Task Offloading and Proactive Resource Allocation in Vehicular Edge Computing via Reinforcement Learning

by

© **Elham Karimi**

A thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Department of Computer Science

Memorial University

August 2022

St. John's, Newfoundland and Labrador, Canada

Abstract

Given the rapid increase of various applications in vehicular networks, it is crucial to consider a flexible architecture to improve the Quality-of-Service (QoS). Utilizing Multi-access Edge Computing (MEC) as a distributed paradigm with computation capabilities closer to the vehicles can be a promising solution to reduce response time in such a network. However, MEC nodes are deprived of processing all tasks offloaded by the vehicles and suffer from limited resources compared with the central cloud. The offloaded tasks usually have different priorities for processing and various resource demands, and they even are dropped when the response time for task processing expires. Due to the workload dynamics at MEC nodes and the randomness of task arrivals, it is challenging to determine proper MEC servers for task offloading and how to manage resources for the application's demands.

This dissertation proposes cooperation between MEC and central cloud decisions for different vehicular application offloading. First, we formulate a new NP-hard resource allocation problem to guarantee the required response time. We transform the environment into a finite Markov decision process that only depends on the current state and action space. We utilize deep reinforcement learning, a proper computational model, to automatically learn the dynamics of the network state and rapidly capture an optimal solution based on the current state and action space. In this dissertation, we develop an intelligent workload prediction of each MEC node utilizing Multivariate Long Short-Term Memory (LSTM) to propose a proactive resource allocation algorithm for various tasks in a dynamic vehicular network. In our algorithm, we classify tasks based on their priorities and migrate the tasks with lower priority to provide service for those with higher priority. Moreover, we apply distributed deep reinforcement learning to solve our problem to increase the efficiency and accuracy of the proactive resource allocation algorithm. Extensive numerical analysis and results

illustrate how our proposed algorithms can increase the ratio of accepted high-priority tasks and reduce response time.

To my lovely parents

Acknowledgements

As this chapter of my life comes to an end, it is a great pleasure to acknowledge several individuals who have contributed to who I am today.

First and foremost, I would like to express my gratitude to my advisor, Prof. Yuanzhu Chen, for his excellent guidance throughout my Ph.D. journey. I would not have come this far without his continuous support and inspiration. Thank you for teaching me to think clearly and independently and present concisely. Also, I acknowledge the financial support provided by Prof. Yuanzhu Chen, the Natural Science and Engineering Research Council (NSERC), and the Dean's Doctoral Award as a top graduate entrance scholarship.

I am deeply indebted to Dr. Behzad Akbari, the best supervisor I have ever known in my entire life, whose expertise, understanding, willingness to help, and tremendous patience made this difficult journey more manageable and more productive. His enthusiasm, support, and belief in me guided me during tough times and added considerably to my graduate experience during my M.Sc. and Ph.D. research.

I am also very grateful to my co-supervisor, Dr. Xianta Jiang, for his kind helps, care, patience, and collaboration. I am also thankful to the chair of the computer science department, Dr. Oscar Meruvia-Pastor, for all his support in making my requests run smoothly.

I thank Wireless Networking and Mobile Computing Laboratory (WineMocol) group members for fruitful discussions and invaluable advice during my research. I also acknowledge Hossein Hassani for his valuable helps in reinforcement learning. My thank also goes to Alzahra group members who provided online religious gatherings to help me overcome my life challenges and be my best companion to make targeted moments in St. John's.

I would like to acknowledge the financial support the School of Graduate Studies provided. My thank also goes to my supervisor, Prof. Yuanzhu Chen, again for giving me a powerful iMac, an excellent PC.

A significant part of my education was in Iran, where my foundations were laid. I wish to offer my sincere thanks to all my teachers at "HojrebneAdi" elementary school, "AmirAlMomenin" middle school, and "Ayandeh Sazan" high school in Karaj, and Qazvin Islamic Azad University in Qazvin. I am also very grateful to all my colleagues at Ghiaseddin Jamshid Kashani University for their collaboration. In particular, very thankful to Prof. Bahman Mehri for his kind help and the many lessons I learned from him.

Last but not least, I am deeply grateful for the support and love I received from my parents, AliHossein Karimi and Soghra Sarlak, and my two brothers, Vahid and Saeed. Thank God for having my cute niece Chamaan Karimi two years old; watching videos of her my family sent to me was the only entertainment that ignited my hope during my research.

Elham Karimi
St. John's, June 2022

Statement of contribution

I, Elham Karimi, hold a principal author status for all the manuscript chapters (Chapter 2 - 5) in this dissertation. However, each manuscript is co-authored by my supervisors, Dr. Yuanzhu Chen, Dr. Behzad Akbari, and Dr. Xianta Jiang, whose contributions have expedited the progress of developing the ideas and their formulation, conducting computational experiments, and refinement of the presentation. The contributions for each chapter are mentioned in the followings:

- Chapter 3:

  E. Karimi, Y. Chen, and B. Akbari, "Task offloading in vehicular edge computing networks via deep reinforcement learning," Computer Communications, 2022.

- Chapter 4:

  (Women in Engineering (WIE) Best Paper Award) E. Karimi, Y. Chen, and B. Akbari, X. Jiang, "An Intelligent Resource Allocation and Scheduling in Vehicular Edge Computing," IEEE Newfoundland Electrical and Computer Engineering Conference (NECEC), 2021.

- Chapter 5:

  E. Karimi, Y. Chen, and B. Akbari, X. Jiang, "An Intelligent, Decentralized and Proactive Resource Allocation Algorithm in Vehicular Edge Computing Networks," Submitted to journal, 2022.

Elham Karimi

Table of contents

List of tables

List of figures

List of symbols

Latin

A	Crucial application category
Acc	Accepted rate
$a(t)$	Action space
B_{acc}	Bandwidth of access link
B_{ni}	Bandwidth of network link
$\widehat{C}(t)$	Running tasks in each servers at time slot t
d_m	Data size of request m
J/j	Number of tasks/index
K	Number of episodes
$\mathcal{M}$/M/m	Set/number/index of vehicles
$\mathcal{N}$/N/n	Set/number/index of MEC servers
$\widehat{P}_b(t)$	Predicted workload of server b
R_i	Vector of resource capacity of server i
R_i^{comp}	Computation capacity of server i
R_i^{stor}	Store capacity of server i
$r_m(t)$	Reward for task m
Rej	Rejected rate
RTT_{acc}	Round trip time between vehicle and MEC
RTT_{ni}	Round trip time between base MEC n and execution node i
$S(t)$	State space
$\mathcal{V}$/i	Set of all MEC servers plus central server v/index
$\widehat{W}(t)$	Application waiting in the buffer of each server in time slot t

List of symbols

Greek

α	Learning rate
β_m^i	Indicator if task m is in the buffer of server i
γ	Discount factor
ϵ_0	First value of ϵ in $\epsilon - greedy$
ϵ_{min}	Min value of ϵ
$\lambda(t)$	Rate of Poisson process
ξ_m^i	Indicator if application from vehicle m is executed on server i
τ_m^{access}	Access latency of application m
τ_m^{est}	Estimated response time of application m
τ_m^{mig}	Migration latency of application m
τ_m^{proc}	Processing latency of application m
τ_m^{queue}	Queuing latency of application m
$\widehat{\phi}$	Application information
ϕ_m	Vector of application resource demand of vehicle m
ϕ_m^{comp}	Application computation demand of vehicle m
ϕ_m^{stor}	Application storage demand of vehicle m
ϕ_m^{res}	Request's tolerable response time of vehicle m
ψ_m	Indicator the accepted request m by the network
Ψ_m	Indicator the rejected request m by the network

List of abbreviations

ARIMA	Auto-Regressive Integrated Moving Average
CA	Crucial Application
CPU	Central Processing Unit
DRL	Deep Reinforcement Learning
DDPG	Deep Deterministic Policy Gradient
DQL	Deep Q-Learning
DSRC	Dedicated Short Range Communications
FMDP	Finite Markov Decision Process
GPS	Global Positioning System
HPA	High Priority Application
LPA	Low Priority Application
LRTF	Least Response Time First
LSTM	Long Short-Term Memory
LTE-V	Long-Term Evolution-Vehicle
MDP	Markov Decision Process
MEC	Multi-access Edge Computing
NP	Nondeterministic Polynomial time
QoS	Quality of Service
RL	Reinforcement Learning
RSU	Road-Side-Unit
RTT	Round Trip Time
TD	Temporal Difference
UAV	Unmanned Aerial Vehicle
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VEC	Vehicular Edge Computing

Chapter 1

Introduction

1.1 Background

Intelligent transportation systems are designed to provide innovative applications and services for vehicles, drivers, and passengers, as well as to facilitate access to information for other systems and users [3]. A significant component of future intelligent transportation systems is the vehicular networks. Vehicular networks have emerged due to advancements in wireless technologies, ad-hoc networking, and the automobile industry. These networks are formed among moving vehicles, Road-Side-Units (RSUs), and pedestrians that carry communication devices. Vehicular networks can be deployed in rural, urban, and highway environments [2]. Vehicular networks are popular as they provide safety, ease of driving, convenience, and greater efficiency.

Vehicular networks have several characteristics, such as highly dynamic traffic, local decision making, execution of ultra low latency applications, lack of previous network knowledge, and the need for real-time decision making. A large number of applications are rapidly growing in vehicular networks. Application scenarios in vehicular networks could include traffic control, path navigation, ultra-low latency service, and entertainment. These applications have specific requirements on computation resources and task processing latency [34, 43, 48, 59, 80]. The emerging vehicular applications with various demands have naturally increased the essential needs of communication, computation, and storage resources while providing the required Quality-of-Service (QoS).

A new computing paradigm, Vehicular Edge Computing (VEC), has been introduced to the vehicular network to grow its computing capacity. However, the vehicles' high mobility and dynamic topology make utilizing computing capacity a challenge. As a result, computation and storage allocation in VEC is a vital issue that should be addressed.

1.1.1 Vehicular edge computing

To better describe Vehicular Edge Computing (VEC), we first introduce vehicular cloud computing and then illustrate the Multi-access Edge Computing (MEC) paradigm.

Cloud computing provides centralized computation and storage services using either a cloud server or many remote servers. Cloud computing benefits the users with virtual servers that allow remote storage capacity and computational facilities [63]. Vehicular cloud computing is a technology that takes advantage of cloud computing to provide several computation services at low cost to the vehicle drivers; The cloud computing paradigm has enabled the exploitation of excess computing power. One critical question arises while dealing with cloud computing, about the delay that takes place while transferring data from vehicles to the cloud server and retrieving the information after being stored and processed.

MEC technology aims at extending cloud computing capabilities. A significant feature of MEC is to drive mobile computing and storage to the network edges closer to the users, which provides low latency [61]. Driven by the benefits of MEC, many efforts have been devoted to integrating vehicular networks with MEC, thereby forming a novel paradigm named VEC [43]. Fig. 1.1 presents a system model of vehicle-assisted MEC Network. VEC has a great potential to enhance traffic safety and improve travel comport by integrating MEC into vehicular networks. VEC literature has seen several works mainly on computation offloading [16, 20, 62].

Vehicles can offload their computation-intensive and latency-sensitive tasks to the edge servers, which can considerably reduce the response time [43]. Road-Side-Units (RSUs) are often equipped with edge servers distributed along the road in a city. For instance, ultra-low latency service must execute immediately in the MEC server, and

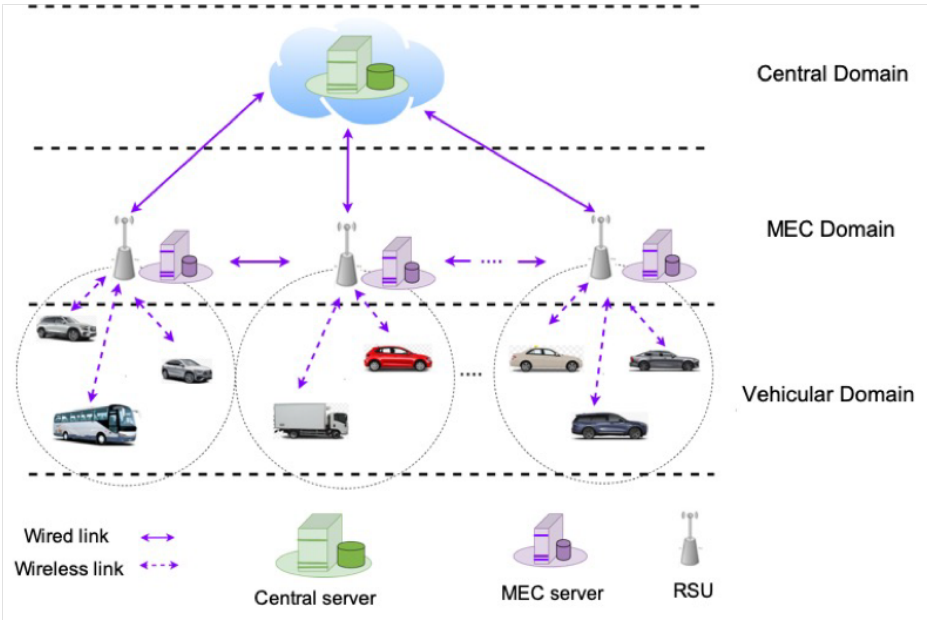


Figure 1.1: System model of vehicle-assisted MEC network.

drivers should be aware of traffic accidents and perform more efficient and effective operations. Furthermore, entertainment applications will benefit from rich computation and storage resources that should migrate to the central cloud for execution.

In vehicular networks, MEC servers are often deployed near RSUs. The computing ability of MEC will also be reduced to a certain degree. Another important property is heterogeneity. Because of the different network environments and communication technologies, MEC should support various hardware and software devices to satisfy the safety demand. In addition, MEC's transmission management and server selection must adapt to the vehicle's mobility to achieve low latency in high-speed mobile environments [47].

MEC facilitates vehicles to offload their tasks to the nearest edge cloud for processing to reduce the response time and increase the ratio of accepted tasks. However, the MEC's restricted computation and storage capacities are considered a barrier that cannot quickly satisfy the increasing resource demands of the vehicles [61]. Despite the development of MEC in vehicular edge computing, there are still two main challenges related to task offloading and resource management. The first challenge is whether an edge cloud, which received a task, should execute it or migrate it to the other MEC to find a proper place to execute it. The second challenge is if an edge cloud decides to relocate the task to the other MEC for processing, which MEC should be chosen.

1.1.2 Computation/storage allocation in VEC and its benefits

VEC networks support an array of applications, which include three main categories: 1) road safety applications (e.g., lowering the risk of accidents); 2) traffic efficiency applications (e.g., reducing travel time and alleviating traffic congestion; 3) value-added applications (e.g., providing infotainment, path planning and Internet access) [43]. The rapid development of vehicular networks will facilitate wide use of smart vehicles, which enables a large number of various types of applications [91, 92]. Fig. 1.2 shows application scenarios of VEC. In vehicular edge computing, the number of tasks that are simultaneously offloaded to the MEC servers may change quickly over time. Therefore, investigating the computation/storage allocation in such a network is significant to guarantee low latency application. In addition, compared to cloud computing, the resources of VEC in terms of computation and storage are limited. If all the vehicles offload their tasks to the same edge server, it will likely be overloaded and degrade network performance. Thus, how to manage these resources is vital. Given dynamic resource demands, diverse application characteristics, and complex traffic environments, optimizing resource allocation is a challenging task [43].

The main focus of MEC is an efficient resource management algorithm to optimize the utilization of edge server resources by considering the distinction of task priorities. In addition, several existing works [12, 18, 79] consider cloud resource provisioning in edge cloud. This dissertation focuses on computation/storage resource allocation and task offloading in VEC.

1.2 Research motivation and challenges

1.2.1 Limited resources in MEC compared with central cloud

A large number of applications are rapidly growing in vehicular networks. The emerging vehicular applications with various demands have naturally increased the essential needs of communication, computation, and storage resources while providing the required QoS.

Compared with central cloud, one of the barriers is the limited computation and

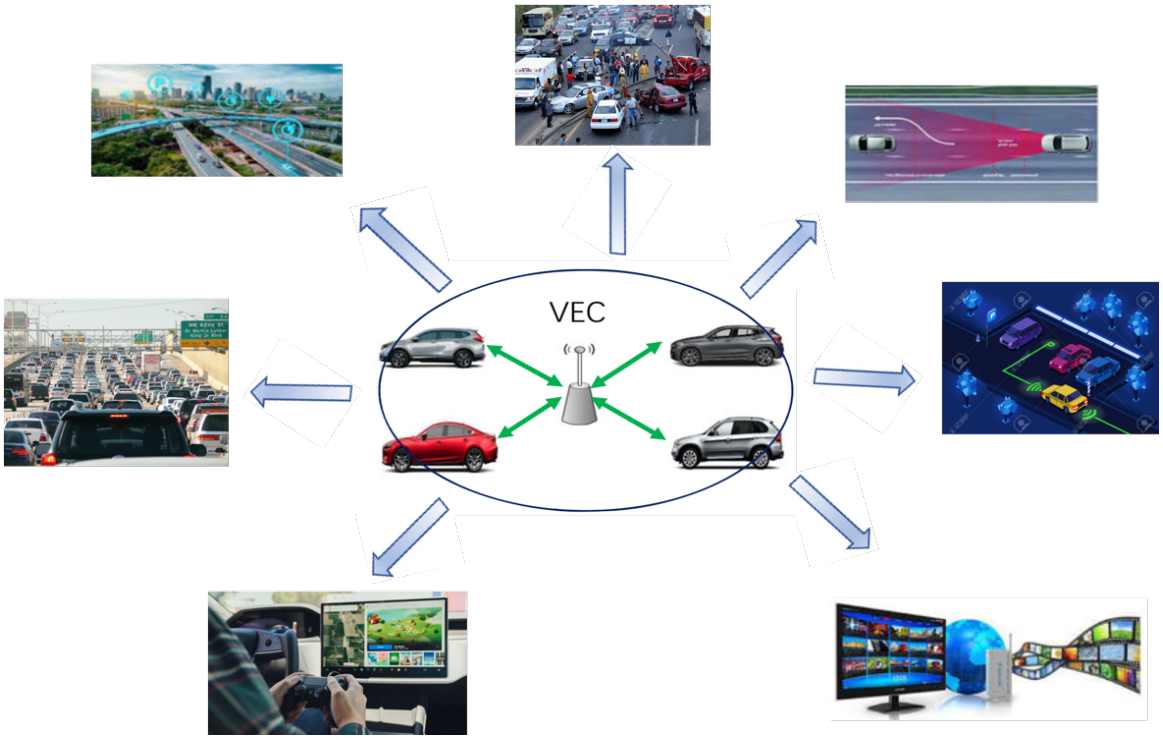


Figure 1.2: Application scenarios of vehicular edge computing.

storage capacities in the MECs, which cannot quickly satisfy the increasing resource demands of such applications, especially those with intensive computation and rigid delay requirements. To tackle these issues, using central cloud computing is widely regarded as a promising solution [13]. Central services are deployed in the remote clouds, and they can get the uploaded information from edge servers. Compared with edge servers, central clouds have large capacities in terms of computation and storage and cover a much broader area. The central cloud paradigm can provide global level management and centralized control, which helps make optimal decisions. Despite the advantages of a central cloud, high data transmission latency is a consequence that is not tolerated in variant types of new emerging applications. Limited capacities in the edge networks and low latency requirements of some applications need efficient allocation of edge resources and leveraging the resources in the central clouds. However, most existing works did not have efficient cooperation between MEC and central servers with optimized task migration.

Motivated by the issues and challenges in limited resources in edge computing, we present a cooperation framework between MEC and central clouds. The offloaded

tasks to the edge servers are migrated between edge and central clouds according to the tasks requirements and the available resources in the edge clouds.

1.2.2 Acceptance of crucial applications

Vehicle applications can be classified into different levels according to their characteristics. Crucial applications are core vehicle system applications or safety-related applications. Because of their importance to the vehicle and the passengers, crucial applications have the highest priority and must be executed immediately. However, there is no distinguishing difference among tasks in the resource allocation policies in existing work.

The number of concurrent tasks offloaded to the MEC servers may change quickly over time in vehicular edge computing. Thus, investigating the workload in resource allocation and task offloading for crucial applications in such a network is important. Moreover, designing a distributed algorithm is challenging because there is no prior knowledge about workloads at MECs [67]. Furthermore, it is significant to proactively manage and allocate resources concerning workload dynamics to satisfy user expectations, especially for crucial tasks. This leads to the demand for a mechanism that is capable of predicting workload variations in MEC servers [52].

To increase the acceptance rate of crucial tasks in high dynamic networks, we propose two approaches in this dissertation. The first approach is the reallocation and scheduling strategy, which classifies vehicular tasks based on their tolerable response time and prioritizes the crucial tasks for resource allocation. The second approach is to present intelligent workload prediction and proactive resource allocation.

1.2.3 Efficient resource allocation and task offloading

Another critical problem in vehicular edge computing is how to decide which MEC servers should accept the offloaded application tasks. Service migration schemes introduced in several studies [19,80] to support reliable computing services. Most of the studies in edge computing suffer a lack of real high dynamic network consideration. The authors did not consider the high volume of task migration and limited resources in MEC for a considerable number of arriving application task offloading requests

and mostly ignored the increased mobility of the vehicles. Several studies [21, 84, 89] considered cooperation among local clouds and remote clouds. However, these works do not address the problem of migration in application tasks.

In dynamic vehicular edge networks, some difficulties arise when resource allocation is defined as an optimization problem. The optimization problem is an NP-hard problem, and achieving the optimal or sub-optimal solutions usually requires exponential time complexity. Moreover, the optimization procedure must be executed at each time slot due to the diversity of request requirements, the number of requests, and resource availability in the MEC server, which are time-varying and highly dynamic. Therefore, the conventional optimization methods using relaxation iteration algorithms incur high computational complexity due to high iterations, and their solutions are often sub-optimal. They would not scale well [64], usually converge slowly, and have prohibitive complexity for real-time implementations [74]. We believe a machine learning-based approach is an effective and attractive solution to tackle this problem.

1.2.4 Distributed and machine learning approach

Resource allocation and task offloading problems as NP-hard problems are difficult to solve via conventional mathematical techniques near-real-time [74, 75]. Moreover, in a dynamic vehicular edge computing network, task requirements and resource capabilities' MEC servers are time-varying; most of the existing approaches need to solve such a problem very frequently to obtain the optimal or suboptimal offloading strategy. However, this introduces significant computational overhead to vehicular edge computing, and it can hardly get the optimal offloading solution in real-time [74].

Due to the time-varying of task requirements and resource capabilities in MEC servers in the dynamic nature of vehicular networks, in this dissertation, we formulate dynamic offloading and resource allocation as Finite Markov Decision Process (FMDP) involving state and action spaces and only depending on the current state and action. FMDP is a prominent part of deep reinforcement learning. Therefore, optimum task offloading using Deep Reinforcement Learning (DRL) could ensure the system performance measured by latency and acceptance rate [75].

One significant characteristic of vehicular systems is local decision-making and

distributed task offloading, and a resource allocation strategy would be an essential need. In most existing works [22, 24, 38, 46, 58, 86], deep learning algorithms for MEC systems have been proposed, while the authors focused on centralized resource allocation and offloading algorithms, and there is no workload consideration in that works. In this dissertation, we develop a resource allocation and task offloading utilizing a distributed DRL where each MEC server learns and acts as an independent agent.

1.3 Research contributions

This dissertation presents the following novel contributions to the resource allocation in vehicular edge computing networks

- We apply an efficient collaboration among MEC, and the central cloud for task offloading and resource allocation. We design a task migration scheme among local and remote servers based on resource restrictions and vehicles' QoS requirements while respecting the tolerable response time of each application.
- We adopt a machine-learning approach, which utilizes a model-free method to find the optimal offloading task scheme and maximize the vehicular application task's acceptance rate. In other words, we develop a central Deep Reinforcement Learning (DRL) deployed in the central cloud.
- We classify three different application tasks with different characteristics and priorities and propose reallocation and scheduling policies based on three categories of vehicle applications. The proposed scheduling policy minimizes the rejection rate of applications. Moreover, to model the dynamic nature of the vehicular network, we generate arriving tasks of each category based on inhomogeneous Poisson distribution.
- We propose an efficient, proactive resource allocation in a highly dynamic vehicular system by developing a decentralized DRL-based task offloading and resource allocation algorithm, enabling each MEC server to find the optimal solution for proactive resource allocation problems. Furthermore, we propose a Multivariate-LSTM approach to predict near-future workload in each MEC server. The prediction workload is based on different application tasks priority.

The workload prediction results provide the opportunity for lower priority tasks migration to release free resources for crucial applications that have the highest priority.

1.4 Thesis outline

The rest of this dissertation is organized as follows. Chapter 2 provides a review of studies on vehicular edge computing. We categorized the studies in five topics as cooperation local/central cloud, cloud resource provisioning, task offloading, task migration, deep learning in VEC, workload prediction. By comparing existing studies on the subject, we lay the groundwork for further research in the next chapters.

In Chapter 3, we propose task offloading with a resource allocation strategy as an optimization problem. The system model includes a cooperation framework between MEC and central clouds. The offloaded tasks to the edge servers are migrated between edge and central clouds according to the tasks requirements and the available resources in the edge clouds. Our strategy respects each application's acceptable response time and maximizes the acceptance rate of applications. We jointly optimize the task offloading decision and computational resource allocation using deep reinforcement learning for a highly dynamic vehicular edge computing environment. After evaluation and getting the results, we propose a complementary algorithm in Chapter 4. The complementary algorithm can reallocate resources for new and previous tasks based on tasks' priorities. In this context, different applications have different requirements, especially response time. We classify vehicle applications into three levels according to their characteristics and apply reallocation and scheduling strategies.

In Chapter 5, we extend our work from Chapter 3. We focus on proactive resource allocation and task offloading in vehicular edge computing. The primary motivation of our proactive resource allocation algorithm is to gain high QoS and to provide a high acceptance rate of crucial vehicular tasks. We develop a distributed deep learning scheme following two objectives. The first is resource allocation and task offloading utilizing a distributed DRL where each MEC server learns and acts as an independent agent. The second is distributed workload prediction based on the Multivariate LSTM method to proactively manage the MEC servers' multi-dimensional resources.

Finally in Chapter 6, we conclude and summarize the contributions presented in

this dissertation, and discuss several potential extensions to our research.

Chapter 2

Related work

Many studies have been conducted on resource allocation and task offloading in vehicular edge computing networks in recent years. There are many optimization tools and methodologies to solve resource allocations. However, in VEC, because of its highly dynamic nature, many challenges arise. Many experimental studies have focused on intelligent resource allocation in this network, mainly on computation and Vehicle-to-Vehicle (V2V)/ Vehicle-to-Infrastructure (V2I) communications. A famous and influential method conducted in research is utilizing deep reinforcement learning.

This chapter overviews the various existing works for task offloading and resource allocation in vehicular edge computing networks. The overview is classified based on key concepts related to this dissertation. We describe related work to capture more about resource allocation, task offloading and migration, central/distributed deep reinforcement learning, and workload prediction. Furthermore, we summarize the related works in the last subsection of this chapter.

2.1 Computation/storage resource allocation

During rush hours, the demands of vehicular applications can constitute a network with strong computing resources that proper resource allocation strategies bring lots of benefits. Many research studies on VEC networks considered applying MEC resources to complete application requests from vehicles on roads and focused on allocating limited resources to achieve multiple purposes for road safety. For instance,

the work in [80] proposed a game-theoretical approach for the system. Furthermore, there is a great challenge in gathering, storing, and processing all the data of these vehicular tasks, and then the management of the resources is yet another task. An edge server must have sufficient resources to execute all application tasks. In reality, an edge node has an inadequate collection of resources and therefore, these can become overloaded when many vehicular requests arrive at once; for instance, at the time of peak traffic, it results in degraded performance. If all the vehicles offload their tasks to the same edge server, it is likely to be overloaded, degrading the network performance. The author of [37] targeted the management strategies in each edge node, channeling resource management in FeRANs. The QoS is improved, focusing on real-time vehicular services, and two schemes were introduced, named fog resource reservation and fog resource reallocation.

The main focus of MEC is an efficient resource management algorithm to optimize the utilization of edge server resources by considering the distinction of task priorities, and several existing works [12, 18, 79] consider cloud resource provisioning. The authors of [12] investigated resource allocation by jointly considering load balancing and offloading in a multi-user, multi-server vehicular environment. This proposed solution allows vehicles to select their preferable edge servers based on their requirements and accounts for the mobility of cars. The work in [79] were considered both the local and remote resource sharing with the collaboration across different data centers. A coalition game was formulated and solved by a game-theoretic algorithm with stability and convergence guarantees to realize resource sharing and cooperation among other servers. Moreover, the work in [18] proposed a new server cooperation scheme where edge servers exploit both the computational and storage resources by proactively caching computation results to minimize the computation latency. The corresponding task distribution problem was formulated as a matching game and solved by an efficient algorithm based on a proposed deferred-acceptance algorithm.

The authors of [87] present a regional cooperative fog computing to provide various services. A localized coordinator supports interoperability and cooperative operation among local fog servers. A hierarchical resource management model is developed to optimize the network performance in the fog computing network. The problem of how to efficiently orchestrate combined edge-cloud applications is, however, incompletely understood, and a wide range of techniques for resource and application management are currently in use [17].

2.2 Optimization tools for resource allocation

A resource allocation problem can be defined as an optimization problem to maximize or minimize some QoS parameters in the network. Several optimization tools are highlighted, including convex optimization, stochastic optimization, game theory, graph theory, and reinforcement learning. We briefly consider them as follows [26] while describing their properties for resource allocation in VEC:

- *Convex optimization:* Convex optimization is a subfield of mathematical optimization that involves the problem of minimizing convex functions over convex sets. Convex optimization technology has been applied to various fields, including communications and networks, signal processing, automatic control, data analysis, modeling, etc. A convex optimization problem is in standard form if it is written as in [6]. Many RA problems are non-convex in VECs, which are hard to solve. Moreover, the complexity of RA is high to find the optimal solution for many vehicles. It is hard to solve dynamic, time-varying optimization problems.

- *Stochastic optimization:* Whereas deterministic optimization problems are formulated with known parameters, vehicular ad hoc network-based resource problems almost invariably include some unknown parameters. Stochastic optimization methods are optimization methods that generate and use random variables. Some stochastic optimization methods use random iterates to solve stochastic problems. The main disadvantage of Stochastic optimization methods is that there is no guarantee for a global optimal solution.

- *Game theory:* Game theory is the study of mathematical models of strategic interaction among rational decision-makers, which has applications in various fields, including logic, system science, and computer science. The assumption that players have the knowledge about their own pay-offs and pay-offs of others is not practical. In a vehicular network, complete information to select a strategy or action may not be available to a vehicular node or MEC server.

- *Graph theory:* Graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. Adjacency matrices consume a massive amount of memory for storing big graphs.

- *Reinforcement learning:* Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment to maximize some notion of cumulative reward. Fig. 2.1 presents the architecture of RL. The general RL problem is formalized as a discrete-time stochastic control process, such as MDP. In general, there are two main types of RL methods, including value-based and policy-based. The value-based RL method tries to find or approximate the optimal value function, which is a mapping between an action and a value, and the most famous algorithm is Q-learning. The policy-based RL method tries to find the optimal policy directly without the Q-value as a middleman. Deep reinforcement learning (DRL) uses deep learning and reinforcement learning principles to create efficient algorithms, and previously some unsolvable problems could be solved by using the powerful DRL model [40]. Taking advantage of the general-purpose framework in decision-making, reinforcement learning is widely used in solving resource allocation problems in VEC networks [66].

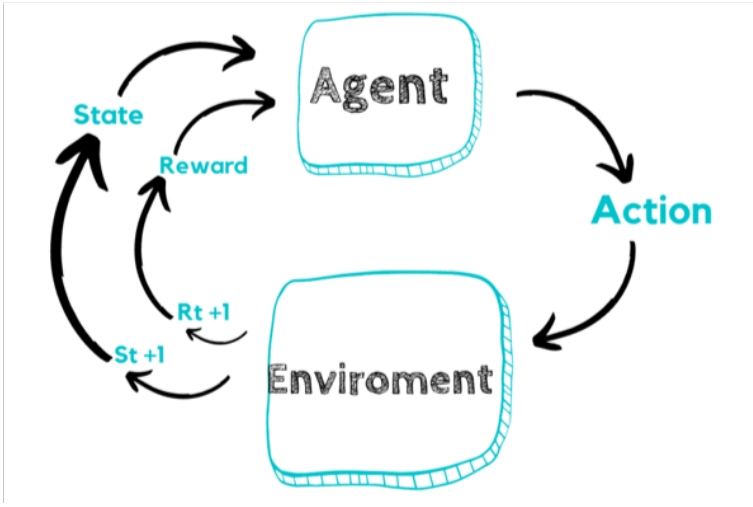


Figure 2.1: Reinforcement learning

The performance metrics, including delay, energy, bandwidth, reliability, etc., are main indicators of the network performance, which guides optimizing the network performance [43]. The authors of [87] present a regional cooperative fog computing to provide various services. A hierarchical resource management model is developed to optimize the network performance in the fog computing network. Two scheduling schemes based on response time and queue length are presented to schedule data

for adapting to the changing network and enhancing data dissemination efficiency, respectively. The work in [28] tries to optimize the utilization of fog computing resource, to fulfill the delay requirement of tasks by assigning each task with the best fog server with the assistance of Software-Defined-Network.

2.3 Task offloading

A critical challenge in VEC is how to find a proper location for task execution. In other words, deciding which cloud, either edge or central, is appropriate for running the tasks. Many vehicular application tasks have stringent requirements in terms of computation and delay. Despite abundant resources, the cloud is unfeasible to support delay-sensitive applications because of the long distance from vehicles. By contrast, VEC is envisioned as a promising solution. Vehicular tasks can be transferred to the MEC nodes with some computation and storage resources, significantly shortening the delay and alleviating the network load [43]. Although making a good decision for task offloading is challenging because of the highly dynamic nature of vehicular networks, much research has been done to investigate task offloading in VEC. Task offloading was considered in [12, 14, 16, 45, 81, 82], where tasks could be offloaded in different domains to improve resource utilization, including servers in RSUs and the central cloud. We classify task offloading models into two models. The first model is edge-based, and the second model is the cooperation of edge and central cloud, which are considered as the following subsections.

2.3.1 Edge-based offloading

Edge servers can reduce transmission costs and generate a fast response in the offloading services because of the closeness to the vehicular users. Despite the rapid response rate, the edge servers usually face the limitation of the resources as compared to the conventional cloud servers, which have a significant computational capacity. The edge servers take a certain time to perform the computation tasks. This is especially true for the edge servers located at the road segments, which have a high density of vehicles in comparison to others [63]. Fig. 2.2 presents a system mode of edge-based task offloading.

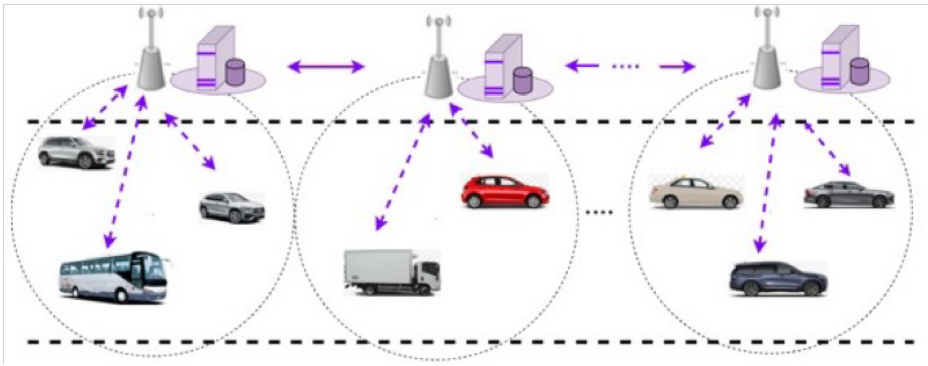


Figure 2.2: System model of edge-based task offloading.

A computation offloading framework that allows a mobile device to offload tasks to multiple MEC servers was proposed in [14], and semidefinite relaxation-based algorithms were also proposed to determine the task allocation decisions and CPU frequency scaling. By considering the requirements of computation tasks and the mobility of vehicles, a predictive-model transmission strategy was introduced in [82] for task offloading, improving the transmission efficiency, and satisfying the required delay. Besides, an optimal offloading scheme based on prediction was designed for accomplishing diverse types of computation tasks. The authors of [12] investigated resource allocation by jointly considering load balancing and offloading. The work in [45] investigated the problem of multiple-user computation offloading on an edge server to reduce communication overhead. The coupling of offloading decisions from vehicles, game theory was used to make optimal offloading decisions and suitable channel selections for vehicles. When determining the offloading strategy, the authors of [16] took into account the profits of both vehicle and edge server simultaneously. There, a dual-side optimization was formulated to minimize their costs. On the side of vehicles, the offloading decisions and local CPU frequencies were jointly optimized; radio resource allocation and service provisioning were both considered on the side of the edge servers. Authors in [81] presented task offloading in vehicular networks. However, they did not consider resource restrictions in MEC.

In [83], a computational offloading infrastructure was analyzed, which stresses the computational effectiveness of the transfer frameworks of V2I and V2V modes of communication. Moreover, an efficient predictive combination-mode relegation scheme while taking into account the time consumption of the execution of the tasks and the mobility of the vehicles was also proposed. In this model, the tasks are

offloaded to the MEC servers over direct uploading and predictive relay transmissions.

The work in [45] investigates the problem of multiple-user computation offloading on an edge server to reduce communication overhead. The computation offloading of each vehicle refers to channel selection for uploading its task to the edge server. Different from the studies in [45] with a focus of single edge server, multi-server scenario is considered in [68], where high reliability and low latency Vehicle-to-Infrastructure (V2I) communication architecture is proposed by jointly optimizing Autonomous Vehicle (AV)-to-Small Base Station (SBS) association and wireless resource management.

2.3.2 Cooperation of edge and cloud

A generic VEC architecture includes cooperation between local and central clouds, and several related works used this architecture [21, 84, 89]. Collaboration between MEC and central cloud leverages both the low communication latency due to the proximity of the MEC server and the low computation latency arising from abundant computational resources at the central-cloud servers. In [89], the server selection problem was studied for a multi-user system comprising a single edge server and a single central cloud. A heuristic scheduling algorithm has been proposed to maximize the total success of offloading probability. The authors designed a threshold-based policy to improve the QoS by cooperation of the local cloud and Internet cloud resources, which simultaneously takes advantage of the local cloud's low latency and abundant computational resources of the Internet clouds. It does not consider the network dynamics, and there is no resource management. In addition, the authors in [21] explored the problem of server selection over multiple MEC servers. The major challenge arises from the correlation between the amounts of the offloaded computation and selected edge servers for various users. To cope with this issue, formulating and solving a congestion game was proposed to minimize the energy consumption of mobile users and edge servers. Furthermore, the Stackelberg game was used in [84] to design an optimal multilevel offloading scheme to maximize the utilities of vehicles and edge servers. A backup server in the vicinity is utilized to supplement insufficient edge server resources. If edge servers are located on dense roads, their constrained capacities may negatively impact the QoS of vehicular users.

2.4 Task migration

Due to the constrained capacity, it is needed for vehicular users to offload computation-intensive and delay-sensitive tasks to edge servers. Considering the dynamic environment and frequently changing topology, optimizing task migration decisions is crucial [43]. For smooth service migration in MEC, an efficient edge server selection algorithm is needed to select the optimal target edge server. In general, two factors should be taken into account: users' trajectory and QoS utility [72]. On the one hand, existing research works rarely explore users' trajectory data and the prediction of their movement and adopts a random mobility model instead [88]. However, users' mobility pattern (e.g., direction and velocity) has a significant influence on the construction of the candidate edge server set, and the users' trajectory data can be used to predict users' movement. On the other hand, existing literature pays less attention to the effect of QoS utility on the selection of edge servers in service migration and therefore hardly select the edge server with the highest QoS utility [29]. Without considering users' trajectory data and QoS utility, the accuracy of edge server selection and the efficiency of service migration decrease. The mobility of mobile users and the limited coverage of edge servers can result in significant network performance degradation, dramatic drop in QoS, and even interruption of ongoing edge services; therefore, it is difficult to ensure service continuity. Service migration has great potential to address the issues which decide when or where these services are migrated following user mobility and the changes in demand.

Task migration among different edge servers has gained increasing attention in existing literature [7, 19, 42, 69, 71]. Multiple edge servers may be considered for one task offloading. The computation migration problem was formulated as an MDP problem based on a random-walk mobility model in [71]. It was shown that the optimal policy has a threshold-based structure, i.e., select the migration only when two thresholds bind the distance of two servers. This work was further extended in [69] where the workload scheduling in edge servers was integrated with the service migration to minimize the average overall transmission and reconfiguration costs using Lyapunov optimization techniques. Another computation migration framework has been presented in [7], where the MEC server can either process offloaded computation tasks locally or migrate them to the central cloud servers. An optimization problem

was formulated to minimize the sum of mobile-energy consumption and computation latency. This problem was solved by a heuristic two-stage algorithm, which first determines the offloading decision for each user by semi-definite relaxation and randomization techniques and then performs resource allocation optimization for all the users. Tao *et al.* [42] have investigated the mobile edge service performance optimization problem and considered the cost of frequent migration. They used Lyapunov optimization to decompose long-term optimization problems into a series of real-time issues, while they did not require prior knowledge, such as user mobility.

As a mobile user moves from one area to another, we can 1) either continue to run the service on the current edge server, and exchange data with a mobile user through the core network or other edge servers, 2) or migrate the service to another edge server that covers the new area. In both two cases, the cost can be incurred [72]. When a user moves through several adjacent or overlapped geographical areas, service migration, should deal with: 1) whether the ongoing service should be migrated out of the current edge server that hosts this service; 2) if the the answer is yes, then which edge server the service should be migrated to; 3) how the service migration process should be carried out, considering the overhead and QoS requirements. The mathematical models, such as MDP, are applied to make efficient service migration decisions. Recently, artificial intelligence technology represented by deep learning and reinforcement learning is developing very fast and can help solve this complex problem [72].

2.5 Deep reinforcement learning in vehicular edge computing

Recently, artificial intelligence-based methods, especially deep learning provides new solutions to solve the resource allocation problems with low-complexity [40, 77]. In VEC networks, some essential features, including vehicular application requirements, network conditions, resource utilization, can be predicted and modeled conveniently by using deep learning methods. Moreover, the extracted features can be adopted to make resource optimization decisions. Therefore, how to efficiently predict and model the resource allocation problem to maximize the revenue is an open issue [26]. In addition, when each MEC can make decisions individually, the distributed deep learning

approaches can be exploited to optimize resources efficiently. In vehicular networks, data are naturally generated and stored across different units in the network, e.g., vehicles, roadside units, and remote clouds. This brings challenges to the applicability of most existing machine-learning algorithms that have been developed under the assumption that data are centrally controlled and easily accessible. As a result, distributed learning methods are desired in vehicular networks that act on partially observed data and have the ability to exploit information obtained from other entities in the network [77].

In order to tackle the resource allocation challenge in vehicular networks, a promising method is for the intelligent agents to leverage the techniques in the field of artificial intelligence, especially reinforcement learning (RL) and deep reinforcement learning (DRL), for decision making. RL has been successfully applied to a variety of domains, it confronts the main challenge when tackling problems with real-world complexity, i.e., the agents must efficiently represent the state of the environment from high-dimensional data, and use this information to learn optimal policies. Therefore, DRL, in which RL is assisted with deep learning (DL), has been developed to overcome the challenge [35]. There are two common DRL algorithms: Deep Q-Learning (DQL) and Deep Deterministic Policy Gradient (DDPG). DQL is a powerful tool for obtaining 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 techniques are applied to stabilize the learning process. However, the method does not work in the network with continuous action space. A summary of classification of central and distributed DRL is provided in Table 2.1. In addition, we classify some existing literature for central and distributed DRL in VEC as the following subsections.

2.5.1 Central deep reinforcement learning in VEC

There have been several works to apply central DRL to task offloading and resource allocation in VEC [11, 22, 24, 44, 86]. A two-level approach to managing the resource allocation of resources in a cloud environment is detailed in [44], with reinforcement learning employing an autoencoder neural network at the global scale, along with a Long Short Term Memory (LSTM) neural network at the local level for workload prediction. In [22], the vehicle's request for video-concerned contents was in the base

Table 2.1: Comparison of DRL algorithms in VEC

Reference	Action type	Central	Distributed	DRL algorithm	Agent location
[22]	discrete	✓	-	DQL	mobile virtual network operator
[44]	discrete	✓	-	DQL	a server cluster
[24]	discrete	✓	-	DQL	a base station
[11]	continuous	✓	-	DDPG	a base station
[86]	discrete	✓	-	DQL	a base station
[46]	discrete	✓	-	DQL	VEC operator
[39]	discrete	-	✓	DQL	V2V agent
[10]	discrete	-	✓	DQL	UAVs
[38]	discrete	✓	-	DDPG	a global controller
[58]	continuous	✓	-	DDPG	a controller
[67]	discrete	-	✓	double-DQN dueling-DQL	mobile devices
[57]	continuous	-	✓	DDPG	MECs

station's cache, or it was retrieved from the Internet. A resource allocation strategy for connected cars using joint networking, caching, and computing was formulated as an optimization problem. However, the mutual consideration of these three factors increases the complexity of the problem. Thus, a deep reinforcement learning method was introduced to solve the optimization problem. In [24], the mutual communication, computation, and caching issues were studied. The resource scheduling was designed by taking into account the mobility of vehicles, and the communication models were vehicle-to-vehicle and vehicle-to-RSU. The authors have proposed a coded caching scheme; each content is encoded into multiple segments, which can be cached in the local vehicle storage or the RSU storage. The deep learning algorithm was utilized to solve the resource allocation problem. In addition, in [24], the vehicle's mobility was modeled by discrete random jumps, and the numbers of contacts between vehicles and RSU follow the Poisson distribution. In [11], by using the Deep Deterministic Policy Gradient (DDPG) algorithm, the authors dealt with joint edge computing and caching resource allocation problems. One of the components that constitute the functioning of this algorithm is replay memory which stores the network experiences. The authors of [86] designed a deep Q-learning approach to make optimal offloading decisions by taking into account the selection of target servers and the determination of data transmission strategies simultaneously in the LTE-V network. The authors used the

vehicle-to-base station, vehicle-to-vehicle, and vehicle-to-RSU communication modes. In [38], the authors proposed a collaborative edge computing framework developed to reduce the computing service latency and improve service reliability for vehicular networks. They adopted DDPG to find the optimal solution in a complex urban transportation network. In their problem, many zones lead to the high dimension in both state and action spaces. Although the action space is discrete, DDPG tackles the problem with a high action dimension by the policy gradient technique.

2.5.2 Distributed deep reinforcement learning in VEC

The distributed approach is more sensible than the centralized approach for resource allocation and task offloading in VEC. Because of the nature of the vehicular network, it is needed to make a local decision immediately. For example, if an accident happens on the road, the vehicle around that area needs to be announced to avoid traffic jams or other alternative accidents. When the single agent of DRL is deployed in a central cloud, it needs to make a global decision with high complexity, processing too much other information coming from the network. However, in the cases that every MEC has its own agent that can make an immediate decision independently with low complexity. With the distributed-based approach, each node can make the decision individually, and the optimization problem can be divided into some subproblems [26].

Designing a distributed algorithm is challenging because there is no prior knowledge about workloads at MECs [67]. To address this challenge, Peng *et al.* [57] have studied multi-dimensional resource management in the MEC- and UAV-assisted vehicular networks utilizing multi-agent RL. Liang *et al.* [39] developed a distributed resource sharing scheme based on multi-agent RL for vehicular networks with multiple V2V links reusing the spectrum of V2I links. Cui *et al.* [10] proposed a stochastic game formulation for the dynamic resource allocation problem of the considered multi-UAV networks for maximizing the expected rewards, where each UAV becomes a learning agent, and each resource allocation solution corresponds to an action taken by the UAVs. The works [57], [39], [10] distributed algorithms were proposed while there was no workload prediction in edge servers.

2.6 Workload prediction and resource provisioning

In vehicular edge computing, the number of concurrent tasks offloaded to the MEC servers may change quickly. Thus, investigating the workload in resource allocation and task offloading algorithms in such a network is crucial. The workload of vehicular applications in VEC changes continuously based on the vehicle requests, and insufficient resource allocation to the application leads to the QoS dropping, loss of safety, and maybe cases of danger. On the other side, allocating unnecessary resources to the application can lead to wastage of cost and energy to maintain the resources. This issue can be solved with the prediction methods, which can predict the future workload of VEC applications in terms of needed resources and allocate those resources in advance, and release the resources when they are not required.

It is significant to proactively manage and allocate resources concerning workload dynamics to satisfy user expectations, especially for crucial tasks. This leads to the demand for a mechanism that is capable of predicting workload variations in MEC servers [52]. Zhang *et al.* [82] introduced a predictive-model transmission strategy for task offloading, improving the transmission efficiency, and satisfying the required delay. Neto *et al.* [51] proposed an estimation-based method, where each device makes its offloading decision based on the estimated processing and transmission capacities. By considering the requirements of computation tasks and the mobility of vehicles, a predictive-model transmission strategy is introduced in [82] for task offloading, improving the transmission efficiency and satisfying the required delay. Besides, an optimal offloading scheme based on prediction is designed for serving diverse types of computation tasks. Workload analysis and prediction have recently become an important research topic, as testified by the significant body of literature and by the presence of a few surveys covering aspects of this field. As outlined in [53], reliable resource provisioning for edge-cloud applications is a complex problem, especially when it is examined in a multi-tenant edge-cloud environment where the infrastructure is utilized to host numerous applications/services owned by different service providers. Each application typically has its own set of requirements, and there is a high possibility that controlling operations or tuning the performance of one application will have some impact on the others.

The newest proposed approaches are based on machine learning techniques. The

machine learning-based methods predict the application behavior in different dimensions. Neural Networks are known to perform well on nonlinear tasks. Because of its versatility due to the large dimension of parameters, and the use of nonlinear activation functions in each layer, the model can adapt to nonlinear trends in the data [54]. Neural network models, e.g., Long Short-Term Memory (LSTM), provide more accurate predictions on average than classical regressive models [17]. LSTM is a special kind of recurrent neural network which makes use of sequential observations and learns from the prior stages to figure future patterns with additional features to memorize the sequence of information [15].

D. Janardhanan *et al.* introduces a hybrid model, LSTM and Auto-Regressive Integrated Moving Average (ARIMA) model [25] for CPU workload prediction. In [31], the authors indicate that the LSTM model could solve the issues faced by cloud systems, as it is fragile and costly in the event of problems such as dynamic scaling of resources and energy consumption. The authors state that if it is possible to determine the precise future workload of a server, resources can be adjusted according to demand and thus maintain both qualities of service and reduce energy consumption. An LSTM network is adopted to predict the moving direction of vehicles [90]. Then, based on Markov decision processes, the optimal caching resource allocation problem is formulated to maximize the reward. Finally, a deep Q-learning-based algorithm is used to solve this problem.

Mainly two methods are used for time series forecasting, univariate and multivariate. In univariate time-series forecasting, there are only two variables: time and the forecasted parameter, depending on time. Multivariate time-series forecasting contains multiple variables, which are time and various parameters that influence the forecasted parameter [8]. In some papers, such as Liang *et al.* [41]; Yang *et al.* [76], the application workload is equivalent to the number of application requests. In this case, the future number of application requests is the output of the prediction methods.

2.7 Summary

Table 2.2 presents an overview of all studies discussed in this chapter. As explained in this chapter, there have been a number of studies on resource allocation and management [12, 17, 18, 37, 79, 80, 87]. Most of this research did not utilize intelligent

techniques, which are decisive for highly dynamic and complex networks. Moreover, various optimization tools were proposed to solve resource allocation, and DRL was more proper to solve resource allocation in the VEC network. As mentioned in this chapter, we can summarize that in the works [12,14,16,45,63,68,81–83], the dynamic of the network with an unknown load (i.e., the number of arriving tasks) is still challenging. There was no cooperation with the central cloud to satisfy the functions with high resource demands.

In addition, in the works [11, 22, 24, 38, 44, 46, 58, 86], DRL algorithms for MEC systems have been proposed. At the same time, the authors focused on centralized resource allocation and offloading algorithms, and there is no workload consideration in those works. In addition, for learning methodology in the works [10,39,57,67], distributed algorithms were proposed while there was no prior knowledge about workload in edge servers. Furthermore, in works [25, 31, 41, 51, 52, 54, 76, 82, 90] prediction and resource provisioning were considered; however, most of them are not beneficial for real-world VEC. Therefore, more realistic methodologies are needed to improve resource allocation and offloading.

Table 2.2: Overview of the related work in the literature

Topic	**Reference**
Resource allocation	[12, 17, 18, 37, 79, 80, 87]
Optimization tools	[6, 26, 40, 66]
Edge-based offloading	[12, 14, 16, 45, 63, 68, 81–83]
Edge and cloud offloading	[21, 84, 89]
Task migration	[7, 19, 29, 42, 69, 71, 72, 88]
DL in VEC	[40, 77]
Central DRL in VEC	[11, 22, 24, 38, 44, 46, 58, 86]
Distributed DRL in VEC	[10, 39, 57, 67]
Workload prediction	[25, 31, 41, 51, 52, 54, 76, 82, 90]

Chapter 3

Efficient task offloading via deep reinforcement learning

Computation and storage capacities in vehicular edge computing networks are significant factor to satisfy vehicular applications demands. One of the barriers is the limited computation and storage capacities in the MECs which cannot quickly satisfy the increasing resource demands of such applications. Another considerable problem in vehicular edge computing is deciding which MEC servers should accept the offloaded application tasks and process them. Service migration schemes have been introduced in several studies [19,80] to support reliable computing services. However, they did not consider a high dynamic network, the high volume of task migration, and limited resources in MEC for a considerable number of arriving application tasks. Several studies [21,84,89] considered cooperation among local clouds and remote clouds. However, these works do not address the problem of migration in application tasks.

In this chapter, we present a cooperation framework between MEC and central clouds, shown in Fig. 3.1. The offloaded task to the edge servers are migrated between edge and central clouds according to the tasks requirements and the available resources in the edge clouds. Vehicles, including traditional manually driven cars and autonomous vehicles, can access the resources at the MEC server through either cellular or Dedicated Short Range Communications (DSRC) technologies. MEC servers are connected to the central cloud in a wired manner. Our offloading problem is formulated as an optimization problem, an NP-hard problem [70]. Such NP-hard problems are difficult to solve via conventional mathematical techniques near-real-time

and within a polynomial time. [74], [75]. Moreover, in our vehicular edge computing network, task requirements and resource capabilities' MEC servers are time-varying; most of the existing approaches need to solve such a problem very frequently to obtain the optimal or suboptimal offloading strategy. However, this introduces significant computational overhead to vehicular edge computing, and it can hardly get the optimal offloading solution in real-time [74].

Due to the time-varying of task requirements and resource capabilities in MEC servers in the dynamic nature of vehicular networks, we formulate dynamic offloading and resource allocation as finite Markov Decision Process (MDP) involving state and action spaces and only depending on the current state and action. MDP is a prominent part of deep reinforcement learning, and both depend on current state space and action. In particular, deep reinforcement learning is deployed at the central cloud to indicate the optimal offloading decision of the vehicles' applications and the computation resource allocation with high accuracy in near-real-time [75]. In this vein, optimum task offloading using deep reinforcement learning could ensure the system performance measured by latency and acceptance rate. In addition, as one of the powerful decision-making algorithms in the artificial intelligence field, deep reinforcement learning can achieve excellent performance and effectiveness in tackling the optimization under dynamic environments [36].

Here, task resource requirements, access delay, queuing delay, migration delay, and delays resulting from execution are considered, which are neglected in most previous works. To the best of our knowledge, this is the first attempt to jointly optimize the task offloading decision and computational resource allocation using deep reinforcement learning for a highly dynamic vehicular edge computing environment. Although offloading in edge computing is well studied and reinforcement learning is well known, our novelty is to propose a feasible solution for the dynamic nature of vehicular networks. We apply deep reinforcement learning to solve dynamic, and time-varying task offloading and resource allocation optimization problems to gain high QoS and a high acceptance rate of application tasks. The main contributions of this chapter are as follows:

- We propose an efficient collaboration among MEC and central cloud for task offloading and task migration in vehicular networks to provide a high acceptance rate and efficient resource management.

- We design a task migration among local and remote servers based on resource restrictions and vehicles' QoS requirements. Our strategy respects the acceptable response time of each application according to a proper server selection and task placement strategy.
- We adopt a machine-learning approach, which utilizes a model-free method to find the optimal offloading task scheme and maximize vehicular application task's acceptance rate. We also develop a deep reinforcement learning process to deal with large network state space, real-time network state transitions, and massive amounts of arriving applications.

3.1 System model

In this section, we first describe the system architecture of vehicular edge computing in different domains and the cooperation of the domains, then explain the system's operation, including cooperation of the request and infrastructure, latency model, and the objective function.

3.1.1 System architecture

Fig. 3.1 illustrates the architecture of vehicular edge computing network in the following domains:

- Vehicle domain: Let $\mathcal{M}(t) = \{1, \ldots, M\}$ be the set of total vehicles in the network at time slot t indexed by m. The total number of vehicles at each time slot can change. Each vehicle can request one task (application) at each time slot. Notably, the types of vehicular tasks are different.
- MEC domain: Let $\mathcal{N} = \{1, \ldots, N\}$ be the set of total MEC servers in the network indexed by n. Each MEC server is located beside one RSU on the road. These edge computing servers suffer from limited resources, such as computing, storage, and buffer but satisfy communications quality because they are close to the vehicles.

- Central domain: Let v stand for the central cloud, which is remotely located. Central cloud has sufficient cloud resources but a considerable end-to-end communications delay. Some of the tasks can be executed on the central cloud according to priority and QoS requirements of vehicles' applications and the lack of sufficient resources in MECs[1].

Based on these domains, we consider a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of MEC servers plus the central server v. Furthermore, $\mathcal{E}$ is the set of links between MECs and central cloud. Each node $i \in \mathcal{V}$ has own resource capacity as $R_i = [R_i^{\text{comp}}$ $R_i^{\text{stor}}]$ where R_i^{comp} and R_i^{stor} shows the computational and storage capacity of node i, respectively and every server is capable of running one or more tasks, simultaneously. Furthermore, cooperation of vehicle, MEC, and central server are as follows:

- *Cooperation of Vehicle and MEC:* Vehicles can access the MEC servers through the wireless link. Thus, vehicles can offload their application tasks to the nearest RSUs. Application tasks can be hosted in the MEC server next to the corresponding RSU through cellular or DSRC technologies. Our system does not consider physical layer characteristics such as signal transmission and radio frequency. However, we investigate bandwidth and latency in the network layer. We assume there is sufficient bandwidth for the cooperation of vehicles and the relative MEC server.

- *Cooperation of MEC and Central Cloud:* To overcome the challenges caused by the limited resources in MEC servers and reduce response time for each vehicle, we apply the MEC servers connected in a wired manner. Our policy cooperatively schedules the resources in the MEC and central cloud. Thus, each MEC server directly connects to the central server in a wired link. Furthermore, the bandwidth deficiency in wired connections is neglected in this thesis.

3.1.2 System's operation

Various vehicles' densities in the city or out of the city can be considered in our policy. The vehicles' direction, speed, and acceleration are not considered in this work. Being

[1]Mobility management and session functionality are done in the central cloud.

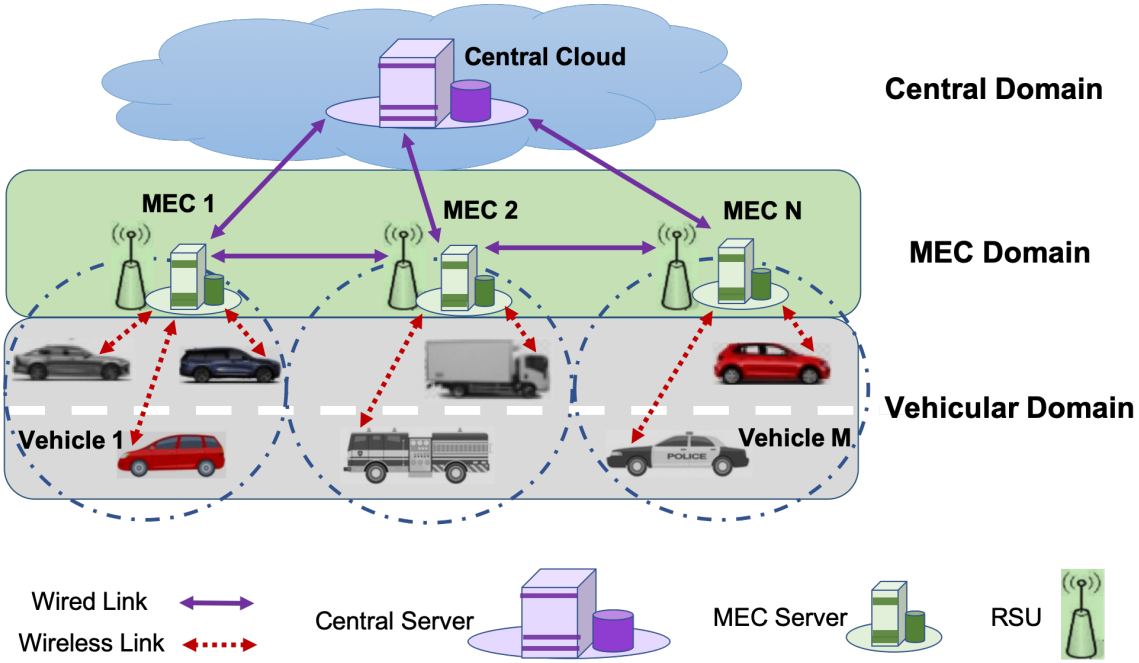


Figure 3.1: An illustration of vehicular edge computing network model

in a coverage area of one MEC server during offloading the requests and downloading the results is a significant factor in our system's vehicles. Our policy delivers the result of the application task to the exact vehicle. We assume a vehicle travels just in the coverage area of one MEC during uploading the request and downloading the result.

We assume that each vehicle can request one application task offloading during each time slot, and the vehicle will receive the result. Each application consists of a request's tolerable response time depicted by ϕ_m^{res}. Moreover, each task has requested resources of computing and storage; it can be denoted by: $\phi_m = [\phi_m^{\text{comp}}$ $\phi_m^{\text{stor}}]$, respectively. Vehicles can offload their application tasks to the nearest RSUs. Application tasks can be hosted in the MEC server next to the corresponding RSU. The task can be migrated to another MEC or central cloud to find the best trade-off between the response time and resource requirements. The considered parameters of this chapter are stated in Table 3.1. In the rest of this subsection, we describe handling requests by the infrastructure, latency model, and objective function. Below, we start with the first.

Table 3.1: Main notations used in this chapter

Notation	Definition
$\mathcal{M}$/M/m	Set/number/index of vehicles
$\mathcal{N}$/N/n	Set/number/index of MEC servers
$\mathcal{V}$/i	Set of all MEC servers plus central server v/index
R_i	Vector of resource capacity of server i
R_i^{comp}	Computation capacity of server i
R_i^{stor}	Store capacity of server i
ϕ_m	Vector of application resource demand of vehicle m
ϕ_m^{comp}	Application computation demand of vehicle m
ϕ_m^{stor}	Application storage demand of vehicle m
ϕ_m^{res}	Request's tolerable response time of vehicle m
τ_m^{access}	Access latency of application m
τ_m^{mig}	Migration latency of application m
τ_m^{proc}	Processing latency of application m
τ_m^{queue}	Queuing latency of application m
Acc	Accepted rate
ξ_m^i	Indicator if application from vehicle m is executed on server i
ψ_m	Indicator the accepted request m by the network

Cooperation of the request and infrastructure

The requests are handled by infrastructure through the operation of the central management. To achieve high usage of resource capacities, we assume that a central orchestrator dynamically makes MEC/central cloud decisions for all the vehicular applications in a centralized manner. The system, which is located in the central cloud, follows the steps below:

- *Monitoring:* In the monitoring step, the orchestrator monitors network situations in central, MEC, and vehicular domains and provides resource usage information to the network. The central orchestrator knows how many server resources are available, how much bandwidth exists in these three domains, how many application tasks are received by MEC servers, and where the vehicle's location aid is in the monitoring phase.

- *Acceptance/Rejection Decision:* When the offloading of the task is complete, the central orchestrator has various options to allocate the request to a proper server by considering the network's condition and the resource requirements of

the task. Thus, the orchestrator determines migration of the task to other MEC or the central cloud or executes in the exact MEC received the task. If there is not a proper server selection to satisfy the resource and QoS demands of the task, then the request will be rejected.

- *Allocation:* The optimal server selection occurs in the allocation phase, where a proper server, among central and MEC servers, is selected for every application task. The significant phase we work on in this chapter is the allocation phase. We utilize a central learning vision for optimal server selection for all application tasks. The machine learning method we adopt is deep reinforcement learning which will be considered later in this chapter.

- *Result Delivery:* After finishing the task computation, the result should migrate to the MEC next to the RSU, which covers the communication range of the vehicle. Based on our assumption, a car does not travel out of the coverage area of a MEC during task execution. In addition, the result should be delivered to the vehicle through the MEC.

Latency model

The total service latency for each application is significant to consider as an end-to-end QoS provision. Here, we are investigating migration, queuing, execution, and access delays during task completion. To clarify, we define two servers: base MEC n, the MEC server receiving requests directly from the vehicle and delivering the results directly to the car, and execution node i, where the task is executed. It is worth noting that these two servers could be the same based on optimization decisions.

- *Access Latency:* The time which is passed through the offloading application from the vehicle to the nearest RSU and downloading the result to the exact vehicle is τ_m^{access}. We assume that to send the request and deliver the result, each vehicle has to be in the coverage area of at least one RSU, and each RSU is located next to one MEC server. Access latency is mainly composed of transmission delay and round trip time (propagation delay). We consider transmission delay as the size of data d_m over the bandwidth of access link B_{acc}

allocated to the vehicle. To this end, access latency is as follows:

$$\tau_m^{\text{access}} = \frac{d_m}{B_{\text{acc}}} + RTT_{\text{acc}} \tag{3.1}$$

where RTT_{acc} is round trip time between vehicle and MEC.

- *Migration Latency:* Migration latency depends on the result of the optimization problem, as denoted by τ_m^{mig}. Each task hosted at base MEC is not necessarily executed on that MEC; it can also be migrated to the other proper node for processing and performance. Hence, migration latency would be equal to the sum of transmission delay and the round trip time delay between base MEC n and execution node i. Then, we can formulate migration delays as seen below

$$\tau_m^{\text{mig}} = \frac{d_m}{B_{ni}} + RTT_{ni} \tag{3.2}$$

where RTT_{ni} is round trip time between base MEC n and execution node i. In addition, B_{ni} is the bandwidth of network link.

- *Processing Latency:* The elapsed time of running each task on an execution node is considered as processing latency denoted by τ_m^{proc}. We assume that each task needs a specific number of CPU cycles, ϕ_m^{comp}, to run on the assigned node. From the physical resource perspective, we assume that each node can provide at most R_i^{comp} CPU cycles. Hence, without loss of generality, processing latency is obtained as follows [48]:

$$\tau_m^{\text{proc}} = \frac{\phi_m^{\text{comp}}}{R_i^{\text{comp}}}, \forall m \in \mathcal{M}(t), \forall i \in \mathcal{V} \tag{3.3}$$

- *Queuing Latency:* Queuing latency denoted as τ_m^{queue} is equal to the number of time slots for each task waiting in the queue of the execution node before starting the execution phase. After the task is offloaded to the dedicated server, it enters its queue. When the execution of other previous tasks in the same queue is completed, the waiting task can leave the queue and be processed. Binary variable $\beta_{m'}^i$ indicates if application task m' is in the buffer of server i prior the task m. Thus, queuing latency is the total time slots consumed for processing all prior tasks in front of task m in the queue. To this end, without

loss of generality, we can formulate queuing latency such as below

$$\tau_m^{\text{queue}} = \sum_{m' \in \mathcal{M}(t)} \frac{\phi_{m'}^{\text{comp}}}{R_i^{\text{comp}}} \cdot \beta_{m'}^i, \forall i \in \mathcal{V} \tag{3.4}$$

Objective function

It would be significant to find a proper server node to execute the application task to reduce the vehicle-perceived latency in vehicular edge computing networks. In other words, task placement in such a network requires considering the limited resources of MEC servers, which is different from central servers with large resource capacities. In this regard, our proposed resource management method provides short required response time applications to execute at the MEC. The applications with high resource demands and safe response time will be placed in the central cloud. In our model, each task is performed entirely using only one server based on (3.5). We define the binary decision variable ξ_m^i which is set to 1 if the application task from vehicle m is executed at server i; otherwise, it is 0. Therefore, we have the following constraint:

$$\sum_{i \in \mathcal{V}} \xi_m^i \leq 1, \forall m \in \mathcal{M}(t) \tag{3.5}$$

Moreover, to ensure that the sum of allocated resources to all vehicles in each server i, do not exceed its resource capabilities R_i, we have the following constraint:

$$\sum_{m \in \mathcal{M}(t)} \xi_m^i \cdot \phi_m \leq R_i, \forall i \in \mathcal{V} \tag{3.6}$$

In general, response time for a task from vehicle m and getting the result to the exact vehicle is the sum of the access, migration, queuing, and processing latency. This sum should satisfy an application's tolerable response time ϕ_m^{res}, so we consider the following constraint for that purpose

$$\tau_m^{\text{access}} + \tau_m^{\text{mig}} + \tau_m^{\text{queue}} + \tau_m^{\text{proc}} \leq \phi_m^{\text{res}} \tag{3.7}$$

Furthermore, we define ψ_m as a decision variable which is set to 1 if the application task from vehicle m is accepted; otherwise, it is 0. Thus, the total accepted applications divided into all arriving applications is considered the accepted rate Acc.

We aim to maximize the acceptance rate respecting response time constraints and resource restrictions. Hence, an optimization problem for resource management can be written as

$$
\begin{aligned}
& \max \quad \mathrm{Acc} = \frac{\sum_{m \in \mathcal{M}(t), i \in \mathcal{V}} \xi_m^i \cdot \psi_m}{\mathcal{M}(t)} \\
\text{s.t.} \quad & \mathrm{C1}: \quad \sum_{i \in \mathcal{V}} \xi_m^i \leq 1, \forall m \in \mathcal{M}(t), \\
& \mathrm{C2}: \quad \sum_{m \in \mathcal{M}(t)} \xi_m^i \cdot \phi_m \leq R_i, \forall i \in \mathcal{V}, \\
& \mathrm{C3}: \quad \tau_m^{\text{access}} + \tau_m^{\text{mig}} + \tau_m^{\text{queue}} + \tau_m^{\text{proc}} \leq \phi_m^{\text{res}}, \\
& \mathrm{C4}: \quad \xi_m^i \in \{0, 1\}, \forall m, i, \\
& \mathrm{C5}: \quad \psi_m \in \{0, 1\}, \forall m,
\end{aligned}
\tag{3.8}
$$

The proposed algorithm for the cooperation of MEC and central servers, including offloading, server selection, processing, and receiving, is presented in Algorithm 1. Based on the algorithm, when request m is offloaded by base MEC n, the resource demand of request m is compared with the available resource capability of base MEC n. If the application resource demand is less than the available resource capability of the server n, then the request m is executed there, and the execution node would be the same base MEC n. Otherwise, if the base MEC server does not have free resources to allocate the request m, server i will be selected based on the resource allocation strategy. The application will migrate there to execute. On line 10, if the execution node is not base n, the result migrates to the base, and from there, the result will be delivered to the vehicle m. Fig. 3.2 illustrates our task offloading operation.

3.2 Learning solution

Problem (3.8) is a mixed-integer linear optimization problem, and such problems are usually considered NP-hard problems [55, 70, 75]. Therefore, we utilize reinforcement learning to solve the formulated optimization problem. In this section, we first describe the implementation of reinforcement learning and then consider the deep reinforcement learning solution.

Algorithm 1 Task offloading algorithm

Vehicle m sends its request to the nearest RSU, during τ_m^{access}
Request m is offloaded by base MEC n
if $\phi_m < R_n(t)$ **then**
    execution node $= n$
else
    Select server i based on resource allocation optimization
    Migrate m to i, during τ_m^{mig}
    execution node $= i$
Execute task on execution node, during $\tau_m^{\text{queue}} + \tau_m^{\text{proc}}$
if execution node is not n **then**
    Migrate the result to n, during τ_m^{mig}
    Deliver the result to vehicle m, during τ_m^{access}
else
    Deliver the result to vehicle m, during τ_m^{access}

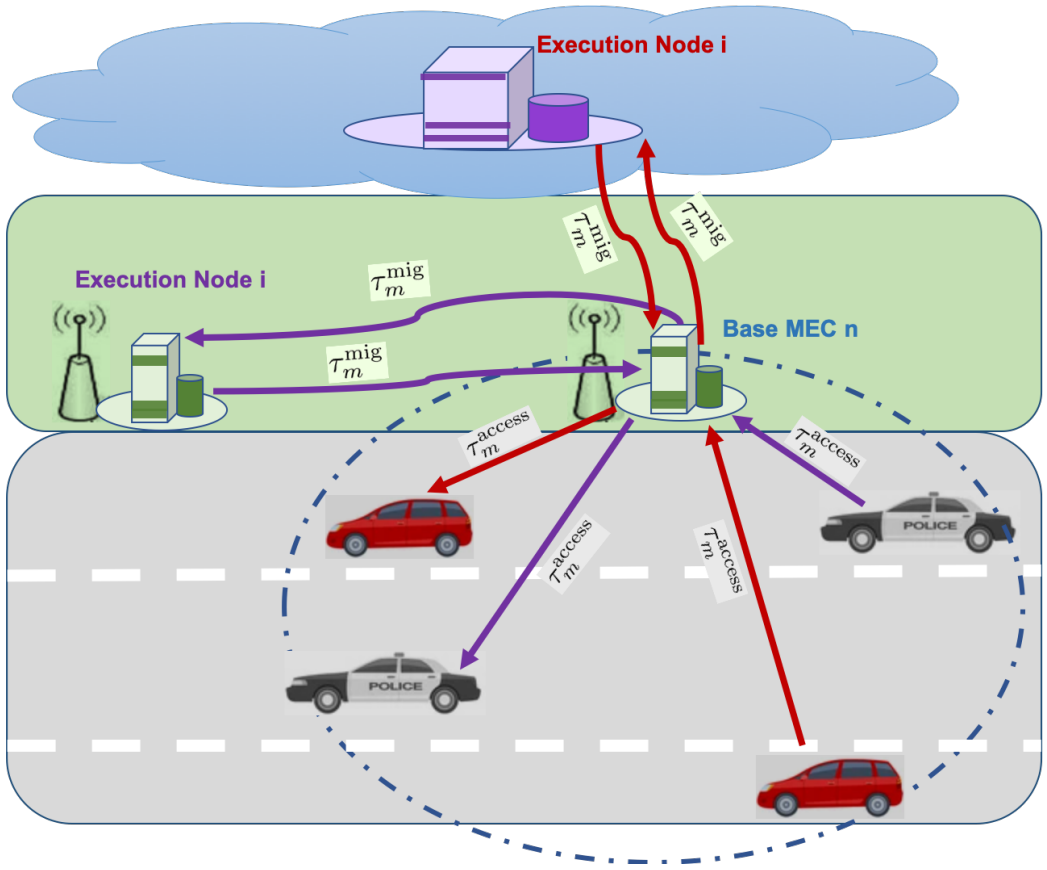


Figure 3.2: An illustration of task offloading algorithm

3.2.1 Reinforcement learning implementation

We use a finite MDP model to capture the dynamic of network state transitions. An MDP can be defined by the set of system states, set of actions, and the set of real-value reward functions [38]. In our problem, the state space $S(t)$, action space $a(t)$, and reward model $r(t)$ in an finite MDP are summarized as follows:

- *State Space:* In our learning solution, each state $S(t) \in \mathcal{S}$ is associated with $S(t) = \{\widehat{\phi}, \widehat{W}(t), \widehat{C}(t)\}$ as following vectors
  - $\widehat{\phi}$: Application information including its resource demands (computational and storage), tolerable response time, and the location of the vehicle.

$$\widehat{\phi} = \begin{bmatrix} \phi_m^{\text{comp}} & \phi_m^{\text{stor}} & \phi_m^{\text{res}} & \phi_m^{\text{loc}} \end{bmatrix} \tag{3.9}$$

  - $\widehat{W}(t)$: Application tasks waiting in the buffer of each server at time slot t with their resource demands including computation and storage ($i \in \mathcal{V}$).

$$\widehat{W}(t) = \begin{bmatrix} W_1^{\text{comp}}(t) & W_2^{\text{comp}}(t) & \dots & W_i^{\text{comp}}(t) \\ W_1^{\text{stor}}(t) & W_2^{\text{stor}}(t) & \dots & W_i^{\text{stor}}(t) \end{bmatrix} \tag{3.10}$$

  - $\widehat{C}(t)$: Running tasks in each servers at time slot t with all resource demands including computation and storage.

$$\widehat{C}(t) = \begin{bmatrix} C_1^{\text{comp}}(t) & C_2^{\text{comp}}(t) & \dots & C_i^{\text{comp}}(t) \\ C_1^{\text{stor}}(t) & C_2^{\text{stor}}(t) & \dots & C_i^{\text{stor}}(t) \end{bmatrix} \tag{3.11}$$

- *Action Space:* The action of the deep reinforcement learning agent is defined as a server selection for each application to be run on the selected server, and it is indexed by the number of MECs and central cloud. The current action is defined as $a_m(t) = \{a_m^1(t), a_m^2(t), \dots, a_m^i(t)\}$ where $a_m^i(t)$ is a decision variable which is set to 1 if requested application from vehicle m is placed on server i to be executed, otherwise, is 0 at time slot t. It is noting that $\sum_{i \in \mathcal{V}} a_m^i(t) = 1$ which means each request can perform and execute only in one server. The vector of action space for all application tasks to be assigned to a proper server

can be depicted as follows

$$a(t) = \begin{bmatrix} a_1^1(t) & a_1^2(t) & \dots & a_1^i(t) \\ a_2^1(t) & a_2^2(t) & \dots & a_2^i(t) \\ \vdots & \vdots & & \vdots \\ a_m^1(t) & a_m^2(t) & \dots & a_m^i(t) \end{bmatrix} \tag{3.12}$$

- *Action Selection:* The agent gets the state information from the environment. The action with the highest value determines the next state. In other words, to select the next state as a proper server for the execution of an application, our algorithm rescans all $Q(S(t), a(t))$ values. It captures them as the outcome of a deep Q-network to gain the maximum value. Moreover, the algorithm adopts the epsilon-greedy action selection method [50] to balance exploration and exploitation by choosing between them randomly.

- *Reward Definition:* Once the agent takes action based on the observed environment state, the environment will return an immediate reward to the agent. Then in the learning stage, the agent updates the resource allocation policy based on the received reward until the algorithm converges [58]. Indicated by Equation (3.13), the estimated response time $\tau_m^{\text{est}}(t)$ (the sum of the access, queuing, migration and execution latencies) is compared with tolerable response time ϕ_m^{res}. If the estimated response time is a less than tolerable response time for each vehicle's request, then the reward would be $-\tau_m^{\text{est}}(t)$ otherwise, the reward would be $-w \cdot \tau_m^{\text{est}}(t)$. Here, $-w$ is a large negative integer. Thus, to maximize the number of offloaded tasks that are completed with satisfying response time by the MEC server at time slot t, we define the following reward element for offloaded task m

$$r_m(t) = \begin{cases} -\tau_m^{\text{est}}(t), & \text{if } \tau_m^{\text{est}}(t) \leq \phi_m^{\text{res}} \\ \\ -w \cdot \tau_m^{\text{est}}(t), & \text{otherwise.} \end{cases} \tag{3.13}$$

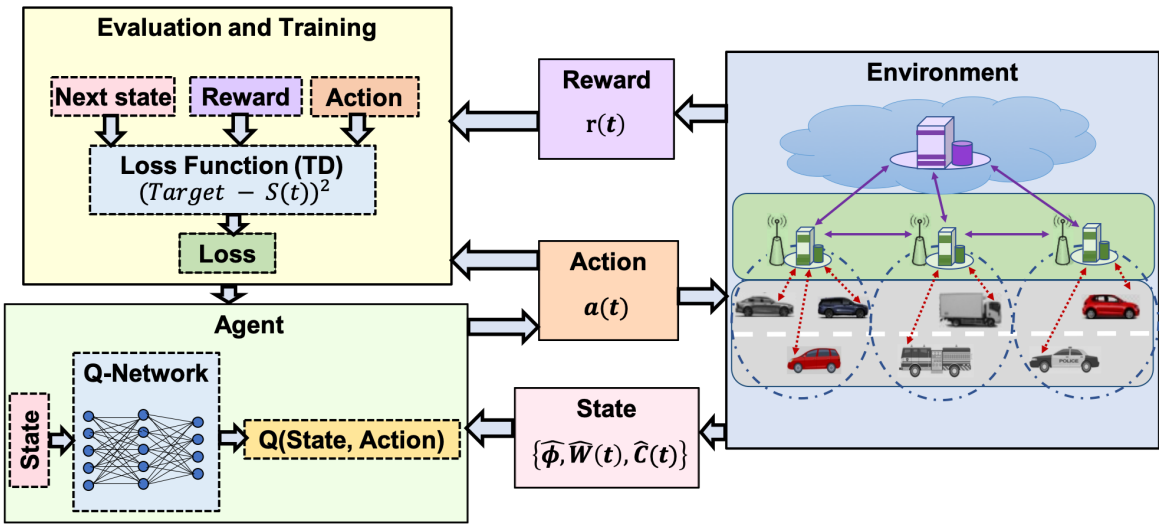


Figure 3.3: Proposed deep reinforcement learning

3.2.2 Deep reinforcement learning solution

Due to the limited knowledge on transition probability between the states and the sizeable state-action space in the network, the traditional dynamic programming cannot find the optimal policy efficiently. Therefore, we adopt DRL to solve the proposed server selection problem [38]. We adopt deep Q-learning as a value-based approach for various reasons. Firstly, the system has a discrete time and is considered every time slot. Secondly, all resources, such as computation capabilities, would be modeled on discrete values and divided into discrete levels. Thirdly, our learning solution has to choose only one server for each time slot as an outcome of deep learning. Fourthly, one of the purposes of using a deep Q-network is to maximize the normalized reward on the episodes. By taking advantage of a deep Q-network, the central orchestrator can manage the system in an effective and efficient way. Fig. 3.3 presents the proposed architecture of deep reinforcement learning. Here, Q-learning updates its Q-value by temporal difference. Temporal difference captures the difference between the current estimate and previous ones and approximates the estimation by comparing them at two consecutive episodes [50]. In a neural network, we use the Multi-Layer Perceptron algorithm as a function approximator for $Q(S(t), a(t))$. We also inject randomness into the approximation model by adding drop-out to the hidden layers of the Multi-Layer Perceptron network. This property increases the robustness and generalized capability of the network besides powering the exploration provided by

this randomness.

Algorithm 2 Deep Q-Learning (Temporal Difference method)

1: Initialize $\alpha, \gamma, \epsilon_0, \epsilon_{min} \in (0, 1)$
2: **for** episode $\leftarrow 1, K$ **do**
3: $\epsilon \leftarrow \max(\epsilon_{min}, \epsilon_0 - \text{episode}/K)$
4: request $\leftarrow$ Generate M requests from the dataset
5: Calculate $S(0)$ based on the first request
6: **for** $t \leftarrow 1, M$ **do**
7: Calculate $S(t)$ based on the request(t)
8: Predict $Q(S(t), a(t))$ from $S(t)$
9: Sample $p \sim$ U(0, 1)
10: **if** $p < \epsilon$ **then**
11: Sample $a(t) \sim$ U$\{0, \mathcal{V}\}$
12: **else**
13: $a(t) \leftarrow \arg\max_a Q(S(t), a(t))$
14: $S(t+1), r(t) \leftarrow$ emulator ($a(t)$, request$(t+1)$)
15: **if** $t < M$ **then**
16: target $\leftarrow r(t) + \gamma . \max Q(S(t+1), a(t))$
17: **if** $t = M$ **then**
18: target $\leftarrow r(t)$
19: Optimize Q-Network based on TD error $(\text{target} - S(t))^2$
20: $S(t) \leftarrow S(t+1)$

Our deep learning solution, depicted in Algorithm 2, repeats the procedure based on the number of episodes until the reward converges. In the first place, initialization occurs for $\alpha,\ \gamma,\ \epsilon_0,\ \epsilon_{min}$ which represent learning rate, discount factor, the first value of ϵ in epsilon-greedy, and a minimum value of the ϵ, respectively. Then, for each episode, M number of requests is generated from a real-world dataset, and the first state $S(0)$ is calculated based on the first request. Furthermore, the epsilon value is updated in each episode, which is used for the epsilon-greedy method. On line 6, the time slot t increases based on the number of requests, and the rest of the instructions occur in each time slot. On line 7, the state of the network is built according to the relevant time slot and the request. Then, $Q(S(t), a(t))$ is predicted based on the neural network and the function approximator. The state $S(t)$ is the input of the neural network. After that, the epsilon-greedy algorithm needs to sample the value of p in a continuous uniform distribution; if p is less than the epsilon, action selection occurs based on the discrete uniform distribution among all MEC servers and

the central server. Otherwise, the selected action is the action that has the highest value. On line 14, the action and subsequent request will be sent to the emulator where there is no decision process, then there is the reward, and the next state will be calculated. Finally, Q-learning updates its Q-value by temporal difference. The temporal difference method needs to find the target, calculated based on the equation on line 16. It is noted that if the request is the last one, then the target equals the reward.

The optimization problem is a mixed-integer linear programming problem, and achieving the optimal or sub-optimal solutions usually requires exponential time complexity. Moreover, the optimization procedure must be executed at each time slot due to the diversity of request requirements, the number of requests, and resource availability in the MEC server, which are time-varying and highly dynamic. Therefore, the conventional optimization methods using relaxation iteration algorithms incur high computational complexity due to numerical iterations, and their solutions are often sub-optimal. They would not scale well [64], usually converge slowly, and have prohibitive complexity for real-time implementations [74]. A machine learning-based approach is an effective and attractive solution to tackle this problem. Furthermore, Since the deep neural network of this chapter employs the full-connection networks, the computational complexity of each training step is $O(\sum_{j=1}^{J} L_{j-1}L_j)$, where L_j represents the neural size of the j-th layer among J layers [32, 36], and the complexity of Q-learning algorithm is $O(T)$, where T is the total number of training steps [27]. As a result, the total complexity of the deep Q-network algorithm in this chapter is $O(2T\sum_{j=1}^{J} L_{j-1}L_j)$.

3.3 Performance evaluation and results

This section presents the experimental settings by adopting a real-world dataset and introduces baselines for comparison. Then, we demonstrate the experimental results and discussions.

3.3.1 Experimental setting and baseline

We use real vehicle traffic flow to evaluate the performance of the proposed edge/central decision resource management approach in a vehicular network. Thus, we utilize a realistic vehicular dataset using mobility traces of taxi cabs in Rome, Italy [4]. It contains GPS coordinates of approximately 320 taxis collected over 30 days, from February 1st to March 2nd, 2014. Traces present the spatial positions of drivers and are collected every 7 seconds. The details of the dataset's content are explained in Table 3.2. The locations of the taxi traces vary from latitude 39.36 to latitude 51.45 and from longitude -0.14 to longitude 41.89.

Moreover, since MEC servers are located on the road close to vehicles, we divide the distance of a maximum and minimum range of vehicles' locations to the number of MEC servers to find the place of each MEC. Based on our policy, each cab's source and destination locations should be in the coverage area of one MEC. Thus, we randomly take a snapshot of each taxi's trajectory during a day using the dataset. In addition, to allocate a geographical location for the central server, we add a high value to the maximum value of latitude and longitude of taxis' locations.

Table 3.2: Dataset content

Parameter	**Value**
Type of vehicle	Taxi
Number of vehicles	320
Duration	30 Days
Year	2014 (Winter)
Location	City (Rome)
Driver ID	An integer number
Time Stamp	Date and Time
Vehicle position	Latitude, Longitude

Furthermore, to make the optimal presentation of our algorithm and comparison, we utilize three models as baselines. The first one, the tabular reinforcement learning, called RL, allocates tasks to a proper server. In the RL method, a large table is implemented to keep all state conditions, and we apply the quantization method to reduce the aspect of the state. The second baseline method is the Greedy method to allocate each task to a server. In the Greedy solution, tasks are assigned to servers with higher resource capabilities. Another baseline for comparison, called Random, means

that all arriving requests are assigned to each node randomly, and its distribution for task offloading and server selection is uniform.

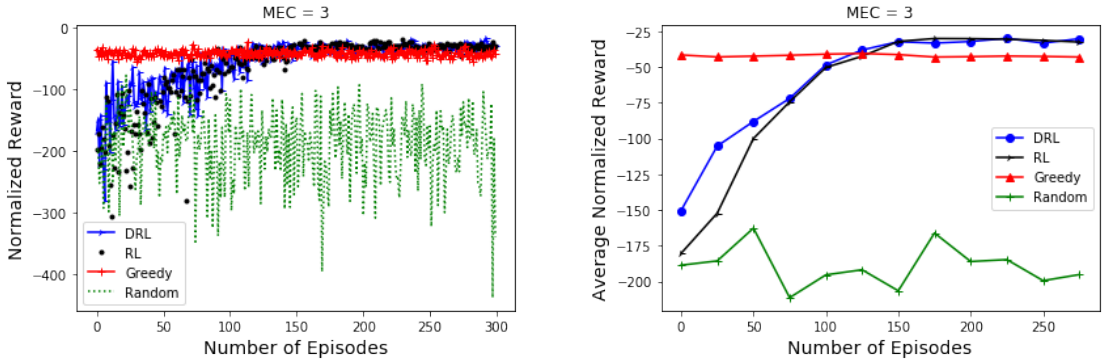
We use TensorFlow [1,58] to implement our proposed deep reinforcement learning. Moreover, we have used four layers of a fully-connected neural network with hidden layers of size 128; ReLU capacity [73] is an activation function added to related layers. This introduces non-linear features to the neural network and improves the deep neural network. Some of the parameters for the proposed resource management strategy and learning process are in Table 3.3. Some of the values are uniformly picked from intervals to show our model works with diverse values. Although our model works for various intervals, the selected intervals could properly determine our model's quality, validity, and capability.

Table 3.3: Parameters used in this chapter

Parameter	**Value**	**Description**
N	1, 3, 6, 9	Number of MEC servers
R_n^{comp}	[1, 9]	CPU cycles per second
R_n^{stor}	[15, 70]	Storage of MEC (Mbits)
R_n^{buf}	[500, 1500]	Buffer of MEC
R_n^u	[2, 10]	Number of cores in MEC
R_v^{comp}	100	CPU cycles per second of central cloud
R_v^{stor}	100	Storage of central cloud (Mbits)
R_v^u	100	Number of cores in central cloud
ϕ_m^{comp}	[100, 260]	CPU cycles' task demand
ϕ_m^{stor}	[3,10]	Storage's task demand
K	300	Number of episodes
ϵ_{min}	0.05	Min value of ϵ
ϵ_0	0.9	First value of ϵ in $\epsilon - greedy$
α	0.0001	Learning rate
γ	0.98	Discount factor

3.3.2 Experimental result and discussion

Implementation results are presented in this subsection to demonstrate the performance of the proposed deep reinforcement learning-based resource management schemes for the vehicular scenarios with MEC and central servers. It is noted that most of the results are captured through 300 episodes. In other words, there are many

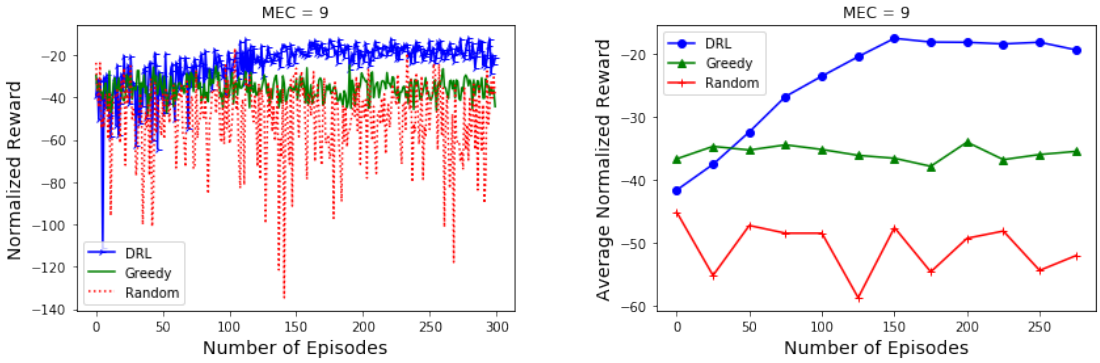


(a) (b)

Figure 3.4: (a) Normalized reward, (b) Average normalized reward per episode in DRL, RL, Greedy, and Random solution for one central server and 3 MECs.

numbers to be presented in one figure. Thus, to improve illustration, we also show the results in the average scope.

Fig. 3.4 shows the convergence performance of the deep reinforcement learning algorithm and other baselines. From a vehicle perspective, satisfying delays and QoS requirements are critical. As we can see from Fig. 3.4, in DRL and RL methods, among the 300 episodes during the learning stage, the total rewards per episode fluctuate sharply and are relatively small in the first 150 episodes and then tend to meet a relatively stable and high value. The learning process starts by updating the parameters of the deep Q-learning. Thus, the total rewards per episode fluctuate sharply at the beginning of the learning process and then increase as the parameters gradually optimize. Moreover, the reward for each task offloading is achieved by Equation (3.13). Fig. 3.4 presents convergence reward for one central server and three MECs. To apply the RL method, we cannot add more MEC servers because tabular RL solution suffers from lack of scalability and it does not work for large settings. Fig. 3.4 demonstrates the reward convergence of the Greedy model for task allocation. Each task is allocated to a server with high resource capacity in the Greedy solution. Like the Random model, Greedy does not follow any learning policy, so there is no improvement about normalized reward values by increasing the number of episodes. However, Greedy presents high reward values as a near-optimal method. Moreover, there is no learning policy to support the convergence of reward in Random strategy; the rewards fluctuate after the total episode. As we can see, this uniformly distributed random model suffers from a small value of rewards, which means it cannot support



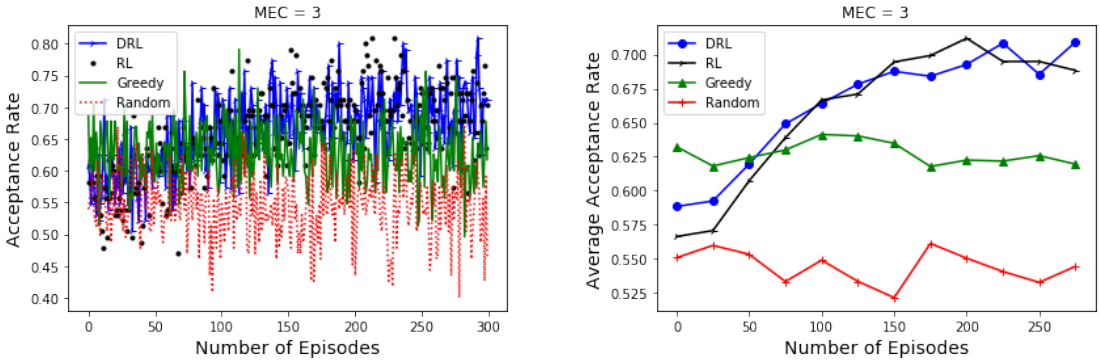
(a) (b)

Figure 3.5: (a) Normalized reward, (b) Average normalized reward per episode in DRL, Greedy, and Random solution for one central server and 9 MECs.

the tolerable delays requested by each task.

Fig. 3.5 demonstrates the convergence performance of DRL, Greedy, and Random solutions for large settings. As we can see, in the DRL method, among the 300 episodes during the learning stage, the total rewards per episode fluctuate sharply and are relatively small in the first 150 episodes and then tend to meet a relatively stable and high value. While Greedy and Random methods do not follow any learning policy, there is no improvement about normalized reward values by increasing the number of episodes. We can observe from Fig. 3.5 that normalized rewards of DRL, Random, and greedy are about -18, -38, and -50, respectively. By following this picture, we can find that the normalized reward of the proposed strategy converges at different episodes. Specifically, reward roughly increases from -42 to -18, converging after the 150th episode. The rewards of Greedy and Random algorithms have no convergence, and they are around -38 and -50, respectively, from the first to 300th episodes. By analyzing this picture, we can obtain a result that the proposed solution based on deep reinforcement learning for vehicular application tasks is efficient and flexible and can converge to a higher value.

Fig. 3.6 represents the acceptance rate and the average acceptance rate of vehicular applications per episode. From the central orchestrator perspective, an efficient resource management scheme should accept as many tasks as possible with the given available resources. As we can see in this figure, the acceptance ratio increases in DRL and RL strategies at the beginning of the learning process. However, after around 150 episodes, the acceptance rate gains a stable value and a higher acceptance rate

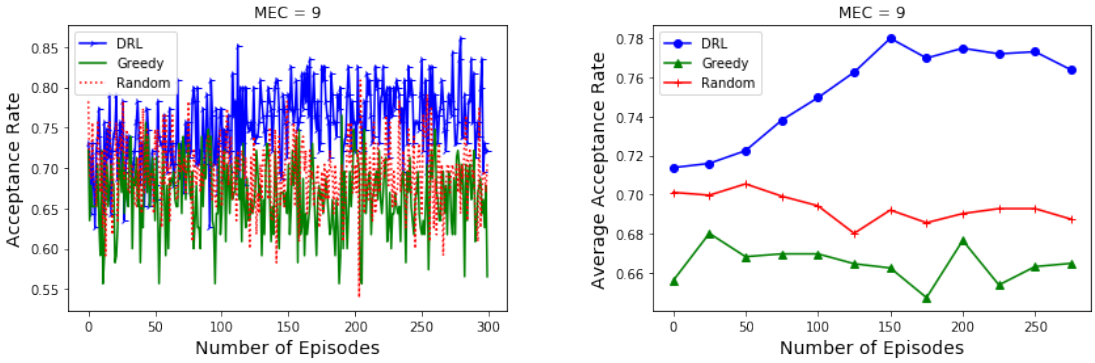


(a) (b)

Figure 3.6: (a) Acceptance ratio, (b) Average acceptance ratio per episode in DRL, RL, Greedy, and Random solutions for one central server and 3 MEC servers.

compared to other baselines. We can find from Fig. 3.6 that our proposed learning and RL solutions are learning how to allocate tasks to proper servers based on available resources while satisfying the task's delay demands. It is worth noting that for a small setting such as three MEC servers RL method is as well as the DRL method. Fig. 3.6 demonstrates that in the Random strategy, there is no improvement even by time passing and increasing the number of episodes. Moreover, the Greedy method has no stable pattern among the first steps of simulation and its last steps. It is worth mentioning that the Greedy model does better than the Random strategy mainly because it makes a difference between servers and gives high priority to servers with high resource abilities.

Fig. 3.7 shows the acceptance rate and the average acceptance rate of vehicular applications per episode for large settings. The RL method does not work in large environments, and the DRL strategy can be a good solution. A large table is implemented in the RL method to keep all state conditions. Tabular RL solution suffers from a lack of scalability, and it does not work for large settings. However, implemented neural networks in DRL can address an abundant space. We can observe from Fig. 3.7 that as the number of episodes increases, the average acceptance rate increases accordingly. This is because increasing episodes can help the DRL algorithm learn effectively. Moreover, the achieved average acceptance rate of both Greedy and Random algorithms is lower than the proposed strategy. In addition, the average acceptance rate of Greedy is the lowest at different values of the episode, followed by the Random methodology, while the proposed strategy is the highest. For example, when

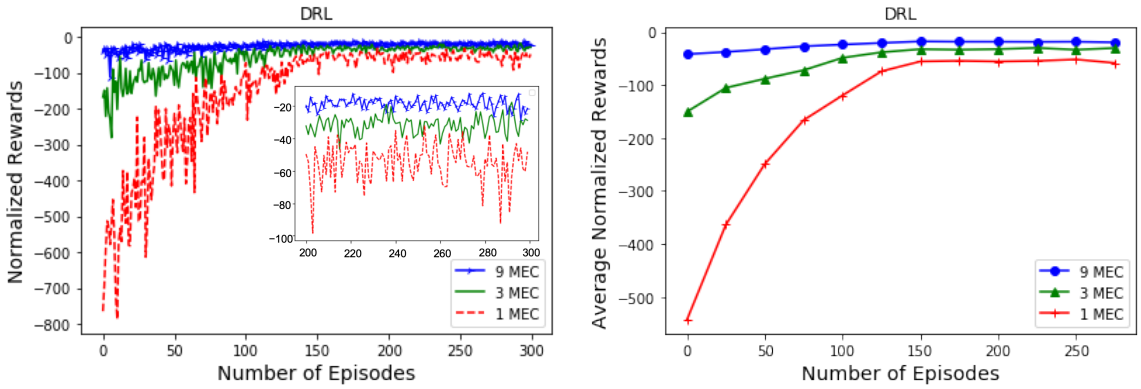


(a) (b)

Figure 3.7: (a) Acceptance ratio, (b) Average acceptance ratio per episode in DRL, Greedy, and Random solutions for one central server and 9 MEC servers

the number of episodes = 50, the DRL is about 2% higher than the Random method is about 5% higher than greedy. When the number of episodes = 300, the DRL is about 10% higher than the Random method is about 13% higher than greedy. This is because the proposed deep Q-Network strategy can determine the optimum resource allocation to achieve the highest acceptance rate. These results further verify that the proposed approach can find a suitable offloading strategy with a shorter response time. Thus, the vehicular networks can provide a flexible way of measuring latency and acceptance rate. The proposed approach can adapt to various dynamic situations and give an optimal solution for resource allocation strategy.

Fig. 3.8 demonstrates the convergence performance of the deep reinforcement learning algorithm for three different settings. The number of MEC servers plays a crucial role in vehicular edge computing networks. As the number of MEC servers increases, QoS satisfaction increases. As it is apparent in Fig. 3.8, DRL represents a higher convergence performance with 9 MEC servers. However, the normalized reward value is small during small settings. We can observe from Fig. 3.8 that normalized rewards of the proposed solution in one MEC, 3 MEC, and 9 MEC servers are about -60, -38, and -20 after the 200th episode. By following this figure, we can see during the first episode to the 20th episode, the value of normalized reward has started from -500, -150, and -58 for one MEC, 3 MEC, and 9 MEC servers scales, respectively. By analyzing this picture, we can obtain the result that the proposed solution based on deep reinforcement learning for vehicular edge networks works more efficiently and flexibly in a large-scale environment.

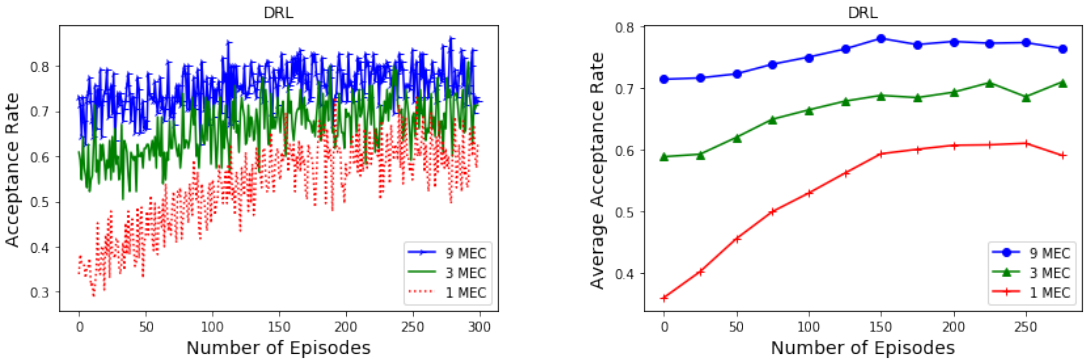


(a) (b)

Figure 3.8: (a) Normalized reward, (b) Average normalized reward per episode in DRL solution for one central server and various number of MECs

Fig. 3.9 illustrates the acceptance rate of vehicular applications per episode for various settings. The number of MEC servers significantly affects QoS and resource allocation strategy in the vehicular edge computing network. An efficient resource management scheme should accept as many tasks as possible with available resources. As we can see in this figure, in the large settings (9 MEC servers), the acceptance rate of vehicular application is around 70 to 80 percent. At the same time, this value decreases with a small number of MEC servers in the vehicular system model. We can observe from Fig. 3.9 that during the first episode to the 20th episode, the value of the acceptance rate has started from 32%, 58%, and 71% for one MEC, 3 MEC, and 9 MEC servers scales, respectively. By following this figure, we can see that the proposed solution converges by increasing the number of episodes. The acceptance rate of the proposed solution in one MEC, 3 MEC, and 9 MEC servers is about 59%, 69%, and 78%, respectively, after the 150th episode. As a whole, we can obtain that the proposed deep reinforcement learning solution for task offloading and resource allocation gain better performance in large-scale settings.

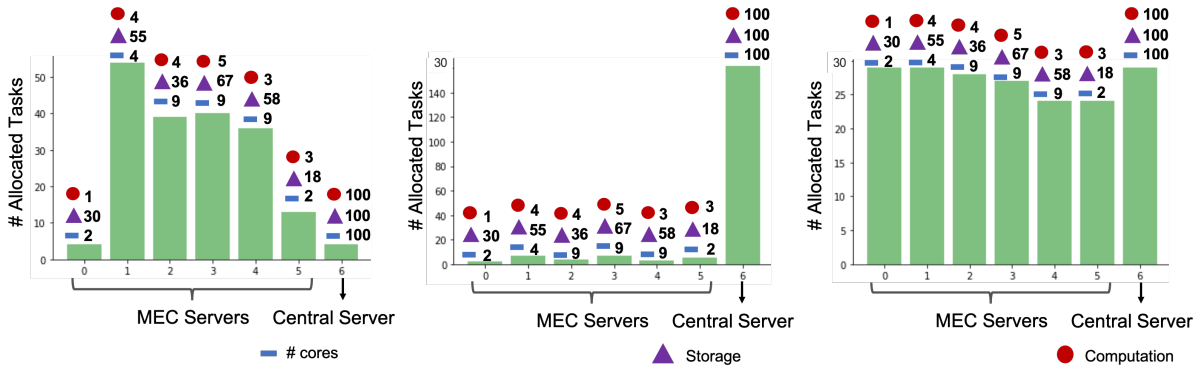
Fig. 3.10 demonstrates task distribution based on server resources. Each number on top of every bar chart reveals the resource ability of the relevant server. The resources are computational capacity and the CPU cycle per second, storage (Mbits) in each server, and the number of cores a server has, which denotes how many tasks each server can execute simultaneously. The main point is depicted using green bars in this figure. The value of each green bar is the sum of the actions, which shows the number of application tasks assigned to each server. As demonstrated in Fig. 3.10 (a), the sum



(a) (b)

Figure 3.9: (a) Acceptance rate, (b) Average acceptance rate per episode in DRL solution for one central server and various number of MECs.

of application tasks assigned to servers is based on the result of our proposed learning resource management solution, which considers total response time and the resource demands of tasks. Moreover, based on the figure, the resource availability of servers is investigated for action selection. Furthermore, as Fig. 3.10 (b) shows, the central server with significant resource ability receives more tasks using the Greedy strategy. Nevertheless, as Fig. 3.10 (c) illustrates, the tasks are distributed uniformly among servers, and the Random method does not investigate QoS demands. Our proposed DRL solution plays well considering its resource capabilities and QoS demands as a final discussion point. Furthermore, two other strategies work as acceptable strategies.



(a) DRL (b) Greedy (c) Random

Figure 3.10: (a) Task distribution based on available resources in DRL solution. (b) Task distribution based on available resources in Greedy solution. (c) Task distribution based on available resources in Random solution.

3.4 Summary

In this chapter, we have considered multi-access edge computing in vehicular networks. We have proposed a task offloading with a resource allocation strategy based on task migration among MEC and central cloud servers. Simulation results showed our proposed approach could provide an efficient and high-reward task offloading with high acceptance rates for arriving requests from vehicles, considering their locations, response times, resource demands, and the current resource utilization. With the help of deep reinforcement learning, we could efficiently handle a large number of real-time arriving requests, especially in a large and dynamic transferred network state space based on a realistic vehicle dataset. Moreover, we could specify QoS constraints in deep reinforcement learning and solve an NP-hard problem instead of mathematical approaches that lead to a non-polynomial time solution.

Chapter 4

Computation re-allocation and dynamic distribution of arriving tasks

Finding an online and optimal resource allocation in vehicular edge networks with high dynamic conditions and users' mobility is challenging. Furthermore, the optimal resource allocation might change over time by entering new applications with dynamic distribution and rate of arriving with different demands to the network. Hence, both allocation and reallocation policies are urgently needed, along with the dynamicity of the network. Each task can be assigned to any server for the execution process. Task migration can address the user's mobility issue, i.e., transferring tasks from one MEC to another to find a proper server [60]. This scenario is challenging because it should be executed immediately when a new time-sensitive task arrives at a MEC server. What is the solution if there is an insufficient resource there? As a result, reallocation policies need to apply to the system architecture of the management, and primary tasks can be migrated to other proper servers.

Dai *et al.* [12] investigated resource allocation by considering the combined effect of load balancing and offloading in a multi-user, multi-server vehicular environment. Nevertheless, there was no cooperation with the central cloud to satisfy the tasks with high resource demands in this work. Yu *et al.* [79] considered both the local and remote resource sharing with the collaboration across different data centers and a coalition game was formulated and solved by a game-theoretic algorithm. However,

the authors did not consider the dynamic and unpredictable load of the environment. Zhang *et al.* [86] studied task offloading in a heterogeneous vehicular network with multiple MEC servers in a deep Q-learning approach. Most of the existing works, such as [12], [79], [86] did not investigate various types of application tasks to formulation their problems, and there were no reallocation and scheduling policies.

In this chapter, motivated by the above issues and challenges, we consider different application tasks and design a complementary algorithm from Chapter 3 for resource allocation optimization. We extend our proposed task offloading algorithm in Chapter 3 where there was an efficient collaboration scheme among MEC and central cloud for task offloading to use the high resource capacity of the central cloud and short delay of MEC simultaneously. The complementary algorithm can reallocate resources for new and previous tasks based on tasks' priorities. In this context, different applications have different requirements, especially response time. We classify vehicle applications into three levels according to their characteristics: Crucial Applications (CA), High-Priority Applications (HPA), and Low-Priority Applications (LPA) [20].

CAs are safety-related applications with a short tolerable response time. CAs have the highest priority and must be immediately executed in the initial MEC server, which is uploaded. HPAs are important, but failures and delays are allowed. HPAs are optional safety-enhancing applications, e.g., routing, navigation, parking navigation. LPAs are of lesser importance to drivers and passengers and the tolerable response time is relatively soft. Cloud-based video games and multimedia applications are some examples of LPAs. The main contributions of this chapter are as follows:

- We formulate a new resource allocation problem concerning required vehicles' QoS and MEC's limited resources to provide a low rejection rate and efficient resource management.
- We propose reallocation and scheduling policies based on three categories of vehicle applications giving the highest priority to crucial applications. In addition, we investigate the dynamic distribution and rate of arriving tasks.
- We adopt a deep reinforcement learning process to find the optimal task offloading scheme and minimize the rejected rate of vehicular application tasks.

4.1 System model and problem formulation

The system model of complementary algorithm and scheduling follows the system model in Chapter 3. The difference with the previous chapter is the number of tasks each vehicles requests in every time slot, which is not restricted to one task and adding dynamic distribution of arriving tasks which will be defined in Section 4.2.3. The architecture of the vehicular edge computing network, as Fig. 4.1 illustrates, has three domains named Vehicle, MEC, and Central domains. In our model each vehicle can request any number of application tasks at each time slot and the vehicle will receive the results. Each vehicle m has the set of application tasks {1,. . . , J} indexed by j.

Each application consists of a tolerable response time depicted by $\phi^{\text{res}}_{m,j}$. Moreover, each task has requested resources of computing and storage; it can be denoted by: $\phi_{m,j} = [\phi^{\text{comp}}_{m,j}\ \phi^{\text{stor}}_{m,j}]$, respectively. Vehicles can offload their application tasks to the nearest RSUs. Application tasks can be hosted in the MEC server next to the corresponding RSU. The task can be migrated to another MEC or central cloud to find the best trade-off between the response time and resource requirements. Some of the considered parameters in this chapter are stated in Table 4.1.

Table 4.1: Main notations used in this chapter

Notation	**Definition**
$\mathcal{M}$/M/m	Set/number/index of vehicles
$\mathcal{N}$/N/n	Set/number/index of MEC servers
$\mathcal{V}$/i	Set of all MEC servers plus central server v/index
J/j	Number of tasks/index
$\phi_{m,j}$	Vector of application resource demand of task j from vehicle m
$\phi^{\text{comp}}_{m,j}$	Application computation demand of task j from vehicle m
$\phi^{\text{stor}}_{m,j}$	Application storage demand of task j from vehicle m
$\phi^{\text{res}}_{m,j}$	Request's tolerable response time of task j from vehicle m
$\tau^{\text{access}}_{m,j}$	Access latency of application j from vehicle m
$\tau^{\text{mig}}_{m,j}$	Migration latency of application j from vehicle m
$\tau^{\text{proc}}_{m,j}$	Processing latency of application j from vehicle m
$\tau^{\text{queue}}_{m,j}$	Queuing latency of application j from vehicle m
`Rej`	Rejection rate
$\xi^{i}_{m,j}$	Indicator if application j from vehicle m is executed on server i
Ψ^{j}_{m}	Indicator the rejected request j from vehicle m by the network

We focus on computational tasks of vehicles, where each task is non-divisible such that it can either be processed in a MEC server or remotely in the central cloud based on (4.1). We define the binary decision variable $\xi^i_{m,j}$ which is set to 1 if application task j from vehicle m is executed at server i, otherwise, it is 0. Therefore, we have the following constraint

$$\sum_{i \in \mathcal{V}} \xi^i_{m,j} \leq 1, \forall m \in \mathcal{M}(t), j \in m \tag{4.1}$$

As before, MEC servers suffer from limited computational and storage resources but satisfy communications quality. In contrast, the central cloud has sufficient cloud resources but a considerable end-to-end communications delay. The central cloud can execute some vehicles' applications based on their priority and QoS requirements, and it can compensate for the lack of sufficient resources in MECs.

It would be significant to find a proper server node to execute the application task to reduce the vehicle-perceived latency in vehicular edge computing networks. In other words, task placement in such a network requires consideration of the limited resources of MEC servers, which is different from central servers with large resource capacities. Furthermore, it should be essential to apply the highest priority to CAs applications to execute in the initial MEC immediately without any migration. In this regard, our proposed intelligent reallocation protocol provides a short required response time, especially for the applications with the kind of CAs. Hence, we define the following restrictions:

- Each task is performed using only one server, according to (4.1).
- The sum of allocated resources to all vehicles in each server does not exceed its resource capabilities. Thus, we have the following constraint:

$$\sum_{m \in \mathcal{M}(t), j \in m} \xi^i_{m,j} \cdot \phi_{m,j} \leq R_i, \forall i \in \mathcal{V} \tag{4.2}$$

- Response time for a task j from vehicle m and getting the result to the same vehicle is the sum of the access $\tau^{\text{access}}_{m,j}$, migration $\tau^{\text{mig}}_{m,j}$, queuing $\tau^{\text{queue}}_{m,j}$, and processing latency $\tau^{\text{proc}}_{m,j}$. This sum should satisfy an application's tolerable response time $\phi^{\text{res}}_{m,j}$. Hence, we consider the following constraint for that purpose

$$\tau_{m,j}^{\text{access}} + \tau_{m,j}^{\text{mig}} + \tau_{m,j}^{\text{queue}} + \tau_{m,j}^{\text{proc}} \leq \phi_{m,j}^{\text{res}} \tag{4.3}$$

- We define Ψ_m^j as a decision variable which is set to 1 if the application task j from vehicle m is a rejected task; otherwise, it is 0.

Furthermore, the total rejected applications divided by the total arriving applications is the rejected rate `Rej`. We aim to minimize the rate of rejection with respect to response time constraints and resource restrictions. Thus, an optimization problem for resource management can be written as

$$\begin{aligned}
&\min \quad \text{Rej} = \frac{\sum_{m\in\mathcal{M}(t)}\sum_{j\in m}\Psi_m^j}{\sum_{m\in\mathcal{M}(t)}\sum_{j\in m}1} \\
&\text{s.t.} \quad \text{C1}: \quad \sum_{i\in\mathcal{V}}\xi_{m,j}^i \leq 1, \forall m \in \mathcal{M}(t), j \in m, \\
&\qquad \text{C2}: \quad \sum_{m\in\mathcal{M}(t), j\in m}\xi_{m,j}^i \cdot \phi_{m,j} \leq R_i, \forall i \in \mathcal{V}, \\
&\qquad \text{C3}: \quad \tau_{m,j}^{\text{access}} + \tau_{m,j}^{\text{mig}} + \tau_{m,j}^{\text{queue}} + \tau_{m,j}^{\text{proc}} \leq \phi_{m,j}^{\text{res}}, \\
&\qquad \text{C4}: \quad \xi_{m,j}^i \in \{0,1\}, \forall m, i, j, \\
&\qquad \text{C5}: \quad \Psi_m^j \in \{0,1\}, \forall m, j \in m
\end{aligned} \tag{4.4}$$

Based on the proposed algorithm, at first, a request is offloaded to the nearest MEC. Then, the resource demand of the request is compared with the available resource capability of the MEC; if the application resource demand is less than the available resource capability of the server, then the request is processed there. Suppose the MEC server does not have free resources to allocate the request and the task belongs CAs. In that case, the previous tasks that belong either to HAs or LAs must be migrated to the other servers to provide free resources for the task with the highest priority.

Moreover, if a MEC offloads new tasks belonging to either HAs or LAs, and the MEC does not have free resources to allocate the request; another server will be selected based on the resource allocation strategy, and the application will migrate there to execute.

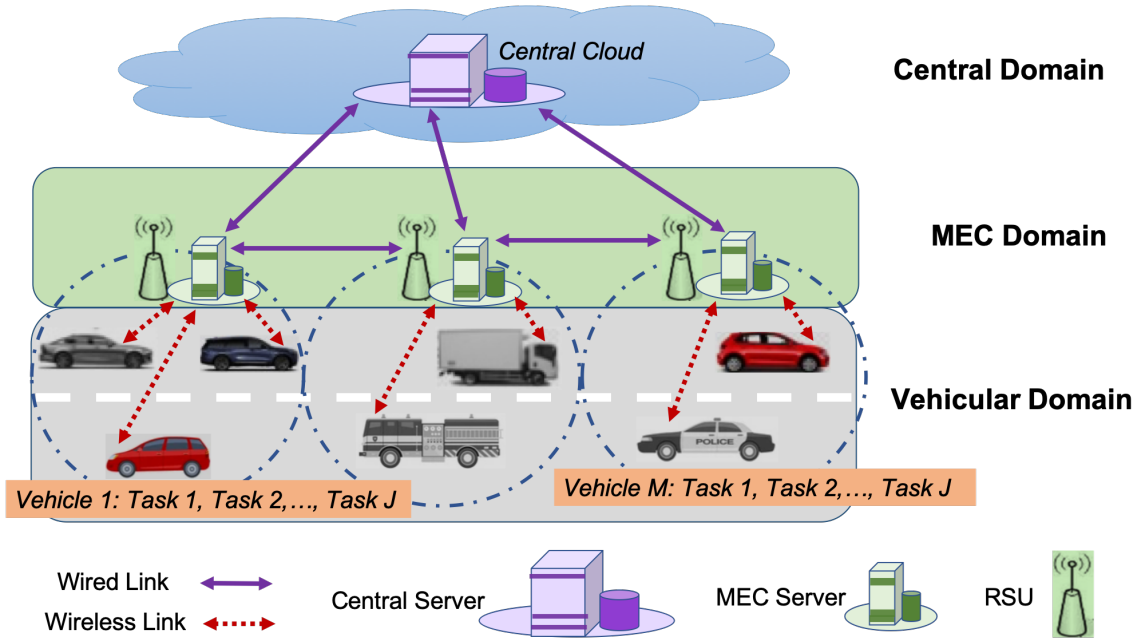


Figure 4.1: A vehicular edge computing network with various tasks

4.2 Re-allocation and dynamic rate of arriving tasks

4.2.1 Resource allocation and offloading based on DRL

Equation (4.4), which has several number of restrictions, as mentioned in 4.1, is a mixed-integer linear optimization problem, and such problems are usually considered NP-hard problems [55, 70]. Therefore, we utilize reinforcement learning to transform the formulated optimization problem and solve it by leveraging deep Q-learning. Our deep reinforcement learning (DRL) characteristics are the same as defined in Section 3.2.1 and are summarized as follows [30]:

State space

In our learning solution, each state $S(t) \in \mathcal{S}$ is associated with $S(t) = \{\widehat{\phi}, \widehat{W}(t), \widehat{C}(t)\}$ as following vectors

- $\widehat{\phi}$: Application information including resource demands (e.g. computational and storage), tolerable response time, and the location of the vehicle.

$$\widehat{\phi} = \left[\phi_{m,j}^{\text{comp}} \quad \phi_{m,j}^{\text{stor}} \quad \phi_{m,j}^{\text{res}} \quad \phi_{m,j}^{\text{loc}}\right] \tag{4.5}$$

- $\widehat{W}(t)$: Application tasks waiting in the buffer of each server at time slot t with their resource demands including computation and storage.
- $\widehat{C}(t)$: Running tasks in each server at time slot t and all their resource demands including computation and storage.

Action space

The current action is defined as $a_{m,j}(t) = \{a^1_{m,j}(t), a^2_{m,j}(t), \ldots, a^i_{m,j}(t)\}$ where $a^i_{m,j}(t)$ is a decision variable which is set to 1 if requested application j from vehicle m is placed on server i to be executed, otherwise, is 0 at time slot t. Note that $\sum_{i\in\mathcal{V}} a^i_{m,j}(t) = 1$ which means each request can perform and execute only in one server. The action of the deep reinforcement learning agent is defined as a server selection for each application to be run on the selected server and it is indexed by the number of MECs and central cloud.

Reward definition

Once the agent takes action based on the observed environment state, the environment will return an immediate reward to the agent. Then in the learning stage, the agent updates the resource allocation policy based on the received reward until the algorithm converges [58]. Indicated by (4.6), the estimated response time for task j from vehicle m, $\tau^{\mathtt{est}}_{m,j}(t)$, (the sum of the access, queuing, migration and execution latencies) is compared with tolerable response time of task j from vehicle m, indicated by $\phi^{\mathtt{res}}_{m,j}$. If the estimated response time is a less than the tolerable response time for request j from vehicle m, then the reward would be $-\tau^{\mathtt{est}}_{m,j}(t)$, otherwise, the reward would be $-w.\tau^{\mathtt{est}}_{m,j}(t)$. Thus, to minimize the number of dropped tasks that do not complete before tolerable response time, we follow reward element for offloaded task j from vehicle m that are corresponding (4.6).

$$r_{m,j}(t) = \begin{cases} -\tau^{\text{est}}_{m,j}(t), & \text{if } \ \tau^{\text{est}}_{m,j}(t) \leq \phi^{\text{res}}_{m,j} \\ \\ -w \cdot \tau^{\text{est}}_{m,j}(t), & \text{otherwise.} \end{cases} \tag{4.6}$$

Deep Q-learning solution

We adopt central deep Q-learning as a value-based approach [38] as mentioned in Section 3.2.2. The agent gets the state information from the environment. The action with the highest value determines the next state. In other words, to select the next state as a proper server for the execution of an application, our algorithm rescans all $Q(S(t), a(t))$ values. It captures them as the outcome of a deep Q-network to gain the maximum value. Moreover, the algorithm adopts the epsilon-greedy action selection method [50] to balance exploration and exploitation by choosing between them randomly.

As mentioned in Section 3.2.2, Q-learning updates its Q-value by temporal difference. Temporal difference captures the difference between the current estimate and previous ones and approximates the estimation by comparing them at two consecutive episodes [50]. In a neural network, we use the Multi-Layer Perceptron algorithm as a function approximator for $Q(S(t), a(t))$.

4.2.2 Re-allocation strategy

Reallocation protocol reallocates resources for new and previous tasks based on tasks priorities when new application tasks arrive. In this context, different applications can have different requirements, especially in terms of response time. Due to the importance of CAs, CAs get the highest priority. The objective of the reallocation strategy is to satisfy all requirements of CAs and reduce response time as much as possible. Thus, when a CA arrives in a MEC, if there is no available resource there, reallocation strategy will provide sufficient resources in the initial MEC for the new incoming CA by migrating previous tasks (HPA or LPA) to the other proper MECs. Consequently, to improve the performance of our system model, we propose reallocation and least-response-time-first (LRTF) scheduling policies described in this section.

Re-allocation scheduling

When an initial MEC server offloads a CA task, it should execute the task there. Hence, after comparing the resource demand of CA, depicted by ϕ_{j_A}, with the available

resource capability of the initial MEC b, indicated by R_b, if the CA resource demand is less than the available resource capability of the server, then the CA is executed there. Otherwise, if the initial MEC server does not have free resources to allocate the CA, another proper server will be selected for LPA, existing in the initial MEC, based on deep Q-Learning algorithm presented in Chapter 3. Finally, the LPA will migrate to a new chosen server. As a consequence, free resources will be provided in the initial MEC to allocate the CA. On some occasions that there is no LPA in the initial MEC to migrate, an HPA will migrate to a fitting selected server. This process will continue until the free resource capacity is available to the CA in the initial MEC. Our Re-allocation policy is presented in Algorithm 3. In addition, Fig. 4.2 illustrates a simple illustration of the re-allocation policy.

Algorithm 3 Re-allocation Policy

```
1: while CAs are offloaded by MEC b do
2:     if R_b < φ_{j_A} then
3:         if LPA ∈ MEC_b then
4:             Find a new execution node for LPA based on Algorithm 2
5:             Migrate LPA to the new execution node
6:             Go to the line (2)
7:         else if HPA ∈ MEC_b then
8:             Find a new execution node for HPA based on Algorithm 2
9:             Migrate HPA to the new execution node
10:            Go to the line (2)
```

LRTF scheduling

After the task offloading process and choosing a proper MEC server for the tasks, they are waiting in the server's buffer for the execution. They will serve in order of their priorities. In essence, our LRTF scheduling says CAs should exit the queue and start the execution phase first until no more CA is in the buffer. Then, HPAs and LPAs will begin to be scheduled for execution in the server, respectively (see Fig. 4.3). The required response times for high-priority tasks will be guaranteed by utilizing reallocation scheduling.

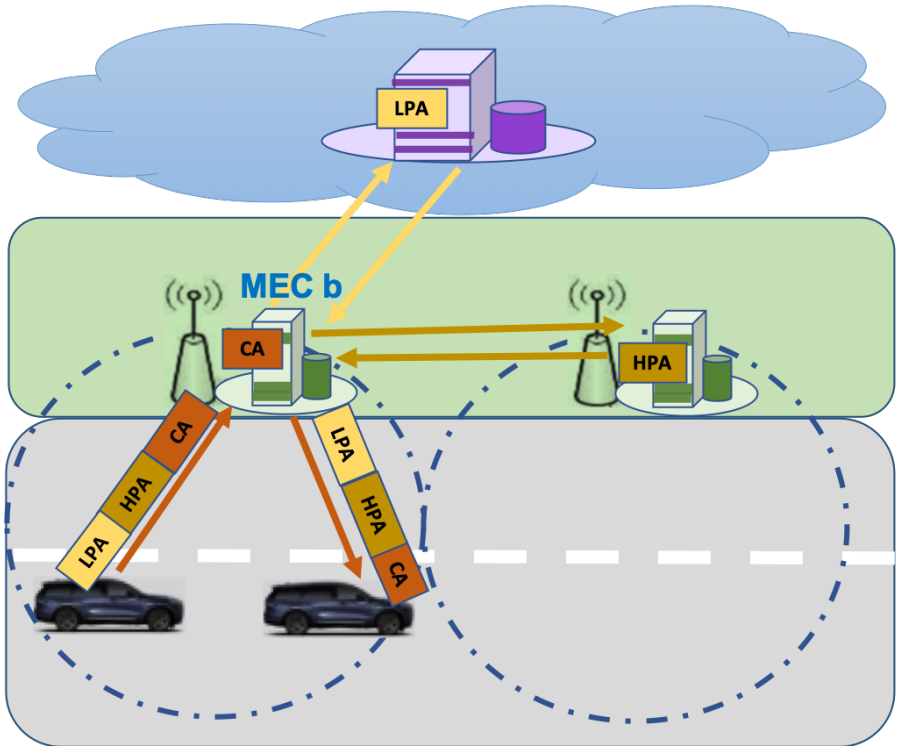


Figure 4.2: A simple illustration of re-allocation policy

4.2.3 Dynamic distribution and rate of arriving tasks

Analyzing and defining a good model for incoming requests would be greatly important in considering the distribution of service requests. To make dynamicity of our model and investigate the dynamic distribution and rate of arriving tasks, we generate tasks based on a Poisson distribution, which is mainly used to model the system better. The time between coming tasks is independent and identically distributed for the Poisson process with λ rate. Moreover, we will adopt inhomogeneous Poisson to consider various loads from vehicles during a 24-hour day. Thus, parameters of $\lambda(t)$ can change over time. We defined $\lambda_A(t)$, $\lambda_B(t)$, and $\lambda_C(t)$ to guarantee the dynamic distribution of arriving CA, HPA, and LPA tasks respectively. Furthermore, $\lambda_A(t)$, $\lambda_B(t)$, and $\lambda_C(t)$ are defined by different functions separately. The functions are time-dependent and continuous (e.g., $sin(t)$, etc.). A visual sample for distributed arriving CAs and HPAs is presented in Fig. 4.4.

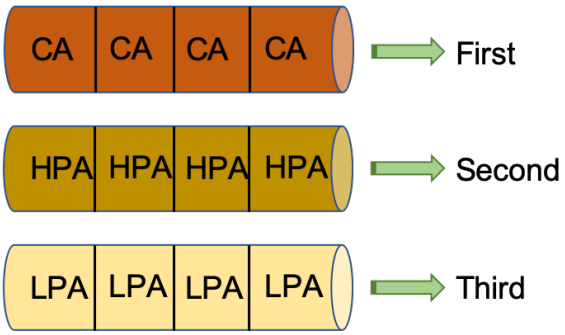


Figure 4.3: LRTF scheduling

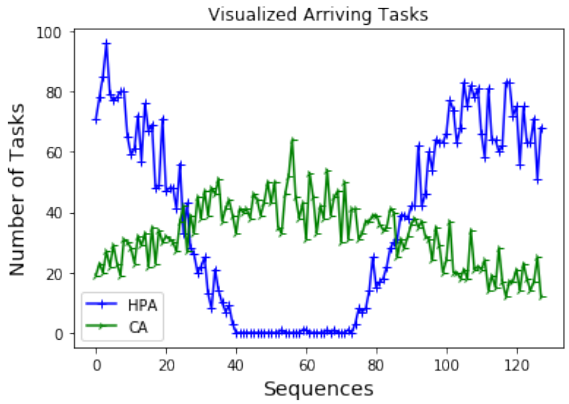


Figure 4.4: Visualized arriving CAs and HPAs.

4.3 Performance evaluation

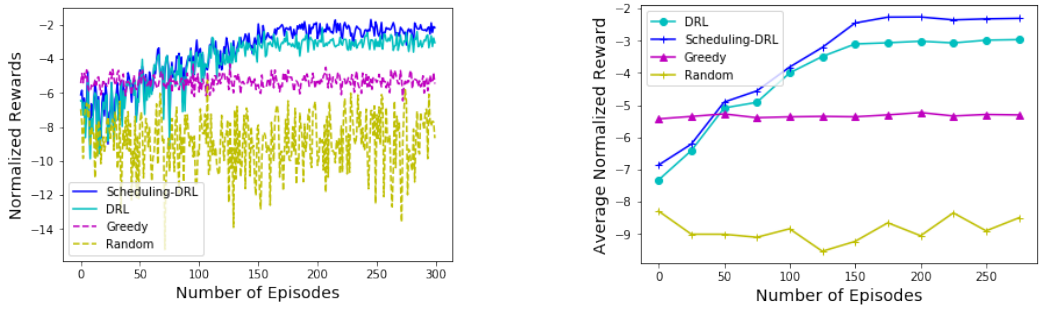
Implementation results are presented in this subsection to illustrate the performance of the proposed reallocation and scheduling task offloading based on deep reinforcement learning for the vehicular scenarios with MEC and central servers. It is noted that to improve illustration, the results are shown in the average scope.

The dataset, setting, and baselines for this chapter are the same as those in Section 3.3.1. Thus, we utilize a realistic vehicular dataset using mobility traces of taxi cabs in Rome, Italy [4] to evaluate the performance of our proposed algorithm. We randomly take a snapshot of each taxi's trajectory during a day using the dataset. In addition, as mentioned in Section 4.2.3, we generate tasks based on inhomogeneous Poisson distribution to create dynamic distribution and rate of arriving tasks.

Furthermore, we utilize three models as baselines to present the comparison and performance for our algorithm named Scheduling-DRL. The first one is the deep

reinforcement learning (DRL) method which is a part of our algorithm without reallocation and LRTF scheduling [30]. The DRL method was proposed in the previous chapter. Our DRL algorithm allocates tasks to a proper server based on task requirements and server capacity. The second baseline method is the Greedy method which assigns tasks to servers with higher resource capabilities. The last baseline for comparison, called Random, means that all arriving requests are assigned to each node randomly.

Fig. 4.5 demonstrates the convergence performance of our Scheduling-DRL algorithm and other baselines. The reward convergence for each task offloading is achieved by Equation (4.6). The learning process starts by updating the parameters of the deep Q-learning. Thus, the total rewards per episode fluctuate sharply at the beginning of the learning process and then increase as the parameters gradually optimize. As we can see from Fig. 4.5, in Scheduling-DRL and DRL methods, among the episodes during the learning stage, the total rewards per episode fluctuate sharply and are relatively small in the first 150 episodes and then tend to meet a relatively stable and high value. In addition, our scheduling algorithm has the highest reward. The Random and Greedy models do not follow any learning policy, so there is no improvement in stability in normalized reward values by increasing the number of episodes there.



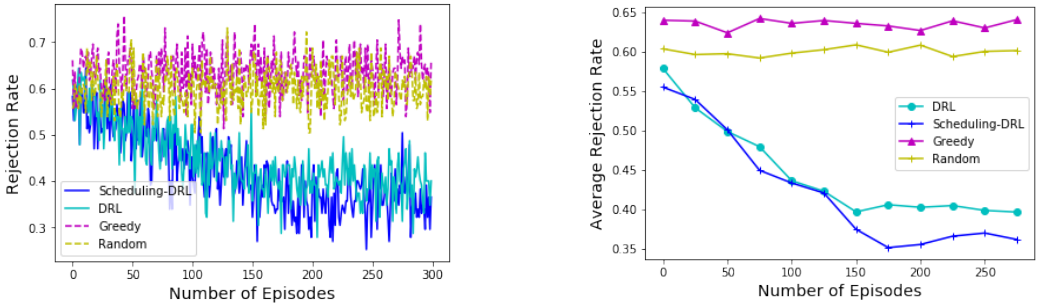
(a) Normalized reward

(b) Average normalized reward

Figure 4.5: Normalized reward per episode in Scheduling-DRL, DRL, Greedy, and Random solution.

Fig. 4.6 represents the rejection rate and the average rejection rate of vehicular applications per episode. From the central orchestrator perspective, an efficient resource management scheme should accept as many tasks as possible with the given available resources. The proposed algorithm achieves a lower ratio of dropped tasks than the other methods. After around 150 episodes, the rejection rate gains a stable value and

a lower rejection rate compared to other baselines. According to our Scheduling-DRL algorithm, we can guarantee that crucial applications with the highest priority in vehicular networks will be immediately processed. There are no reallocation policies and the LRTF scheduling, and it works based on a first-come-first-served policy in which requests are served in the order of their arrival in other base lines. It is worth mentioning that the Greedy model does better than the Random strategy mainly because it makes a difference between servers and gives high priority to servers with high resource abilities.



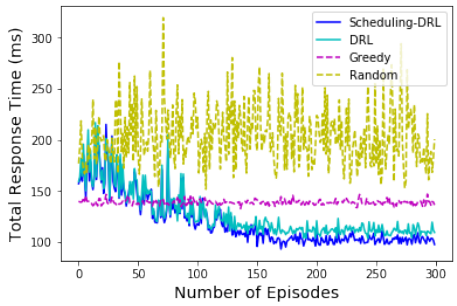
(a) Rejection rate

(b) Average rejection rate

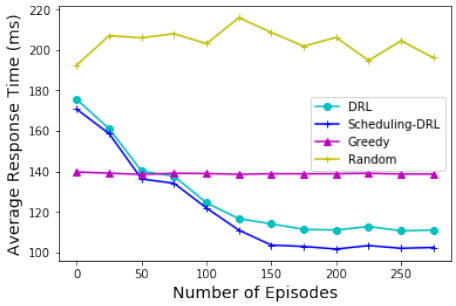
Figure 4.6: Rejection ratio per episode in Scheduling-DRL, DRL, Greedy, and Random solutions.

As shown in Fig. 4.7, the proposed algorithm consistently achieves a lower total response time than the other methods. From a vehicle perspective, satisfying delays and QoS requirements are critical. When the number of episodes increases, the average response time decreases accordingly. This is because increasing episodes can help the DRL and scheduling algorithms learn effectively. These results further verify that the proposed scheduling approach can find a suitable offloading strategy with a shorter response time. Thus, the vehicular networks can provide a flexible way of measuring latency and rejection rate. The proposed approach can adapt to various dynamic situations and give an optimal solution for resource allocation strategy. Random and Greedy models do not follow any learning policy, so there is no improvement in normalized reward, rejection rate, and response time values by increasing the number of episodes.

Fig. 4.8 demonstrates task distribution based on server resources. We randomly choose episode 297 to consider task distribution. Servers 0 to 5 represent MEC 0 to MEC 5, and server 6 shows the central cloud. The resource of servers includes



(a) Total response time



(b) Average response time

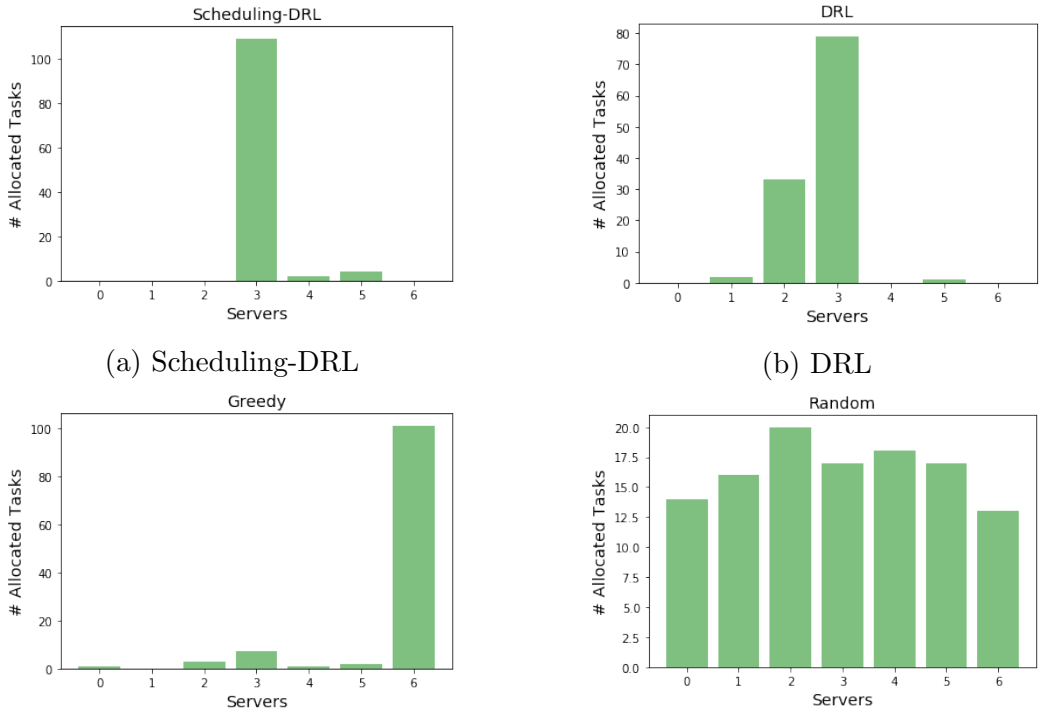
Figure 4.7: Response time per episode in Scheduling-DRL, DRL, Greedy, and Random solution.

computational capacity and the CPU cycle per second, storage (Mbits) in each server, and the number of cores a server has, which denotes how many tasks each server can execute simultaneously. The resource ability of each MEC and central server is listed in Table 4.2.

Table 4.2: Servers resource ability for Fig. 4.8

Server	**Computation**	**Storage**	**# Core**
MEC 0	1	30	2
MEC 1	4	55	4
MEC 2	4	36	9
MEC 3	5	67	9
MEC 4	3	58	9
MEC 5	3	18	2
Central 6	100	100	100

Fig. 4.8a shows the number of actions allocated to the servers based on Scheduling-DRL. It is based on the number of tasks waiting in the queue of a server that can satisfy tasks requirement. According to Table 4.2, MEC 3 has more computation and storage compared with other edge clouds; thus, it can process tasks much quicker than others. As a result, our Scheduling-DRL method allocates more tasks to the MEC 3 to satisfy both tolerable delay and computation demands of applications. As demonstrated in Fig. 4.8b, the sum of application tasks assigned to servers is based on the result of our proposed learning resource management solution, which considers total response time and the resource demands of tasks. Moreover, based on the figure, the resource availability of servers is investigated for action selection. Furthermore,



(a) Scheduling-DRL

(b) DRL

(c) Greedy

(d) Random

Figure 4.8: The number of allocated tasks to the MEC (0 to 5) and central servers (6) in Scheduling-DRL, DRL, Greedy, and Random methods.

as Fig. 4.8c shows, the central server with significant resource ability receives more tasks using the Greedy strategy. Nevertheless, as Fig. 4.8d illustrates, the tasks are distributed uniformly among servers, and the Random method does not investigate QoS demands. Our proposed scheduling plays well considering its resource capabilities and QoS demands as a final discussion point.

4.4 Summary

In this chapter, we have studied resource allocation and application task offloading in a highly dynamic of vehicular edge computing. We proposed reallocation and LRTF scheduling policies to ensure that crucial applications with the highest priority in vehicular networks are immediately processed. In addition, we utilized inhomogeneous Poisson distribution to investigate the dynamic distribution and rate of arriving tasks. Moreover, we proposed deep reinforcement learning to solve an NP-hard problem

considering QoS constraints and resource restrict capacity. Simulation results showed that our proposed algorithm could reduce the ratio of dropped tasks and response time compared with other methods.

Chapter 5

Decentralized and proactive resource allocation algorithm

This chapter focuses on proactive resource allocation and task offloading in vehicular edge computing. Different from existing works, we aim to propose a distributed DRL algorithm to provide various benefits such as local decision making. Our DRL-based distributed algorithm presents an accurate workload prediction. We classify vehicular tasks based on their priorities into three categories, Crucial Applications (CAs), High-Priority Applications (HPAs), and Low-Priority Applications (LPAs) [20], and migrate the application tasks with lower priority to provide service for CAs. CAs are core vehicle system applications or safety-related applications. Because of their importance to the vehicle and the passengers, CAs have the highest priority and must be executed immediately. The primary motivation of our proactive resource allocation algorithm is to gain high QoS and to provide a high acceptance rate of crucial vehicular tasks. The task offloading as a part of the algorithm is usually considered an NP-hard optimization problem [70], and it is complicated to find an optimal global solution by traditional approaches. We develop a distributed deep learning scheme following two objectives. The first is resource allocation and task offloading utilizing a distributed DRL where each MEC server learns and acts as an independent agent. The second is distributed workload prediction based on the Multivariate LSTM method.

Due to the time-varying of task requirements and resource capabilities in MEC servers in the dynamic nature of vehicular networks, we formulate dynamic offloading and resource allocation as Finite Markov Decision Process (FMDP) involving state

and action spaces and only depending on the current state and action. One promising approach is to apply deep learning to solve this optimization problem by training a deep learning model to learn the mapping between the problem input parameters and the optimal solution [75]. In particular, the agents of distributed DRL are deployed at the MEC server to indicate the optimal offloading decision of the vehicles' applications and the computation resource allocation with high accuracy in near-real-time [75]. In this vein, optimum task offloading using decentralized DRL could ensure the system performance measured by latency and acceptance rate. In addition, as one of the powerful decision-making algorithms in the artificial intelligence field, DRL performs dynamic programming to achieve excellent performance and effectiveness in tackling the optimization under dynamic environments [36].

In the distributed DRL, we propose a model-free distributed Q-learning algorithm for cooperative multi-agent-decision processes. The set of all possible actions is discrete and finite, and the action space in each agent is the same. In addition, the reward function in each agent is the same; in this situation, distributed Q-learning algorithm can find the optimal policy and converge [33, 85]. The authors in [33] advocate a distributed Q-learning algorithm that converges for deterministic finite multi-agent Markov decision process where each agent maintains local action and successively takes maximization over the joint action. There is no need for each agent to acquire other agents' actions and histories. Moreover, the convergence of the optimal Q-function for cooperative multi-agent RL settings involving homogeneous agents has been established in [85]. In our case, each MEC server has its learning agent who runs a standard Q-learning procedure independently based on a multi-agent reinforcement learning algorithm, and all agents conduct a decision algorithm independently and simultaneously determine an optimal strategy for the FMDP [9, 49, 85]. The main contributions of this chapter are as follows:

- *Proactive resource allocation in a highly dynamic environment*: We propose an efficient, proactive resource allocation and task offloading in a highly dynamic vehicular system. We utilize machine learning methods to find an efficient resource allocation and workload prediction to maximize the acceptance rate of the tasks. This way, a task migration among local and remote servers is allowed based on the vehicles application's priority and QoS requirements.

- *Decentralised DRL-based task offloading algorithm*: In a highly dynamic environment, we propose a model-free decentralized DRL-based task offloading and resource allocation algorithm. Our decentralized learning enables each MEC server to find the optimal solution for proactive resource allocation problems. The problem is to maximize the acceptance rate of the vehicular tasks, particularly crucial application tasks.

- *Migration based on workload prediction*: In our proactive resource allocation algorithm, we propose Multivariate-LSTM to predict near-future workload in each MEC server. The workload prediction is based on different application tasks priority. According to the result of the workload prediction, migration occurs to release free resources for crucial applications that have the highest priority.

5.1 System model and assumptions

The system architecture in this chapter is the same as the system architecture of Chapter 4. The number of tasks each vehicle requests in every time slot is not restricted to one task. We consider three different domains in vehicular edge computing networks (see Fig. 5.1). The first domain is the central domain consists of a central cloud v. The second domain is the MEC domain includes a set of MEC servers $\mathcal{N} = \{1, \ldots, N\}$. Each MEC server belongs to at least one cluster in the MEC domain, and each MEC is located beside one RSU in the road. The third domain is the vehicle domain consisting of a set of vehicles $\mathcal{M}(t) = \{1, \ldots, M\}$ in the network at time slot t indexed by m. In addition, Fig. 5.1 illustrates vehicles access the MEC servers through the wireless link, and the MEC servers are connected through a backhaul network. In the system, during MEC servers' deployment, a backhaul network is made, and MEC servers that are in close distance are clustered. Furthermore, each MEC server directly connects to the central server in a wired manner. The only difference between this chapter and the two previous chapters is system management. In other words, in Chapter 3 and 4, the central management is assumed to be deployed in the central cloud; however, in this chapter, system management is distributed, which is deployed in each MEC server.

Like previous chapters in this thesis, we consider a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$

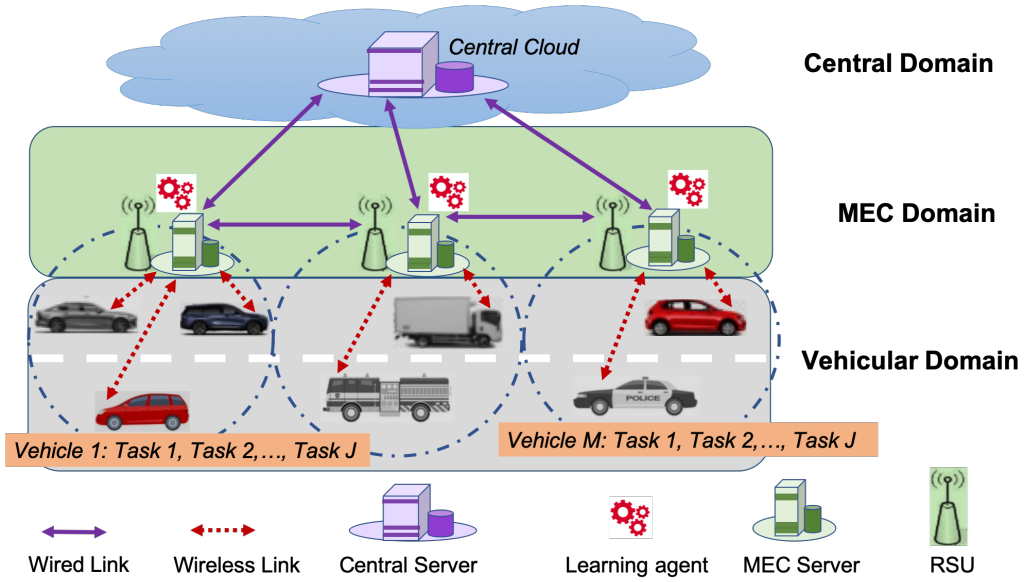


Figure 5.1: An illustration of vehicular edge computing network.

is the set of MEC servers plus the central server v and $\mathcal{E}$ is the set of links between MECs and central cloud. Each node $i \in \mathcal{V}$ has own resource capacity as $R_i = [R_i^{\text{comp}}\ R_i^{\text{stor}}]$ where R_i^{comp} and R_i^{stor} shows the computational and storage capacity of node i, respectively and every MEC server is capable of running one or more tasks, simultaneously. The considered parameters of this chapter are stated in Table 5.1. We present task characteristics, architecture components in each MEC, and problem formulation in the following subsections.

5.1.1 Task characteristics

In our model each vehicle can request any number of application tasks at each time slot. Let $m = \{1, \ldots, J\}$ be the set of application tasks vehicle m requests indexed by j. We focus on computational tasks of vehicles, where each task is non-divisible such that it can either be processed in a MEC server or remotely in the central cloud based on (5.1). We define the binary decision variable $\xi_{m,j}^i$ which is set to 1 if application task j from vehicle m is executed at server i, otherwise, it is 0. Therefore, we have following constraint:

$$\sum_{i \in \mathcal{V}} \xi_{m,j}^i \leq 1, \forall m \in \mathcal{M}(t), j \in m \tag{5.1}$$

Table 5.1: Network parameters and notations used in this chapter

Notation	Definition
$\mathcal{M}$/M/m	Set/number/index of vehicles
$\mathcal{N}$/N/n	Set/number/index of MEC servers
$\mathcal{V}$/i	Set of all MEC servers plus central server v/index
J/j	Number of tasks/index
R_i	Vector of resource capacity of server i
R_i^{comp}	Computation capacity of server i
R_i^{stor}	Store capacity of server i
$\phi_{m,j}$	Vector of application resource demand of task j from vehicle m
ϕ_m^{comp}	Application computation demand of task j from vehicle m
$\phi_{m,j}^{\text{stor}}$	Application storage demand of task j from vehicle m
$\phi_{m,j}^{\text{res}}$	Request's tolerable response time of task j from vehicle m
$\tau_{m,j}^{\text{access}}$	Access latency of application j from vehicle m
$\tau_{m,j}^{\text{mig}}$	Migration latency of application j from vehicle m
$\tau_{m,j}^{\text{proc}}$	Processing latency of application j from vehicle m
$\tau_{m,j}^{\text{queue}}$	Queuing latency of application j from vehicle m
Acc	Accepted rate of crucial applications
A	Crucial applications category
$\xi_{m,j}^{i}$	Indicator if application j from vehicle m is executed on server i
ψ_m^j	Indicator the accepted request j from vehicle m by the network

In the first place, each task can offload its requests to the nearest MEC where the vehicle is in the coverage area through a wireless link, and then it can migrate to a proper server for processing through a wired connection. Like Chapter 4, we classify vehicle applications into three levels according to their characteristics: Crucial Applications (CAs), High-Priority Applications (HPAs), and Low-Priority Applications (LPAs) [20]. In this context, different tasks can have different importance based on their tolerable response time. CAs have the highest priority due to the importance of CAs, which have the shortest tolerable response time. Tolerable response time (in time slots) of task j from vehicle m is denoted by $\phi_{m,j}^{\text{res}}$. Moreover, task j from vehicle m has requested resources; it can be denoted by: $\phi_{m,j} = [\phi_{m,j}^{\text{comp}}\ \phi_{m,j}^{\text{stor}}]$, where $\phi_{m,j}^{\text{comp}}$ and $\phi_{m,j}^{\text{stor}}$ shows the requested computational and storage resources of task j from vehicle m. The cooperation of the task and Infrastructure is considered as follows:

- *Task between vehicle and MEC domains:* In our system model, vehicles are allowed to access the MEC servers through the wireless link. Each car can

generate more than one task simultaneously for offloading to its nearest MEC in each time slot and receiving the results through the maybe other MEC in the other time slots. This communication between vehicle and MEC server imposes access latency for task j from vehicle m depicted by $\tau_{m,j}^{\text{access}}$ which is composed of transmission delay and round trip time. We have two assumptions here: first, a vehicle can travel through the coverage area of more than one MEC while uploading the tasks and downloading the results. Second, there is sufficient bandwidth for the cooperation of cars and the relative MEC servers.

- *Task between MEC and central domains:* To overcome the challenges caused by the limited resources in MEC servers and to achieve high resource capacities usage, the task may migrate to another MEC or central cloud. The MEC, which received the application, determines whether it is executed or migrated to the other MEC or the central cloud. If there is not a proper server selection to satisfy the resource and QoS demands of the task, then the task will be dropped. We utilize a distributed learning vision in each MEC server for optimal server selection for all application tasks. The machine learning method we adopt is a combination of DRL and LSTM, which will be considered later in this chapter. During task migration, offloading, and processing, the total service latency for each application is significant to consider as an end-to-end QoS provision. Based on the latency model in Chapter 3, we investigate migration latency $\tau_{m,j}^{\text{mig}}$, processing latency $\tau_{m,j}^{\text{proc}}$, and queuing latency $\tau_{m,j}^{\text{queue}}$ for task j from vehicle m in this work, as well.

5.1.2 Architecture components in each MEC

Fig. 5.2 illustrates the architecture components of each MEC server which are defined as following modules:

- *Clustering Agent:* During deployment of MEC servers and making backhaul network, the MEC servers in the close physical distance are clustered with each other. Moreover, at the beginning of each time slot, the clustering agent broadcasts its necessary information, such as its available computation and storage resources, to the other MEC servers in the same cluster.

- *Monitoring:* In the beginning of each time slot, the MEC's Monitoring module receives information from the environment such as the characteristics of arriving tasks and information from other MEC servers which are in the same cluster.
- *Receiving Request:* In each time slot, the Receiving module of MEC server receives all requests from vehicles that are in the coverage area of the MEC.
- *Learning Agent:* Every MEC server has its own learning agent independent of other MECs or central cloud. Intuitively, each MEC adopts information from other MEC servers in the same cluster for efficient learning. Efficient learning provides an optimum solution for resource allocation problem in a highly dynamic vehicular environment. The sub-components of each learning agent are as follows:
  - *Workload Prediction:* The learning agent of an MEC server predicts its own workload of the next time step. In other words, the learning agent predicts how many CA, HPA, and LPA will arrive in next time slot. For this purpose, the learning agent utilizes the history of its own workload and the workload history of other MECs in the same cluster.
  - *Migration:* There are three types of migrations in migration modules. The first one is *Migration based on Prediction*. The workload prediction results provide MEC server the ability to manage resources for the next time slot. Intuitively, suppose there are insufficient computation and storage resources for new arriving CAs with the highest importance of application tasks. The MEC can free resources by migrating lower priority tasks (LPA and HPA) to the central cloud or other MEC servers. The second one is *Request Migration*. If there is not enough resource capacity in the MEC that receives the request, the request has to be migrated to a proper server for execution. The third one is *Result Migration*. After finishing the execution process, if the vehicle is not in the coverage area of the MEC that executed the task, the result has to be migrated to MEC, which is very close to the car. As the new location of vehicles is updated continuously, it would be easy to capture the new place of the car and the destination MEC.
  - *Acceptance Decision:* When new application tasks arrive to a MEC, the MEC considers available resources of its own and neighbors and tasks'

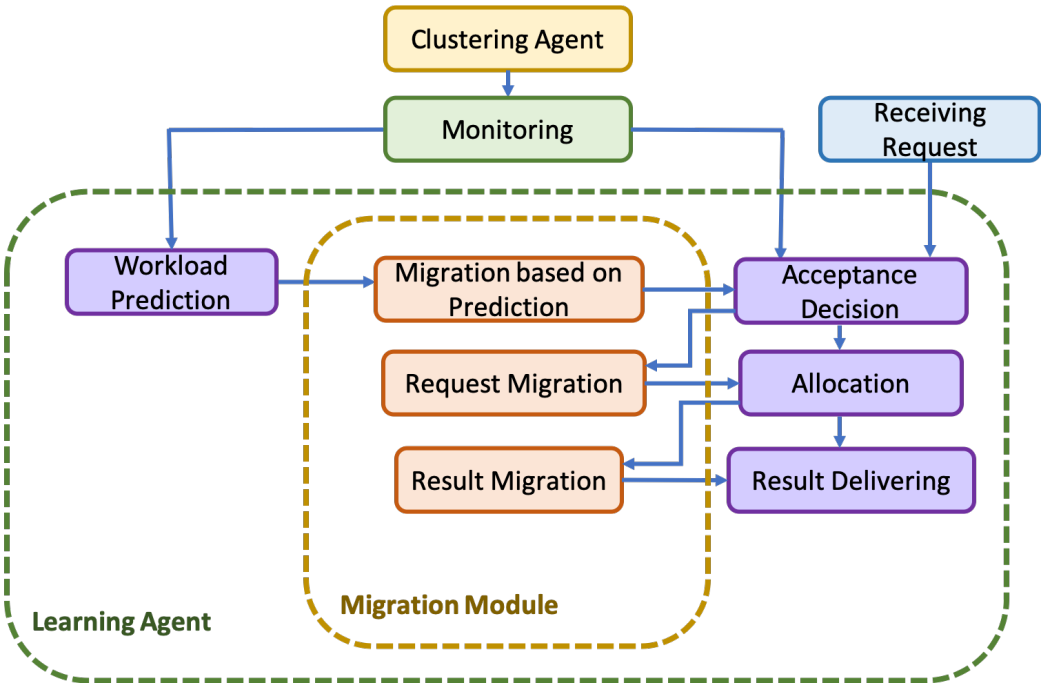


Figure 5.2: MEC node architecture.

demands then decides whether to accept the tasks or reject them.

- *Allocation:* If an arriving task is not rejected at the arriving time, the Allocation module in the MEC server finds a proper location for task execution utilizing learning methodology.

- *Result Delivery:* After accomplishing task execution, the result must be delivered to the same vehicle that sent the request request. The Result Delivery module delivers the result to the vehicle.

5.1.3 Problem formulation

Finding a proper server node to execute the application task would significantly reduce the vehicle-perceived latency in vehicular edge computing networks. In other words, task offloading in such a network requires considering the limited resources of MEC servers, which is different from central servers with large resource capacities. Furthermore, knowing the incoming workload of each MEC in the near future, it is essential to reserve resources for the task with the highest priority and migrate tasks

with low priority to the other MEC servers. In this regard, our proposed proactive resource allocation algorithm provides a short required response time, especially for the applications with the kind of CAs to execute at the MEC. In contrast, applications with high resource demands and more extended response time will be placed in the central cloud. To ensure the sum of allocated resources to all vehicles in each server i, does not exceed its resource capabilities R_i, we have the following constraint

$$\sum_{m \in \mathcal{M}(t), j \in m} \xi_{m,j}^{i} \cdot \phi_{m,j} \leq R_i, \forall i \in \mathcal{V} \tag{5.2}$$

In general, the response time for a task j from vehicle m and the time required to get the result to the same vehicle is the sum of the times for access, migration, queuing, and processing latency. This sum time should satisfy an application's tolerable response time $\phi_{m,j}^{\text{res}}$, so we consider the following constraint for that purpose

$$\tau_{m,j}^{\text{access}} + \tau_{m,j}^{\text{mig}} + \tau_{m,j}^{\text{queue}} + \tau_{m,j}^{\text{proc}} \leq \phi_{m,j}^{\text{res}} \tag{5.3}$$

Furthermore, we define ψ_m^j as a decision variable which is set to 1 if the application task j from vehicle m which belongs to CA categories is accepted; otherwise, it is 0. Thus, the total accepted applications of CA categories divided by the total arriving CA applications, which we indicate with A, is considered the accepted rate of crucial tasks Acc. We aim to maximize the rate of CAs acceptance respecting response time constraints and resource restrictions. Hence, an optimization problem for resource management would be as

$$\begin{aligned}
\max \quad & \text{Acc} = \frac{\sum_{m \in \mathcal{M}(t), i \in \mathcal{V}} \sum_{j \in m \cap A} \xi_{m,j}^{i} \cdot \psi_m^j}{\sum_{m \in \mathcal{M}(t)} \sum_{j \in m \cap A} 1} \\
\text{s.t.} \quad & \text{C1}: \quad \sum_{i \in \mathcal{V}} \xi_{m,j}^{i} \leq 1, \forall m \in \mathcal{M}(t), j \in m, \\
& \text{C2}: \quad \sum_{m \in \mathcal{M}(t), j \in m} \xi_{m,j}^{i} \cdot \phi_{m,j} \leq R_i, \forall i \in \mathcal{V}, \\
& \text{C3}: \quad \tau_{m,j}^{\text{access}} + \tau_{m,j}^{\text{mig}} + \tau_{m,j}^{\text{queue}} + \tau_{m,j}^{\text{proc}} \leq \phi_{m,j}^{\text{res}}, \\
& \text{C4}: \quad \xi_{m,j}^{i} \in \{0,1\}, \forall m, i, j, \\
& \text{C5}: \quad \psi_m^j \in \{0,1\}, \forall m, j \in m \cap A
\end{aligned} \tag{5.4}$$

Fig. 5.3 is a simple illustration of the task offloading management in our proactive resource allocation. In addition, the proposed proactive resource allocation and task offloading process is presented in Fig. 5.4, in which green boxes show the parts performing machine learning. Each MEC server has its own learning agent, which can predict arriving workload during the next time slot. Because the CA has the highest priority compared to other vehicular applications, it would be considered if there is sufficient resource capacity in the base MEC b for execution CA or not. Base MEC b is the server that receives tasks directly from the vehicles. If there are insufficient resources in MEC b to satisfy CA's resource demand, then MEC b can free resources. MEC b migrates lowest priority tasks LPA to the central cloud or other MEC servers utilizing the DRL algorithm, which will be explained later in this paper. If there is no LPA in the MEC, HPA will be migrated to another server. The workload prediction policy can guarantee the CAs to be immediately executed once being received by MEC b. The proactive resource allocation is presented in Algorithm 4.

In general, based on the algorithm, once task j from vehicle m is received by the base MEC b, it will be executed there if the task resource demand is less than the available resource capability of the server b. Therefore, the execution node would be the same MEC b. If there are insufficient resources to be allocated to the request j in MEC b, the DRL-based optimization algorithm will find the proper server for request j and request j migrates to the relative server named execution node i. After finishing the execution, j should be delivered to vehicle m. If the execution node i is not the same destination MEC, task j has to migrate from execution node i to destination MEC to deliver to vehicle m. The task offloading algorithm is presented in Algorithm 5.

5.2 Intelligent and proactive resource allocation algorithm

The formulated problem (5.4) is a mixed-integer linear optimization problem, and such problems are usually considered as NP-hard problems [55, 70]. We use a finite MDP model to capture the dynamic of network state transitions. An MDP can be defined by the set of current system states, set of actions, and the set of real-value reward functions [38]. Therefore, we utilize distributed DRL to transform the formulated

Algorithm 4 Proactive resource allocation

1: A $\leftarrow$ Find coming CAs based on workload prediction
2: **for** all $j_A \in A$ **do**
3: **if** $\phi_{j_A} < R_b$ **then**
4: Go to the next time slot
5: **else if** $LPA \in MEC_b$ **then**
6: Find a new execution node for LPA based on Algorithm 6
7: Migrate LPA to the new execution node
8: Go to the line (3)
9: **else if** $HPA \in MEC_b$ **then**
10: Find a new execution node for HPA based on Algorithm 6
11: Migrate HPA to the new execution node
12: Go to the line (3)

Algorithm 5 Task offloading algorithm

1: Task j from vehicle m is offloaded by base MEC_b
2: **if** $\phi_{m,j} < R_b(t)$ **then**
3: execution node = b
4: **else**
5: Select server i based on Algorithm 6
6: Migrate task j to node i
7: execution node = node i
8: Execute task on execution node
9: **if** execution node is not destination node **then**
10: Migrate the result to destination node
11: Deliver the result to vehicle m
12: **else**
13: Deliver the result to vehicle m

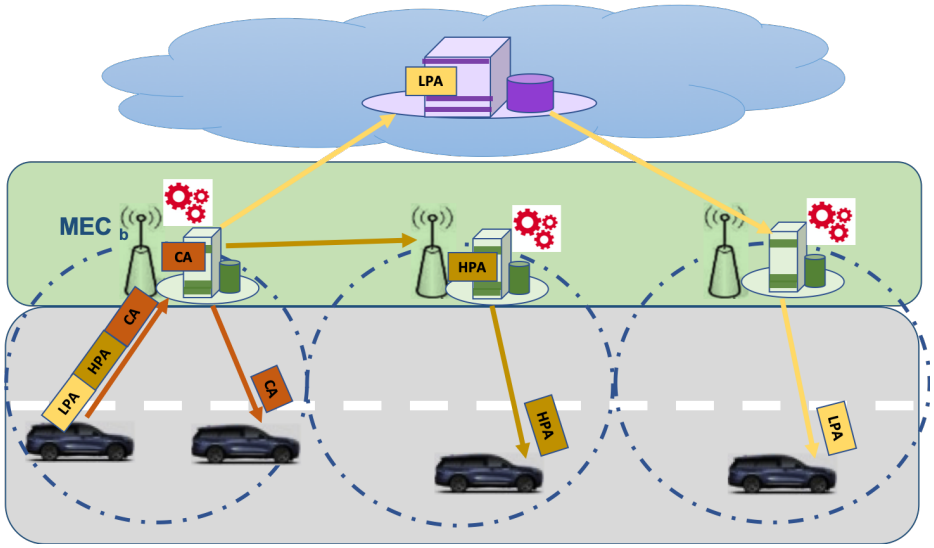


Figure 5.3: A simple illustration of task offloading in proactive resource allocation.

optimization problem. In addition, to maximize the crucial vehicular application acceptance rate, we implement a proactive resource strategy by adopting workload prediction methodology.

5.2.1 Decentralized multi-agent DRL and workload prediction

In the system model, MEC servers located at close physical distances create a cluster and broadcast their state information in the cluster through the overlay network in every time slot. Each cluster runs multi-agent DRL and solves it by leveraging deep Q-learning. Furthermore, predicting its variation workload is crucial for developing a proactive resource allocation algorithm. In each MEC, a learning mechanism is performed that efficiently decides how much and when to allocate tasks and where to place and migrate tasks. Thus, we utilize a Multivariate Long Short-Term Memory (LSTM) to perform workload prediction in each MEC.

Decentralized multi-agent DRL

The distributed DRL algorithm is designed to perform an effective proactive resource allocation algorithm eradicating difficulties in modeling and computation that can

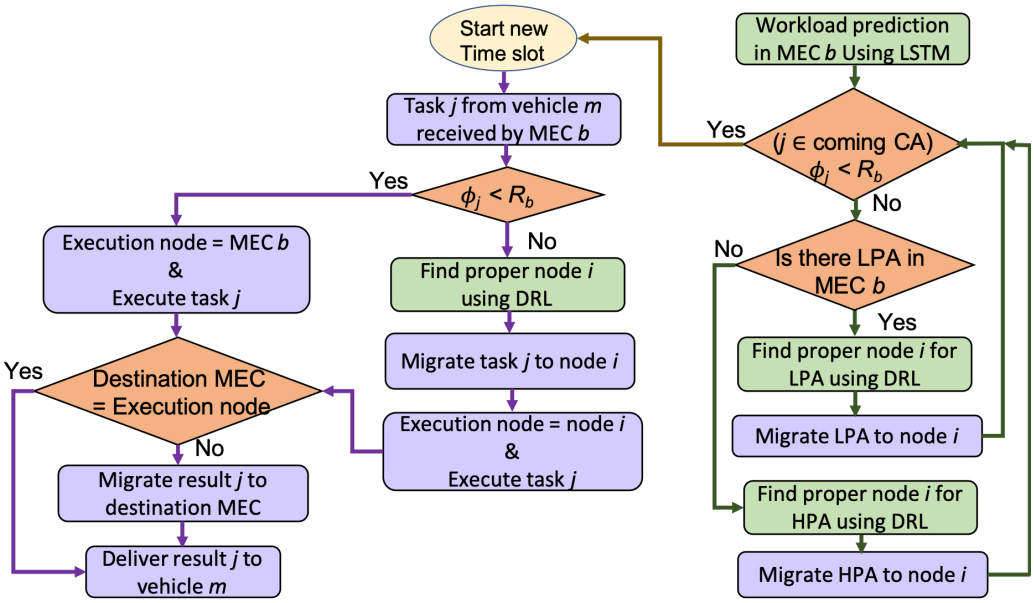


Figure 5.4: Proactive resource allocation and task offloading process

be handled in distributed manners [10]. In our distributed learning method, every MEC server learns and acts as an agent in each cluster and independently conducts a decision algorithm. We propose a model-free distributed Q-learning algorithm for cooperative multi-agent-decision processes. The set of all possible actions is discrete and finite. Space action is common among all agents, and the reward function is the same in each agent; in this situation, distributed Q-learning algorithm can find the optimal policy and converge [33, 85]. Note that each MEC runs the Q-learning procedure independently based on multi-agent reinforcement learning algorithm [9, 49]. All agents behave individually and are governed by the deep Q-network policy, and have the same objective. However, other agents' actions result in a new state, which is considered as the environment for formulating the agent's policy. Each agent has access only to local information, which in turn is affected by the decisions of other agents [56, 78]. This independent learner approach brings the benefit that the size of the state-action space is linear with the number of agents [49].

Workload prediction based on Multivariate LSTM

It is crucial to predict variation workload in MEC that the Multivariate LSTM method is applied to develop a proactive resource allocation algorithm. In essence, the LSTM

network learns the temporal dependence of sequential observations and predicts the future time series [67]. As vehicles move across a geographical area, workload changes at MEC at a close physical distance may introduce correlations between MECs' workloads. Investigating this correlation can be utilized to improve the prediction performance [52]. Intuitively, the historical workload information of MECs in close physical distance to anticipate the workload for each MEC is needed. Moreover, the workload information is categorized into three types of vehicular tasks. Thus, a Multivariate LSTM is considered for workload prediction.

In essence, the LSTM network will output the information for various application tasks that indicate the future workload dynamics. The outputs will pass to the state of the DRL for further learning.

We assume workload is a numeric value abstracting the number of different application tasks that come from vehicles and are offloaded by the MEC server. Hence, at every time slot t, the total workload in a particular MEC corresponds to three numbers, A, B, and C, which present the number of CA, HPA, and LPA tasks coming to it. Given a vehicular edge network topology with geographical distributed MECs, let $\widehat{H}(t-1)$ be a $(n+1)\times 3$ dimensional time series vector. $\widehat{H}(t-1)$ is constructed by $3(n+1)$ time series record the historical workload. This historical workload is related to the predicted MEC b and its n neighbors at each time interval (i.e. 3 is the number of task types and $n+1$ is the predicted MEC b and its n neighbors). Assume $H_i^{\mathrm{A}}(t-1), H_i^{\mathrm{B}}(t-1), H_i^{\mathrm{C}}(t-1)$ are the time series containing the observed historical workload of MEC i up to time t for CA, HPA, and LPA tasks respectively. Each data point in $\widehat{H}(t-1)$ is sampled with a specific time window (e.g., 100 milliseconds, 15 seconds, etc). The goal is to estimate workload of MEC b at the near future.

To develop a real-time prediction, one step ahead prediction is considered for a quick response and preventing the error accumulation as occurring in multi-step ahead prediction. Specifically, the Multivariate LSTM network takes the matrix $\widehat{H}(t-1)$ as input for learning the workload dynamics (Fig. 5.5). The Multivariate LSTM network will output three numbers $p_b^{\mathrm{A}}(t)$, $p_b^{\mathrm{B}}(t)$, $p_b^{\mathrm{C}}(t)$ indicating the estimated number of CA, HPA, and LPA tasks respectively in MEC b at time t. Output vector $P_b(t)$ as the predicted workload of MEC b is as follows:

$$P_b(t) = \begin{bmatrix} p_b^{\mathrm{A}}(t) & p_b^{\mathrm{B}}(t) & p_b^{\mathrm{C}}(t) \end{bmatrix} \tag{5.5}$$

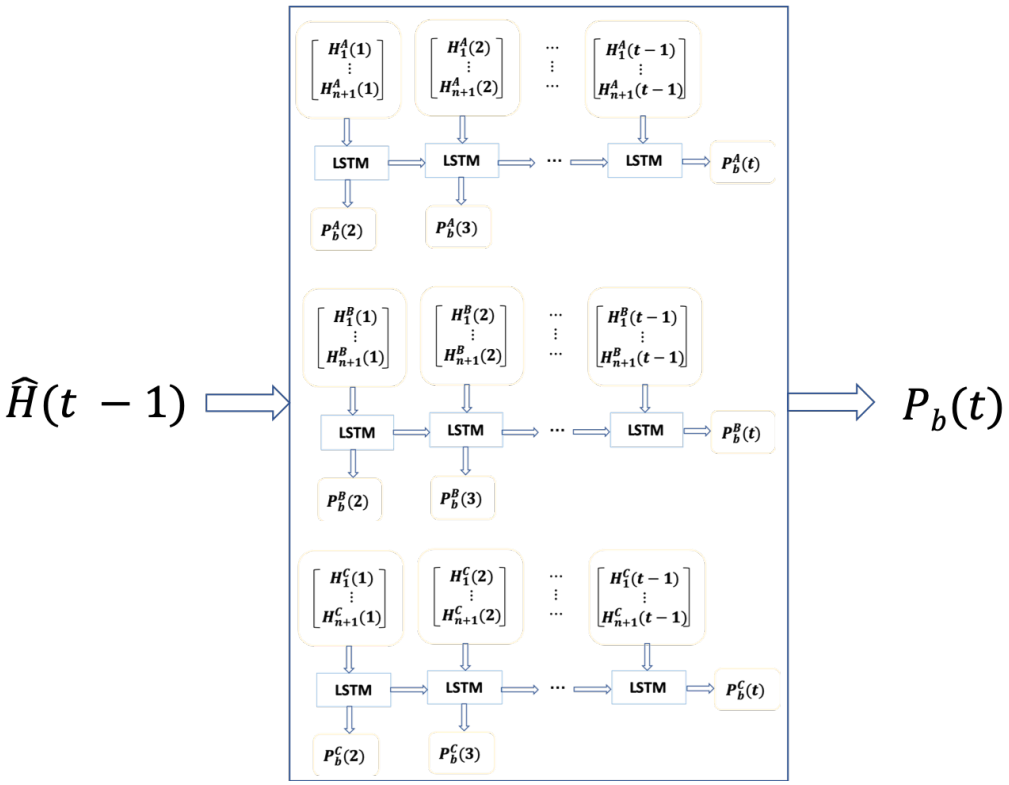


Figure 5.5: An illustration of input/output of multivariate LSTM network.

In the Multivariate LSTM-based workload prediction method and the data preparation stage, the vector of the dataset is divided into train and test sets [5]. Note that in LSTM, the number feature defines the different measures observed at the time of observation. Here, for three application tasks in the network and n neighbor MECs, the number feature equals $3 \times (n+1)$.

5.2.2 Intelligent task offloading and proactive resource allocation solution

Task offloading is a part of our proactive resource allocation algorithm. As aforementioned, the distributed DRL algorithm is designed to enable each MEC server to act as an agent and make the decision based on local observations, some information broadcasting from other agents in their close vicinity, and the output of Multivariate LSTM at the beginning of time slot t. Each MEC server has its own state space $S(t)$, action space $a(t)$, and reward model $r(t)$ in a finite Markov decision process [38],

which are summarized as follows for MEC b:

- *State Space:* Each state $S_b(t) \in \mathcal{S}$ is associated with $S_b(t) = \{\widehat{\phi}_b, \widehat{P}_b(t), \widehat{W}_b(t), \widehat{C}_b(t)\}$ as following vectors

  - $\widehat{\phi}_b$: Task information including its resource demands (computational and storage), tolerable response time $\phi^{\text{res}}_{m,j}$, and the location of the vehicle $\phi^{\text{loc}}_{m,j}$. This task information is observed by MEC b.

$$\widehat{\phi}_b = \begin{bmatrix} \phi^{\text{comp}}_{m,j} & \phi^{\text{stor}}_{m,j} & \phi^{\text{res}}_{m,j} & \phi^{\text{loc}}_{m,j} \end{bmatrix} \tag{5.6}$$

  - $\widehat{P}_b(t)$: Predicted workload of each MEC server at time slot t including the estimated number of applications of type A (CA), type B (HPL), and type C (LPA) in the near future as the output of Multivariate LSTM network. Notably, CA applications have the highest priority among vehicular tasks; thus, we consider A (CA) application to the DRL state as an input. In this way, each MEC server has a predicted workload plus a predicted workload of n number of other MEC servers located in its close physical distance.

$$\widehat{P}_b(t) = \begin{bmatrix} P^{\text{A}}_1(t) & P^{\text{A}}_2(t) & \dots & P^{\text{A}}_{n+1}(t) \end{bmatrix} \tag{5.7}$$

  - $\widehat{W}_b(t)$: Application tasks waiting in the buffer of the MEC server and other MEC servers located in the close physical distance at time slot t with their resource demands, including computation and storage. Here n is the number of MEC servers located in the close physical distance of MEC b doing learning.

$$\widehat{W}_b(t) = \begin{bmatrix} W^{\text{comp}}_1(t) & W^{\text{comp}}_2(t) & \dots & W^{\text{comp}}_{n+1}(t) \\ W^{\text{stor}}_1(t) & W^{\text{stor}}_2(t) & \dots & W^{\text{stor}}_{n+1}(t) \end{bmatrix} \tag{5.8}$$

  - $\widehat{C}_b(t)$: Running tasks in the MEC server and other MEC server located in close physical distance at time slot t with all resource demands including computation and storage.

$$\widehat{C}_b(t) = \begin{bmatrix} C^{\text{comp}}_1(t) & C^{\text{comp}}_2(t) & \dots & C^{\text{comp}}_{n+1}(t) \\ C^{\text{stor}}_1(t) & C^{\text{stor}}_2(t) & \dots & C^{\text{stor}}_{n+1}(t) \end{bmatrix} \tag{5.9}$$

- *Action Space:* The action is defined as server selection for each task to be run on the selected server, and it is indexed by the number of MECs located in their close distance plus the central cloud. The current action for task j from vehicle m is defined as $a_{m,j}(t) = \{a^1_{m,j}(t), a^2_{m,j}(t), \ldots, a^{n+1}_{m,j}(t)\}$ where $a^i_{m,j}(t)$ is a decision variable which is set to 1 if requested application j from vehicle m is placed on server i to be executed, otherwise, is 0 at time slot t. Note that $\sum_{i=1}^{n+1} a^i_{m,j}(t) = 1$ which means each request can perform and execute only in one server. The vector of action space for all application tasks to be assigned to a proper server can be depicted as follows

$$a_b(t) = \begin{bmatrix} a^1_{1,1}(t) & a^2_{1,1}(t) & \ldots & a^{n+1}_{1,1}(t) \\ a^1_{1,2}(t) & a^2_{1,2}(t) & \ldots & a^{n+1}_{1,2}(t) \\ \vdots & \vdots & & \vdots \\ a^1_{m,j}(t) & a^2_{m,j}(t) & \ldots & a^{n+1}_{m,j}(t) \end{bmatrix} \tag{5.10}$$

- *Action Selection:* The agent gets the state information from the environment. The action with the highest value determines the next state. In other words, to select the next state as a proper server for the execution of an application, our algorithm rescans all $Q_b(S(t), a_b(t))$ values. It captures them as the outcome of a deep Q-network to gain the maximum value. Moreover, the algorithm adopts the epsilon-greedy action selection method [50] to balance exploration and exploitation by choosing between them randomly.

- *Reward Definition:* Once the agent takes action based on the observed environment state, the environment will return an immediate reward to the agent. Then in the learning stage, the agent updates the resource allocation policy based on the received reward until the algorithm converges [58]. Indicated by (5.11), the estimated response time $\tau^{\text{est}}_{m,j}(t)$ (the sum of the access, queuing, migration and execution latencies) is compared with the tolerable response time $\phi^{\text{res}}_{m,j}$. If the estimated response time is less than tolerable response time for each vehicle's request, then the reward would be $-\tau^{\text{est}}_{m,j}(t)$ otherwise, the reward would be $-w \cdot \tau^{\text{est}}_{m,j}(t)$. Here, $-w$ is a large negative integer. Thus, to maximize the number of offloaded tasks that are completed with satisfying response time by the MEC server at time slot t, we define the following reward element for offloaded task j from vehicle m Equation (5.11).

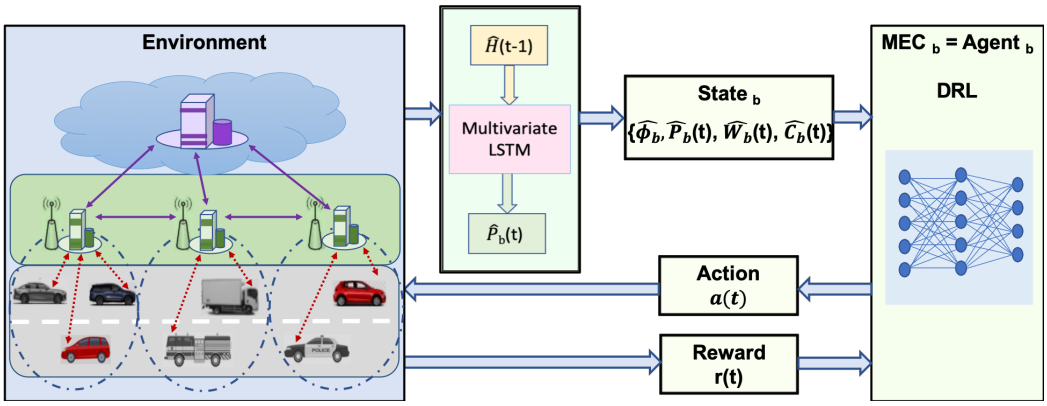


Figure 5.6: Proposed proactive deep reinforcement learning.

$$r^b_{m,j}(t) = \begin{cases} -\tau^{\text{est}}_{m,j}(t), & \text{if } \tau^{\text{est}}_{m,j}(t) \leq \phi^{\text{res}}_{m,j} \\ \\ -w \cdot \tau^{\text{est}}_{m,j}(t), & \text{otherwise.} \end{cases} \tag{5.11}$$

Fig. 5.6 presents the proposed architecture of proactive deep reinforcement learning. Here, Q-learning updates its Q-value by temporal difference. Temporal difference captures the difference between the current estimate and previous ones and approximates the estimation by comparing them at two consecutive episodes [50]. In a neural network, we use the Multi-Layer Perceptron algorithm as a function approximator for $Q(S_b(t), a_b(t))$. We also inject randomness into the approximation model by adding drop-out to the hidden layers of the Multi-Layer Perceptron network. This property increases the robustness and generalized capability of the network besides powering the exploration provided by this randomness.

Our distributed deep learning solution can be summarized in Algorithm 6 which repeats the procedure based on the number of episodes until the reward converges. The algorithm is executed in each MEC server while using the other MEC neighbors to calculate the state space. As mentioned in the above section, the workloads arriving at the MEC in the next time slot are considered in the state space. Moreover, every vehicle can send more than one request at each time slot. Indicated by Algorithm 6, in the first place, we have initialization for learning rate, discount factor γ, the epsilon first value ϵ_0 in epsilon-greedy, and minimum epsilon value $\epsilon_{\min}$. Then, for each episode, requests from vehicles arrive, and the total number of requests is counted in variable len, and the first state $S_b(0)$ is computed according to the first request.

Furthermore, for the epsilon-greedy method, the epsilon value needs to be updated in each episode (line 3). In addition, a deep Q-learning procedure is called in each episode. The Q-learning procedure repeats according to the number of requests. In other words, the time slot t grows based on the number of requests; On line 10, the state of MEC b is computed according to local observation and the relevant information from other MECs in close physical distance. Then, $Q(S_b(t), a_b(t))$, the output of our neural network, is estimated based on the state. During lines 12 to 16, the action selection process occurs in cooperation with the epsilon-greedy algorithm. Line 16 reveals the selected action is the action that has the highest Q-value. On line 17, the reward and the next state are calculated. Finally, TD method is used to optimize Q-Network.

Algorithm 6 DRL algorithm at MEC server b

1: Initialize learning rate, discount factor γ, the epsilon first value ϵ_0 in epsilon-greedy, and minimum epsilon value $\epsilon_{\min}$
2: **for** episode $\leftarrow 1, K$ **do**
3: Calculate new value for ϵ by $\max(\epsilon_{\min}, \epsilon_0 - \text{episode}/K)$
4: request $\leftarrow$ Arriving requests from the vehicles
5: len $\leftarrow$ Length of request
6: Compute $S_b(0)$ according to the first arriving request
7: $S_b(t) \leftarrow$ Q-Learning(request, len, $S_b(0)$)
8: **procedure** Q-Learning(request, len, $S_b(0)$)
9: **for** $t \leftarrow 1, \text{len}$ **do**
10: Compute $S_b(t)$ according to the request(t)
11: Estimate $Q(S_b(t), a_b(t))$ based on $S_b(t)$
12: Sample continuous uniform $p \sim$ U(0, 1)
13: **if** $p < \epsilon$ **then**
14: Sample discrete uniform $a_b(t) \sim$ U$\{0, n+1\}$
15: **else**
16: $a_b(t) \leftarrow \text{argmax}_a Q(S_b(t), a_b(t))$
17: $S_b(t+1), r_b(t) \leftarrow$ emulator ($a_b(t)$, request$(t+1)$)
18: **if** $t <$ len **then**
19: target $\leftarrow r_b(t) + \gamma.\max Q(S_b(t+1), a_b(t))$
20: **if** $t =$ len **then**
21: target $\leftarrow r_b(t)$
22: Optimize Q-Network based on TD error $(\text{target} - S_b(t))^2$
23: $S_b(t) \leftarrow S_b(t+1)$
24: **return** $S_b(t+1)$

The optimization problem is a mixed-integer linear programming problem, and achieving the optimal or sub-optimal solutions usually requires exponential time complexity. Moreover, the optimization procedure must be executed at each time slot due to the diversity of request requirements, the number of requests, and resource availability in the MEC server, which are time-varying and highly dynamic. Therefore, the conventional optimization methods using relaxation iteration algorithms incur high computational complexity due to numerical iterations, and their solutions are often sub-optimal. They would not scale well [64], usually converge slowly, and have prohibitive complexity for real-time implementations [74]. A machine learning-based approach is an effective and attractive solution to tackle this problem. Furthermore, Since the deep neural network of Q-learning employs the full-connection networks, the computational complexity of each training step is $O(\sum_{f=1}^{F} L_{f-1}L_f)$, where L_f represents the neural size of the f-th layer among F layers [32, 36], and the complexity of Q-learning algorithm is $O(T)$, where T is the total number of training steps [27]. The complexity of the multi-agent deep Q-network algorithm in this chapter for each cluster is $O(2NT\sum_{f=1}^{F} L_{f-1}L_f)$, where N is the number of agents in each cluster. In addition, LSTM is local in space and time, which means that the input length does not affect the storage requirements of the network and for each time step, the time complexity per weight is $O(1)$. Therefore, the overall complexity of an LSTM per time step is equal to $O(X)$, where X is the number of weights [23, 65]. As a result, the total complexity of the proactive resource algorithm in this chapter for each cluster is $O(2NT\sum_{f=1}^{F} L_{f-1}L_f + NX)$.

5.3 Performance evaluation and results

This section presents the experimental setting, baselines, results, and discussion to validate the proposed intelligent and proactive resource allocation algorithm.

5.3.1 Experimental setting and baseline

We consider a scenario with 7 MEC servers located at a close physical distance and one central server. Some of the parameter settings are in Table 5.2. Although our model works for various intervals, the selected intervals could properly determine the quality, validity, and capability of our model [30].

Table 5.2: Parameters used in this chapter

Parameter	Value	Description
N	7	Number of MEC servers
R_i^{comp}	[3, 11]	CPU cycles per second
R_i^{stor}	[15, 70]	Storage of MEC (Mbits)
R_i^{buf}	[500, 1500]	Buffer of MEC
R_i^u	[2, 10]	Number of cores in MEC
R_v^{comp}	100	CPU cycles per second of central cloud
R_v^{stor}	100	Storage of central cloud (Mbits)
R_v^u	100	Number of cores in central cloud
$\phi_{m,j}^{\text{comp}}$	[100, 260]	CPU cycles' task demand
$\phi_{m,j}^{\text{stor}}$	[3,10]	Storage's task demand

Analyzing and defining a good model for incoming requests would be greatly important to consider the distribution of service requests. We generate tasks based on Poisson distribution, mainly used to model the system better. The time between arriving tasks is independent and identically distributed for the Poisson process with λ rate. Moreover, we will adopt inhomogeneous Poisson to consider various loads from vehicles during a 24-hour day. Thus, parameters of $\lambda(t)$ can change over time. We defined $\lambda_A(t)$, $\lambda_B(t)$, and $\lambda_C(t)$ to guarantee the dynamic distribution of arriving CA, HPA, and LPA tasks respectively. Furthermore, $\lambda_A(t)$, $\lambda_B(t)$, and $\lambda_C(t)$ are defined with different functions separately. The functions are time-dependent and continuous (e.g., $sin(t)$, etc.), mainly because of a better illustration of workload prediction using Multivariate LSTM.

We use real vehicle mobility and assign the various application tasks to the vehicles. Thus, like previous chapters, we utilize a realistic vehicular dataset using mobility traces of taxi cabs in Rome, Italy [4]. As mentioned in Section 3.3.1, we randomly take a snapshot of each taxi's trajectory during a day using the dataset. Furthermore, two separate neural networks are applied for our proactive resource allocation algorithm; one is for DRL, a feedforward neural network with 4 fully connected layers. The other is for workload prediction, which is a recurrent neural network and LSTM. Deep learning settings are given in Table 5.3. Additionally, ReLU capacity [73] is an activation function added to hidden layers in DRL, which introduces non-linear features to the neural network and improves the deep neural network. Moreover, in the data preparation stage of the Multivariate LSTM method, the vector of the dataset is divided into train and test sets with the 90% and 10% ratio, respectively. Meanwhile,

we use TensorFlow [1, 58] to implement our proposed DRL and Multivariate LSTM schemes.

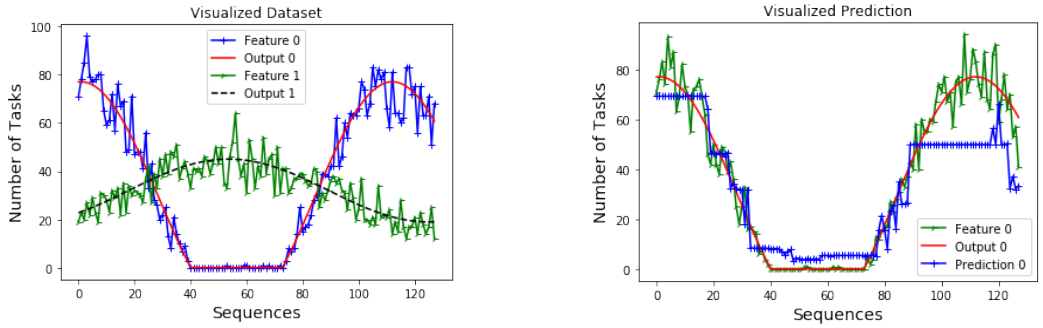
Table 5.3: Deep learning settings

DRL		
Notation	**Value**	**Description**
K	400	Number of episodes
$\epsilon_{\min}$	0.05	Min value of ϵ
ϵ_0	0.9	First value of ϵ in $\epsilon - greedy$
γ	0.98	Discount factor
-	0.0001	Learning rate
-	128	Hidden layer size
-	Temporal Difference Error (TD)	Loss function
-	Adam	Optimizer
Multivariate LSTM		
-	0.005	Learning rate
-	80	Hidden layer size
-	3×number of servers	Number of features
-	Mean Square Error	Loss function
-	Adam	Optimizer

We illustrate our proposal model in two parts Proactive-DRL and Predictive-DRL. Proactive-DRL is the whole work we proposed in this chapter according to Fig 5.4. Indeed, Proactive-DRL is the combination of the scheduling policy that we presented in Chapter 4 and workload prediction as a state-space of DRL. However, the Predictive-DRL algorithm consists of workload prediction as one part of state space in the DRL algorithm. As a result, we compare our proposed proactive and intelligent algorithm, named Proactive-DRL, with several benchmark methods, including Predictive-DRL, DRL, Greedy, and Random methods. DRL method is what we did in Chapter 3 which is an intelligent method for task offloading [30]. In other words, the DRL algorithm finds the best server based on the resource capability of servers and the task's resource requirements concerning the QoS, and there is no prediction policy in it. In the Greedy solution, tasks are assigned to servers with higher resource capabilities. In the Random method, arriving tasks are assigned to each node randomly, and its distribution for task offloading and server selection is uniform.

5.3.2 Experimental results and discussion

The experimental results are presented in this subsection to demonstrate the performance of the proposed proactive resource allocation in vehicular edge networks. In the proposed Multivariate LSTM, every MEC has three features showing the number of application tasks (CA, HPA, and LPA) as its workload plus three features of n MEC neighbors. Fig. 5.7 (a) presents a visual demonstration of the dataset for feature 0 (number of CA tasks) and feature 1 (number of HPA tasks) and their output. The output of prediction using LSTM determines the number of different applications. To better illustrate of prediction process, we use the $\lambda(t)$ as output. Fig. 5.7 (b) presents a visualized prediction as to the result of the LSTM scheme. Although one run prediction function will result in prediction in one time, we run the prediction function 128 times to draw the prediction results.



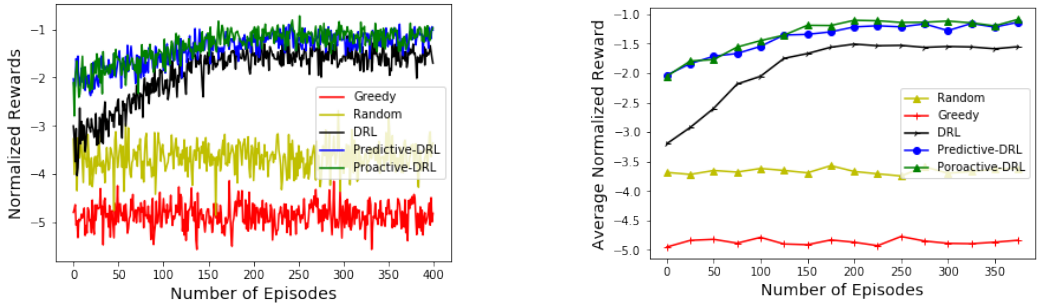
(a) Visualized incoming applications

(b) Prediction in multivariate LSTM

Figure 5.7: Visualized incoming applications and prediction in multivariate LSTM

Fig. 5.8 shows the convergence of the proposed algorithm Proactive-DRL with other policies. As we can see from Fig. 5.8, Proactive-DRL and Predictive-DRL work well with a good reward convergence. In all three learning methods, Proactive-DRL, Predictive-DRL, and DRL methods, among the 400 episodes during the learning stage, the total rewards per episode fluctuate sharply and are relatively small in the first 150 episodes and then tend to meet a relatively stable and high value. The Proactive-DRL and Predictive-DRL algorithm reach a higher value than the DRL algorithm. The learning process starts by updating the parameters of the deep Q-learning. Thus, the total rewards per episode fluctuate sharply at the beginning of the learning process and then increase as the parameters gradually optimize. Furthermore, Fig. 5.8 shows in the Greedy solution that each task is allocated to a server with high resource capacity.

Like the Random model, Greedy does not follow any learning policy, so there is no improvement in stability about normalized reward values by increasing the number of episodes. Moreover, there is no learning policy to support the convergence of rewards; the rewards fluctuate after the total episode. As we can see, this uniformly distributed random model suffers from a small value of rewards, which means it cannot support the tolerable response time requested by each task.



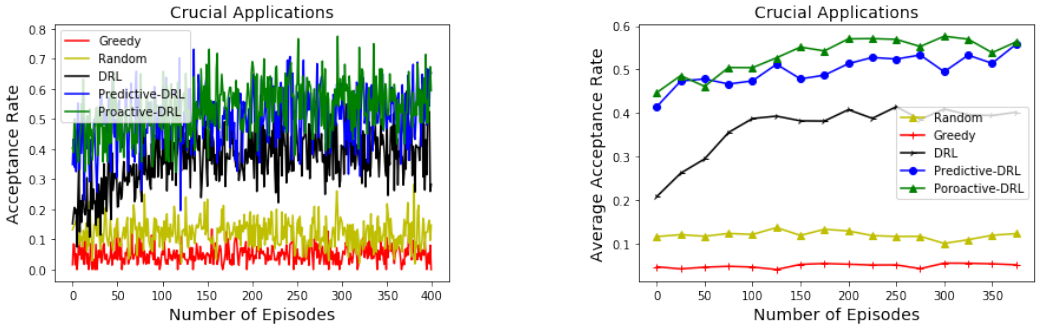
(a) Normalized reward

(b) Average normalized reward

Figure 5.8: Normalized reward per episode in Proactive-DRL, Predictive-DRL, DRL, Greedy, and Random solution.

Moreover, we consider two performance metrics: the ratio of accepted tasks (i.e., the percentage of the number of accepted tasks to the number of total task arrivals) and the response time (i.e., the average response time of the tasks which have been processed). Fig. 5.9 represents the acceptance rate and the average acceptance rate of tasks per episode. An efficient resource management scheme should accept as many tasks as possible with the given available resources. As we can see in this figure, the acceptance ratio increases in Proactive-DRL, Predictive-DRL, and DRL strategies at the beginning of the learning process. However, after around 200 episodes, the acceptance rate gains a stable value and a higher acceptance rate than other baselines. Based on Fig. 5.9, our proposed learning solution gets the highest acceptance rate in comparison with other methods. Proactive-DRL is learning how to allocate tasks to proper servers based on available resources and utilizing prediction of the workload and scheduling in the near future. Proactive-DRL gets a higher acceptance rate than Predictive-DRL mainly because Proactive-DRL has both workload prediction and scheduling, but Predictive-DRL has just workload prediction as one part of state-space. In the Random strategy, as observable in this figure, there is no improvement, even after passing the time and increasing the number of episodes. The acceptance

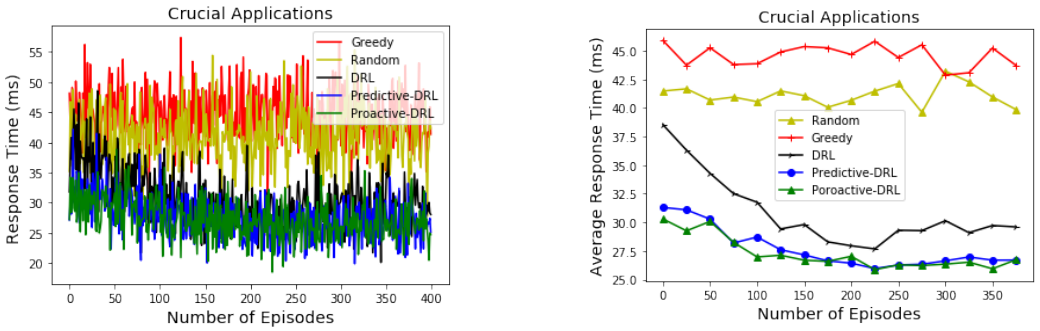
rate is around 1%, which is very low. Moreover, the Greedy method has no stable pattern between the first steps of simulation and its last steps. In the Greedy policy, the acceptance rate of crucial applications is near zero. This method allocates tasks to a server with a higher resource capacity in the central cloud and does not consider the tolerable response time of crucial tasks.



(a) Acceptance ratio of CA (b) Average acceptance ratio of CA

Figure 5.9: Acceptance ratio of crucial tasks per episode in Proactive-DRL, Predictive-DRL, DRL, Greedy, and Random solution.

Fig. 5.10 demonstrates the total response time per episode in each algorithm. In vehicular networks, satisfying tolerable response time and QoS demand of tasks are critical. As Fig. 5.10 represents, Proactive-DRL reduces response time more than other baselines.

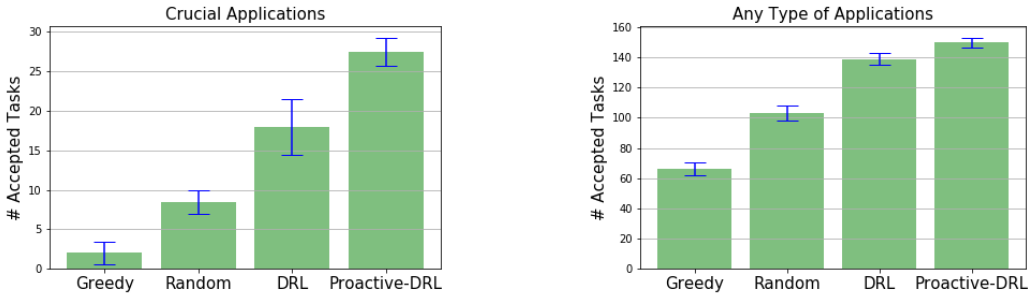


(a) Response time of CA (b) Average response time of CA

Figure 5.10: Total response time of crucial tasks per episode in Proactive-DRL, Predictive-DRL, DRL, Greedy, and Random solution.

The number of accepted tasks is presented in Fig. 5.11. We calculate the mean and variance of the number of accepted applications in the last four episodes. Fig. 5.11 (a)

demonstrates the number of crucial applications which are satisfied and accepted. The proactive-DRL algorithm has a higher acceptance number. As Fig. 5.11 (b) represents, Proactive-DRL and DRL are equivalent in the number of accepted tasks. However, this result is just for the last four episodes. In addition, the Greedy algorithm is the worse strategy in terms of acceptance number.



(a) Number of accepted crucial tasks in last episodes

(b) Number of accepted any type of tasks in last episodes

Figure 5.11: Number of accepted tasks in Proactive-DRL, DRL, Greedy, and Random solution.

5.4 Summary

In this chapter, we considered resource allocation and task offloading in vehicular edge computing networks. We proposed an intelligent and proactive resource allocation algorithm for various types of tasks in vehicular edge computing networks. We designed a distributed DRL algorithm for our resource allocation and task offloading problem that enables MEC servers to decentralize their offloading decisions, respecting tasks' demands. Furthermore, each MEC could learn how to predict its workload quickly utilizing Multivariate LSTM. Simulation results showed that our proposed algorithm could increase the ratio of accepted tasks and reduce the average response time compared with several benchmark methods.

Chapter 6

Conclusion and future work

In this final chapter, we summarize the contributions presented in this dissertation and discuss several potential extensions to our work.

6.1 Conclusion

Resource allocation and task offloading in vehicular edge computing networks to increase QoS and maximize the acceptance rate of vehicular tasks is restricted by several challenges. Much research is tackling these challenges in the MEC network; however, some reasonable solutions were not proposed for the VEC network. It is mainly because of the dynamic nature of vehicular networks and the limited resource capacity in edge servers. We proposed resource allocation and proactive resource allocation considering vehicular edge computing restrictions. We classified applications based on their requirements, and our proposed solutions work based on different categories of applications. To find the optimum resource allocation solution, we utilized machine learning methods and presented proactive resource allocation using intelligent predictions. In addition, we adopt a distribution model to generate vehicular requests to make a dynamic environment. The following conclusions can be drawn from this dissertation:

- We have considered multi-access edge computing in vehicular networks. We have proposed a task offloading with a resource allocation strategy based on task migration among MEC and central cloud servers. In other words, we consider

cooperation between MEC and the central cloud. We can benefit from a central cloud with sufficient resource capacity for vehicular tasks and MEC servers close to the vehicles providing a short response time. The central server is proper for tasks with plenty of resource demands and tasks that are not significantly delay-sensitive, and MEC servers are suitable for tasks with rigid delay requirements.

- With the help of deep reinforcement learning, we could efficiently handle a large number of real-time arriving requests, especially in a large and dynamic transferred network state space based on a realistic vehicle dataset. Furthermore, we model our network based on FMDP, which depends on the current and previous state and works based on deep reinforcement learning.

- We could specify QoS constraints in deep reinforcement learning and solve an NP-hard problem instead of mathematical approaches that lead to a non-polynomial time solution. In other words, we defined the reward function in deep reinforcement learning according to the tolerable response time of application tasks. Moreover, limited resources in MEC server is considered providing that the resource capacity of MEC servers should satisfy the computation and storage resource demand of application tasks.

- Simulation results showed that our proposed approach could provide an efficient and high-reward task offloading with high acceptance rates for arriving requests from vehicles, considering their locations, response times, resource demands, and the current resource utilization. Furthermore, we could reduce the total response time and minimize the rejection rate. We could gain a proper task distribution among servers to find an optimal task offloading.

- We proposed an intelligent and proactive resource allocation algorithm for various types of tasks in vehicular edge computing networks. Task offloading decisions with a central learning policy requires interaction between all vehicles, MEC servers, and the central cloud. Central learning can lead to time complications and an increase in some expenses. Therefore, we designed a distributed DRL algorithm for our resource allocation and task offloading problem that enables MEC servers to decentralize their offloading decisions, respecting tasks' demands.

- In our proactive resource allocation algorithm, each MEC could learn how to

predict its workload quickly utilizing Multivariate LSTM. In the intelligent workload prediction method, the correlation between MEC servers in their close physical distance was considered. Each MEC server can predict how many crucial tasks, high priority tasks, and low priority tasks will arrive at the next time slot. According to this prediction, the learning agent in the MEC server can make a proper resource allocation for crucial applications and find the optimum crucial task offloading. In this vein, the acceptance rate of crucial applications is guaranteed. Simulation results showed that our proposed algorithm could increase the ratio of accepted tasks and reduce the average response time compared with several benchmark methods.

- We proposed reallocation and LRTF scheduling policies to ensure that crucial applications with the highest priority in vehicular networks are immediately processed. For instance, if a time-sensitive application task arrives at a MEC server, it should run immediately, but what is the solution if there is an insufficient resource there? Our reallocation policies applied to the system architecture of the management, and primary tasks migrate to other proper servers to address this issue. Simulation results showed that our proposed algorithm could reduce the ratio of dropped tasks and response time compared with other methods.

- We classified vehicle applications into three levels according to their characteristics: crucial, high-priority, and low-priority. Different applications have different requirements, especially response time. Hence, proactive resource allocation works based on these three types of vehicular tasks. In addition, to make more real-work dynamic nature of the vehicular network, we generate arriving requests based on inhomogeneous Poisson distribution. Furthermore, to find the mobility of vehicles and their actual trajectory, we utilized a real-world dataset of taxi trajectories from Rome, Italy.

- We ran simulations in Python. We used both Spyder and Jupyter, which are running on Anaconda-Navigator. In addition, we used TensorFlow for implementing the deep neural network part of the machine learning solution. All coding ran in the iMac with the 3.5 GHz Intel Core i5 for the processor and 8 GB 1600 MHz DDR3 for the memory.

6.2 Future work

The research on resources allocation and task offloading in VEC is still in the early stage. Many issues remain to be solved well. There are several directions to extend this dissertation, which can be briefly outlined as follows:

- One of the elements affecting the transmission latency is the available bandwidth allocated to the vehicle; we considered a constant value of the bandwidth. In practice, the bandwidth of the access link assigned to the vehicle is not fixed due to the changeable vehicles' density during time slots. Thus, the bandwidth allocation ratio should be considered as an optimization parameter in the future. Energy efficiency, power consumption, and throughput are the other QoS parameters that should be investigated well in resource allocation.

- To enhance the performance of the proposed algorithm finding the new locations of the vehicles to deliver the result is a significant issue. One way to get the new vehicle's location is that the vehicle's speed and direction information are constantly monitored. Alternatively, mobility predictions utilizing machine learning techniques would be a promising solution. Adding intelligent mobility prediction to our proposed models can allow the learning agent to make a better decision for task offloading.

- We can extend our work for task offloading management and resource allocation by applying other machine learning methods such as federated learning. Utilizing federated learning in VEC is still in the early stages and is mainly used in radio frequency parts. However, we can apply federated learning as a central orchestrator to manage task offloading in the central and edge clouds. In other words, federated learning can be used in the same architecture of our work with different vehicular applications in a highly dynamic network. In addition, we can apply reinforcement learning in federated learning.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] E. Ahmed and H. Gharavi. Cooperative vehicular networking: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 19(3):996–1014, 2018.

[3] A. Boukerche and E. Robson. Vehicular cloud computing: Architectures, applications, and mobility. *Computer networks*, 135:171–189, 2018.

[4] L. Bracciale, M. Bonola, P. Loreti, G. Bianchi, R. Amici, and A. Rabuffi. CRAWDAD dataset roma/taxi. `https://crawdad.org/roma/taxi/20140717`, 2014.

[5] J. Brownlee. *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery, 2018.

[6] S. Bubeck et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[7] M.-H. Chen, M. Dong, and B. Liang. Joint offloading decision and resource allocation for mobile cloud with computing access point. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3516–3520. IEEE, 2016.

[8] R. Chuentawat and Y. Kan-ngan. The comparison of PM2. 5 forecasting methods in the form of multivariate and univariate time series based on support vector machine and genetic algorithm. In *2018 15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pages 572–575. IEEE, 2018.

[9] J. Cui, Y. Liu, and A. Nallanathan. The application of multi-agent reinforcement learning in UAV networks. In *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2019.

[10] J. Cui, Y. Liu, and A. Nallanathan. Multi-agent reinforcement learning-based resource allocation for UAV networks. *IEEE Transactions on Wireless Communications*, 19(2):729–743, 2019.

[11] Y. Dai, D. Xu, S. Maharjan, G. Qiao, and Y. Zhang. Artificial intelligence empowered edge computing and caching for internet of vehicles. *IEEE Wireless Communications*, 26(3):12–18, 2019.

[12] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang. Joint load balancing and offloading in vehicular edge computing and networks. *IEEE Internet of Things Journal*, 6(3):4377–4387, 2018.

[13] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[14] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek. Offloading in mobile edge computing: Task allocation and computational frequency scaling. *IEEE Transactions on Communications*, 65(8):3571–3584, 2017.

[15] B. Dissanayake, O. Hemachandra, N. Lakshitha, D. Haputhanthri, and A. Wijayasiri. A comparison of arimax, var and lstm on multivariate short-term traffic volume forecasting. In *Conference of Open Innovations Association, FRUCT*, number 28, pages 564–570. FRUCT Oy, 2021.

[16] J. Du, F. R. Yu, X. Chu, J. Feng, and G. Lu. Computation offloading and resource allocation in vehicular networks based on dual-side cost minimization. *IEEE Transactions on Vehicular Technology*, 68(2):1079–1092, 2018.

[17] T. L. Duc, R. G. Leiva, P. Casari, and P.-O. Östberg. Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey. *ACM Computing Surveys (CSUR)*, 52(5):1–39, 2019.

[18] M. S. Elbamby, M. Bennis, and W. Saad. Proactive edge computing in latency-constrained fog networks. In *2017 European conference on networks and communications (EuCNC)*, pages 1–6. IEEE, 2017.

[19] P. Fang, Y. Zhao, Z. Liu, J. Gao, and Z. Chen. Resource allocation strategy for MEC system based on VM migration and RF energy harvesting. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–6. IEEE, 2020.

[20] J. Feng, Z. Liu, C. Wu, and Y. Ji. AVE: Autonomous vehicular edge computing framework with ACO-based scheduling. *IEEE Transactions on Vehicular Technology*, 66(12):10660–10675, 2017.

[21] Y. Ge, Y. Zhang, Q. Qiu, and Y.-H. Lu. A game theoretic resource allocation for overall energy minimization in mobile cloud computing system. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pages 279–284, 2012.

[22] Y. He, N. Zhao, and H. Yin. Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, 67(1):44–55, 2017.

[23] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[24] R. Q. Hu et al. Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 67(11):10190–10203, 2018.

[25] D. Janardhanan and E. Barrett. CPU workload forecasting of machines in data centers using LSTM recurrent neural networks and ARIMA models. In *2017 12th international conference for internet technology and secured transactions (ICITST)*, pages 55–60. IEEE, 2017.

[26] X. Jiang, F. R. Yu, T. Song, and V. C. Leung. Resource allocation of video streaming over vehicular networks: a survey, some research issues and challenges. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[27] C. Jin, Z. Allen-Zhu, S. Bubeck, and M. I. Jordan. Is Q-learning provably efficient? *Advances in neural information processing systems*, 31, 2018.

[28] A. J. Kadhim and S. A. H. Seno. Maximizing the utilization of fog computing in internet of vehicle using sdn. *IEEE Communications Letters*, 23(1):140–143, 2018.

[29] C. A. Kamienski, F. F. Borelli, G. O. Biondi, I. Pinheiro, I. D. Zyrianoff, and M. Jentsch. Context design and tracking for iot-based energy management in smart cities. *IEEE Internet of Things Journal*, 5(2):687–695, 2017.

[30] E. Karimi, Y. Chen, and B. Akbari. Task offloading in vehicular edge computing networks via deep reinforcement learning. *Computer Communications*, 2022.

[31] J. Kumar, R. Goomer, and A. K. Singh. Long short term memory recurrent neural network (LSTM-RNN) based workload forecasting model for cloud datacenters. *Procedia Computer Science*, 125:676–682, 2018.

[32] S. Lai, R. Zhao, S. Tang, J. Xia, F. Zhou, and L. Fan. Intelligent secure mobile edge computing for beyond 5G wireless networks. *Physical Communication*, 45:101283, 2021.

[33] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*. Citeseer, 2000.

[34] M.-A. Lèbre, F. L. Mouël, E. Ménard, J. Dillschneider, and R. Denis. Vanet applications: Hot use cases. *arXiv preprint arXiv:1407.4088*, 2014.

[35] L. Lei, Y. Tan, K. Zheng, S. Liu, K. Zhang, and X. Shen. Deep reinforcement learning for autonomous internet of things: Model, applications and challenges. *IEEE Communications Surveys & Tutorials*, 22(3):1722–1760, 2020.

[36] C. Li, J. Xia, F. Liu, D. Li, L. Fan, G. K. Karagiannidis, and A. Nallanathan. Dynamic offloading for multiuser muti-CAP MEC networks: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, 70(3):2922–2927, 2021.

[37] J. Li, C. Natalino, D. P. Van, L. Wosinska, and J. Chen. Resource management in fog-enhanced radio access network to support real-time vehicular services. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 68–74. IEEE, 2017.

[38] M. Li, J. Gao, L. Zhao, and X. Shen. Deep reinforcement learning for collaborative edge computing in vehicular networks. *IEEE Transactions on Cognitive Communications and Networking*, 6(4):1122–1135, 2020.

[39] L. Liang, H. Ye, and G. Y. Li. Spectrum sharing in vehicular networks based on multi-agent reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 37(10):2282–2292, 2019.

[40] L. Liang, H. Ye, G. Yu, and G. Y. Li. Deep-learning-based wireless resource allocation with application to vehicular networks. *Proceedings of the IEEE*, 108(2):341–356, 2019.

[41] Q. Liang, J. Zhang, Y.-h. Zhang, and J.-m. Liang. The placement method of resources and applications based on request prediction in cloud data center. *Information Sciences*, 279:735–745, 2014.

[42] G. Liu, J. Wang, Y. Tian, Z. Yang, and Z. Wu. Mobility-aware dynamic service placement for edge computing. *EAI Endorsed Transactions on Internet of Things*, 5(19), 2019.

[43] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang. Vehicular edge computing and networking: A survey. *Mobile networks and applications*, 26(3):1145–1168, 2021.

[44] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*, pages 372–382. IEEE, 2017.

[45] Y. Liu, S. Wang, J. Huang, and F. Yang. A computation offloading algorithm based on game theory for vehicular edge networks. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[46] Y. Liu, H. Yu, S. Xie, and Y. Zhang. Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks. *IEEE Transactions on Vehicular Technology*, 68(11):11158–11168, 2019.

[47] X. Ma, J. Zhao, Y. Gong, and Y. Wang. Key technologies of mec towards 5g-enabled vehicular networks. In *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pages 153–159. Springer, 2017.

[48] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358, 2017.

[49] L. Matignon, G. J. Laurent, and N. Le Fort-Piat. Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31, 2012.

[50] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[51] J. L. D. Neto, S.-Y. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci. Uloof: A user level online offloading framework for mobile edge computing. *IEEE Transactions on Mobile Computing*, 17(11):2660–2674, 2018.

[52] C. Nguyen, C. Klein, and E. Elmroth. Multivariate LSTM-based location-aware workload prediction for edge data centers. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 341–350. IEEE, 2019.

[53] P.-O. Östberg, J. Byrne, P. Casari, P. Eardley, A. F. Anta, J. Forsman, J. Kennedy, T. Le Duc, M. N. Marino, R. Loomba, et al. Reliable capacity provisioning for distributed cloud/edge/fog computing applications. In *2017 European conference on networks and communications (EuCNC)*, pages 1–6. IEEE, 2017.

[54] S. Ouhame and Y. Hadi. Multivariate workload prediction using vector autoregressive and stacked LSTM models. In *Proceedings of the New Challenges in Data Sciences: Acts of the Second Conference of the Moroccan Classification Society*, pages 1–7, 2019.

[55] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[56] B. Peng, G. Seco-Granados, E. Steinmetz, M. Fröhle, and H. Wymeersch. Decentralized scheduling for cooperative localization with deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 68(5):4295–4305, 2019.

[57] H. Peng and X. Shen. Multi-agent reinforcement learning based resource management in MEC-and UAV-assisted vehicular networks. *IEEE Journal on Selected Areas in Communications*, 39(1):131–141, 2020.

[58] H. Peng and X. S. Shen. Deep reinforcement learning based resource management for multi-access edge computing in vehicular networks. *IEEE Transactions on Network Science and Engineering*, 7(4):2416 – 2428, 2020.

[59] H. Peng, Q. Ye, and X. S. Shen. SDN-based resource management for autonomous vehicular networks: A multi-access edge computing approach. *IEEE Wireless Communications*, 26(4):156–162, 2019.

[60] J. Plachy, Z. Becvar, and E. C. Strinati. Dynamic resource allocation exploiting mobility prediction in mobile edge computing. In *27th annual international symposium on personal, indoor, and mobile radio communications (PIMRC)*, pages 1–6. IEEE, 2016.

[61] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb. Survey on multi-access edge computing for internet of things realization. *IEEE Communications Surveys & Tutorials*, 20(4):2961–2991, 2018.

[62] G. Qiao, S. Leng, K. Zhang, and Y. He. Collaborative task offloading in vehicular edge multi-access networks. *IEEE Communications Magazine*, 56(8):48–54, 2018.

[63] S. Raza, S. Wang, M. Ahmed, and M. R. Anwar. A survey on vehicular edge computing: architecture, applications, technical issues, and future directions. *Wireless Communications and Mobile Computing*, 2019, 2019.

[64] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato. Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective. *IEEE Communications Surveys & Tutorials*, 22(1):38–67, 2019.

[65] H. Sak, A. Senior, and F. Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

[66] M. A. Salahuddin, A. Al-Fuqaha, and M. Guizani. Reinforcement learning for resource provisioning in the vehicular cloud. *IEEE Wireless Communications*, 23(4):128–135, 2016.

[67] M. Tang and V. W. Wong. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing*, 2020.

[68] M. M. K. Tareq, O. Semiari, M. A. Salehi, and W. Saad. Ultra reliable, low latency vehicle-to-infrastructure wireless communications with edge computing. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2018.

[69] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung. Dynamic service migration and workload scheduling in edge-clouds. *Performance Evaluation*, 91:205–228, 2015.

[70] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang. Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. *IEEE Transactions on Wireless Communications*, 16(8):4924–4938, 2017.

[71] S. Wang, R. Urgaonkar, T. He, M. Zafer, K. Chan, and K. K. Leung. Mobility-induced service migration in mobile micro-clouds. In *2014 IEEE military communications conference*, pages 835–840. IEEE, 2014.

[72] S. Wang, J. Xu, N. Zhang, and Y. Liu. A survey on service migration in mobile edge computing. *IEEE Access*, 6:23511–23528, 2018.

[73] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang. NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning. In *Proceedings of the International Symposium on Quality of Service*, pages 1–10, 2019.

[74] B. Yang, X. Cao, J. Bassey, X. Li, and L. Qian. Computation offloading in multi-access edge computing: A multi-task learning approach. *IEEE transactions on mobile computing*, 20(9):2745–2762, 2020.

[75] B. Yang, X. Cao, K. Xiong, C. Yuen, Y. L. Guan, S. Leng, L. Qian, and Z. Han. Edge intelligence for autonomous driving in 6G wireless system: Design challenges and solutions. *IEEE Wireless Communications*, 28(2):40–47, 2021.

[76] J. Yang, C. Liu, Y. Shang, B. Cheng, Z. Mao, C. Liu, L. Niu, and J. Chen. A cost-aware auto-scaling approach using the workload prediction in service clouds. *Information Systems Frontiers*, 16(1):7–18, 2014.

[77] H. Ye, L. Liang, G. Y. Li, J. Kim, L. Lu, and M. Wu. Machine learning for vehicular networks: Recent advances and application examples. *ieee vehicular technology magazine*, 13(2):94–101, 2018.

[78] B. Yongacoglu, G. Arslan, and S. Yüksel. Learning team-optimality for decentralized stochastic control and dynamic games. *arXiv preprint arXiv:1903.05812*, 2019.

[79] R. Yu, J. Ding, S. Maharjan, S. Gjessing, Y. Zhang, and D. H. Tsang. Decentralized and optimal resource cooperation in geo-distributed mobile cloud computing. *IEEE Transactions on Emerging Topics in Computing*, 6(1):72–84, 2015.

[80] R. Yu, Y. Zhang, S. Gjessing, W. Xia, and K. Yang. Toward cloud-based vehicular networks with efficient resource management. *IEEE Network*, 27(5):48–55, 2013.

[81] J. Zhang, H. Guo, and J. Liu. Adaptive task offloading in vehicular edge computing networks: a reinforcement learning based scheme. *Mobile Networks and Applications*, 25(5):1736–1745, 2020.

[82] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. Zhang. Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading. *IEEE Vehicular Technology Magazine*, 12(2):36–44, 2017.

[83] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. Zhang. Predictive offloading in cloud-driven vehicles: using mobile-edge computing for a promising network paradigm. *IEEE Vehicular Technology Magazine*, 12(2), 2017.

[84] K. Zhang, Y. Mao, S. Leng, S. Maharjan, and Y. Zhang. Optimal delay constrained offloading for vehicular edge computing networks. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2017.

[85] K. Zhang, Z. Yang, and T. Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control*, pages 321–384, 2021.

[86] 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 of Things Journal*, 6(5):7635–7647, 2019.

[87] W. Zhang, Z. Zhang, and H.-C. Chao. Cooperative fog computing for dealing with big data in the internet of vehicles: Architecture and hierarchical resource management. *IEEE Communications Magazine*, 55(12):60–67, 2017.

[88] Y. Zhang, D. Niyato, and P. Wang. Offloading in mobile cloudlet systems with intermittent connectivity. *IEEE Transactions on Mobile Computing*, 14(12):2516–2529, 2015.

[89] T. Zhao, S. Zhou, X. Guo, Y. Zhao, and Z. Niu. A cooperative scheduling scheme of local cloud and internet cloud for delay-aware mobile cloud computing. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2015.

[90] Z. Zhao, W. Chen, X. Wu, P. C. Chen, and J. Liu. LSTM network: a deep learning approach for short-term traffic forecast. *IET Intelligent Transport Systems*, 11(2):68–75, 2017.

[91] Z. Zhou, C. Gao, C. Xu, Y. Zhang, S. Mumtaz, and J. Rodriguez. Social big-data-based content dissemination in internet of vehicles. *IEEE Transactions on Industrial Informatics*, 14(2):768–777, 2017.

[92] Z. Zhou, H. Yu, C. Xu, Y. Zhang, S. Mumtaz, and J. Rodriguez. Dependable content distribution in d2d-based cooperative vehicular networks: A big data-integrated coalition game approach. *IEEE Transactions on Intelligent Transportation Systems*, 19(3):953–964, 2018.