location are independent. The expected length of input bits for the tasks generated by a user by $\mathbb{E}[X]$. Thus, the expected length of computing bits generated inside the sub-area $\mathcal{A}_i$ is

$$\mathbb{E}[I_i] = \mathbb{E}[X]\mathbb{E}[N_i] = \mathbb{E}[X] \int_{(x,y) \in \mathcal{A}_i} \hat{f}(x, y) dx dy, \tag{4.34}$$

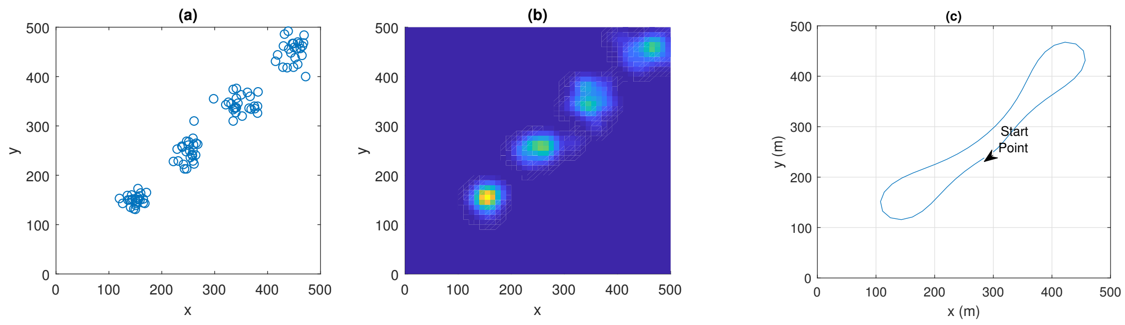
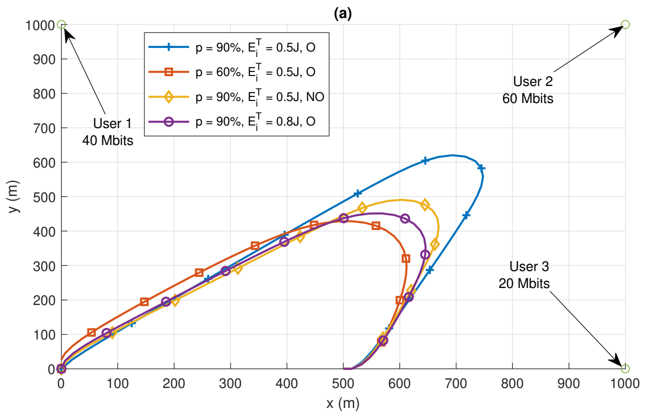


Figure 4.2: (a) Sampled mobile users location. (b) Estimated spatial distribution. (c) Corresponding UAV trajectory.

where $\mathbb{E}[N_i]$ denotes the expected number of users in the sub-area $\mathcal{A}_i$. Our proposed approach can now be adopted to solve the problem: In the new problem, there are G^2 virtual users participating in the computing task offloading, and virtual user i has $\mathbb{E}[I_i]$ computation load to be done in a cycle. The location of user i is fixed at the center of the sub-area. For the orthogonal channel model, the virtual user i shares a portion of $\mathbb{E}[N_i]/N$ of the channel bandwidth. As G increases, the performance of the estimation will be improved correspondingly. An example of the estimation is shown in Fig. 4.2, where the size of operation area $\mathcal{A}$ is set as 500m$\times$ 500m. As shown in Fig. 4.2(a), a hundred users are located in the area with a certain pattern. The estimated distribution results are shown in Fig. 4.2(b), where the optimal bandwidth is [270;318]. The corresponding UAV trajectory is shown in Fig. 4.2(c).

Table 4.1: Parameter settings for the computing scenario with three IoT nodes

Parameter	Value	Parameter	Value
B	3 MHz	κ	10^{-28}
σ^2	-80 dBm/Hz	γ_1	0.0037
χ_i	1550.7	γ_2	500.206
Δ	1.5 s	H	100 m
$a_{\max}$	50 m/s^2	P	100 mW
v_{max}	35 m/s	K	50



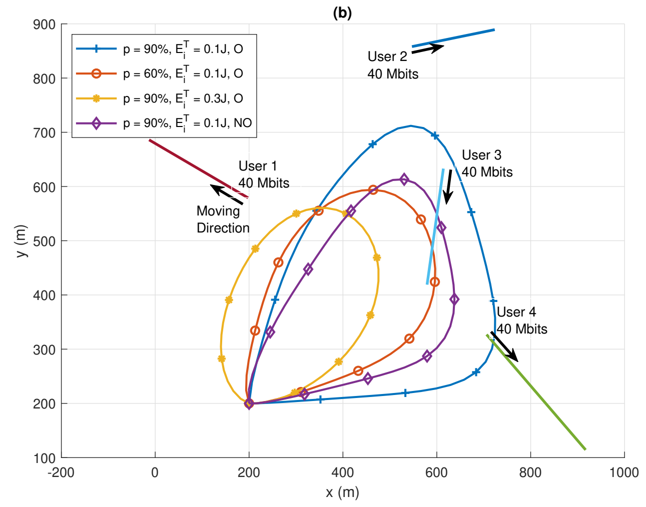


Figure 4.3: Optimal UAV trajectories with different parameter settings: (a) the three-node scenario; (b) the four-node scenario with user mobility, where the solid straight lines represent user trajectories, and the arrows represent user moving directions.

4.6 Numerical Results

In this section, we evaluate the performance of our proposed optimization approach. The parameter settings are given in Table 4.1. The channel gain parameter g_0 is -70 dB. Let the term p represent the percentage of computing tasks that have to be offloaded to the cloudlet, *i.e.*, $p = (\check{I}_i/I_i) * 100\%$. We consider that users have homogeneous offloading requirements in the simulation, *i.e.*, E_i^T and p are identical for all user. The term "NO" represents the non-orthogonal channel access scheme, and the term "O" represents the

Table 4.2: Parameter settings for the computing scenarios in Fig. 4.5

Index	p	E_i^T	Index	p	E_i^T	Index	Radius	E_i^T
1	90%	0.5 J	4	60%	0.5 J	7	200 m	0.5 J
2	90%	0.8 J	5	60%	0.8 J	8	200 m	0.8 J
3	90%	1.1 J	6	60%	1.1 J	9	200 m	1.1 J

orthogonal channel access scheme. We also consider the circular trajectory scheme as the benchmark, where the UAV moves around a circle within a cycle, with the circle center located at (0.5,0.5) km, and the radius is predefined. Two network scenarios are considered: a three-node scenario and a four-node scenario. In the three-node scenario, there are three users located at (0,1) km, (1,1) km, and (1,0) km, as shown in Fig. 4.3(a). At the beginning of the cycle, the UAV moves from the location (0,0) at an initial speed (-10,0) m/s. By the end of the cycle, the UAV returns to the final designated position at (0.5,0) km. In the four-node scenario, there are four users located at the randomly generated locations. The users travel at constant speeds which are random selected from [-3,-3] m/s to [3,3] m/s, as shown in Fig. 4.3(b). The UAV moves from the location (200,200) m at an initial speed (-10,0) m/s and returns to the initial position at the end of the cycle.

The UAV trajectory results obtained by the proposed approach are shown in Fig. 4.3. In the three-node case shown in Fig. 4.3(a), the UAV takes most of the time moving towards and stays around the location of user 2 due to high computing task loads of the user. With a higher minimum offloading requirement p, the UAV moves closer to users in order to collect more offloading tasks. Similarly, with a lower maximum communication energy requirement E_i^T, the UAV also moves closer to users to reduce the user's offloading communication energy consumption. Moreover, since the non-orthogonal access method has a higher channel capacity, under the same condition, the trajectory of the non-orthogonal case is shrunk to preserve the mechanical energy consumption compared to the orthogonal channel case. Similar results can be obtained in the four-node case, as shown in Fig. 4.3(b).

The comparisons of the energy efficiency with different settings are shown in Fig. 4.4. In Figs. 4.4(a) and 4.4(b), the x-axis represents the iteration number of the SCA-based algorithm loop. As shown in Fig. 4.4(a), the energy efficiency converges at $t = 30$ in the three-node scenario, while the number of iterations till convergence is increased in the four-node scenario. Moreover, for both scenarios, with loose user offloading requirements, the energy efficiency is improved due to the expanded optimization feasible set. In contrast, with tight user offloading requirements, the energy efficiency is decreased significantly due to high energy consumption for the UAV to move closer to the users.

For the three-node case, the ratio between the offloaded data amount and the overall

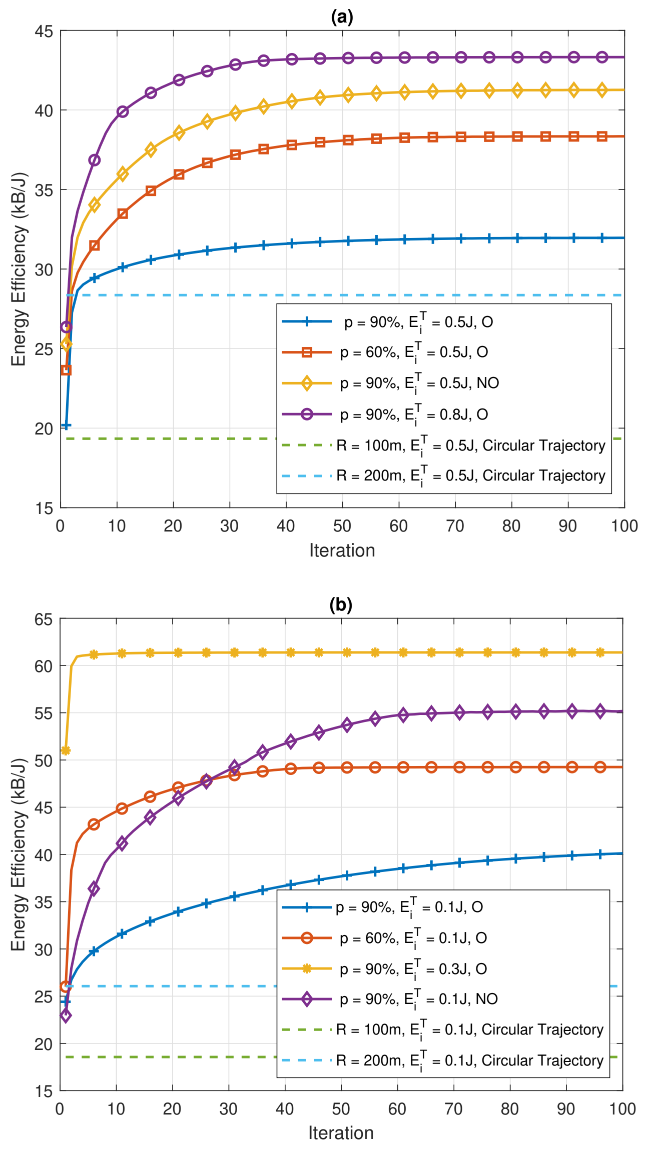


Figure 4.4: Energy efficiency versus main loop iteration number with different trajectory designs: (a) the three-node scenario; (b) the four-node scenario with user mobility.

computing data amount is shown in Fig. 4.5. The parameter settings for the indexes are given in Table 4.2, where the results by the proposed approach are shown in 1-6, and the results by the circular trajectory are shown in 7-9. For all scenarios, the proposed approach

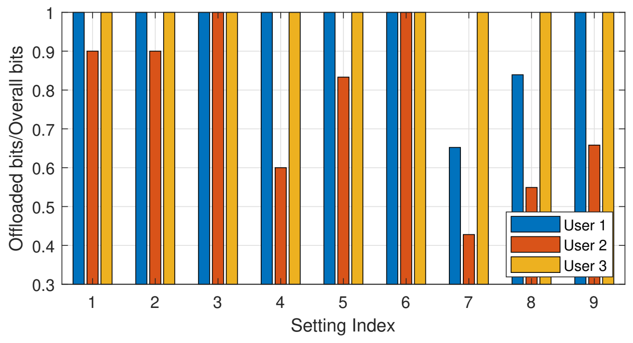


Figure 4.5: The ratio between the offloaded task data amount and the overall computing task data among generated by users with different parameter settings.

can achieve the minimum offloading requirement, while the circular trajectory scheme cannot guarantee to achieve the requirement. Moreover, when the maximum communication energy requirement E_i^T is increased, the UAV can collect more data even though its trajectory is far away from users compared to the case with a low E_i^T. The UAV also collects the extra offloaded tasks, which is beyond the users' requirement, to improve its energy efficiency.

The trade-off between the maximum offloading energy, *i.e.*, E_i^T, and the energy efficiency in the three-node case is shown in Fig. 4.6(a). As E_i^T increases, the energy efficiency of the UAV is increased at first and hits the ceiling in a high E_i^T. At that point, E_i^T is not the factor that limits the energy efficiency performance since all user's computing data is collected as shown in Fig. 4.6(c). When the energy efficiency reaches the maximum value, the UAV will find a path that has minimum energy consumption given that all tasks are offloaded. Furthermore, our proposed approach can improve the energy efficiency significantly compared to the circular trajectory.

The magnitudes of the UAV acceleration and velocity in the three-node case are shown in Fig. 4.7(a) and Fig. 4.7(b), respectively. The final velocity is constrained to be equal to the initial velocity. Note that the optimal velocity cannot be zero due to the characteristic of fixed-wing UAV. With the lower maximum energy requirement, both magnitudes of acceleration and velocity are increased, such that the UAV can move closer to users. With the higher energy requirement, the fluctuation on velocity and acceleration decreases to reduce the propulsion energy consumption of the UAV.

The ratio of the actual allocated transmit power to the maximum power, $\delta_{i,k}$, for the three users in a cycle is shown in Fig. 4.8(a). Note that the overall offloading communication energy is limited. For the user with high offloading demands, *i.e.*, user 2, the

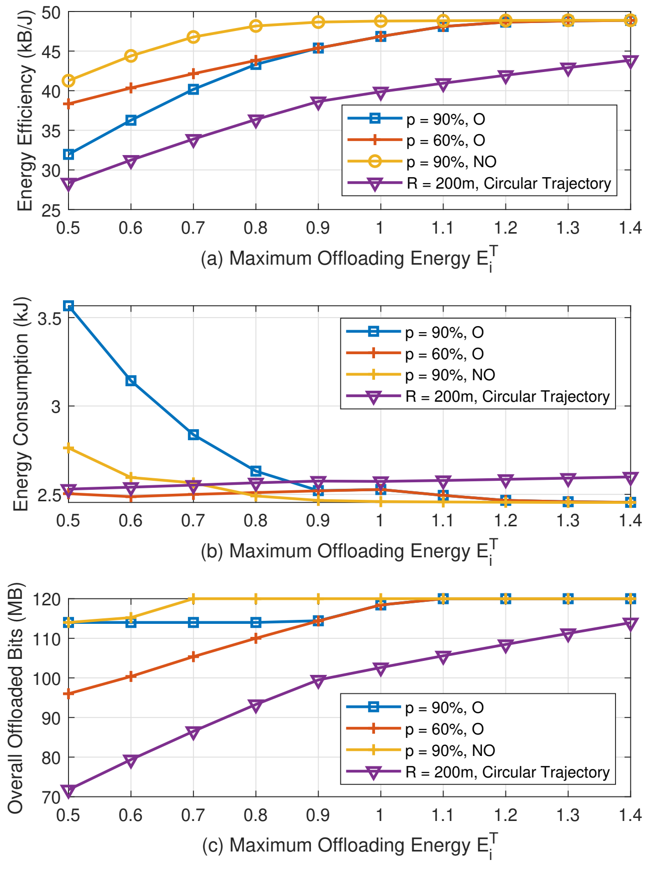


Figure 4.6: (a) Energy efficiency versus the maximum offloading communication energy with different settings. (b) Overall energy consumption in a cycle versus the maximum offloading communication energy. (c) Overall offloaded bits in a cycle versus the maximum offloading communication energy.

ratio is maximized when the UAV moves adjacent to it, while the ratio is minimized when the UAV moves away from it. The user tends to preserve the communication energy and starts the offloading only when the data rate is high. However, for user 3, the transmit power is still allocated when the UAV is far away from the location of the user for two reasons: Firstly, the maximum communication energy of the user allows user uploading the data even though the user transmission efficiency is low. Secondly, the UAV-mounted cloudlet prefers collecting the data in advance such that it can balance the computation

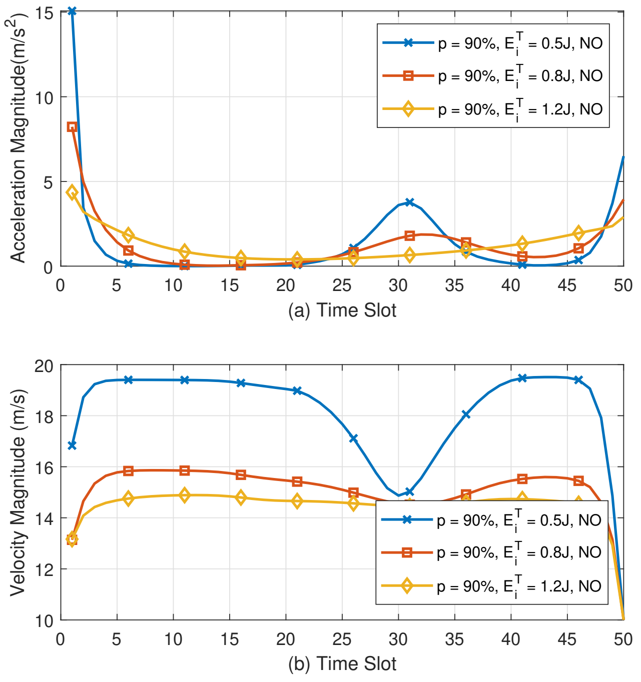


Figure 4.7: (a) The acceleration of the UAV in the cycle. (b) The speed of the UAV in the cycle.

load to reduce the computing energy cost. The computation load allocation of the cloudlet in the three-node case is shown in Fig. 4.8(b). Since the energy consumption is cubically increased as the computation load in a unit time increased (based on (4.8) and (4.13)), the computation load is preferred to be balanced among time slots. However, the computation load can only be executed after the corresponding tasks are offloaded into the cloudlet. Therefore, in the case with limited maximum communication energy, the allocated computation load is increased only when the new offloaded tasks are received. In contrast, with the loose maximum communication energy constraint, the workload fluctuation is reduced significantly to minimize the computing energy consumption.

Since the considered problem is non-convex and the proposed approach would achieve local optimums, optimization results depend on the initial value settings. We compare energy efficiency achieved by the proposed approach with different initial values of UAV speeds in Fig. 4.9. Specifically, in initial speed profiles 1 to 3, the initial UAV speeds $\mathbf{v}_k^0$ are $[-5\times\mathbf{1}_{(2\times K/2)}, 2.5\times\mathbf{1}_{(2\times K/2)}]$, $[5\times\mathbf{1}_{(2\times K/2)}, -2.5\times\mathbf{1}_{(2\times K/2)}]$, and $[10\times\mathbf{1}_{(2\times K/2)}, -5\times\mathbf{1}_{(2\times K/2)}]$, respectively, where $\mathbf{1}_{(2\times K/2)}$ is a $(2 \times K/2)$ matrix with all elements in the value of 1. As

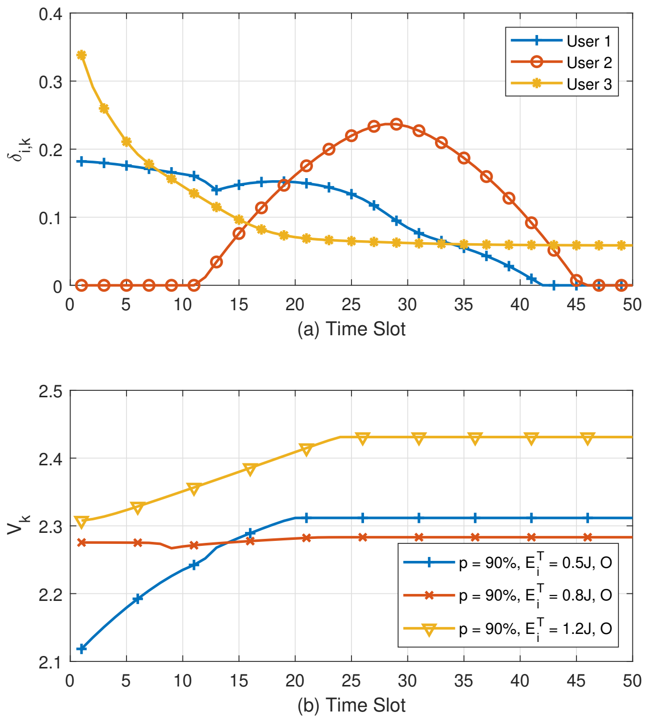


Figure 4.8: (a) The transmit power allocation among three users, where $p = 90\%$, and $E_i^T = 0.5$ J under orthogonal channel scenario. (b) The workload allocation with different settings.

shown in Fig. 4.9, the performance is different with various initial value settings. For speed profile 2, the proposed approach is trapped into local optimums when $E_i^T = 0.7$ and 0.8. One method to eliminate the effects from initial value settings in non-convex optimization is using the simulated annealing method [107], which runs the optimization approach with different initial values several times and selects the resource management policy with the best performance.

4.7 Summary

In this chapter, an optimization approach has been proposed to maximize the energy efficiency of a UAV-assisted MEC system, where the UAV trajectory design and resource allocation have been jointly considered. The non-convex and non-linear energy efficiency

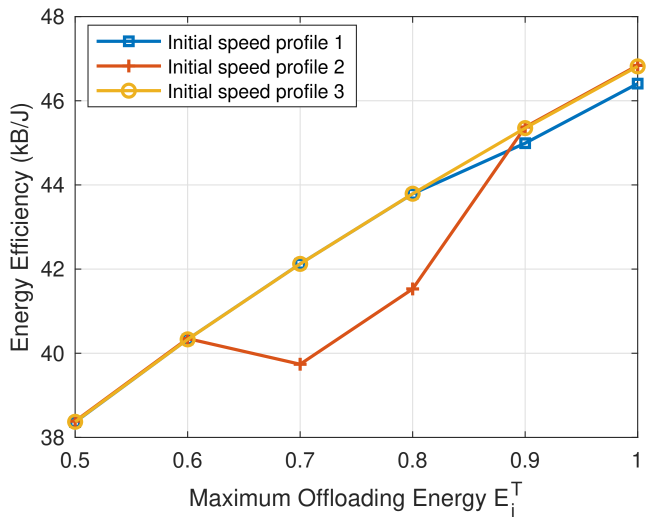


Figure 4.9: Energy efficiency with different initial value of UAV speeds for optimization and different E_i^T , where $p = 40\%$.

maximization problem has been solved in a distributed manner. Moreover, the node mobility estimation has been adopted to design a proactive UAV trajectory when the knowledge of user trajectory is limited.

Chapter 5

Collaborative Edge Computing in Vehicular Networks

In Chapters 3 and 4, we have investigated the computation offloading and task scheduling under an edge server. In this chapter, we investigate computing resource management for multiple edge servers to support seamless computing services for MUDs with high mobility. Vehicles traverse the communication coverage of edge servers and offload computing tasks to these servers. We aim to avoid service discontinuity due to mobility through collaboration among edge servers. The proposed collaborative computing includes two aspects: computing task scheduling algorithm, which schedules computing tasks offloaded by vehicles given a computation offloading policy, and AI-based collaborative computing approach, which determines the optimal computation offloading policy based on the dynamic network environment. Specifically, Section 5.1 introduces the background and motivation of the work, in which the contribution of the work is addressed. Section 5.2 describes the system model. Section 5.3 formulates the service delay minimization problem. In Section 5.4, we present the task partition and scheduling scheme, followed by an AI-based collaborative computing approach in Section 5.5. Section 5.6 presents simulation results.

5.1 Background and Motivation

Vehicular communication networks have drawn significant attention from both academia and industry in the past decade. Conventional vehicular networks aim to improve the driving experience and enable safety applications via data exchange in Vehicle-to-Everything

(V2X) communications. In the era of 5G, the concept of vehicular networks has been extended to IoV, in which intelligent and interactive applications are enabled by communication and computing technologies [78, 108]. A myriad of onboard applications can be implemented in the context of IoV, such as assisted/autonomous driving and platooning, urban traffic management, and onboard infotainment services [44, 109].

Although IoV technologies are promising, realizing the IoV applications still faces challenges. One of the obstacles is the limited onboard computation capability at vehicles. For example, a self-driving car with ten high-resolution cameras may generate 2 gigapixels per second of data, while 250 trillion computation operations per second are required to process the data promptly [110]. Processing such compute-intensive applications on vehicular terminals is energy-inefficient and time-consuming. To overcome the limitation, MEC is an emerging paradigm that provides fast and energy-efficient computing services for vehicle users [58, 61, 111]. Via Vehicle-to-Infrastructure (V2I) communications, resource-constrained vehicle users are allowed to offload their compute-intensive tasks to highly capable edge servers co-located with RSUs for processing. Meanwhile, compared to the conventional mobile cloud computing, the network delay caused by task offloading can be significantly reduced in MEC due to the proximity of the edge server to vehicles [112]. Consequently, some applications that require high computation capability, such as path navigation, video stream analytics, and object detection, can be implemented in vehicular networks with edge servers [113].

Despite the advantage brought by MEC-enabled vehicular networks, new challenges have emerged in task offloading and computing. One critical problem in MEC is to decide which edge servers should their computing tasks be offloaded to. In vehicular networks, the highly dynamic communication topology leads to unreliable communication links [114]. Due to the non-negligible computing time and the limited communication range of vehicles, a vehicle may travel out of the coverage area of an edge server during a service session, resulting in a service disruption. To support reliable computing services for high-mobility users, a service migration scheme has been introduced in [63]. Under this scope, when a user moves out of the communication area of the edge that the computing task was offloaded, the computing process will be interrupted, and the corresponding VM will be migrated to a new edge according to the radio association. In the urban area, where highly dense infrastructure are deployed, frequent service interruption would happen due to the dynamically changing radio association, which can significantly increase the overall computing service latency.

Alternatively, computing service reliability can be achieved by cooperation among edge servers. Different from service migration, which achieves service reliability by migrating the computing service according to the vehicle's trajectory, service cooperation improves the service reliability by accelerating task processing time. The computing task can be divided and computed by multiple servers in parallel or fully offloaded to a server with high computation capability at the cost of communication overhead [52, 115]. In this regard, the computing task can be forwarded to the edge server which is out of the user's communication range. Compared to service migration, in which edge servers only execute the task offloaded by the vehicles under their communication coverage, service cooperation allows edge servers processing the tasks offloaded by the vehicles out of their coverage for reducing the overall computing time. Nevertheless, multi-hop communications could result in significant transmission delay and waste communication spectrum resources in the task offloading process. The tradeoff between the communication overhead and the computation capability increases the complexity of the server assignment problem. In addition, although computing service latency can be reduced by cooperative computing, it is hard to guarantee service reliability for the vehicles with high mobility. The uncertainty of vehicle moving trajectories poses significant challenges in computing result delivery.

Motivated by the issues in the existing service migration and computing cooperation schemes, we present a computing collaboration framework to provide reliable low-latency computing in an MEC-enabled vehicular network. Once an edge server receives the computing tasks offloaded by a vehicle, it may partially or fully distribute the computing workload to another edge server to reduce computing latency. Furthermore, by selecting proper edge servers to deliver the computing results, vehicle users are able to obtain computing results without service disruption caused by mobility. Under this framework, we propose a novel task offloading and computing approach that reduces the overall computing service latency and improves service reliability. To achieve this objective, we firstly formulate a task partition and scheduling optimization problem, which allows all received tasks in the network to be executed with minimized latency given the offloading strategy. A heuristic task partition and scheduling approach is developed to obtain a near-optimal solution of the non-convex integer problem. In addition, we formulate the radio and computing association problem into a MDP. By characterizing stochastic state transitions in the network, MDP is able to provide proactive offloading policy for vehicles. An AI approach, DRL, is adopted to cope with the curse of dimensionality in MDP and unknown network state transitions caused by vehicle mobility. Specifically, a Convolutional Neural Network (CNN) based DRL is developed to handle the high-dimensional state space, and the Deep Deterministic Policy Gradient (DDPG) algorithm is adopted to handle the high-dimensional action space in the proposed problem. The major contributions of this work are:

1) We develop an efficient collaborative computing framework for MEC-enabled vehicular networks to provide low-latency and reliable computing services. To overcome the

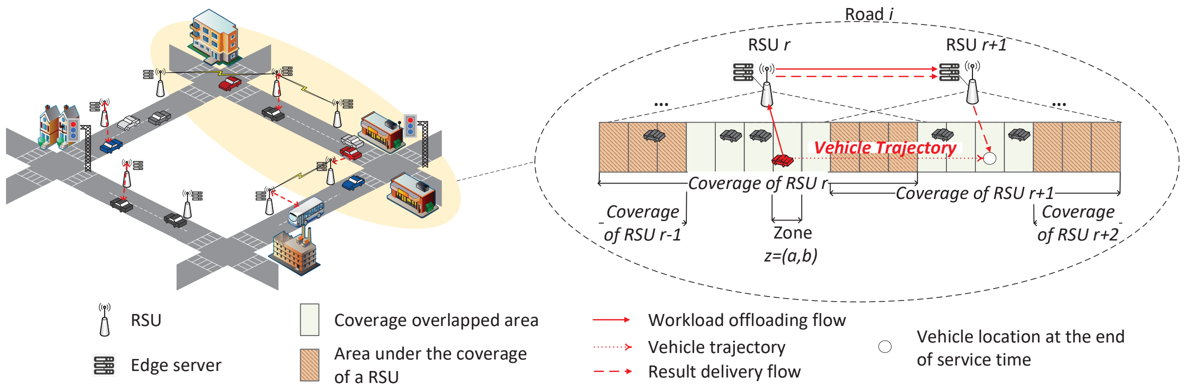


Figure 5.1: Collaboration computing framework.

complexity brought by the dynamic network topology, we propose a location-aware task offloading and computing strategy to guide MEC server collaboration.

2) We devise a task partition and scheduling scheme to divide the computing workload among edge servers and coordinate the execution order for tasks offloaded to the servers. Given the offloading strategy, our scheme can minimize the computing time by finding a near-optimal task scheduling solution with low time-complexity.

3) We further propose an AI-based collaborative computing approach, which utilizes a model-free method to find the optimal offloading strategy and MEC server assignment in a 2-dimensional transportation system. A CNN based DDPG technique is developed to capture the correlation of the state and action among different zones and accelerate the learning speed.

5.2 System Model

In this section, we present the system model of the considered problem.

5.2.1 Collaborative Edge Computing Framework

An MEC-enabled vehicular network is illustrated in Fig. 5.1. A row of RSUs, equipped with computing resources, provide seamless communication and computing service coverage for vehicles on the road. An RSU can also communicate with other RSUs within its communication range via wireless links. The set of RSUs is denoted by $\mathcal{R}$, where the index of RSUs is denoted by $r \in \mathcal{R}$. We assume that a global controller has full knowledge of the transportation network and makes offloading and computing decisions for all the vehicles in a centralized manner. In our model, a computing session for a task includes three steps:

1) Offloading: When a computing task is generated at a vehicle, the vehicle selects an RSU, which is under its communication range, and offloads the computing data of the task to the RSU immediately. In the example shown in Fig. 5.1, RSU r is selected to offload the computation load. Such RSU is referred to as the *receiver RSU* for the task.

2) Computing: After the computing task is fully offloaded, the receiver RSU can process the whole computing task or select another RSU to share the computation load. The RSU, which is selected to process the task collaboratively with the receiver RSU, is referred to as the *helper RSU* for the task.

3) Delivering: A vehicle may travel out of the communication range of its receiver RSU. Therefore, the controller may select an RSU, which could connect with the vehicle at the end of service session, to gather and transmit computing results. The RSU is referred to as the *deliver RSU*. To reduce the overhead, we limit the deliver RSU to be either the receiver RSU or the helper RSU of the task. In the example shown in Fig. 5.1, RSU $r + 1$ behaves as both the helper RSU and the deliver RSU for the computing task offloaded by the vehicle.

To reduce the decision space in task offloading and scheduling, instead of providing the offloading and computing policy to individual vehicles, we consider location-based offloading and computing policy. We divide each road into several zones with equal length, where the set of zones is denoted by $\mathcal{Z}$. The index of the zones is denoted by $z = (a, b) \in \mathcal{Z}$. The terms a and b represent the index of the roads and the index of the segments on the road, respectively, where $a \in \{1, \dots, A\}$, and $b \in \{1, \dots, B\}$. As the vehicle drives through the road, it traverses the zones consecutively. We assume that all vehicles in the same zone follow the same offloading and computing policy.[1] For simplicity, we evaluate the aggregated tasks for vehicles in each zone at a time slot, and refer to the tasks offloaded by zone z as task z in the remainder of the chapter. We suppose that the vehicle will not travel out of a zone during the time duration of a time slot, and vehicles can complete the offloading process of a task generated in a zone before it travels out of the zone. Denote

[1]The accuracy of vehicle locations will be improved when the length of the zone is reduced. In consideration of the length of a car, the length of a zone is larger than 5 m.

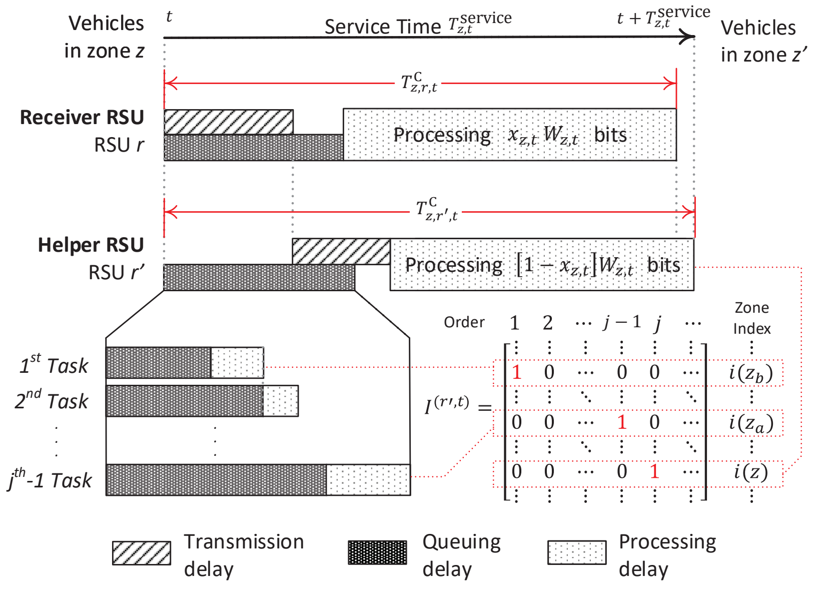


Figure 5.2: An example of the task offloading and computing process.

the set of vehicles in zone z and time slot $t \in \mathcal{T}$ as $\mathcal{V}_{z,t}$. The offloading decision for vehicles in zone z and time slot t is represented by a vector $\boldsymbol{\alpha}_{z,t} \in \mathbb{Z}_{+}^{|\mathcal{R}|}$, where $\sum_{r=1}^{|\mathcal{R}|} \alpha_{z,r,t} = 1$. The element $\alpha_{z,r,t}$ is 1 if RSU r is selected as the receiver RSU for the vehicles in zone z and time slot t, and 0 otherwise. Similarly, the collaborative computing decision for vehicles in zone z and time slot t is represented by a vector $\boldsymbol{\beta}_{z,t} \in \mathbb{Z}_{+}^{|\mathcal{R}|}$, where $\sum_{r=1}^{|\mathcal{R}|} \beta_{z,r,t} = 1$. The element $\beta_{z,r,t}$ is 1 if RSU r is selected as the helper RSU for the vehicles in zone z and time slot t, and 0 otherwise. In addition, the decision on result delivery is denoted by a binary variable $\gamma_{z,r,t}$, where $\gamma_{z,r,t}$ is 0 if the computing results are delivered by RSU r for task z in time slot t, and $\gamma_{z,r,t}$ is 1 if the computing results are delivered by RSU r.

5.2.2 Cost Model

The system cost includes two parts: the service delay and the penalty caused by service failure.

Service Delay

We adopt the task partition technique during task processing. Once a receiver RSU receives the offloaded task from vehicles in a zone, it immediately divides the task and offloads a part of the workload to the helper RSU of the corresponding zone. We denote the computing delay of task z corresponding to the receiver or helper RSU r in time slot t as $T^{\mathrm{C}}_{z,r,t}$. As shown in Fig. 5.2, the computing delay includes task offloading delay, queuing delay, and processing delay. Since the amount of output data is usually much smaller compared to the amount of input data, we neglect the transmission delay in result delivery [79, 97].

Firstly, task offloading comprises two steps: offloading tasks from vehicles to their receiver RSU and offloading the partial workload from the receiver RSU to the helper RSU. According to the propagation model in 3GPP standards [116], the path loss between a transmitter and a receiver with distance d (km) can be computed as:

$$\begin{aligned} L(d) = {} & 40(1 - 4 \times 10^{-3} D^{hb}) \log_{10} d - 18 \log_{10} D^{hb} \\ & + 21 \log_{10} f + 80 \text{ (dB)}, \end{aligned} \tag{5.1}$$

where the parameter f is the carrier frequency in MHz, and the parameter D^{hb} represents the antenna height in meter. We do not consider the shadowing effect of the channel. Denote the distance between the center point of zone z and the location of RSU r as $D_{z,r}$, and the distance between RSU r and r' as $D_{r,r'}$. The data rate for vehicles in zone z offloading task to RSU r is

$$r_{z,r} = B^{\mathrm{Z}} \log_2 \Big(1 + \frac{P^{\mathrm{V}} 10^{-L(D_{z,r})/10}}{\sigma_v^2}\Big), \tag{5.2}$$

where the parameter σ_v^2 denotes the power of the Gaussian noise in the V2I channel, P^{V} represents the vehicle transmit power, and B^{Z} represents the bandwidth reserved for vehicles in a zone. As the receiver RSU for task z, a signal-to-noise ratio threshold should be satisfied, where

$$\frac{P^{V} 10^{-L(D_{z,r})/10}}{\sigma_v^2} \geq \alpha_{z,r,t} \delta^{\mathrm{O}}, \forall t, z, r, \tag{5.3}$$

where δ^{O} is the signal-to-noise ratio threshold for data offloading. Assume that vehicles in a zone are scheduled to offload the tasks successively, and the channel condition is fixed in the duration of any computing task offloading. The transmission delay for offloading the computing data in zone z to the receiver RSU is:

$$T^{\mathrm{T}}_{z,t} = \sum_{r\in\mathcal{R}} \frac{\alpha_{z,r,t} W_{z,t}}{r_{z,r}}, \tag{5.4}$$

where $W_{z,t}$ represents the overall computing data generated by vehicles in zone z, *i.e.*, task z, and time slot t. In addition, the data rate between RSU r and RSU r' for forwarding the computing data offloaded from a zone is

$$r_{r,r'} = B^{\mathrm{R}} \log_2\Big(1 + \frac{P^{\mathrm{R}} 10^{-L(D_{r,r'})/10}}{\sigma_r^2}\Big), \tag{5.5}$$

where the parameter σ_r^2 represents the power of the Gaussian noise in the RSU to RSU channel, P^{R} represents the RSU transmit power, and B^{R} represents the bandwidth reserved for forwarding data offloaded from a zone. In data forwarding, the signal-to-noise constraint is also required to be satisfied, where

$$\frac{P^{R} 10^{-L(D_{r,r'})/10}}{\sigma_r^2} \geq \beta_{z,r',t} \delta^{\mathrm{O}}, \forall t, z, r, r'. \tag{5.6}$$

For computing task z in time slot t, the portion of workload to be processed by the receiver RSU and the helper RSU is denoted by $x_{z,t}$ and $1 - x_{z,t}$, respectively. Thus, the delay for forwarding the data to the deliver RSU is:

$$T^{\mathrm{R}}_{z,t} = \sum_{r\in\mathcal{R}} \sum_{r'\in\mathcal{R}} \frac{\alpha_{z,r,t} \beta_{z,r',t} (1 - x_{z,t}) W_{z,t}}{r_{r,r'}}. \tag{5.7}$$

Furthermore, after the task is offloaded to edge servers, the queuing delay may be experienced. Let set $\mathcal{Z}^{r,t}$ denote the zones which have tasks offloaded to RSU r, *i.e.*, $\{z|\alpha_{z,r,t} = 1\} \cup \{z|\beta_{z,r,t} = 1\}$, and let $i(z)$ represent the index of zone z in set $\mathcal{Z}^{r,t}$. We denote $N_{r,t}$ as the number of tasks offloaded in time slot t and assigned to the RSU r, where $N_{r,t} = \sum_z \alpha_{z,r,t} + \beta_{z,r,t}$. Then, a matrix, $\mathbb{I}^{(r,t)} \in \mathbb{Z}_+^{N_{r,t}\times N_{r,t}}$, can be defined to imply the processing order of tasks offloaded to RSU r in time slot t, where $I^{(r,t)}_{i(z),j} = 1$ if the task offloaded from zone z is scheduled as the j-th task to be processed among the other tasks offloaded in the same time slot. As shown in Fig. 5.2, the queuing delay of a task depends on the computing time of the task scheduled priorly. For the first task to be processed among the tasks offloaded in time slot t, the queuing delay stems from the computing time for the tasks offloaded in previous time slots. Thus, the queuing delay of task z in RSU r

can be formulated as follows:

$$T^{\mathrm{Q}}_{z,r,t} = \begin{cases} T^{\mathrm{Q0}}_{r,t}, \text{ if } I^{(r,t)}_{i(z),1} = 1, \\ \sum_{z'} \sum_j I^{(r,t)}_{i(z),j} I^{(r,t)}_{i(z'),j-1} T^{\mathrm{C}}_{z',r,t}, \text{ otherwise.} \end{cases} \tag{5.8}$$

The term $T^{\mathrm{Q0}}_{r,t}$ represents the latency for finishing the tasks offloaded in previous time slots $\{1, \ldots, t-1\}$, where

$$T^{\mathrm{Q0}}_{r,t} = \max\big\{ \sum_{z'} I^{(r,t)}_{i(z'),N_{r,t-1}} T^{\mathrm{C}}_{z',r,t-1} - \epsilon, 0 \big\}, \tag{5.9}$$

where ϵ is the length of a time slot.

We consider that data transmission and task processing run in parallel. After the task is offloaded and other tasks scheduled priorly are completed, the task can be processed by the dedicated server. The delay for processing task z offloaded to RSU r in time slot t can be formulated as

$$T^{\mathrm{P}}_{z,r,t} = \frac{\chi W_{z,t}[\alpha_{z,r,t} x_{z,t} + \beta_{z,r,t}(1 - x_{z,t})]}{C_r}, \tag{5.10}$$

where C_r denotes the computation capability (CPU-cycle frequency) of RSU r, and χ denotes the number of computation cycles needed to execute 1 bit of data.

Given the offloading delay, queuing delay, and processing delay, the computing delay for task z on RSU r can be formulated as follows:

$$T^{\mathrm{C}}_{z,r,t} = \max\{T^{\mathrm{T}}_{z,t} + \beta_{z,r,t} T^{\mathrm{R}}_{z,t}, T^{\mathrm{Q}}_{z,r,t}\} + T^{\mathrm{P}}_{z,r,t}. \tag{5.11}$$

Denote the overall service delay for the task offloaded from zone z in time slot t as $T^{\mathrm{service}}_{z,t}$. As shown in Fig. 5.2, the overall service delay depends on the longest computing time between the receiver RSU and the helper RSU. Thus, we have

$$T^{\mathrm{service}}_{z,t} = \max\{\sum_r \alpha_{z,r,t} T^{\mathrm{C}}_{z,r,t}, \sum_r \beta_{z,r,t} T^{\mathrm{C}}_{z,r,t}\}. \tag{5.12}$$

Service Failure Penalty

The mobility of vehicles brings uncertainty in result downloading. Service failure may occur if a vehicle is out of the coverage of its deliver RSU during the service session. Denote the zone that vehicle v is located when its computing result is delivered as m_v, *i.e.*, the location of vehicle $v \in \mathcal{V}_{z,t}$ in time slot $t + T^{\mathrm{Service}}_{z,t}$. Also, we denote the signal-to-noise ratio threshold for result delivering as δ^{D}. We introduce a variable $\mathbf{1}_{z,t}$ to indicate whether the computing service for task z offloaded in time slot t is successful or not, where

$$\mathbf{1}_{z,t} = \begin{cases} 1, \text{ if } P^{R} 10^{-L(D_{m_v,r})/10} \geq \sigma_r^2 \gamma_{z,r,t} \delta^{\mathrm{D}}, \forall v \in \mathcal{V}_{z,t} \\ 0, \text{ otherwise.} \end{cases} \tag{5.13}$$

5.3 Problem Formulation

Our objective is to minimize the weighted sum of the overall computing service delay for vehicle users and service failure penalty. The corresponding objective function can be formulated as follows:

$$\min_{\substack{\{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\mathbf{x},\\ \{\mathbf{I}^{(r,t)},\forall r,t\}\}}} \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{z\in\mathcal{Z}} \left\{ T_{z,t}^{\text{service}} \mathbf{1}_{z,t} + \lambda W_{z,t}(1-\mathbf{1}_{z,t}) \right\} \tag{5.14a}$$

$$\text{s.t. } (5.3), (5.6), \tag{5.14b}$$

$$\sum_{r\in\mathcal{R}} \alpha_{z,r,t} = 1, \sum_{r\in\mathcal{R}} \beta_{z,r,t} = 1, \sum_{r\in\mathcal{R}} \gamma_{z,r,t} = 1 \tag{5.14c}$$

$$\sum_{i=1}^{N_{r,t}} I_{i,j}^{(r,t)} = 1, \sum_{j=1}^{N_{r,t}} I_{i,j}^{(r,t)} = 1 \tag{5.14d}$$

$$0 \leq x_{z,t} \leq 1, \tag{5.14e}$$

$$\boldsymbol{\alpha}_{z,t}, \boldsymbol{\beta}_{z,t} \in \mathbb{Z}_{+}^{|\mathcal{R}|}, \tag{5.14f}$$

$$\boldsymbol{I}^{(r,t)} \in \mathbb{Z}_{+}^{N_{r,t}\times N_{r,t}}, \tag{5.14g}$$

where λ represents per-unit penalty, for the case when the computation offloading service fails.

The optimization variables include three aspects: edge server selection, *i.e.*, $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$, task partition, *i.e.*, $\mathbf{x}$, and task scheduling, *i.e.*, $\{\mathbf{I}^{(r,t)}, \forall r, t\}$. It can be seen that Problem (5.14) is a mixed-integer nonlinear optimization problem. Solving the above problem directly by conventional optimization methods is challenging. Furthermore, the decision dimension of the problem is too large to apply model-free techniques directly. Taking the variable of task execution order as an example, *i.e.*, $\mathbf{I}^{(r,t)}$, there are $N_{r,t} \times N_{r,t}$ number of decisions to be determined for a server in a time slot. The number of combinations of scheduling decisions is at least $(|\mathcal{Z}|/|\mathcal{R}|)! \times |\mathcal{R}| \times |\mathcal{T}|$, in which tasks are evenly assigned to servers and each task is processed by only one server. Thus, to reduce the decision dimension of the problem, we divide Problem (5.14) into two sub-problems: i) task partition and scheduling problem, and ii) edge server selection problem. In the task partition and scheduling problem, we aim to obtain the optimal task partition ratio and the execution order to minimize the computing latency given the offloading policy $\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$. After that, we re-formulate the edge server selection problem as an MDP and utilize the DRL technique to obtain the optimal offloading and computing policy.

5.4 Task Partition and Scheduling

Multiple tasks offloaded from different zones can be received by an edge server in a time slot. The computing tasks can be processed only if the tasks scheduled priorly are executed. As a result, the overall computing time may vary depending on the task execution order in edge servers. In addition, the workload of a task can be divided and offloaded to two edge servers, *i.e.*, receiver and helper RSUs. Workload allocation for a task also affects the overall service time. Therefore, we study task partition and scheduling to minimize the service latency given the offloading policy $\{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$. Based on Problem (5.14), the delay minimization problem can be formulated as follows:

$$\min_{\mathbf{x}, \{\mathbf{I}^{(r,t)}, \forall r,t\}} \sum_{z \in \mathcal{Z}} T^{\text{service}}_{z,t} \tag{5.15a}$$

$$\text{s.t.} \;\; (5.14d), (5.14e), (5.14g). \tag{5.15b}$$

Problem (5.15) is a mixed-integer programming, which involves a continuous variable $\mathbf{x}$ and an integer matrix variable $\{\mathbf{I}^{(r,t)}, \forall r, t\}$. Moreover, even if $\mathbf{x}$ is known, the remaining integer problem is a variation of the traveling salesman problem, which is an NP-hard problem. To reduce the time-complexity in problem-solving, we exploit the properties of task partition and scheduling and develop a heuristic algorithm to obtain an approximate result efficiently. To simplify the notations, we eliminate the time index t in the remainder of the section since we consider the scheduling scheme for the tasks offloaded in one time slot. We further denote $r(z)$ and $h(z)$ as the index of receiver and helper RSUs for task z, respectively.

Lemma 6. *If no task is queued after task z for both the receiver RSU and the helper RSU, the optimal partition ratio for the task x^*_z is* $\min\{\max\{0, \hat{x}_z\}, 1\}$, *where $\hat{x}_z$ can be* *determined by Eq.* (5.16)*.*

$$\hat{x}_z = \begin{cases} \frac{T^Q_{z,h(z)} - \max\{T^Q_{z,r(z)}, T^T_z\} + \chi W_z / C_{h(z)}}{\chi W_z / C_{r(z)} + \chi W_z / C_{h(z)}}, & \textit{if } T^Q_{z,h(z)} - \max\{T^Q_{z,r(z)}, T^T_z\} \geq \frac{\chi W_z}{C_{r(z)}} \\ & \quad -\chi R_{r(z),h(z)} (T^Q_{z,h(z)} - T^T_z)(\frac{1}{C_{r(z)}} + \frac{1}{C_{h(z)}}) \\ \frac{T^T_z - \max\{T^Q_{z,r(z)}, T^T_z\} + \chi W_z / C_{h(z)} + W_z / R_{r(z),h(z)}}{\chi W_z / C_{r(z)} + \chi W_z / C_{h(z)} + W_z / R_{r(z),h(z)}}, & \textit{otherwise.} \end{cases} \tag{5.16}$$

Proof. Without considering the tasks queued later, the service time of task z can be minimized by solving the following problem:

$$\min \max\{T^{\mathrm{C}}_{z,r(z)}, T^{\mathrm{C}}_{z,h(z)}\} \qquad \text{s.t. (5.14e).} \tag{5.17}$$

Given that $0 < x_z < 1$, the optimal task partition strategy exists when $T^{\mathrm{C}}_{z,r(z)} = T^{\mathrm{C}}_{z,h(z)}$. The optimal task partition ratio is $x^*_z = \hat{x}_z$. In addition, $x^*_z = \max\{0, \hat{x}_z\} = 0$ when the helper RSU can fully process task z in a shorter service time comparing to the queuing time in the receiver RSU, *i.e.*, $\max\{T^{\mathrm{Q}}_{z,r(z)}, T^{\mathrm{T}}_z\} \geq \max\{T^{\mathrm{Q}}_{z,h(z)}, T^{\mathrm{T}}_z + \frac{\chi W_z}{R_{r(z),h(z)}}\} + \frac{\chi W_z}{C_{h(z)}}$. Otherwise, $x^*_z = \min\{1, \hat{x}_z\} = 1$, when the receiver RSU can process task z by itself in a shorter service time comparing to the queuing time in the helper RSU, *i.e.*, $\max\{T^{\mathrm{Q}}_{z,r(z)}, T^{\mathrm{T}}_z\} \leq T^{\mathrm{Q}}_{z,h(z)} - \frac{\chi W_z}{C_{r(z)}}$. □

Lemma 6 shows the optimal partition ratio from the individual task perspective. However, multiple tasks could be offloaded from different zones to an RSU, where the role of the RSU could be different for those tasks. The task partition strategy for a single task could affect the computing latency for the task queued later. Therefore, we will investigate the optimality of the task partition scheme in Lemma 6 in terms of minimizing the overall service time for all tasks $z \in \mathcal{Z}$.

Lemma 7. *Assume that the following conditions are met:*

- *The computation capability C_r is identical for all edge servers.*
- *The receiver RSU and helper RSU are different for each task, i.e., $r(z) \neq h(z)$.*
- *For the helper RSUs for all tasks, the queuing time is not shorter than the offloading time, i.e., $T^Q_{z,h(z)} \geq T^T_{z,r(z)} + T^R_{r(z),h(z)}, \forall z, r$.*

Then, given the execution order of tasks, the optimal solution of Problem (5.15) *follows the results shown in Lemma 6, i.e., $x^*_z = \min\{\max\{0, \hat{x}_z\}, 1\}, \forall z$.*

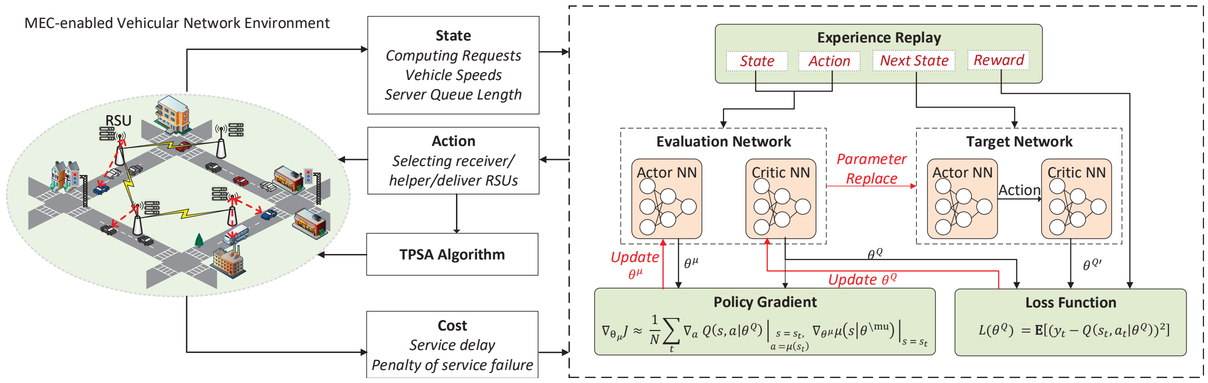


Figure 5.3: AI-based collaborative computing approach.

Proof. See Appendix C. □

We have proved that, given the task execution order, the partition ratio in Lemma 1 is the optimal solution for Problem (5.15) under certain assumptions. Next, we will explore the optimal scheduling order given the workload allocation policy.

Lemma 8. *Consider only one available RSU in the system, i.e., $r(z) = h(z)$. Under the assumption in which the offloading time is proportional to the size of the task, the optimal task execution order is to schedule the task with the shortest service time first.*

Proof. See Appendix D. □

According to the properties provided in Lemmas 6-8, we design a heuristic algorithm to schedule the task execution order and allocate workload among RSUs. The full algorithm, i.e., Task Partition and Scheduling Algorithm (TPSA), is presented in Algorithm 6. In the algorithm, we allocate the task that has the shortest service time first. For each task, we divide the workload between the receiver RSU and helper RSU according to the optimal partition ratio in Lemma 6. In the worst case, in which all zones have tasks to offload in a time slot, the algorithm requires $|\mathcal{Z}|(|\mathcal{Z}|+1)/2$ iterations to compute the task partition and scheduling results, which can still provide fast responses in the dynamic environment.

5.5 AI-Based Collaborative Computing Approach

To deal with the server selection problem, we utilize a DRL technique to conduct the complex decision-making problem in a dynamic environment. To implement the DRL

Algorithm 6 Task Partition and Scheduling Algorithm (TPSA)

```
1: At time slot $t$, initialize set $\mathcal{S} = \{z|W_{z,t} \neq 0\}$.
2: Initialize $\psi_r = T^{\mathrm{Q0}}_{r,t}$, $\mathbf{I}^{(r,t)} = \mathbf{0}$, and $j_r = 1, \forall r$.
3: while $|\mathcal{S}| \neq 0$ do
4:     Initialize $Q_z = 0, \forall z \in \mathcal{S}$.
5:     for Task $z = 1 : |\mathcal{S}|$ do
6:         Update $r(z) = \{r|\alpha_{z,r,t} = 1\}$ and $h(z) = \{r|\beta_{z,r,t} = 1\}$.
7:         Update partition ratio $x_z = \min\{\max\{0, \hat{x}_z\}, 1\}$, where $\hat{x}$ is obtained by (5.16).
8:         Update $\hat{\psi}_{z,r(z)} = \psi_{r(z)} + T^{\mathrm{C}}_{z,r(z)}$.
9:         Update $\hat{\psi}_{z,h(z)} = \psi_{h(z)} + T^{\mathrm{C}}_{z,h(z)}$.
10:        If $x_z = 1$, then $Q_z = \hat{\psi}_{z,h(z)}$.
11:        If $x_z = 0$, then $Q_z = \hat{\psi}_{z,r(z)}$.
12:        If $0 < x_z < 1$, then $Q_z = (\hat{\psi}_{z,r(z)} + \hat{\psi}_{z,h(z)})/2$.
13:    end for
14:    Find $z^* = \mathrm{argmin}_z Q_z$.
15:    Update $\psi_{r(z^*)} = \hat{\psi}_{z^*,r(z^*)}$ and $\psi_{h(z^*)} = \hat{\psi}_{z^*,h(z^*)}$.
16:    Update order matrix $I^{r(z^*),t}_{z^*,j_{r(z^*)}} = 1$, and $I^{h(z^*),t}_{z^*,j_{h(z^*)}} = 1$.
17:    Update $j_{r(z^*)} = j_{r(z^*)} + 1$, and $j_{h(z^*)} = j_{h(z^*)} + 1$.
18:    $\mathcal{S} = \mathcal{S} \backslash \{z^*\}$.
19: end while
20: $T^{\mathrm{Q0}}_{r,t+1} = \psi_r - \epsilon, \forall r$.
```

method, we first re-formulate the problem into an MDP. An MDP can be defined by a tuple $(\mathbb{S}, \mathbb{A}, \mathbb{T}, \mathbb{C})$, where $\mathbb{S}$ represents the set of system states; $\mathbb{A}$ represents the set of actions; $\mathbb{T} = \{p(s_{t+1}|s_t, a_t)\}$ is the set of transition probabilities; and $\mathbb{C}$ is the set of real-value cost functions. The term $C(s, a)$ represents the cost when the system is at state $s \in \mathbb{S}$ and an action $a \in \mathbb{A}$ is taken. A policy π represents a mapping from $\mathbb{S}$ to $\mathbb{A}$. In our problem, the state space, action space, and cost model in an MDP are summarized as follows:

1) State space: In time slot t, the network state, s_t, includes the computing data amount in zones, *i.e.*, $\{W_{z,t}, \forall z\}$, the average vehicle speed, *i.e.*, $\{v_{z,t}, \forall z\}$, and the delay for edge servers to finish the tasks offloaded in previous time slots $\{1, \ldots, t-1\}$, *i.e.*, $\{T^{\mathrm{Q0}}_{r,t}, \forall r\}$.

2) Action space: For zone z and time slot t, the action taken by the network includes

three elements: the index of receiver RSU, helper RSU, and deliver RSU, which can be represented by $\{a^1_{z,t}, a^2_{z,t}, a^3_{z,t}\}$, receptively.

3) Cost model: Given the state-action pair, the overall service time can be available by the TPSA algorithm. Thus, according to the objective function (5.14), the cost function can be formulated as

$$C(s_t, a_t) = \sum_{z \in \mathcal{Z}} \Big\{ T^{\text{service}}_{z,t} \mathbf{1}_{z,t} + \lambda W_{z,t}(1 - \mathbf{1}_{z,t}) \Big\}. \tag{5.18}$$

Then, to obtain the expected long-term discounted cost, the value function V of state s is

$$V(s, \pi) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t C(s_t, a_t) | s_0 = s, \pi \Big], \tag{5.19}$$

where the parameter γ is a discount factor. By minimizing the value function of each state, we can obtain the optimal offloading and computing policy π^*; that is,

$$\pi^*(s) = \arg\min_a \sum_{s'} p(s'|s, a)[C(s, a) + \gamma V(s', \pi^*)]. \tag{5.20}$$

Due to the limited knowledge on transition probability between the states and the sizeable state-action space in the network, the traditional dynamic programming method is not able to compute the optimal policy efficiently. Therefore, we adopt DRL to solve the proposed server selection problem. There are three common DRL algorithms: DQN, Actor-Critic (AC), and DDPG. DQN is a powerful tool to obtain the optimal policy with a high dimension in the state space. Besides an online neural network (evaluation network) to learn the Q value, a frozen network (target network) and the experience replay technique are applied to stabilize the learning process. However, the method shows the inefficiency on the network with a high dimension in the action space, while in our problem, the large number of zones leads the high dimension in both state and action spaces. On the other hand, both AC and DDPG tackle the problem with a high action dimension by the policy gradient technique. Two networks, *i.e.*, actor and critic networks, are adopted, in which the critic evaluates the Q value, and the actor updates policy parameters in the direction suggested by the critic. Moreover, DDPG combines the characteristics of DQN on top of the AC algorithm to learning the Q value and the deterministic policy by the experience relay and the frozen network, thereby helping reach the fast convergence [117]. In this chapter, we exploit the DDPG algorithm to obtain the optimal collaborative computing policy in vehicular networks.

The illustration of our AI-based collaborative computing approach is shown in Fig. 5.3. The system states are observed from the MEC-enabled vehicular network. After state s_t is obtained, the optimal server selection policy can be computed by the DDPG algorithm. According to the server selection results, the corresponding task partition and scheduling policy can be obtained by the proposed TPSA algorithm. Then, the cost of the corresponding state-action pair and the next system state can be observed from the environment. The state transition set (s_t, a_t, r_t, s_{t+1}) is stored in the replay memory for training the neural networks. In DDPG, four neural networks are employed. Two of the four networks are evaluation networks, where the weights are updated when the neural network is trained, and the other two networks are target networks, where the weights are replaced periodically from the evaluation network. For both evaluation and target networks, two neural networks, *i.e.*, actor and critic networks, are adopted to evaluate the optimal policy and Q value, respectively. The weights in evaluation and target critic networks are denoted by θ^Q and $\theta^{Q'}$, and the weights in evaluation and target actor networks are denoted by θ^μ and $\theta^{\mu'}$, respectively.

In each training step, a batch of experience tuples are extracted from the experience replay memory, where the number of tuples in a mini-batch is denoted by N. The critics in both evaluation and target networks approximate the value function and compute the loss function L, where

$$L(\theta^Q) = \mathbf{E}\Big[\big(y_t - Q(s_t, a_t|\theta^Q)\big)^2\Big]. \tag{5.21}$$

The term $Q(s_t, a_t|\theta^Q)$ represents the Q function approximated by the evaluation network. The value of y_t is obtained from the value function approximated by the target network, where

$$y_t = C(s_t, a_t) + \gamma Q(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'}). \tag{5.22}$$

The term $\mu'(s_{t+1}|\theta^{\mu'})$ represents the action taken at s_{t+1} given by the target actor network. By minimizing the loss function (5.21), the weights in the evaluation critic, *i.e.*, θ^Q, can be updated. On the other hand, to update the weights of the evaluation actor network, the policy gradient can be represented as

$$\nabla_{\theta_\mu} J \approx \frac{1}{N}\sum_t \nabla_a Q(s, a|\theta^Q)|_{\substack{s=s_t,\\ a=\mu(s_t)}} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t}. \tag{5.23}$$

From (5.23), it can be seen that actor weights are updated in each training step according to the direction suggested by the critic.

Although the DDPG algorithm is able to tackle the problem with a high dimension of state and action spaces, it is inefficient to apply the DDPG algorithm directly in our

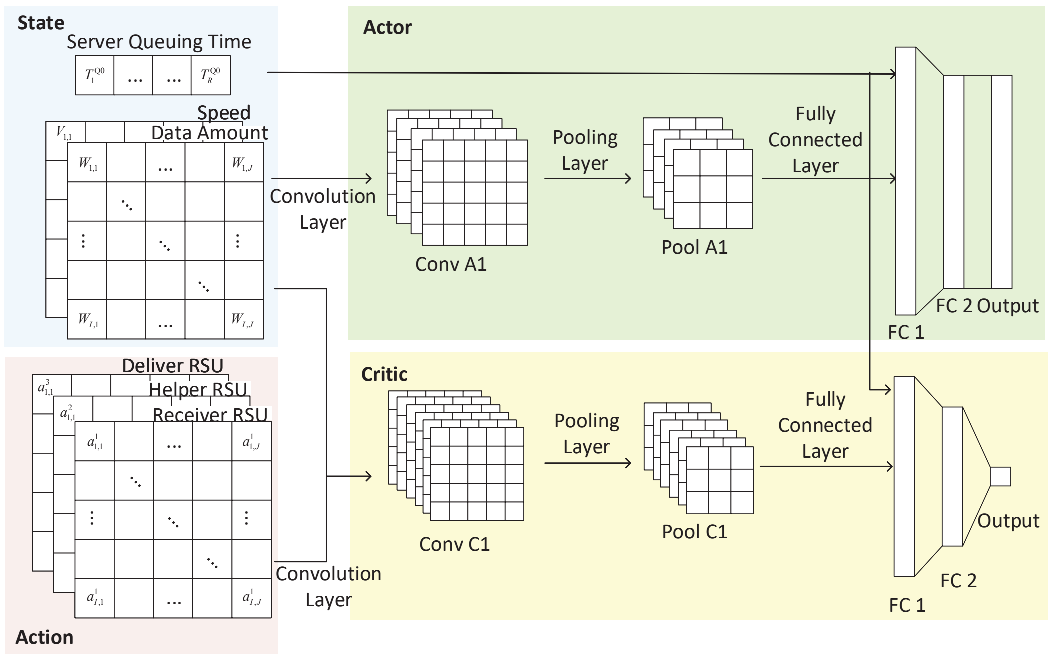


Figure 5.4: The structure of actor and critic neural network.

problem due to the 2-dimensional transportation network and the multiple dimensions of the input. A huge number of neurons in a network will be deployed if the conventional neural network with fully connected layers is adopted. To improve the algorithm efficiency, we utilize CNN in both actor and critic networks to exploit the correlation of states and actions among different zones. The structure of actor and critic networks is shown in Fig. 5.4. Before fully connected layers, convolution layers and pooling layers are applied to learn the relevant features of the inputs among zones. Due to the weight sharing feature of CNN filters, the number of training parameters can be significantly reduced compared to the network with fully connected layers [118]. After several convolution and pooling layers, the output of the CNN combines the state of edge servers and forwards to fully connected layers.

The proposed AI-based collaborative computing approach is provided in Algorithm 7, where τ is a small number less than 1. In our algorithm, to learn the environment efficiently, the system will continuously train the parameter by N_t times after N_e time step, where $N_e > N_t$.

Algorithm 7 AI-based Collaborative Computing Approach

```
1: Initialize critic network $Q(s_0, a_0|\theta^Q)$ and actor network $\mu(s_0|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
2: Initialize target network with weights $\theta^{Q'} = \theta^Q$ and $\theta^{\mu'} = \theta^\mu$.
3: Initialize the experience replay buffer.
4: Initialize a random vector $\mathcal{N}$ as the noise for action exploration.
5: for episode = 1:G do
6:     Initialize environment, and observe the initial state $s_0$.
7:     for time slot $t = 1 : T$ do
8:         Select action $a_t = \mu(s|\theta^\mu) + \mathcal{N}$.
9:         Let $\alpha_{z,a^1_{z,t},t}$, $\beta_{z,a^2_{z,t},t}$, and $\gamma_{z,a^3_{z,t},t}$ equal to 1.
10:        Compute the task partition and scheduling results by Algorithm 1.
11:        Observe next state $s_{t+1}$ and cost $C(s_t, a_t)$.
12:        Store transition $(s_t, a_t, r_t, s_{t+1})$ into the experience replay buffer. Delete the
           oldest transition set if the buffer is full.
13:        if $k$ mod $N_e == 0$ then
14:            for $j = 1 : N_t$ do
15:                Sample a mini-batch of $N$ samples.
16:                Update $y_t$ by (5.22).
17:                Update the weights in the evaluation critic network by minimizing the
                   loss in (5.21).
18:                Update the weights in the evaluation actor network using sampled policy
                   gradient presented in (5.23).
19:                Update target networks: $\theta^{Q'} = \tau\theta^Q + (1-\tau)\theta^{Q'}$; $\theta^{\mu'} = \tau\theta^\mu + (1-\tau)\theta^{\mu'}$.
20:            end for
21:        end if
22:    end for
23: end for
```

5.6 Performance Evaluation

In this section, we first present the efficiency of the proposed TPSA algorithm in task partition and scheduling. Then, we evaluate the performance of the proposed AI-based collaborative computing approach in a vehicular network simulated by VISSIM [119], where TPSA is applied to schedule computing tasks according to the policy given by the DDPG algorithm.

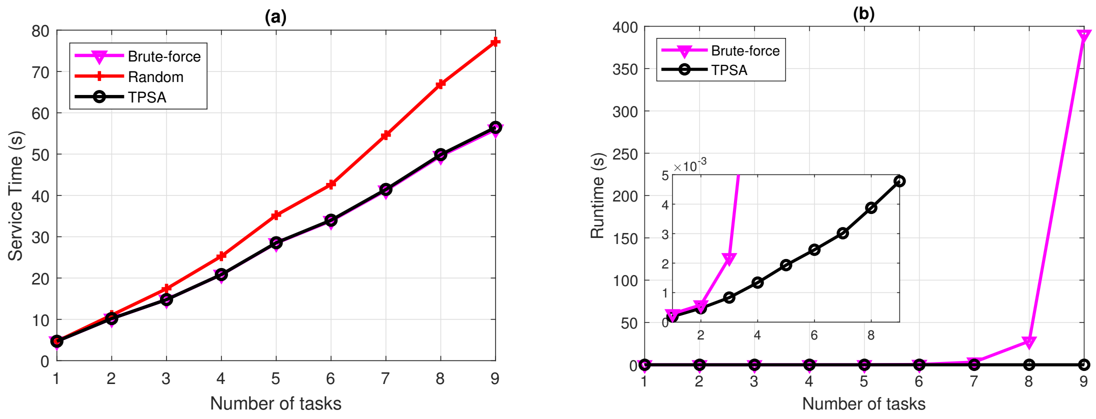


Figure 5.5: (a) Average service delay among the three task partition and scheduling schemes with respect to the number of tasks. (b) Average computation runtime among the three task partition and scheduling schemes with respect to the number of tasks.

Table 5.1: Network parameters

P^V	P^R	σ_r^2, σ_v^2	λ	ϵ
27 dBm	37 dBm	-93 dBm	50	1 s
f	χ	N_e, N_t	δ^{O}	δ^{D}
2800 MHz	1200 C/bits	80, 25	7 dB	7 dB

5.6.1 Task Partition and Scheduling Algorithm

We first evaluate the performance of the proposed TPSA algorithm. In the simulation, we consider that tasks can be offloaded to five edge servers with an identical offloading rate of 6 Mbits/s. The communication rate among the servers is 8 Mbits/s. We set that the computation capability of the servers is 8 GC/s, and the number of computation cycles needed for processing 1 Mbit is 4 GC. The computing data amount of tasks is uniformly distributed in the range of [1,21] Mbits. For each task, the receiver and helper RSUs are randomly selected from the five servers. We compare the proposed TPSA algorithm with *brute-force* and *random* schemes. In the brute-force scheme, we utilize an exhaustive search for finding the optimal scheduling order. In the random scheme, we randomly assign the scheduling order of the tasks. Note that, for both *brute-force* and *random* schemes, we adopt the optimal task partition ratio in workload allocation. The simulation results presented in Figs. 5.5(a) and 5.5(b) are averaged over 200 rounds of Monte Carlo simulations.

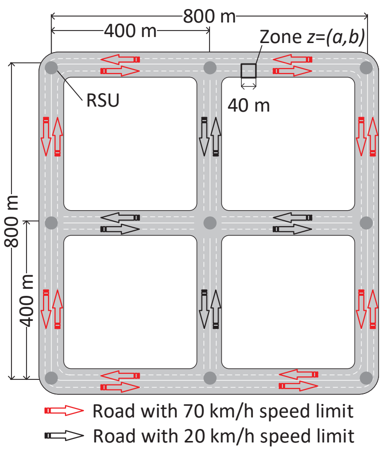


Figure 5.6: The transportation network topology for simulation.

The service delay performance of the proposed algorithm is shown in Fig. 5.5(a). It can be seen that an increase in the task number leads to increasing overall service time, and the increasing rate of the random scheme is the highest among the three schemes. The proposed TPSA algorithm can achieve a performance very close to the brute-force scheme. Moreover, we compare the runtime between the proposed TPSA and the brute-force scheme. As shown in Fig. 5.5(b), as the number of the task increases, the runtime of brute-force scheme increases exponentially, while the proposed TPSA algorithm has imperceptible runtime to compute the scheduling result that is close to the optimal one. In summary, the proposed TPSA algorithm can achieve a near-optimal performance for task partition and scheduling with low computation complexity.

5.6.2 AI-based Collaborative Computing Approach

In this subsection, we evaluate the performance of the proposed AI-based collaborative computing approach. In the simulation, we consider an 800 m $\times$ 800 m transportation system, where the transportation topology is shown in Fig. 5.6. Nine RSUs with edge servers are deployed, as shown in the figure. We generate vehicle traffic by VISSIM [119], where 200 vehicles are traveling in the area. The speed of vehicles depends on the speed limit on the road and the distance to the vehicle ahead. For each vehicle, the computing tasks are generated using a Poisson process, and the input data amount of each task is uniformly distributed in the range of [2,5] Mbits. The length and width of a zone are 40 m and 10 m (2 driving lanes), respectively. Other network parameter settings are presented

Table 5.2: Neural network structure in DDPG

Actor Network		
Layer	Number of neurons	Activation function
CONV1	5×1×2×10, stride 1	relu
POOL1	2×1	none
Data Concatenation and Batch Normalization Layer		
FC1	1400	tanh
FC2	1400	tanh
FC3	5×$A \times B$	tanh
Critic Network		
Layer	Number of neurons	Activation function
CONV1	5×1×2×40, stride 1	relu
POOL1	2×1	none
CONV2	3×1×40×10, stride 1	relu
POOL2	2×1	none
Data Concatenation and Batch Normalization Layer		
FC1	640	relu
FC2	512	relu
FC3	128	none
FC4	1	relu

in Table 5.1. We test the system performance within a duration of 20 seconds.

The neural network structures of the DDPG algorithm are presented in Table 5.2. The initial learning rates of the actor and critic networks are 1e-5 and 1e-4, respectively, and the learning rates are attenuated by 0.991 in every 500 training steps. The experience replay buffer can adopt 8,000 state-action transitions, and in each training step, the number of transition tuples selected for training, *i.e.*, the batch size, is 128. We adopt a soft parameter replacement technique to update the parameters in the target network, where τ is 0.01. We compare the performance of the proposed AI-based collaborative computing approach with three approaches. In the *Greedy* approach, vehicles always offload their tasks to the RSU with the highest SNR, and the received computing tasks will not be collaboratively computed with other RSUs. In the *Greedy+TPSA* approach, a vehicle offload their tasks to the RSU with the highest SNR, and the RSU randomly selects another RSU to compute the task collaboratively. The task partition and scheduling policy follows the TPSA algorithm, and the computing results are delivered by the receiver RSU. In the *Random+TPSA* approach, the receiver, helper, and deliver RSUs are selected randomly,

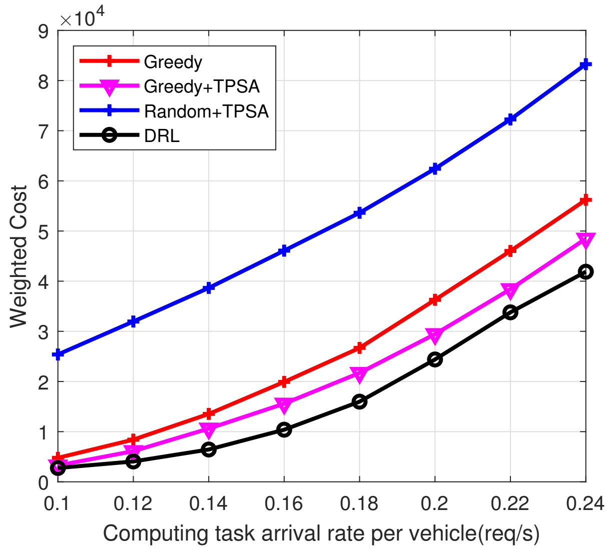


Figure 5.7: Average weighted computing delay cost versus computing task arrive rate per vehicle.

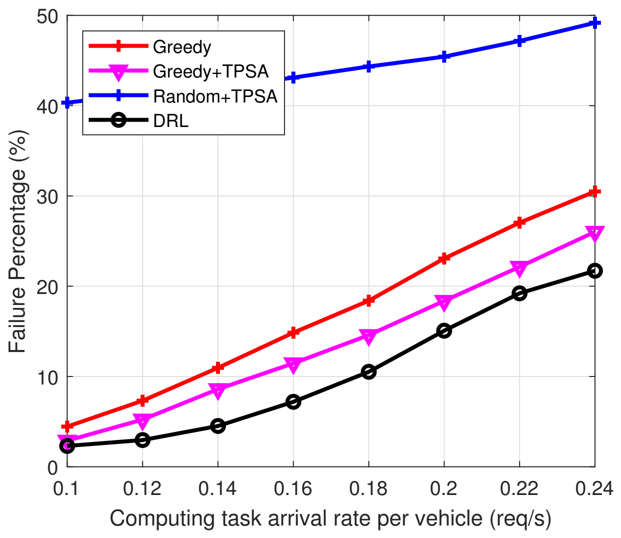


Figure 5.8: Average percentage of service failure versus computing task arrive rate per vehicle.

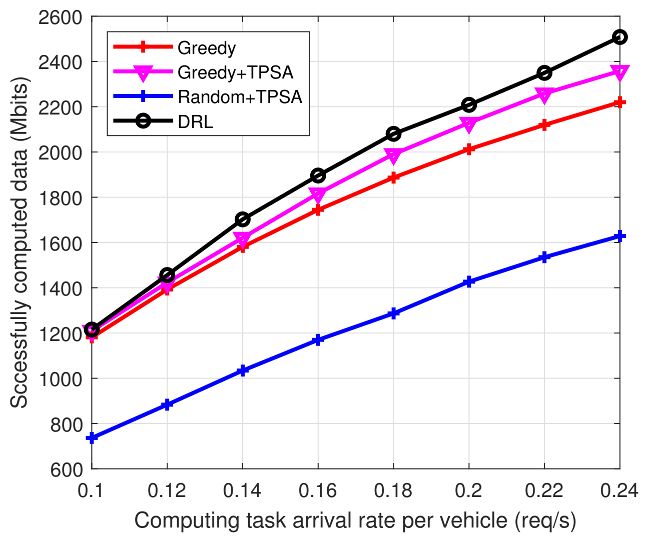


Figure 5.9: Average computing data amount which is successfully computed versus computing task arrive rate per vehicle.

and the TPSA algorithm is applied to determine the task partition ratio and the execution order.

The overall weighted computing cost with respect to task arrival rates is shown in Fig. 5.7. Our proposed approach can achieve the lowest computing cost compared to the other three approaches. The random approach suffers the highest cost compared to others due to the inefficient server selection in the scheme. Moreover, the greedy+TPSA approach achieves a lower cost compared to the greedy approach. The reason is that parallel computing is able to reduce the overall service time, and the proposed TPSA is able to achieve near-optimal task partition and scheduling results. However, the greedy approach selects the servers according to the instantaneous cost of the network rather than the value in the long term. Therefore, the greedy+TPSA approach cannot attain a lower cost compared to the proposed AI-based approach.

As shown in Eq. (5.18), the service cost consists of the service delay and the failure penalty. The results of the service failure percentage is shown in Fig. 5.8. Similar to the service cost, the proposed AI-based approach achieves the lowest failure percentage among the four approaches. Correspondingly, as shown in Fig. 5.9, the proposed approach can successfully process the highest amount of data among the four approaches. On the other hand, the results of the average service delay for 1 Mbits successful computed data are

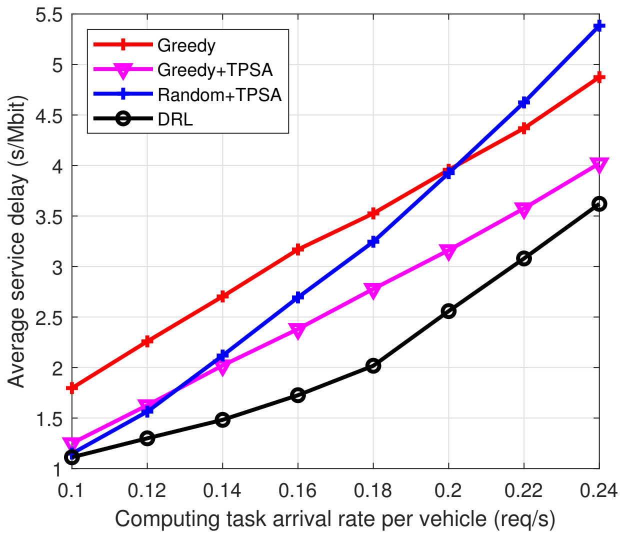


Figure 5.10: Average service delay for 1 Mbits successful computed data versus computing task arrive rate per vehicle.

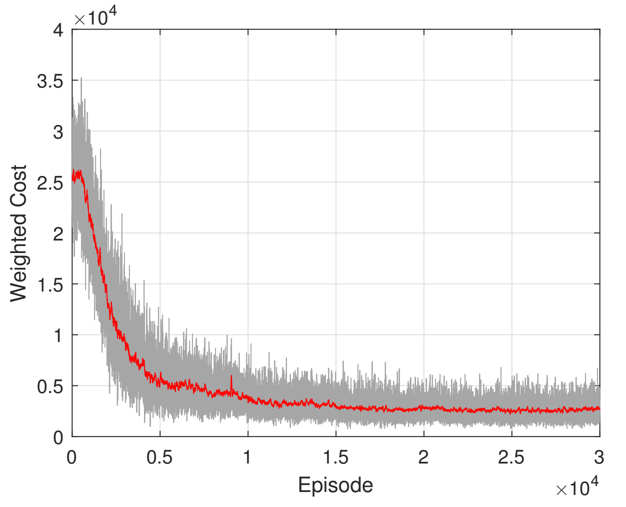


Figure 5.11: Convergence performance of the proposed algorithm, where the task arrival rate is 0.1 request/sec.

shown in Fig. 5.10. Compared to the other three approaches, the proposed scheme reduces the service delay significantly. Furthermore, the delay of the random approach increases exponentially since less amount of data can be successfully computed when the task arrival rate is high.

The convergence performance of the proposed AI-based approach is shown in Fig. 5.11, where the highlighted line represents the moving average from 50 samples around the corresponding point. Note that in our algorithm, we explore multiple times in each training step. It can be seen that our approach converges after 10,000 episodes, or equivalently, after the network being trained by around 3,000 episodes, *i.e.*, 60,000 training steps.

5.7 Summary

We have introduced a novel collaboration computing framework to reduce computing service latency and improve service reliability in MEC-enabled vehicular networks. The proposed framework addresses the challenge of maintaining computing service continuity for vehicle users with high mobility. As a result, our collaborative computing approach is able to support proactive decision making for computation offloading through learning the network dynamics.

Chapter 6

Conclusions and Future Works

In this chapter, we summarize the main results and contributions of this thesis and present our future research directions.

6.1 Main Research Contributions

In this thesis, we have investigated computation offloading and task scheduling for MEC in 5G and beyond. In specific, three resource management strategies have been developed for different computing scenarios in MEC. First, an adaptive computing scheduling scheme has been proposed to support real-time applications in autonomous vehicles. Then, an energy-efficient resource management strategy has been proposed for UAV-assisted IoT networks. Last, a collaborative edge computing strategy has been developed to achieve efficient computing resource management for MUDs with high mobility. The main contributions of this thesis are summarized as follows.

1) An adaptive index-based task scheduling scheme has been proposed for an edge server. The proposed scheme identifies the characteristics of vehicles, i.e., mobility, and schedules the computing resources according to these characteristics to improve the overall computing service performance. We have designed an index-based scheduling scheme to determine the order of computing at an edge server and developed a novel DRL approach to determine the index based on the mobility dynamics of MUDs. The proposed index-based scheduling scheme can adapt to real-time mobility and provide

optimal resource scheduling policy with low computing complexity and communication overhead. The work provides an efficient MEC solution for supporting real-time applications generated by a number of autonomous vehicles in future networks.

2) An optimal multi-resource management strategy has been proposed to maximize the energy efficiency of a UAV-mounted cloudlet in MEC system. In the strategy, we have jointly considered the optimization of UAV trajectory, computation offloading, and task scheduling. To solve the corresponding complex optimization problem, we have developed a distributed optimization solution, which allows UAV and IoT nodes cooperatively solving the complex problem with less local information sharing. The work contributes valuable insights for computing resource management in an energy-constrained network scenario, i.e., IoT in remote areas.

3) A novel collaboration computing framework and the corresponding resource management strategy have been developed to improve computing service reliability and reduce service latency. With the proposed framework, edge servers can efficiently avoid computing service discontinuity caused by frequent radio association changes in the vehicular networks. Based on the framework, we have proposed a AI-based resource management strategy that enables intelligent and proactive decision-making for computation offloading and scheduling in vehicular networks. The proposed work provides a novel mobility management solution for MEC, and the proposed strategy can be applied to implement MEC in a complex network environment, such as urban transportation systems.

In summary, this thesis has investigated resource management for MEC in three aspects: optimizing the QoE for MUDs, improving energy efficiency for edge servers, and collaborating computing resources for MUD mobility adaptivity. All the three of research issues in this thesis have explored the nature of mobility in an MEC-enabled system, and we have proposed resource management strategies to deal with mobility dynamics, including mobility-aware computing resource scheduling, UAV trajectory optimization, and mobility management in vehicular networks. The proposed approaches and theoretical resources can provide valuable guidelines for implementing MEC in 5G and beyond.

6.2 Future Research Directions

Motivated by diversifying mobile applications, wireless networks keep developing, resulting in new computing service requirements to be satisfied by MEC. For future research on MEC, there are some interesting and promising research directions listed as follows:

1) Service-oriented computing resource provisioning: Different services would have differentiated computing service requirements. For example, safety-related applications require MEC to support high-reliability computing, while VR video streaming demands high-speed communication and computing. In 5G, network slicing has been developed to support multiple isolated and independent logical networks for different services, i.e. slices, on the same physical network infrastructure [120, 121]. With network slicing, network resources are reserved for services to guarantee their service level agreements (SLAs). With the development of MEC, the network slicing architecture is expected to be inherited in sixth-generation networks and will be implemented in RAN, i.e., RAN slicing, to support diversified services on the network edge. In edge servers, communication, computing, and caching resources are reserved by multiple slides for service provisioning, and these reserved resources are further scheduled to individual users in real time. However, RAN slicing faces several challenges. For example, effective resource reservation strategies for edge servers should adapt to service demands, thereby avoiding resource over-provisioning or SLA violation [122], while high network traffic dynamics in RAN make resource reservation challenging. Moreover, the configurable communication topology in RAN provides a higher levels of flexibility in network slicing, while it also increases the difficulty of finding optimal resource management solutions for slices. Therefore, how to enabling effective service-oriented resource provisioning by MEC calls for further investigation.

2) MEC for AI services: AI can play an essential role in network management; meanwhile, AI can also be a new type of service to be supported by networks. Examples of AI services in future networks include language processing, video surveillance, and autonomous driving [123, 124]. Since AI services will need to gather or generate a vast amount of data, AI service provisioning on network edge, i.e., edge intelligence, has attracted extensive interest as it moves AI closer to MUDs and alleviates data traffic load in the core network. Empowered by distributed learning techniques and MEC technologies, the communication, computing, and storage resources at each edge server can be leveraged to perform data processing and inference for AI services [125]. In contrast with conventional compute-intensive services, the service performance of AI services, such as accuracy and training convergence rate, strongly relates to data generated by MUDs, thus resulting in the revolution in computing service provisioning by MEC. Specifically, data can be considered as a novel resource in the application layer [26]. Edge servers should allocate communication, computing, and caching resources to manage data and execute AI services accordingly, which requires a novel resource management strategy for MEC to support AI services.

3) Intelligent network control: In addition to supporting mobile applications, MEC can be an enabler technology to achieve hybrid and hierarchical network control architecture. Since edge servers are close to MUDs, local network controllers can interface with edge servers to collect real-time data in high granularity, such as locations and transmission buffer states of each MUD. The local controller can compute network control policies for individual MUDs and aggregate MUD data for further large-scale network control in the global controller at the cloud server [120,126]. By coordinating the controllers in edge and cloud servers, intelligent and scalable network management can be achieved in future networks, while how to effectively take advantage of computation capability on the network edge to maximize entire network performance remains an open issue for future networks.

4) Synergy between data-driven and model-driven methods in resource management: AI techniques facilitate data-driven methods for resource management. Via data-driven methods, network characteristics can be abstracted by exploiting the features of network data. Thus, data-driven methods can be used to solve complex resource management problems without having to obtain a closed-form mathematical model. However, the performance of data-driven methods is questionable whenever the characteristics of the network change and the network environment is not stationary. Furthermore, the explainability of data-driven methods remains an open question in academia, especially for deep neural networks. Although learning models are fully trained, data-driven methods cannot always provide accurate inference results. One potential solution to address the limitations of data-driven methods is to integrate model-driven methods with data-driven methods in resource management. Model-driven methods provide closed-form solutions that are generalized to diverse network environments and are explainable, which can be combined with data-driven methods to improve the adaptivity on network environment changing and the accuracy of inference results. However, how to implement the synergy between data-driven and model-driven methods in resource management is still an open issue.

References

[1] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on multi-access edge computing for internet of things realization," *IEEE Commun. Surveys Tuts.*, vol. 20, pp. 2961–2991, June 2018.

[2] Y. Liu, M. Peng, G. Shou, Y. Chen, and S. Chen, "Toward edge intelligence: Multi-access edge computing for 5G and internet of things," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6722–6747, June 2020.

[3] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. Comput. Sci., Univ. of California, Berkeley, Rep. UCB/EECS*, vol. 28, no. 13, pp. 2009, 2009.

[4] S. Appleby L. Han and K. Smith, "Problem statement: Transport support for augmented and virtual reality applications," Tech. Rep., Mar. 2020, Working Draft, IETF Secretariat, Available on: `https://www.etsi.org/deliver/etsi_gs/MEC-IEG/001_099/006/01.01.01_60/gs_MEC-IEG006v010101p.pdf`.

[5] ETSI, GS MEC-IEG, "006: Mobile edge computing market acceleration MEC metrics best practice and guidelines v1. 1.1. 2017," Available on: `https://www.etsi.org/deliver/etsi_gs/MEC-IEG/001_099/006/01.01.01_60/gs_MEC-IEG006v010101p.pdf`.

[6] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5G," White paper, ETSI, 2014.

[7] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–61, Apr. 2017.

[8] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 38–43, Mar. 2017.

[9] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, Fourthquarter 2017.

[10] "Edge computing internet of things solution," `https://e.huawei.com/en/solutions/business-needs/enterprise-network/agile-iot`, Accessed: 2021-05-25.

[11] "Azure iot edge," `https://azure.microsoft.com/en-us/services/iot-edge/`, Accessed: 2021-05-25.

[12] 3GPP, "Study on enhancement of support for Edge Computing in the 5G Core network (5GC)," Tech. rep., Dec. 2020, Rel. 17, 3GPP TR 23.748.

[13] 3GPP, "System architecture for the 5G System (5GS)," Tech. specification, Mar. 2021, Rel. 17, 3GPP TS 23.501 V17.0.0.

[14] Z. Zhang, Y. Xiao, Z. Ma, M. Xiao, Z. Ding, X. Lei, G. K. Karagiannidis, and P. Fan, "6G wireless networks: Vision, requirements, architecture, and key technologies," *IEEE Veh. Technol. Mag.*, vol. 14, no. 3, pp. 28–41, July 2019.

[15] B. Zong, C. Fan, X. Wang, X. Duan, B. Wang, and J. Wang, "6G technologies: Key drivers, core requirements, system architectures, and enabling technologies," *IEEE Veh. Technol. Mag.*, vol. 14, no. 3, pp. 18–27, July 2019.

[16] L. Kong, M. Khan, F. Wu, G. Chen, and P. Zeng, "Millimeter-wave wireless communications for IoT-cloud supported autonomous vehicles: Overview, design, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 1, pp. 62–68, Jan. 2017.

[17] "Advancing ADAS development with massive data and machine learning," `https://www.quantum.com/en/solutions/autonomous/`, Accessed: 2021-05-25.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Prov. 2016 IEEE Conf. Comp. Vision Pattern Recognit. (CVPR)*, 2016, pp. 779–788.

[19] N. Mohamed, J. Al-Jaroodi, I. Jawhar, H. Noura, and S. Mahmoud, "UAVFog: A UAV-based fog computing for Internet of Things," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Computed, Scalable Comput. Commun., Cloud Big Data Comput., Internet People Smart City Innov*, Aug. 2017, pp. 1–8.

[20] W. Z. Khan, M. Y. Aalsalem, M. K. Khan, M. S. Hossain, and M. Atiquzzaman, "A reliable Internet of Things based architecture for oil and gas industry," in *Proc. 19th Int. Conf. Adv. Commun. Technol.*, Feb. 2017, pp. 705–710.

[21] M. C. Domingo, "An overview of the internet of underwater things," *J. Netw. and Comput. Appl.*, vol. 35, no. 6, pp. 1879–1890, 2012.

[22] B. Płaczek, "Selective data collection in vehicular networks for traffic control applications," *Transp. Res. Part C: Emerg. Technol.*, vol. 23, pp. 14–28, Aug. 2012.

[23] H. He, H. Shan, A. Huang, and L. Sun, "Resource allocation for video streaming in heterogeneous cognitive vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 65, no. 10, pp. 7917–7930, Mar. 2016.

[24] P. Zhou, W. Zhang, T. Braud, P. Hui, and J. Kangasharju, "ARVE: Augmented reality applications in vehicle to edge networks," in *Proc. Workshop Mobile Edge Commun.*, 2018, p. 25–30.

[25] J. Du, F. R. Yu, G. Lu, J. Wang, J. Jiang, and X. Chu, "MEC-assisted immersive VR video streaming over terahertz wireless networks: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9517–9529, June 2020.

[26] M. Li, J. Gao, C. Zhou, W. Zhuang, and X. Shen, "Slicing-based AI service provisioning on network edge," *arXiv preprint arXiv:2105.07052*, 2021.

[27] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[28] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[29] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Mar. 2017.

[30] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.

[31] J. Wang, D. Feng, S. Zhang, A. Liu, and X. Xia, "Joint computation offloading and resource allocation for MEC-enabled IoT systems with imperfect CSI," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3462–3475, Sept. 2021.

[32] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, Mar. 2016.

[33] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6774–6785, Aug. 2019.

[34] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 939–951, Dec. 2021.

[35] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, Dec. 2018.

[36] M. Qin, N. Cheng, Z. Jing, T. Yang, W. Xu, Q. Yang, and R. R. Rao, "Service-oriented energy-latency tradeoff for IoT task partial offloading in MEC-enhanced multi-RAT networks," *IEEE Internet Things J.*, vol. 8, no. 3, pp. 1896–1907, Aug. 2021.

[37] L. Lei, Z. Zhong, K. Zheng, J. Chen, and H. Meng, "Challenges on wireless heterogeneous networks for mobile cloud computing," *IEEE Wireless Commun.*, vol. 20, no. 3, pp. 34–44, July 2013.

[38] N. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen, "Space/aerial-assisted computing offloading for IoT applications: A learning-based approach," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1117–1129, May 2019.

[39] C. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4132–4150, June 2019.

[40] T. G. Rodrigues, K. Suto, H. Nishiyama, N. Kato, and K. Temma, "Cloudlets activation scheme for scalable mobile edge computing with transmission power control and virtual machine migration," *IEEE Trans. Comput.*, vol. 67, no. 9, pp. 1287–1300, Sept. 2018.

[41] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through VM migration and transmission power control," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 810–819, May 2017.

[42] U. Saleem, Y. Liu, S. Jangsher, Y. Li, and T. Jiang, "Mobility-aware joint task scheduling and resource allocation for cooperative mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 360–374, Sept. 2021.

[43] Y. Sun, X. Guo, J. Song, S. Zhou, Z. Jiang, X. Liu, and Z. Niu, "Adaptive learning-based task offloading for vehicular edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3061–3074, Apr. 2019.

[44] H. Peng and X. Shen, "Deep reinforcement learning based resource management for multi-access edge computing in vehicular networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 4, pp. 2416–2428, Mar. 2020.

[45] Y. Liu, J. Liu, A. Argyriou, and S. Ci, "MEC-assisted panoramic VR video streaming over millimeter wave mobile networks," *IEEE Trans. Multimedia*, vol. 21, no. 5, pp. 1302–1316, Oct. 2019.

[46] W. Wu, N. Chen, C. Zhou, M. Li, X. Shen, W. Zhuang, and X. Li, "Dynamic RAN slicing for service-oriented vehicular networks via constrained learning," *IEEE J. Sel. Areas Commun.*, pp. 1–14, 2020, to appear.

[47] Z. Hu, F. Zeng, Z. Xiao, B. Fu, H. Jiang, and H. Chen, "Computation efficiency maximization and QoE-provisioning in UAV-enabled MEC communication systems," *IEEE Trans. Netw. Sci. Eng.*, pp. 1–15, 2021.

[48] X. Hu, K. Wong, and Y. Zhang, "Wireless-powered edge computing with cooperative UAV: Task, time scheduling and trajectory design," *IEEE Trans. Wireless Commun.*, vol. 19, no. 12, pp. 8083–8098, Sept. 2020.

[49] F. Zhao, Y. Chen, Y. Zhang, Z. Liu, and X. Chen, "Dynamic offloading and resource scheduling for mobile edge computing with energy harvesting devices," *IEEE Trans. Netw. Service Manag.*, pp. 1–13, 2021, to appear.

[50] G. Zhang, W. Zhang, Y. Cao, D. Li, and L. Wang, "Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4642–4655, June 2018.

[51] Y. Chen, Y. Zhang, Y. Wu, L. Qi, X. Chen, and X. Shen, "Joint task scheduling and energy management for heterogeneous mobile edge computing with hybrid energy supply," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8419–8429, May 2020.

[52] J. Cao, L. Yang, and J. Cao, "Revisiting computation partitioning in future 5G-based edge computing environments," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2427–2438, Apr. 2019.

[53] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 668–682, Mar. 2019.

[54] J. Feng, Z. Liu, C. Wu, and Y. Ji, "AVE: Autonomous vehicular edge computing framework with ACO-based scheduling," *IEEE Trans. Veh. Technol.*, vol. 66, no. 12, pp. 10660–10675, June 2017.

[55] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4854–4866, Oct. 2019.

[56] R. Kaewpuang, D. Niyato, P. Wang, and E. Hossain, "A framework for cooperative resource management in mobile cloud computing," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 2685–2700, Dec. 2013.

[57] R. Yu, J. Ding, S. Maharjan, S. Gjessing, Y. Zhang, and D. H. K. Tsang, "Decentralized and optimal resource cooperation in geo-distributed mobile cloud computing," *IEEE Trans. Emerg. Topics Comput.*, vol. 6, no. 1, pp. 72–84, Jan. 2018.

[58] P. Liu, J. Li, and Z. Sun, "Matching-based task offloading for vehicular edge computing," *IEEE Access*, vol. 7, pp. 27628–27640, Feb. 2019.

[59] Y. Kao, B. Krishnamachari, M. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Trans. Mobile Comput.*, vol. 16, no. 11, pp. 3056–3069, Nov. 2017.

[60] J. L. J. Laredo, F. Guinand, D. Olivier, and P. Bouvry, "Load balancing at the edge of chaos: How self-organized criticality can lead to energy-efficient computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 517–529, Feb. 2017.

[61] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7635–7647, Oct. 2019.

[62] J. Zhou, D. Tian, Y. Wang, Z. Sheng, X. Duan, and V. C. M. Leung, "Reliability-optimal cooperative communication and computing in connected vehicle systems," *IEEE Trans. Mobile Comput.*, vol. 19, no. 5, pp. 1216–1232, Mar. 2020.

[63] T. Taleb and A. Ksentini, "An analytical model for follow me cloud," in *2013 IEEE Global Commun. Conf (GLOBECOM)*, Dec. 2013, pp. 1291–1296.

[64] T. Taleb and A. Ksentini, "Follow me cloud: interworking federated clouds and distributed mobile networks," *IEEE Netw.*, vol. 27, no. 5, pp. 12–19, Sep. 2013.

[65] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Perform. Eval.*, vol. 91, pp. 205 – 228, 2015.

[66] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on Markov decision process," *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1272–1288, June 2019.

[67] H. Liang, X. Zhang, J. Zhang, Q. Li, S. Zhou, and L. Zhao, "A novel adaptive resource allocation model based on SMDP and reinforcement learning algorithm in vehicular cloud system," *IEEE Trans. Veh. Technol.*, vol. 68, no. 10, pp. 10018–10029, Oct. 2019.

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

[69] L. Li, Y. Li, and R. Hou, "A novel mobile edge computing-based architecture for future cellular vehicular networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2017, pp. 1–6.

[70] W. Zhuang, Q. Ye, F. Lyu, N. Cheng, and J. Ren, "SDN/NFV-empowered future IoV with enhanced communication, computing, and caching," *Proc. IEEE*, vol. 108, no. 2, pp. 274–291, Feb. 2020.

[71] S. Zhang, J. Chen, F. Lyu, N. Cheng, W. Shi, and X. Shen, "Vehicular communication networks in the automated driving era," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 26–32, Sept. 2018.

[72] M. Li, J. Gao, L. Zhao, and X. Shen, "Adaptive computing scheduling for edge-assisted autonomous driving," *IEEE Trans. Veh. Technol.*, pp. 1–14, 2021, to appear.

[73] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Comput. Surv.*, vol. 52, no. 6, Oct. 2019.

[74] Queensland government's transport department, "Stopping distances on wet and dry roads," Available on: `https://www.qld.gov.au/transport/safety/road-safety/driving-safely/stopping-distances/graph`.

[75] S. Maheshwari, D. Raychaudhuri, I. Seskar, and F. Bronzino, "Scalability and performance evaluation of edge cloud systems for latency constrained applications," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, 2018, pp. 286–299.

[76] 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 Veh. Technol. Mag.*, vol. 12, no. 2, pp. 36–44, June 2017.

[77] L. T. Tan and R. Q. Hu, "Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 10190–10203, Nov. 2018.

[78] M. Li, J. Gao, L. Zhao, and X. Shen, "Deep reinforcement learning for collaborative edge computing in vehicular networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 6, no. 4, pp. 1122–1135, Dec. 2020.

[79] X. Wang, Z. Ning, and L. Wang, "Offloading in Internet of Vehicles: A fog-enabled real-time traffic management system," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4568–4578, Oct. 2018.

[80] I. Kadota, A. Sinha, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, "Scheduling policies for minimizing age of information in broadcast wireless networks," *IEEE/ACM Trans. Netw*, vol. 26, no. 6, pp. 2637–2650, Dec. 2018.

[81] K. Liu and Q. Zhao, "Indexability of restless bandit problems and optimality of whittle index for dynamic multichannel access," *IEEE Trans. Inf. Theory*, vol. 56, no. 11, pp. 5547–5567, Nov. 2010.

[82] P. Whittle, "Restless bandits: activity allocation in a changing world," *J. Appl. Probability*, vol. 25, no. A, pp. 287–298, 1988.

[83] D. P. Bertsekas, *Dynamic programming and optimal control*, Athena scientific Belmont, MA, 1995.

[84] Y. Qian, C. Zhang, B. Krishnamachari, and M. Tambe, "Restless poachers: Handling exploration-exploitation tradeoffs in security domains," in *Proc. Int. Conf. Auton. Agents & Multiagent Syst.*, 2016, pp. 123–131.

[85] J. Gao, L. Zhao, and X. Shen, "Service offloading in terrestrial-satellite systems: user preference and network utility," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–6.

[86] Y. Wu, B. Shi, L. P. Qian, F. Hou, J. Cai, and X. Shen, "Energy-efficient multi-task multi-access computation offloading via noma transmission for IoTs," *IEEE Trans. Ind. Informat.*, vol. 16, no. 7, pp. 4811–4822, Oct. 2020.

[87] S. Fu, L. Zhao, X. Ling, and H. Zhang, "Maximizing the system energy efficiency in the blockchain based internet of things," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–6.

[88] T. Samad, J. S. Bay, and D. Godbole, "Network-centric systems for military operations in urban terrain: The role of UAVs," *Proc. IEEE*, vol. 95, no. 1, pp. 92–107, Jan. 2007.

[89] S. Fu, L. Zhao, Z. Su, and X. Jian, "UAV based relay for wireless sensor networks in 5G systems," *Sensors*, vol. 18, 2018.

[90] Y. Zhou, N. Cheng, N. Lu, and X. Shen, "Multi-UAV-aided networks: Aerial-ground cooperative vehicular networking architecture," *IEEE Veh. Technol. Mag.*, vol. 10, no. 4, pp. 36–44, Dec. 2015.

[91] W. Shi, J. Li, N. Cheng, F. Lyu, S. Zhang, H. Zhou, and X. Shen, "Multi-drone 3-D trajectory planning and scheduling in drone-assisted radio access networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 8145–8158, Aug. 2019.

[92] N. Cheng, W. Xu, W. Shi, Y. Zhou, N. Lu, H. Zhou, and X. Shen, "Air-ground integrated mobile edge networks: Architecture, challenges, and opportunities," *IEEE Commun. Mag.*, vol. 56, no. 8, pp. 26–32, Aug. 2018.

[93] Q. Hu, Y. Cai, G. Yu, Z. Qin, M. Zhao, and G. Y. Li, "Joint offloading and trajectory design for UAV-enabled mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1879–1892, Apr. 2019.

[94] M. Li, N. Cheng, J. Gao, Y. Wang, L. Zhao, and X. Shen, "Energy-efficient UAV-assisted mobile edge computing: Resource allocation and trajectory optimization," *IEEE Trans. Veh. Technol.*, vol. 69, no. 3, pp. 3424–3438, Mar. 2020.

[95] Y. Zeng and R. Zhang, "Energy-efficient UAV communication with trajectory optimization," *IEEE Trans. Wireless Commun.*, vol. 16, no. 6, pp. 3747–3760, June 2017.

[96] S. Jeong, O. Simeone, and J. Kang, "Mobile edge computing via a UAV-mounted cloudlet: Optimization of bit allocation and path planning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 3, pp. 2049–2063, Mar. 2018.

[97] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the internet of things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan. 2018.

[98] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 149–163, Oct. 2003.

[99] T. Lipp and S. Boyd, "Variations and extension of the convex–concave procedure," *Optimization and Eng.*, vol. 17, no. 2, pp. 263–287, 2016.

[100] W. Dinkelbach, "On nonlinear fractional programming," *Manage. Sci.*, vol. 13, no. 7, pp. 492–498, 1967.

[101] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011.

[102] R. Nishihara, L. Lessard, B. Recht, A. Packard, and M. Jordan, "A general analysis of the convergence of ADMM," in *Int.Conf. Machine Learning*, 2015, pp. 343–352.

[103] W. Deng, M. Lai, Z. Peng, and W. Yin, "Parallel multi-block admm with O(1/k) convergence," *J. of Scientific Comput.*, vol. 71, no. 2, pp. 712–736, 2017.

[104] K. Wang, A. M. So, T. Chang, W. Ma, and C. Chi, "Outage constrained robust transmit optimization for multiuser MISO downlinks: Tractable approximations by conic optimization," *IEEE Trans. Signal Process.*, vol. 62, no. 21, pp. 5690–5705, Nov. 2014.

[105] A. Hakiri, P. Berthou, A. Gokhale, and S. Abdellatif, "Publish/subscribe-enabled software defined networking for efficient and scalable IoT communications," *IEEE Commun. Mag.*, vol. 53, no. 9, pp. 48–54, Sep. 2015.

[106] R. Cao, A. Cuevas, and W. G. Manteiga, "A comparative study of several smoothing methods in density estimation," *Comput. Statist. & Data Anal.*, vol. 17, no. 2, pp. 153–176, 1994.

[107] P. Van Laarhoven and E. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*, pp. 7–15. Springer, 1987.

[108] N. Cheng, N. Zhang, N. Lu, X. Shen, J. W. Mark, and F. Liu, "Opportunistic spectrum access for CR-VANETs: A game-theoretic approach," *IEEE Trans. Veh. Technol.*, vol. 63, no. 1, pp. 237–251, July 2014.

[109] J. Gao, M. Li, L. Zhao, and X. Shen, "Contention intensity based distributed coordination for V2V safety message broadcast," *IEEE Trans. Veh. Technol.*, vol. 67, no. 12, pp. 12288–12301, Dec. 2018.

[110] "Self driving safety report," Tech. Rep., Nvidia, Santa Clara, CA, USA, 2018.

[111] N. Zhang, S. Zhang, P. Yang, O. Alhussein, W. Zhuang, and X. Shen, "Software defined space-air-ground integrated vehicular networks: Challenges and solutions," *IEEE Commun. Mag.*, vol. 55, no. 7, pp. 101–109, July 2017.

[112] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 3, pp. 361–373, Sep. 2017.

[113] J. Zhang and K. B. Letaief, "Mobile edge intelligence and computing for the Internet of Vehicles," *Proc. IEEE*, vol. 108, no. 2, pp. 246–261, 2020.

[114] F. Lyu, H. Zhu, H. Zhou, L. Qian, W. Xu, M. Li, and X. Shen, "MoMAC: Mobility-aware and collision-avoidance MAC for safety applications in VANETs," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 10590–10602, Nov. 2018.

[115] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1584–1607, Aug. 2019.

[116] J. Chen, W. Xu, N. Cheng, H. Wu, S. Zhang, and X. Shen, "Reinforcement learning policy for adaptive edge caching in heterogeneous vehicular network," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.

[117] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[118] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. 2012.

[119] "VISSIM," `http://vision-traffic.ptvgroup.com/`.

[120] X. Shen, J. Gao, W. Wu, K. Lyu, M. Li, W. Zhuang, X. Li, and J. Rao, "AI-assisted network-slicing based next-generation wireless networks," *IEEE Open J. Veh. Technol.*, vol. 1, pp. 45–66, Jan. 2020.

[121] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2429–2453, Mar. 2018.

[122] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Pérez, "DeepCog: Optimizing resource provisioning in network slicing with AI-based capacity forecasting," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 361–376, Nov. 2020.

[123] W. Saad, M. Bennis, and M. Chen, "A vision of 6G wireless systems: Applications, trends, technologies, and open research problems," *IEEE Netw.*, vol. 34, no. 3, pp. 134–142, Oct. 2020.

[124] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G networks: Use cases and technologies," *IEEE Commun. Mag.*, vol. 58, no. 3, pp. 55–61, Mar. 2020.

[125] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, June 2019.

[126] X. You et al., "Towards 6G wireless communication networks: Vision, enabling technologies, and new paradigm shifts," *Sci. China Inf. Sci.*, vol. 64, no. 1, pp. 1–74, Nov. 2021.

[127] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, Mar. 2017.

[128] "Edge computing – introduction in phases," https://www.ericsson.com/en/edge-computing, Accessed: 2021-05-25.

Appendices

Appendix A

Proof of Lemma 2

Firstly, to deal with the non-convex function on the numerator, *i.e.*, $R_{i,k}(\delta_{i,k}, \mathbf{Q}_k)$, we introduce the auxiliary variable $\check{R}_{i,k}$ to indicate the lower bound of the data rate for user i in slot k. Moreover, we introduce two auxiliary variables: the term $\xi_{i,k}$, where $\xi_{i,k} \leq \delta_{i,k}P/l_{i,k}$, and the term $l_{i,k}$, where $l_{i,k} \geq N_0/h_{i,k}$. Thus, the following relation can be established

$$\check{R}_{i,k} \leq \frac{B\Delta}{N}\log(1+\xi_{i,k}) \leq R_{i,k}(\delta_{i,k}, \mathbf{Q}_k), \tag{A.1}$$

where $\check{R}_{i,k}$ is the epigraph form of $R_{i,k}(\delta_{i,k}, \mathbf{Q}_k)$. When (4.15a) is maximized, *i.e.*, the numerator $\check{R}^*_{i,k}$ is maximized, we have $l^*_{i,k} = 1/g^*_{i,k}$, $\xi^*_{i,k} = \delta^*_{i,k}P/l^*_{i,k}$, and $\check{R}^*_{i,k} = R_{i,k}(\delta^*_{i,k}, \mathbf{Q}^*_k)$.

Furthermore, to deal with the non-linear function on the denominator, *i.e.*, $E^F_k(\mathbf{Q})$, we introduce an auxiliary variable $\hat{E}^F_k$ to indicate the upper bound of the UAV propulsion energy in slot k. For the non-linear part of the function, we introduce two auxiliary variables: the term ω_k, where $\omega_k^2 \leq \|\mathbf{v}_k(\mathbf{Q})\|_2{}^2$, and the term $A_{i,k}$, where $A_{i,k} \geq (1/\omega_k)(1+\|a_k(\mathbf{Q})\|_2^2/g^2)$. Thus, we have

$$\begin{aligned}\hat{E}^F_k &\geq \gamma_1\|\mathbf{v}_k(\mathbf{Q})\|_2^3 + \gamma_2 A_k \\ &\geq \gamma_1\|\mathbf{v}_k(\mathbf{Q})\|_2^3 + \gamma_2\frac{1}{\omega_k}(1+\frac{\|a_k(\mathbf{Q})\|_2^2}{g^2}) \geq E^F_k(\mathbf{Q}).\end{aligned} \tag{A.2}$$

Similarly, when (4.15a) is maximized, *i.e.*, the denominator $E_k^F(\mathbf{Q})$ is minimized, $\hat{E}_k^{F*} = E_k^F(\mathbf{Q}^*)$. Therefore, problem (4.15) is equivalent to problem (4.14), and $\eta^* = \check{\eta}^*$.

Appendix B

Proof of Lemma 3

Constraint (4.15c) can be transformed into the following equivalent form:

$$(\xi_{i,k} + l_{i,k})^2 - (\xi_{i,k} - l_{i,k})^2 \leq 4\delta_{i,k} P, \tag{B.1}$$

which is difference of convex functions [99]. Then, we approximate the second part of the equation by the Taylor expansion:

$$(\xi_{i,k} - l_{i,k})^2 \approx (\xi_{i,k}^t - l_{i,k}^t)^2 + \begin{bmatrix} 2\xi_{i,k}^t - 2l_{i,k}^t \\ 2l_{i,k}^t - 2\xi_{i,k}^t \end{bmatrix}^T \begin{bmatrix} \xi_{i,k} - \xi_{i,k}^t \\ l_{i,k} - l_{i,k}^t \end{bmatrix} \tag{B.2}$$

Then, we further reformulate the approximated equation as the constraints shown in (4.17) with a cone expression. Moreover, constraint (4.15g) is approximated by constraint (4.19) in a similar way. Constraints (4.15f) and (4.16) are approximated by (4.18) and (4.20) respectively by first order Taylor expansion to obtain the lower bound on the squared norm and the subtracted term, respectively.

All the approximated constraints (4.17)-(4.20) are stricter than their original counterparts, guaranteeing that the solution of the approximated problem is strictly smaller than the original optimum. For example, consider the optimal $\xi_{i,k}$ and $l_{i,k}$ obtained by solving the approximated problem, which is denoted by $\xi_{i,k}^a$ and $l_{i,k}^a$. These two variables are bounded by constraint (4.17) in the approximated problem. Comparing (4.17) with the original constraint (4.15c) and considering the property of the Taylor expansion, we have

$\xi^a_{i,k} l^a_{i,k} + \Delta_{approx} \leq \delta_{i,k} P$, where $\Delta_{approx} \geq 0$. Thus,

$$\frac{B\Delta}{N} \log(1 + \xi^a_{i,k}) \leq \frac{B\Delta}{N} \log(1 + \frac{\delta_{i,k} P}{l^a_{i,k}}) \tag{B.3}$$

Moreover, due to $l^a_{i,k} \geq 1/g_{i,k}$, we have

$$\frac{B\Delta}{N} \log(1 + \frac{\delta_{i,k} P}{l^a_{i,k}}) \leq R_{i,k}(\delta_{i,k}, \mathbf{Q}_k). \tag{B.4}$$

Therefore, the approximation on constraint (4.15c) will leads to $\check{R}^*_{i,k} < R_{i,k}(\delta_{i,k}, \mathbf{Q}_k)$. Other approximated constraints can be proven similarly to show that the proposed approximated objective function provides the global lower bound for original objective function (4.14). Moreover, due to the gradient consistency in the first order estimation, the SCA algorithm will be stopped when a local optimizer is found.

Appendix C

Proof of Lemma 7

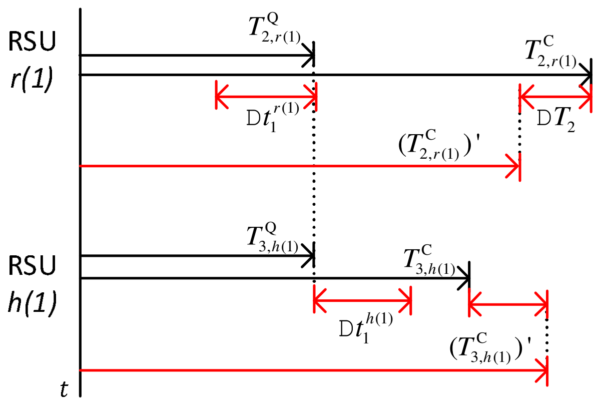
An illustration of task partition is shown in Fig. C.1. Consider that the workload of all tasks are divided and shared among RSUs following the results in Lemma 6. We focus on a single task which is numbered as task 1 as shown in the figure. As indicated in the second and the third assumptions in Lemma 7, the computation load of task 1 is shared between RSU $r(1)$ and $h(1)$. Tasks 2 and 3 are scheduled after task 1 in RSU $r(1)$ and $h(1)$, respectively. In addition, $T^{Q}_{2,h(2)} \geq T^{T}_{2,r(2)} + T^{R}_{r(2),h(2)}$. We then prove that, under the assumption in Lemma 7, the overall service time will be increased if the partition ratio of task 1 does not follow the policy presented in Lemma 6.

Consider that, for task 1, the workload assigned to RSU $r(1)$ is decreased by Δx. Correspondingly, the computing time of task 1 in server $r(1)$ is reduced by $\Delta t_1^{r(1)} = \Delta x / C_{r(1)}$, while the computing time of task 1 in server $h(1)$ is increased by $\Delta t_1^{h(1)} = \Delta x / C_{h(1)}$. Thus, the service time of task 1 is increased by $\Delta T_1 = \Delta x / C_{h(1)}$. Denote the new partition ratio of task 2, after task partition ratio x_1 is decreased by Δx, as $\hat{x}_2$. We then list following cases to analyze the time deduction from the tasks queued after the task 1:

- *Case 1:* Task 2 regards RSU $r(1)$ as the receiver RSU, *i.e.*, $r(1) = r(2)$, and $\hat{x}_2 < 1$. According to Eq. (5.12) and Lemma 6, the optimal service time of task 2 is

$$T_2^{\text{service}} = \max\{T_2^{\text{T}}, T^{\text{Q}}_{2,r(1)}\} + \frac{(T^{\text{Q}}_{2,h(2)} - \max\{T_2^{\text{T}}, T^{\text{Q}}_{2,r(1)}\})C_{h(2)} + \chi W_2}{C_{r(1)} + C_{h(2)}}. \tag{C.1}$$

After task partition ratio x_1 is decreased by Δx, task 2 can be processed by RSU



RSU
r(1)
$T^{\mathrm{Q}}_{2,r(1)}$
$T^{\mathrm{C}}_{2,r(1)}$
$\Delta t_1^{r(1)}$
ΔT_2
$(T^{\mathrm{C}}_{2,r(1)})'$
RSU
h(1)
$T^{\mathrm{Q}}_{3,h(1)}$
$T^{\mathrm{C}}_{3,h(1)}$
$\Delta t_1^{h(1)}$
$(T^{\mathrm{C}}_{3,h(1)})'$
t

Figure C.1: An illustration of task partition.

$r(1)$ in advance by $\Delta t_1^{r(1)}$. The new optimal service time of task 2 is

$$(T_2^{\text{service}})' = \max\{T_2^{\mathrm{T}}, T^{\mathrm{Q}}_{2,r(1)} - \Delta t_1^{r(1)}\} + \frac{(T^{\mathrm{Q}}_{2,h(2)} - \max\{T_2^{\mathrm{T}}, T^{\mathrm{Q}}_{2,r(1)} - \Delta t_1^{r(1)}\})C_{h(2)} + \chi W_2}{C_{r(1)} + C_{h(2)}}. \tag{C.2}$$

The service time deduction on task 2 can be obtained by subtracting Eq. (C.1) by Eq. (C.2). We found the reduced service time $\Delta T_2 \le \Delta t_1^{r(1)} C_{r(1)}/(C_{r(1)} + C_{h(2)})$, where equality can be reached when $T_2^{\mathrm{T}} \le T^{\mathrm{Q}}_{2,r(1)} - \Delta t_1^{r(1)}$.

- *Case 2:* Task 2 regards RSU $r(1)$ as the receiver RSU, *i.e.*, $r(1) = r(2)$, and $\hat{x}_2 = 1$. In this case, the new optimal service time of task 2 is

$$(T_2^{\text{service}})' = \max\{T_2^{\mathrm{T}}, T^{\mathrm{Q}}_{2,r(1)} - \Delta t_1^{r(1)}\} + \frac{\chi W_2}{C_{r(1)}}. \tag{C.3}$$

Via subtracting Eq. (C.1) by Eq. (C.3), we have

$$\begin{aligned}\Delta T_2 &\le \Delta t_1^{r(1)} - \frac{(\chi W_z/C_{r(1)} - T^{\mathrm{Q}}_{2,h(2)} + T^{\mathrm{Q}}_{2,r(1)})C_{h(2)}}{C_{r(1)} + C_{h(2)}} \\ &\le \frac{\Delta t_1^{r(1)} C_{r(1)}}{(C_{r(1)} + C_{h(2)})},\end{aligned} \tag{C.4}$$

where equality can be achieved when $T_2^{\mathrm{T}} \leq T_{2,r(1)}^{\mathrm{Q}} - \Delta t_1^{r(1)}$.

- *Case 3:* Task 2 regards RSU $r(1)$ as the helper RSU, *i.e.*, $r(1) = h(2)$. In this case, the new optimal service time of task 2 is

$$(T_2^{\mathrm{service}})' = \max\{T_2^{\mathrm{T}}, T_{2,r(2)}^{\mathrm{Q}} - \Delta t_1^{r(2)}\} + \frac{(T_{2,r(1)}^{\mathrm{Q}} - \Delta t_1^{r(2)} - \max\{T_2^{\mathrm{T}}, T_{2,r(2)}^{\mathrm{Q}}\})C_{r(1)} + \chi W_2}{C_{r(2)} + C_{r(1)}}. \tag{C.5}$$

  Similar as case 1, the reduced service time for task 2 is $\Delta T_2 = \Delta t_1^{r(1)} C_{r(1)}/(C_{r(1)} + C_{r(2)})$.

Considering that the computation capabilities C_r are identical for all servers (the first assumption in Lemma 7), the maximum service time deduction for task 2 is $\Delta t_1^{r(1)}/2$. For all tasks queued after task 1 in RSU $r(1)$, the overall service time deduction is less than $\Delta t_1^{r(1)}[1/2 + (1/2)^2 + (1/2)^3 + \ldots]$, which is always less than $\Delta t_1^{r(1)}$. We omit the proof for the case when the workload assigned in RSU $h(1)$ is decreased by Δx due to the similarity. Therefore, we obtain that, under the assumptions presented in Lemma A, the overall service time will be increased if the workload allocation does not follow the task partition ratio presented in Lemma 7.

Appendix D

Proof of Lemma 8

Suppose the tasks in edge server r are scheduled by the shortest-task-first rule, and task 2 is queued after the task 1. Then, we have

$$\max\{T^{\mathrm{Q}}_{1,r}, T^{\mathrm{T}}_{1,r}\} + T^{\mathrm{P}}_{1,r} \leq \max\{T^{\mathrm{Q}}_{1,r}, T^{\mathrm{T}}_{2,r}\} + T^{\mathrm{P}}_{2,r}. \tag{D.1}$$

If the order of task 1 and task 2 are switched with each other, the service time of task 2 will be decreased by

$$D = \max\{\max\{T^{\mathrm{Q}}_{1,r}, T^{\mathrm{T}}_{1,r}\} + T^{\mathrm{P}}_{1,r}, T^{\mathrm{T}}_{2,r}\} - \max\{T^{\mathrm{Q}}_{1,r}, T^{\mathrm{T}}_{2,r}\}. \tag{D.2}$$

On the other hand, the service time of task 1 will be increased by

$$I = \max\{T^{\mathrm{Q}}_{1,r}, T^{\mathrm{T}}_{2,r}\} + T^{\mathrm{P}}_{2,r} - \max\{T^{\mathrm{Q}}_{1,r}, T^{\mathrm{T}}_{1,r}\}. \tag{D.3}$$

From (D.1), we can derive that $I \geq T^{\mathrm{P}}_{1,r}$. Then, the overall service time of tasks 1 and 2 will be increased by

$$\begin{aligned} I - D \geq T^{\mathrm{P}}_{1,r} - \max\{\max\{T^{\mathrm{Q}}_{1,r}, T^{\mathrm{T}}_{1,r}\} + T^{\mathrm{P}}_{1,r}, T^{\mathrm{T}}_{2,r}\} \\ + \max\{T^{\mathrm{Q}}_{1,r}, T^{\mathrm{T}}_{2,r}\}. \end{aligned} \tag{D.4}$$

We then list the three scenarios on $T^{\mathrm{T}}_{2,r}$:

- *Case 1:* $T^{\mathrm{T}}_{2,r} \geq \max\{T^{\mathrm{Q}}_{1,r}, T^{\mathrm{T}}_{1,r}\} + T^{\mathrm{P}}_{1,r}$. In this case, $I - D \geq T^{\mathrm{P}}_{1,r} \geq 0$.
- *Case 2:* $T^{\mathrm{Q}}_{1,r} \leq T^{\mathrm{T}}_{2,r} \leq \max\{T^{\mathrm{Q}}_{1,r}, T^{\mathrm{T}}_{1,r}\} + T^{\mathrm{P}}_{1,r}$. In this case, $I - D \geq T^{\mathrm{T}}_{2,r} - \max\{T^{\mathrm{Q}}_{1,r}, T^{\mathrm{T}}_{1,r}\}$. According the assumption, where $T^{\mathrm{T}}_{1,r} \leq T^{\mathrm{T}}_{2,r}$, we then have $I - D \geq 0$.

- *Case 3:* $T^{\mathrm{T}}_{2,r} \leq T^{\mathrm{Q}}_{1,r}$. In this case, $I - D \geq T^{\mathrm{Q}}_{1,r} - \max\{T^{\mathrm{Q}}_{1,r}, T^{\mathrm{T}}_{1,r}\} = 0$.

Therefore, we can obtain the conclusion that the service time will be increased if the task execution order does not follow a shortest-task-first rule under the assumptions.