University of Technology Sydney
Faculty of Engineering and Information Technology
School of Electrical and Data Engineering

AI-empowered Communications

Huynh Nguyen Van

A Thesis Submitted
in Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

under the supervision of

Dr. Diep N. Nguyen
Dr. Hoang Dinh
Prof. Eryk Dutkiewicz

Sydney, Australia

September 2021

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, HUYNH NGUYEN VAN declare that this thesis, is submitted in fulfilment of the requirements for the award of DOCTOR OF PHILOSOPHY, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Student Name: Huynh Nguyen Van

Student Signature: Production Note: Signature removed prior to publication.

Date: December 13, 2021

ABSTRACT

AI-empowered Communications

by

Huynh Nguyen Van

Artificial Intelligence (AI) has been successfully applied to various areas and received great attention from both industry and academia. The recent advances in deep learning, convolutional neural networks, and reinforcement learning hold significant promise for solving intractable problems in future communication systems. This thesis aims to develop novel AI-based solutions to address different problems in communications, including resource allocation, security, and secure and effective computing.

Firstly, we propose an optimal and fast real-time resource slicing framework that maximizes the long-term profit of the network provider while considering the uncertainty of resource demand from tenants. To obtain the optimal resource allocation policy under the dynamics of slicing requests, e.g., uncertain service time and resource demands, we develop a deep reinforcement learning-based solution with an advanced deep learning architecture, called deep dueling. Extensive simulations show that the proposed solution yields up to 40% higher long-term average profit while being few thousand times faster, compared with state-of-the-art network slicing approaches.

Secondly, we introduce an optimal anti-jamming framework that allows wireless transceivers to effectively defeat jamming attacks. Specifically, while being attacked, wireless devices can either harvest energy from the jamming signals or backscatter the jamming signals to transmit data by using the ambient backscatter communication technique. Then, the deep dueling algorithm is adopted to learn about the jammer and obtain the optimal countermeasures thousand times faster than traditional reinforcement learning algorithms. Extensive simulations demonstrate that our solution can successfully defeat jamming attacks even with very high attack power levels/budgets. Interestingly, we show that by leveraging the jamming signals, the more frequently the jammer attacks the channel, the greater performance the system can achieve.

Finally, we propose a joint optimal coding and scheduling framework for secure and effective distributed learning (DL) over wireless edge networks. In particular, we use the coded computing technique to encode learning tasks by adding data/computing redundancy. As such, a learning task can be completed without waiting for straggling nodes. To account for the dynamics and uncertainty of wireless connections and edge nodes, several reinforcement learning algorithms are proposed to jointly obtain the optimal coding scheme and the best set of edge nodes for different learning tasks. Simulations show that the proposed framework reduces the average learning delay in wireless edge computing up to 66% compared with other DL approaches.

Acknowledgements

First and foremost, I would like to take this opportunity to express my deepest gratitude to my supervisors, Dr. Diep N. Nguyen, Dr. Hoang Dinh, and Prof. Eryk Dutkiewicz, for all the support, guidance, and encouragement. Without them, this dissertation would have been impossible. During my study, they not only guided me to pursue great and impactful research but also encouraged me to go beyond my boundary to do things that seem impossible. I am truly privileged and lucky to be supervised by them.

I would like to thank all my colleagues and friends at the University of Technology Sydney for their support, discussion, and friendship. They made my PhD life more colorful. My thanks also go to the SEDE admin team for handling all the paperwork and forms during my PhD study. I would like to thank the university and the Faculty of Engineering and Information Technology (FEIT) for giving me the International Research Scholarship and FEIT Scholarship. I also would like to thank Google for supporting my research through the Google PhD Fellowship program.

I would especially like to thank my dearest family for encouraging and supporting me. Their endless love gives me strength and power to overcome difficulties in my life. My final words of love and gratitude are dedicated to my beloved wife, Hong Ngoc, for her infinite love, sacrifice, and patience.

Contents

Certificate of Original Authorship ii

Abstract iii

Acknowledgments v

Table of Contents vi

List of Publications x

List of Figures xv

Abbreviation xix

1 Introduction and Literature Review 1

1.1 Motivations 1

1.2 Literature Review and Contributions 3

1.2.1 Resource Allocation for Future Communication Systems . . . 3

1.2.2 Defeating Jamming Attacks in Wireless Communication Systems 7

1.2.3 Effective Distributed Computing 13

1.3 Thesis Organization 21

2 Background 23

2.1 Deep Learning 23

2.2 Reinforcement Learning 25

2.2.1 Markov Decision Process 26

2.2.2 Q-learning 26

2.3 Deep Reinforcement Learning . . . 30

2.3.1 Deep Q-learning . . . 30

2.3.2 Deep Dueling . . . 38

2.3.3 Complexity Analysis . . . 41

3 Optimal and Fast Real-time Resource Slicing with Deep Dueling Neural Networks 43

3.1 System Model . . . 44

3.2 Problem Formulation . . . 47

3.2.1 Decision Epoch . . . 47

3.2.2 State Space . . . 48

3.2.3 Action Space . . . 48

3.2.4 State Transition Probability . . . 49

3.2.5 Reward Function . . . 52

3.3 Performance Evaluation . . . 56

3.3.1 Parameter Setting . . . 56

3.3.2 Simulation Results . . . 57

3.4 Conclusion . . . 66

4 "Jam Me If You Can": Defeating Jammers with Deep Dueling Neural Network Architecture and Ambient Backscattering Augmented Communications 68

4.1 System Model . . . 69

4.1.1 Smart Jammer with Self-Interference Suppression Capability . 70

4.1.2 Ambient Backscattering-Augmented Communications . . . 72

4.1.3 System Operation . . . 75

4.2 Problem Formulation 78
4.2.1 State Space 79
4.2.2 Action Space 79
4.2.3 Immediate Reward 80
4.2.4 Optimization Formulation 81
4.3 Performance Evaluation 82
4.3.1 Parameter Setting 82
4.3.2 Simulation Results 84
4.4 Conclusion 94

5 Joint Coding and Scheduling Optimization for Distributed Learning over Wireless Edge Networks 96

5.1 System Model 98
5.1.1 Coded Computing for Distributed Learning over Wireless Edge Networks 100
5.1.2 Communication and Computation Models 101
5.1.3 Learning-Task Delay Minimization Problem 104
5.2 Coded Computing for Distributed Learning Formulation 107
5.2.1 State Space 107
5.2.2 Action Space 108
5.2.3 Immediate Reward 109
5.2.4 Long-Term Delay Minimization Formulation 110
5.3 Performance Analysis and Simulation Results 111
5.3.1 Parameter Setting 111
5.3.2 Simulation Results 114

5.4 Conclusion 123

6 Conclusions and Future Work 124

6.1 Conclusion 124

6.2 Future Works 126

A Proofs in Chapter 2 129

A.1 The proof of Theorem 2.1 129

B Proofs in Chapter 3 131

B.1 The proof of Theorem 3.1 131

B.2 The proof of Lemma 3.1 132

C Proofs in Chapter 5 133

C.1 The proof of Theorem 5.2 133

Bibliography 134

List of Publications

Books

B-1. D. T. Hoang, D. Niyato, D. I. Kim, **N. V. Huynh**, and S. Gong, *Ambient Backscatter Communication Networks*, Cambridge University Press, (Authored book) 2019. (*Partly corresponding to Chapter 4)*)

B-2. D. T. Hoang, **N. V. Huynh**, D. N. Nguyen, E. Hossain, and D. Niyato, *Deep Reinforcement Learning for Wireless Communications and Networking: Theory, Applications and Implementation*, Wiley, expected third quarter of 2022. (*Corresponding to Chapter 2)*)

Patent

P-1. "Jamming Signal Resistant Method and System," D. T. Hoang, D. N. Nguyen, **N. V. Huynh**, and E. Dutkiewicz, *UTS Provisional Patent Filed*, Feb. 2019. (*Partly corresponding to Chapter 4)*)

Journal Papers

J-1. **N. V. Huynh**, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "Optimal and Fast Real-Time Resource Slicing With Deep Dueling Neural Networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1455-1470, Jun. 2019. (*Corresponding to Chapter 3)*)

J-2. **N. V. Huynh**, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz, "Jam Me If You Can: Defeating Jammer with Deep Dueling Neural Network Architecture and Ambient Backscattering Augmented Communications," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 11, pp. 2603-2620, Nov. 2019. (*Corresponding to Chapter 4)*)

J-3. **N. V. Huynh**, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "DeepFake: Deep Dueling-based Deception Strategy to Defeat Reactive Jammers," *IEEE Transactions on Wireless Communications*, Early Access, May 2021 (*Partly corresponding to Chapter 4)*)

J-4. **N. V. Huynh**, D. N. Nguyen, D. T. Hoang, E. Dutkiewicz, and M. Mueck, "Ambient Backscatter: A Novel Method to Defend Jamming Attacks for Wireless Networks," *IEEE Wireless Communications Letters*, vol. 9, no. 2, pp. 175-178, Feb. 2020. (*Partly corresponding to Chapter 4)*)

J-5. **N. V. Huynh**, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "Joint Coding and Scheduling Optimization for Distributed Learning over Wireless Edge Networks," *IEEE Journal on Selected Areas in Communications*, accepted, 2021. (*Corresponding to Chapter 5)*)

J-6. **N. V. Huynh**, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz, "Optimal Beam Association for High Mobility mmWave Vehicular Networks: Lightweight Parallel Reinforcement Learning Approach," *IEEE Transactions on Communications*, Early Access, Jun. 2021.

J-7. **N. V. Huynh**, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, D. Niyato, and P. Wang, "Optimal and Low-Complexity Dynamic Spectrum Access for RF-Powered Ambient Backscatter System with Online Reinforcement Learning," *IEEE Transactions on Communications*, vol. 67, no. 8, pp. 5736-5752, Aug. 2019. (Q1, IF = 5.646)

J-8. **N. V. Huynh**, D. N. Nguyen, D. T. Hoang, T. X. Vu, E. Dutkiewicz, and S. Chatzinotas, "Defeating Super-Reactive Jammers With Deception Strategy: Modeling, Signal Detection, and Performance Analysis," *IEEE Transactions on Wireless Communications*, **under major revision**.

Conference Papers

C-1. **N. V. Huynh**, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "Real-Time

Network Slicing With Uncertain Demand: A Deep Learning Approach," *IEEE ICC*, Shanghai, China, 20-24 May 2019. (*Corresponding to Chapter 3)*)

C-2. **N. V. Huynh**, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz, "Defeating Reactive Jammers with Deep Dueling-based Deception Mechanism," *IEEE ICC*, Montreal, Canada, 14-18 June 2021. (*Corresponding to Chapter 4)*)

C-3. **N. V. Huynh**, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, and M. Mueck, "Defeating Smart and Reactive Jammers with Unlimited Power," *IEEE WCNC*, Virtual Conference, 25-28 May 2020. (*Partly corresponding to Chapter 4)*)

C-4. **N. V. Huynh**, D. N. Nguyen, D. T. Hoang, E. Dutkiewicz, M. Mueck, and S. Srikanteswara, "Defeating Jamming Attacks with Ambient Backscatter Communications," *IEEE ICNC*, Big Island, Hawaii, USA, 17-20 Feb. 2020. (*Partly corresponding to Chapter 4)*)

C-5. **N. V. Huynh**, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "Dynamic optimal coding and scheduling for distributed learning over wireless edge networks," *IEEE GLOBECOM*, Madrid, Spain, Dec. 2021. (*Corresponding to Chapter 5)*)

C-6. **N. V. Huynh**, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz, "Optimal Beam Association in mmWave Vehicular Networks with Parallel Reinforcement Learning," *IEEE GLOBECOM*, Taipei, Taiwan, 7 - 11 December, 2020.

C-7. **N. V. Huynh**, D. T. Hoang, D. N. Nguyen, E. Dutkiewicz, D. Niyato, and P. Wang, "Reinforcement Learning Approach for RF-Powered Cognitive Radio Network with Ambient Backscatter," *IEEE GLOBECOM*, Abu Dhabi, UAE, 9-13 Dec. 2018.

Other Papers

O-1. C. T. Nguyen, **N. V. Huynh**, N. H. Chu, Y. M. Saputra, D. T. Hoang, D. N. Nguyen, Q. V. Pham, D. Niyato, E. Dutkiewicz, and W. J.Hwang, "Transfer Learning for Future Wireless Networks: A Comprehensive Survey," submitted to *Proceedings of the IEEE*, **under review**.

O-2. N.-T. Nguyen, D. N. Nguyen, D. T. Hoang, **N. V. Huynh**, E. Dutkiewicz, N.-H. Nguyen, and Q.-T. Nguyen, "Time Scheduling and Energy Trading for Heterogeneous Wireless-Powered and Backscattering-based IoT Networks," *IEEE Transactions on Wireless Communications*, accepted 2021.

O-3. C. T. Nguyen, Y. M. Saputra, **N. V. Huynh**, N.-T. Nguyen, T. V. Khoa, B. M. Tuan, D. N. Nguyen, D. T. Hoang, T. X. Vu, E. Dutkiewicz, S. Chatzinotas, and B. Ottersten, "A Comprehensive Survey of Enabling and Emerging Technologies for Social Distancing — Part I: Fundamentals and Enabling Technologies," *IEEE Access*, vol. 8, pp. 153479-153507, Aug. 2020.

O-4. C. T. Nguyen, Y. M. Saputra, **N. V. Huynh**, N.-T. Nguyen, T. V. Khoa, B. M. Tuan, D. N. Nguyen, D. T. Hoang, T. X. Vu, E. Dutkiewicz, S. Chatzinotas, and B. Ottersten, "A Comprehensive Survey of Enabling and Emerging Technologies for Social Distancing — Part II: Emerging Technologies and Open Issues," *IEEE Access*, vol. 8, pp. 154209-154236, Aug. 2020.

O-5. N. T. Nguyen, D. N. Nguyen, D. T. Hoang, **N. V. Huynh**, N. H. Nguyen, Q. T. Nguyen, and E. Dutkiewicz, "Energy Trading and Time Scheduling for Energy-Efficient Heterogeneous Low-Power IoT Networks," *IEEE GLOBECOM*, Taipei, Taiwan, 7 - 11 December, 2020.

O-6. N. T. Nguyen, **N. V. Huynh**, D. T. Hoang, D. N. Nguyen, N. H. Nguyen, Q. T. Nguyen, and E. Dutkiewicz, "Energy Management and Time Scheduling for Heterogeneous IoT Wireless-Powered Backscatter Networks," *IEEE ICC*, Shanghai, China, 20-24 May 2019.

O-7. N. H. Chu, D. T. Hoang, D. N. Nguyen, **N. V. Huynh**, and E. Dutkiewicz, "Fast or Slow: An Autonomous Speed Control Approach for UAV-Assisted IoT Data Collection Networks," *IEEE WCNC*, Nanjing, China, 29 Mar - 1 Apr 2021.

O-8. T. T. Vu, **N. V. Huynh**, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz , "Offloading Energy Efficiency with Delay Constraint for Cooperative Mobile Edge Computing Networks," *IEEE GLOBECOM*, Abu Dhabi, UAE, 9-13 Dec. 2018.

List of Figures

2.1 Typical deep neural network architecture. 24

2.2 Reinforcement learning. 25

2.3 Q-learning model. 27

2.4 Deep Q-learning model. 31

2.5 Flow chart of the deep Q-learning algorithm. 34

2.6 Deep dueling neural network architecture. 36

3.1 Network resource slicing system model. 44

3.2 Uniformization technique. 50

3.3 The average reward when optimizing with one resource and three resources. 58

3.4 The average reward of the system when the immediate reward of class-3 is varied. 59

3.5 The number of request running in the system of (a) greedy algorithm, (b) Q-learning algorithm, and (c) deep dueling algorithm when the immediate reward of class-3 is varied. 60

3.6 The average reward of the system when the immediate reward of class-3 is varied. 61

3.7 The number of request running in the system of (a) Q-learning algorithm (10^6 iterations), (b) Q-learning algorithm (10^7 iterations), and (c) deep dueling algorithm (20,000 iterations) when the immediate reward of class-3 is varied. The dash lines are results of the greedy algorithm. 62

3.8 The probabilities of accepting a request from classes when the maximum available resources of the system is (a) 4 times, (b) 10 times, and (c) 20 times of resources requested by a slice. 64

3.9 The convergence of reinforcement learning algorithms when the radio, computing, storage resources are 400 Mbps, 8 CPUs, and 4 GB, respectively with (a) 10^6 iteration and (b) 20,000 iterations. . . . 65

3.10 The convergence of reinforcement learning algorithms when (a) the radio, computing, storage resources are 1 Gbps, 20 CPUs, and 10 GB, respectively and (b) the radio, computing, storage resources are 2 Gbps, 20 GB, and 40 CPUs, respectively. 66

3.11 The performance of deep dueling algorithm with different learning rates. 67

4.1 System model 70

4.2 Function and circuit diagram of the proposed anti-jamming system . 72

4.3 Average throughputs of the proposed solution and the RA technique vs. P_{avg}. 84

4.4 (a) Average throughput (packets/time unit), (b) Packet loss (packets/time unit), (c) Average number of packets in the data queue, (d) PDR, (e) Delay (time/units) vs. η. 86

4.5 (a) Average throughput (packets/time unit), (b) Packet loss (packets/time unit), (c) Average number of packets in the data queue, (d) PDR, (e) Delay (time/units) vs. P_{avg}. 88

4.6 (a) Average throughput (packets/time unit), (b) Packet loss (packets/time unit), (c) Average number of packets in the data queue, (d) PDR, (e) Delay (time/units) vs. $\widehat{d}_{\mathrm{t}}$. 89

4.7 (a) Average throughput (packets/time unit), (b) Packet loss (packets/time unit), (c) Average number of packets in the data queue, (d) PDR, (e) Delay (time/units) vs. λ. 91

4.8 (a) Average throughput (packets/time unit), (b) Packet loss (packets/time unit), (c) Average number of packets in the data queue, (d) PDR, (e) Delay (time/units) vs. t_{th}. 92

4.9 Convergence rates when (a) $D = E = 10$ and (b) $D = E = 20$. 93

5.1 System model for coded distributed learning over wireless edge network. Here, we illustrate the case when learning task $\mathcal{D}^{(2)}$ is processed with $(n = 4, k = 2)$ MDS code. The sub-learning tasks are sent to edge nodes $1, 2, 3,$ and N to process. Then, when edge node 2 is disconnected, and edge node N is straggling, the learning task $\mathcal{D}^{(2)}$ still can be completed by using computed results from edge nodes 1 and 3. 97

5.2 Convergence rates of learning algorithms. 114

5.3 (a) Average number of tasks waiting in the queue, (b) task dropping probability, and (c) average delay of learning tasks in the system vs. task arrival probability. 115

5.4 (a) Average number of tasks waiting in the queue, (b) task dropping probability, and (c) average delay of learning tasks in the system vs. processing time. 117

5.5 (a) Average number of tasks waiting in the queue, (b) task dropping probability, and (c) average delay of learning tasks in the system vs. disconnection probability of links. 119

5.6 (a) Average number of tasks waiting in the queue, (b) task dropping probability, and (c) average delay of learning tasks in the system vs. the rate parameter λ (in the exponential distribution) of the stochastic computing time of edge nodes. 121

5.7 (a) Average number of tasks waiting in the queue, (b) task dropping probability, and (c) average delay of learning tasks in the system vs. task size. 122

Abbreviation

AI	Artificial Intelligence
DL	Distributed learning
M2M	Machine-to-machine
QoS	Quality of service
SINR	Signal-to-interference-plus-noise ratio
SDN	Software-defined networking
NFV	Network functions virtualization
SMDP	Semi-Markov decision processes
FHSS	Frequency-hopping spread spectrum
DSSS	Direct sequence spread spectrum
RA	Rate adaptation
RF	Radio frequency
MDP	Markov decision process
MDS	Maximum distance separable
ARC	Aligned repetition coding
AMC	Aligned minimum distance separable coding
MEC	Mobile edge server
RMO	Resource management and orchestration
URLLC	Ultra-reliable and low-latency communications
SiS	Self-interference suppression
BER	Bit error rate
PDR	Packet delivery ratio
ARP	Action-replay process

Chapter 1

Introduction and Literature Review

This chapter first provides the background on the development and current challenges of communication technologies. Then, the up-to-date solutions in addressing these issues are comprehensively discussed. Finally, the main contributions as well as the structure of this thesis are highlighted at the end of this chapter.

1.1 Motivations

Communication technologies have been explosively evolving over the past years to support various aspects of our daily life, from healthcare, smart cities, logistics to transportation. This has opened new frontiers for the future's data-centric society. However, these new applications generate untraditional workloads and require more efficient and reliable infrastructure. The latest Cisco Visual Networking Index [2] forecasts the number of networked devices will be 29.3 billion by 2023, of which 45% will be mobile-connected. Machine-to-machine (M2M) is expected to be the fastest growing mobile connection type as IoT applications continue to gain attraction in consumer and business environments. However, legacy mobile networks are mostly designed to provide services for mobile broadband users and are unable to meet adjustable parameters like priority and quality of service (QoS) of emerging services with different requirements. Consequently, service providers may find difficulties in concurrently managing multiple interconnected resources and dynamically allocating them to users in a real-time manner to maximize their long-term revenue.

In addition, connecting billions of wireless devices to the Internet faces unprecedented security issues, especially for resource-constrained devices in IoT networks. In particular, due to its broadcast nature, wireless communications are particularly vulnerable to jamming attacks. By injecting interference to the wireless communication channel (i.e., deliberate jamming), a jammer can degrade the effective signal-to-interference-plus-noise ratio (SINR), thereby disrupting or even bringing down legitimate communications links. The jamming attacks can be easily launched by using commercial off-the-shelf products [3, 4] and have a significant detriment to wireless applications, especially for mission-critical systems (e.g., cyber-physical systems in traffic safety, industry automation or military missions). Therefore, there is an urgent demand for effective countermeasures to prevent and mitigate the impacts of jamming attacks in future wireless communication networks.

In future communication systems, computing and caching resources are expected to be deployed at the network edge to support time-sensitive applications and reduce energy consumption, resulting in a highly sophisticated network [5], [6]. In this case, the network can be considered as a distributed environment in which edge nodes can collect data and learn locally without sending them to the cloud. This approach can significantly reduce the latency, communication costs, and preserve data privacy compared to conventional centralized solutions. However, one major challenge in this distributed paradigm is the straggling problems at both edge nodes and wireless links that can significantly prolong the computation delay of the system.

Facing these new challenges and demands, conventional approaches expose several limitations. First, with the massive number of users' devices, the expansion of network scale, and the diversity of services in the new era of communications, the amount of data generated by applications, users, and networks is expected to experience an explosive growth [2]. However, conventional solutions are infeasible to process and/or utilize the data to learn useful information to improve the system performance. Second, existing algorithms are not effective in dealing with the dynamic and uncertainty of the network environments, resulting in poor performance [1]. Finally, with conventional solutions, the complete information of the system is usually required to effectively obtain solutions. Nevertheless, this information may not be available in advance in practice, and thus limiting the applications of conventional approaches.

To address the limitations of the current solutions, AI is a promising approach. In particular, AI has been emerging as a disruptive technique and architectural framework to intelligently solve complex and large-scale problems in many areas, including search engines and speech recognition, medical diagnosis, and computer vision. Thus, AI approaches can be adopted to manage the growing complexity and scale of future communication networks. This thesis aims to develop AI-based solutions to effectively and intelligently address the aforementioned issues in communications, including resource allocation, security, and effective and secure distributed computing.

1.2 Literature Review and Contributions

This section first reviews the advantages and limitations of existing works in addressing the aforementioned issues. Then, the main contributions of this thesis are highlighted.

1.2.1 Resource Allocation for Future Communication Systems

1.2.1.1 Literature Review

In order to enhance operators' products for vertical enterprises and provide service customization for emerging massive connections, as well as to give more control to enterprises and mobile virtual network operators, the concept of network slicing has been recently introduced to allow the independent usage of a part of network resources by a group of mobile terminals with special requirements. Network slicing was introduced by Next Generation Mobile Networks Alliance [7], and it has quickly received paramount attention from both academia and industry. In general, network slicing is a novel virtualization paradigm that enables multiple logical networks, i.e., slices, to be created according to specific technical or commercial demands and simultaneously run on top of the physical network infrastructure. The core idea of the network slicing is using software-defined networking (SDN) and network functions virtualization (NFV) technologies for virtualizing the physical infrastructure and controlling network operations. In particular, SDN provides a separation between the network control and data planes, improving the flexibility of network function management and efficiency of data transfer. Meanwhile, NFV allows various network functions to be virtualized, i.e., in virtual machines. As a result, the functions can be moved to different locations, and the corresponding virtual machines can be migrated to run on commoditized hardware dynamically depending on the demand and requirements [8], [9].

A number of research works have been introduced recently to address the network slicing resource allocation problem for the network provider [10]- [20]. In particular, the authors in [10] and [11] developed a two-tier admission control and resource allocation model to answer two fundamental questions, i.e., whether a slice request is accepted and how much radio resource is allocated to the accepted slice. To address this problem, the authors in [10] used an extensive searching method to achieve the globally optimal resource allocation solution for the network provider. However, this searching method cannot be applied to complex systems with a large number of resources. To address this problem, a heuristic scheme with three main steps was introduced in [11] to effectively allocate resources to the users. Yet this heuristic scheme cannot guarantee to achieve the optimal solution for the network provider. In addition, both network slicing resource allocation solutions proposed in [10] and [11] are heuristic methods with only radio resource taken into consideration. Thus, these solutions may not be appropriate to implement in dynamic network slicing resource allocation systems with a wide range of resource demands and services.

To deal with the dynamic of services, e.g., users' resource demands and their occupation time, the authors in [17] proposed a model to predict the future demand of slices, thereby maximizing the system resource utilization for the provider. The key idea of this approach is to use the Holt-Winters approach [21] to predict network slices' demands through tracking the traffic usage of users in the past. However, the accuracy of this prediction depends largely on the heavy-tailed distribution functions along with many control parameters such as scale factor, least-action trip planning, and potential gain. Furthermore, this approach only considers the short-term reward for the provider, and thus the long-term profit may not be able to obtain. Therefore, the authors in [18], [19], and [20] proposed reinforcement learning algorithms to address these problems. Among dynamic resource allocation methods, reinforcement learning has been considering to be the most effective way to maximize the long-term reward for dynamic systems as this method allows the network controller to adjust its actions in a real-time manner to obtain the optimal policy through the trial-and-error learning process [22]. However, this method often takes a long period to converge to the optimal solution, especially for a large-scale system.

1.2.1.2 Contributions

In all aforementioned work, the authors considered optimizing only radio resources, while other resources are completely ignored. However, as stated in [7, 23, 24], a typical network slice is composed of three main components, i.e., radio, computing, and storage. Consequently, considering only radio resources when orchestrating slices may be not able to achieve the optimal solution. Therefore, in this thesis, we introduce a Semi-Markov decision process (SMDP) framework [25] which allows the network provider to effectively allocate all three resources, i.e., radio, computing, and storage, to the users in a real-time manner. However, when we jointly consider combinatorial resources, i.e., radio, storage, and computing resources, together with the uncertainty of demands, the optimization problem becomes very complex as we need to simultaneously deal with a very large state space with multi-dimension and real-time dynamic decisions. Thus, we propose a novel network slicing framework with an advanced deep learning architecture using two streams of fully connected hidden layers, i.e., deep dueling neural network [26], combined with Q-learning method to effectively address this problem. It is worth noting that the VNF placement, routing, and connectivity resource allocation problems have been well investigated in the literature. Hence, in this thesis, we focus on dealing with the uncertainty, dynamics, and heterogeneity of slice requests. Specifically, the properties of slice requests may not be available in advance to the service provider. Moreover, different types of services have different resource requirements and these requirements can also be changed over time. Finally, each slice request can require different types of resources, e.g., radio, storage, and computing. Note that, our system model can be straightforwardly extended to the case with diverse connectivity among servers and data centers by accommodating additional states to the system state space. Note that, our proposed framework can handle very well a large state space (one of our key contributions in this thesis).

The main contributions are summarized as follows:

- We develop a dynamic network resource management model based on SMDP framework which allows the network provider to jointly allocate computing, storage, and radio resources to different slice requests in a real-time manner and maximize the long-term reward under a number of available resources.
- To find the optimal policy under the uncertainty of slice service demands, we deploy the Q-learning algorithm which can achieve the optimal solution

through reinforcement learning processes. However, the Q-learning algorithm may not be able to effectively achieve the optimal policy due to the curse-of-dimensionality problem when we jointly optimize multiple resources concurrently. Thus, we develop a deep double Q-learning approach which utilizes the advantage of a neural network to train the learning process of the Q-learning algorithm, thereby attaining much better performance than that of the Q-learning algorithm.

- To further enhance the performance of the system, we propose the novel network slicing approach with the deep dueling neural network architecture [26], which can outperform all other current reinforcement learning techniques in managing network slicing. The key idea of the deep dueling is using two streams of fully connected hidden layers to concurrently train the learning process of the Q-learning algorithm, thereby improving the training process and achieving an outstanding performance for the system.

- Finally, we perform extensive simulations with the aim of not only demonstrating the efficiency of proposed solutions in comparison with other conventional methods but also providing insightful analytical results for the implementation of the system. Importantly, through simulation results, we demonstrate that our proposed framework can improve the performance of the system up to 40% compared with other current approaches.

1.2.2 Defeating Jamming Attacks in Wireless Communication Systems

1.2.2.1 Literature Review

Anti-jamming has a very rich literature, originating from the early days of wireless communications. With most conventional anti-jamming solutions like frequency hopping or spread spectrum, legitimate transceivers often tend to "escape" or "hide" themselves from jammers. As an example, frequency-hopping spread spectrum

(FHSS) [27]- [34] allows a wireless device to quickly switch its operating frequency to other frequency channels. As soon as the jammer attacks the channel, the device will quickly change its operating frequency, thereby avoiding the jamming attack. In [27], the authors proposed an integrated bit-level FHSS for low-power wireless communication systems. The key idea of this approach is exploiting the frequency agility of bulk acoustic wave resonators. In [28], the authors proposed a hybrid approach using FHSS and direct sequence spread spectrum (DSSS) to cope with fast-following jammers. Using a stochastic game framework, the authors of [30] studied the strategic interaction between jammers and legitimate users. In particular, the jammer and the transmitter are considered as two players playing with each other to obtain the optimal attack and defense strategies, respectively. Through the minimax-Q learning algorithm, the transmitter can gradually obtain the optimal defense policy, i.e., how to switch between different channels. The simulation results demonstrated that the proposed framework can maximize the spectrum-efficient throughput. Similarly, a game theory based anti-jamming framework for frequency hopping wireless communications was considered in [31]. Nevertheless, these game models require complete information of the jammer, which may not be available in advance in practice. In [32] and [33], the authors adopted the Q-learning and deep Q-learning algorithms that allow the transmitter to choose frequencies to hop when the jammer attacks the channel. In [34], the authors proposed a mode-frequency hopping scheme which jointly uses the mode hopping and the traditional FH for anti-jamming in cognitive radio networks. However, the main limitation of the FHSS technique is that it requires extra spectrum resources (for hopping to evade the jammers). In addition, with powerful jammers which can attack multiple channels simultaneously, FHSS may be less effective.

Besides the FHSS and DSSS techniques, the rate adaptation (RA) technique is also widely adopted, e.g., [35]- [37]. The key idea of the RA technique is to proactively or adaptively account for jamming attacks by operating at a lower transmission rate. In [35], the authors proposed an RA algorithm together with power control to mitigate jamming. Specifically, the algorithm consists of two modules: (i) a rate module for RA and (ii) a power control module for controlling the transmit power at legitimate transmitters. The experimental results demonstrated that the proposed algorithm can improve the network throughput by 150% under jamming attacks. However, in [36], the authors revealed that the RA technique is not effective on a single channel. In [37], the authors investigated the performance of several state-of-the-art RA algorithms under different scenarios. The experimental results demonstrated that the existing RA algorithms are not effective to combat smart jamming attacks. Similar to the RA method, legitimate transmitters can also adapt (i.e., bump/push) their transmit power or beamforming vectors/matrices (to improve the effective SINR) to overcome or mitigate the effect of excessive interference. Nevertheless, this solution is either power-inefficient or not viable for low-power or hardware-constrained devices (e.g., in IoT applications).

In [3], the authors proposed a joint RA and FHSS technique to mitigate attacks from a reactive-sweep jammer. In particular, the jammer can sweep through a set of channels and sense the activities of legitimate transmitters to attack. To combat the jammer, the legitimate transmitters can either hop to a new channel and/or adapt their transmission rates. The authors modeled the system as a zero-sum Markov game and obtained the optimal policies for the transmitters by solving a constrained Nash equilibrium problem. Similar to [30, 31], this work also assumed complete information of the jammer in deriving the defense strategy. Another widely adopted approach in the literature is to use the ultra-wideband communications to hide the legitimate signals [38] under the noise.

1.2.2.2 Contributions

It is worth noting that, to allow the transmitters to effectively escape or hide from the jammer, most aforementioned solutions require additional resources (in spectrum bandwidth, or transmit power, or hardware capability). This fact limits the practical applications of these conventional methods, especially for low-power and/or low-cost communications systems (e.g., in IoT). In this thesis, we present a novel anti-jamming framework for these low-power and/or wireless-power communications devices. Such a framework allows these wireless transceivers to not only survive jamming attacks without requiring additional resources but also leverage the jamming signals to improve their transmission rate. To that end, we first observe that most existing anti-jamming solutions are reactive ones that are constrained by the lack of timely knowledge of jamming attacks (especially from smart jammers). Bringing together the latest advances in neural network architectures and ambient backscattering communications, this work allows wireless nodes to effectively "face" the jammer (instead of escaping) by first learning its jamming strategy, then adapting the rate and transmitting information right on the jamming signals (i.e., backscattering modulated information on the jamming signals). In our design, transmitters are augmented with an ambient backscattering communication circuit [39,40] and an energy harvester. When a jammer attacks the communication channel, the transmitter can leverage the jamming signals to backscatter information to the gateway or harvest energy from the jamming signal.

Our key idea is inspired by the latest advances in ambient backscatter communications and RF energy harvesting. An ambient backscatter communication-capable transceiver can modulate/backscatter the RF ambient signals (e.g., FM, AM radio signals) to transmit its own information. Interested readers of ambient backscatter communications are referred to [39], [40] and therein references. Note that ambient backscatter communications, unlike bistatic backscattering communications, does not require a dedicated RF source. Specifically, in bistatic backscattering communications, e.g., [41], [42], [43], [44], backscatter devices transmit information by backscattering the RF signals generated by a dedicated RF source which is controllable. With the recent development of RF energy harvesting, the transmitter can harvest energy from RF signals with a high efficiency. In particular, in [45], [46], and [47], the authors proposed novel designs for the rectenna, i.e., rectifier and antenna, and RF-DC converter to improve the amount of harvested RF energy from RF energy sources. The experimental results demonstrated that with the proposed rectenna, a tag can harvest RF energy and convert the harvested energy to DC with 70% efficiency under a wide range of input power. In [48], low power circuit designs for the voltage regulator and resistor to digital converter are also proposed. Differently, in [49] and [50], the authors aimed to maximize the amount of harvested energy by considering the joint information and energy cooperative problem with channel constraints.

To deal with the uncertainty (or unknowns) of jamming attacks and ambient RF signals, existing work often relies on reinforcement learning algorithms, e.g,. Q-learning under the framework of a Markov decision process (MDP). However, the Q-learning algorithm is notorious for its slow convergence to the optimal policy, especially when the state and action spaces are large. This makes the Q-learning algorithm pragmatically inapplicable. To overcome this problem, we design a novel deep reinforcement learning algorithm using a new dueling neural network architecture. The key idea of this algorithm is to separately estimate the advantage and value functions of each state-action pair. In this way, the learning rate can be significantly improved, allowing the transmitter to effectively learn about the jammer and attain the optimal countermeasures (e.g., adapt the transmission rate or backscatter or harvest energy or stay idle) thousand times faster than that of the conventional Q-learning algorithm. The transmitters not only successfully defeat jamming attacks, but also leverage jamming signals to significantly improve the performance for the system. Specifically, extensive simulation results show that our design (using ambient backscattering and the deep dueling neural network architecture) can improve the average throughput (under smart and reactive jamming attacks) by up to 426% and reduce the packet loss by 24%. By augmenting the ambient backscattering capability on devices and using our algorithm, it is interesting to observe that the (successful) transmission rate increases with the jamming power.

The major contributions can be summarized as follows.

- We propose a novel smart anti-jamming design using ambient backscatter, energy harvesting, and rate adaption techniques to defeat smart and reactive jammers. Based on this method, when a jammer attacks the channel, the transmitter can leverage jamming signals to transmit information using the ambient backscattering communication technique or harvest energy from the jamming signals. Alternatively, the transmitter can also choose to adapt the transmission rate to *actively*[*] transmit data.
- To lay a theoretical foundation for our design, we develop a dynamic approach using the MDP framework and Q- and deep Q-learning algorithms to maximize the average long-term throughput for systems under the uncertainty of jamming attacks and ambient RF signals. Unlike existing rate adaption methods that require the explicit or implicit knowledge of the interference/jamming level, our reinforcement learning-based RA framework does not require such information. It is also worth emphasizing that such an RA method is not susceptible/subject to the imperfect estimation/observations of the jamming signals (e.g., misdetection or false alarms).

[*]It refers to the conventional radio transmission that is different from the ambient backscattering transmission.

- To provide a practical solution for the theoretical Q-learning based approach above, we design a deep dueling neural network architecture which allows the system to quickly approach to the optimal defend policy. The key idea of this approach is implementing two streams of fully-connected hidden layers to separately and concurrently estimate the values of states and advantages of actions. As a result, the proposed deep dueling algorithm converges thousands times faster than Q-learning based algorithms.
- Finally, we perform extensive simulations with the aims of not only demonstrating the efficiency of proposed solutions in comparison with other conventional methods, but also providing insightful analytical results for the implementation of our framework.

1.2.3 Effective Distributed Computing

1.2.3.1 Literature Review

Recently, distributed computing has been introduced to offload complex computing task to edge devices for processing. In particular, a highly-complex task can be partitioned into multiple sub-tasks, and then these sub-tasks can be transmitted to several edge nodes for executing. In this way, the computation load at the centralized server can be offloaded to multiple edge nodes, and thus reducing the computation delay. As a result, distributed computing over wireless edge networks finds its applications in various emerging machine learning services that demand low delays such as autonomous vehicle, augmented reality, and virtual reality [51].

Although distributed learning over wireless edge networks has many advantages and applications in practice, it has been facing some technical challenges. First, it is pointed out that the performance of a distributed system is greatly affected by the straggling problem at the edge computing devices [52–54]. In particular, this straggling problem can cause unpredictable computing latency due to several factors such as resource sharing, maintenance activities, power regulations, and hardware configurations [52]. Consequently, the computing latency is usually determined by the slowest computing edge node. In the worst case, if an edge node is highly-straggling, the learning task will stay in the system for a long time. Consequently, the computing latency of the whole system will be significantly increased. Second, the data privacy protection of the conventional distributed learning is not guaranteed as the edge nodes can derive information from the assigned sub-learning tasks. Moreover, transmitting sub-learning tasks over wireless links may lead to another security concern as an attacker can eavesdrop the transmitted data over the wireless links. These security problems are very serious as private information such as finance data and medical records can be leaked to the third party. Third, distributed learning over wireless edge networks suffers from wireless link failures. Re-transmissions can be performed for failed messages. However, this may significantly increase the training time for the system.

To overcome the aforementioned challenges, the coded computing technique [52] has been introduced recently as a highly-effective solution. Specifically, the principle of the coded computing is utilizing advanced coding theoretic mechanisms to inject and leverage data/computation redundancy in order to mitigate the effects of the straggling problems as well as to protect the learning tasks' privacy at the edge nodes and over the wireless links [52, 54, 55]. With the coded computing technique, the computation latency is now determined by a group of the fastest edge computing devices [52, 53]. In other words, the coded computing technique does not require all assigned edge nodes to send back their computed results as in the traditional distributed edge computing. Similarly, the effects of unstable wireless links can be mitigated as the coded computing mechanism may ignore computed results from edge nodes with unstable wireless links if it has received sufficient computed results from other edge nodes with good wireless connections. Finally, the sub-learning tasks are encoded before sending to the assigned edge nodes, resulting in a high data privacy protection.

Recently, several works in the literature have been proposed to improve the performance of the coded computing mechanisms for distributed learning systems [52, 56–63]. In [52], the authors propose a new maximum distance separable (MDS) code design for matrix multiplication which is the most common operation in machine learning algorithms. In particular, the MDS code aims to encode k learning tasks into n coded learning tasks, where $n \geq k$. These encoded tasks are then distributed to n workers to execute. As soon as k workers complete their assigned tasks and send the results to the master node, the master node can decode them to obtain the expected results. In this way, the effect of straggling workers can be significantly mitigated. The authors then demonstrate that with n homogeneous workers, the MDS code can speed up the distributed matrix multiplication by a factor of $\log n$. The authors in [56] then extend the MDS code for large-scale matrix multiplication. Specifically, the key idea is to partition a large-scale matrix into sub-matrices. Then, the MDS code is applied for each sub-matrix. Although the matrix multiplication delay is similar to that of the conventional MDS code [52], thanks to shorter MDS codes, the proposed scheme can achieve a lower delay in encoding and decoding compared to that of the conventional MDS code. Similarly, the authors in [59] propose a gradient coding method based on the MDS code [52] for the synchronous gradient descent method. By using this code, the server can obtain the final gradient of any loss function even if a number of workers do not return their gradient results. The experimental results then demonstrate that the proposed gradient coding mechanism can significantly mitigate the straggling problems at workers compared to those of the uncoded schemes. Unlike [59], the authors in [60] propose to encode the dataset with built-in data redundancy for linear regression tasks. At every training step, the missing results from straggling nodes can be compensated by using the structured computing redundancy added by the proposed coding mechanism. Experiments then show that the proposed coding mechanism can significantly reduce the system computing delay.

It is worth noting that aforementioned works and others in the literature mostly focus on optimizing coding mechanisms only. However, their applications to wireless edge computing are not straightforward due to the inherent uncertainty of wireless channel quality. In particular, when the wireless link between the server and an edge node is disconnected, transmitted data (i.e., sub-learning tasks sent from the server and results sent from the edge node) need to be re-transmitted. This consequently drags out the training time of the whole system. For that, the authors in [54] introduce an effective coded computing framework for non-linear distributed machine learning, namely CodedFedL, that adds structured coding redundancy to mitigate straggling problems in both edge nodes and wireless links. Specifically, each edge node privately generates a matrix from a probability distribution with mean 0 and variance 1. This matrix is then applied on the weighted local dataset to compute a local parity dataset. All local parity datasets of edge nodes are then combined at the server to obtain a global parity dataset. Gradient over the global parity dataset will be used to replace missing gradient updates from straggling edge nodes. The size of the local parity datasets is the coding redundancy. The authors then formulate an optimization problem to find the optimal amount of coding redundancy based on the the conditions of edge nodes and wireless links. Numerical experiments then show that CodedFedL can speed up the training time of federated learning by up to 15 times compared to those of other approaches. In [64], the authors propose two coding schemes, namely Aligned Repetition Coding (ARC) and Aligned Minimum Distance Separable Coding (AMC), to mitigate the effect of straggling communication links. In particular, several reliable helper nodes are deployed to help the server in gradient aggregation operations. Under the ARC scheme, each node divides its gradient and sends to multiple helpers. In this way, gradient components from different nodes are aligned at the helpers, and thus mitigating the effect of straggling problems on server-to-node links. Differently, the AMC scheme allows each node to partition its gradient and encode with an MDS code and with the same generator matrix for all nodes. In [65], the authors point out that not only wireless computing nodes, but access points can also become stragglers, especially when the congestion occurs. As such, the authors propose a hierarchical code that can jointly mitigate the straggling problems at edge nodes and access points. Alternatively, the authors in [66] consider the strangling problems in coded distributed computing caused by link failures. In particular, the authors first model the link between the server and nodes as packet erasure channels. Then an MDS code is designed based on the packet erasure probability to reduce the system delay. Furthermore, the authors in [67] propose a new method to convert encoded data in a resource-efficient manner, namely convertible code. With this code, the authors can reduce the overhead in erasure-coded storage systems.

In [68], the authors introduce an extension of the MDS code considering delay/latency caused by unstable wireless links. In particular, the authors point out that under dynamically changing edge environments, wireless edge nodes may not be able to complete their computations within a given deadline. Thus, the authors propose a new coding mechanism that can incorporate partially-finished computations from edge nodes into the computation recovery at the server. Similarly, the authors in [69] aim to minimize the communication and computing delays by considering both wireless and computing impairments. In particular, the number of edge nodes for executing learning tasks is optimized based on the interference, imperfect channel state information, and straggling processors. Nevertheless, all the aforementioned solutions and others in the literature require complete environment information in advance, which may not be practical to implement. In reality, environment related parameters like link failures and straggling are dynamic and uncertain, especially wireless channel-dependent ones. They can randomly occur at both the edge nodes and wireless links due to unpredictable factors such as maintenance activities, hardware errors, random obstacles, and interference. Without considering these factors, existing solutions may not be able to achieve a highly reliable, efficient and robust performance for distributed learning over wireless edge networks. More importantly, all the current works only optimize the number of edge nodes to execute learning tasks (i.e., optimal values of (n, k) code) and overlook the fact that different edge nodes may have different computing resources, wireless connections, and hardware configurations. As such, selecting the best set of nodes, instead of the number of nodes to execute learning tasks given the current status of the whole system is very critical to further improve the performance of coded distributed learning in wireless edge networks.

1.2.3.2 Contributions

Given the above, in this thesis, we will propose a jointly optimal coding and scheduling framework for distributed learning over wireless edge networks. In particular, we consider a wireless edge network consisting of a mobile edge computing (MEC) server connected to various edge nodes with different hardware configurations via different wireless links. When a learning task arrives at the MEC server, it will be encoded into sub-learning tasks by using an MDS-based code[†]. Then, these sub-learning tasks are sent to a set of n selected edge nodes to execute. When a predefined number of edge nodes (i.e., k where $k \leq n$) complete their assigned sub-learning tasks, their results can be aggregated to obtain the final result of the

[†]Note that our proposed solution can not only apply for the MDS code proposed in [52] but also can apply for other codes. In particular, most of the coded computing techniques aim to optimize the amount of coding redundancy, e.g., [54], which is similar to k in the MDS code. Therefore, our proposed solution can be straightforwardly extended to other coding techniques.

original learning task. However, finding an optimal MDS code (i.e., a pair of n and k) *and* the best edge nodes (referred to as the optimal scheduling) for each learning task under the dynamic of edge nodes (e.g., available or unavailable) and wireless environment (e.g., good or bad channel condition) is a challenging problem. Solving such a problem in practice is even more difficult as one also needs to account for the uncertainty of wireless links and edge nodes. This is the unpredictable failures or straggling links/devices. To the best of our knowledge, all current works cannot effectively address all these problems.

To tackle the above problem, we first develop an MDP framework to capture the aforementioned dynamics and uncertainty of the system such as diverse learning tasks, computing resources, straggling issues at different edge nodes, and wireless channel conditions [‡]. To minimize the communication and computing delays, one can rely on the Q-learning algorithm to obtain the optimal coding and edge node scheduling policy. The key idea of this algorithm is learning through interactions with the environment and gradually finds the optimal policy. Nevertheless, the Q-learning algorithm usually takes a long time to converge to the optimal policy, especially for distributed learning systems which usually involve with high-dimensional state and action spaces. Moreover, if the state space is continuous, the conventional Q-learning algorithm may not be able to effectively address the dynamic optimization problem. Therefore, we propose a highly-effective deep reinforcement learning algorithm based on the idea of using the deep dueling neural network architecture [26] to facilitate the learning process of the distributed learning system. In particular, as the Q-function of each state-action pair is estimated by the deep dueling neural network instead of Q-table as in the conventional Q-learning algorithm, our proposed algorithm can effectively handle the continuous state space. Moreover,

[‡]The straggling issues at edge nodes and wireless links are provided in detail in (5.2) and (5.4) in Chapter 5.

different from conventional deep reinforcement learning approaches, this proposed algorithm separately estimates the advantage and value functions for each state-action pair with two streams of hidden layers in the deep dueling neural network architecture [26]. These two functions are then combined at the output layer to derive the optimal action, i.e., coding and scheduling policy. In this way, the learning process is significantly improved and stable as the unnecessary relations between the values of states and the advantages of corresponding actions are mitigated. For example, selecting MDS codes with high values of n, i.e., processing learning tasks in many edge nodes, only benefit when learning task sizes are large. Extensive simulation results show that the proposed solution can jointly obtain the optimal code and the best edge nodes to perform learning tasks given the uncertainty and dynamic of wireless channels and straggling computing at edge nodes. Under the optimal policy, the average latency for learning tasks can be reduced by 66% compared to those of the conventional coded distributed learning methods. The major contributions are highlighted as follows:

- Propose a highly effective distributed learning framework leveraging outstanding advantages of coded computing as well as abundant computing resources from multiple collaborative edge nodes to securely and effectively execute learning tasks.

- Propose a jointly optimal coding and scheduling framework for distributed learning over wireless edge networks. Under this framework, one can simultaneously select the optimal code as well as the optimal edge nodes for each learning task given the uncertainty of the edge nodes and wireless links. To the best of our knowledge, our thesis is the first work which can jointly optimize both coding and edge node scheduling for coded computing.

- Develop a highly-effective deep reinforcement learning algorithm for coded computing over wireless edge networks by utilizing the advanced deep dueling neural network architecture [26] to address the slow-convergence and non-discrete problems of conventional reinforcement learning algorithms (e.g., Q-learning and deep Q-learning algorithms). By separately estimating the advantage and value functions, unnecessary relations between the values of states and the advantages of corresponding actions are mitigated, resulting in a high learning rate. This feature is especially useful as the sever needs not only to optimize the code, but also to select the best edge nodes to execute learning tasks at the same time.

- Perform extensive simulations to show the efficiency of our proposed solution compared to those of the conventional approaches (e.g., [52]). Moreover, we discuss and analyze various scenarios to provide insightful designs for distributed learning over wireless edge networks with the coded computing mechanism.

1.3 Thesis Organization

The rest of this thesis is organized as follows.

- Chapter 2: This chapter provides the fundamental background of deep learning, reinforcement learning, and deep reinforcement learning. In particular, Section 2.1 highlights the key information about Deep Learning as well as its advantages. Section 2.2 provides the fundamentals of reinforcement learning, including MDP, SMDP, and Q-learning. Finally, Section 2.3 introduces several deep reinforcement learning algorithms that are used in this thesis, including deep Q-learning, deep double Q-learning, and deep dueling algorithms.

- Chapter 3: This chapter presents our proposed resource allocation framework for network slicing. Specifically, the network slicing system model is introduced in Section 3.1. Then, the system is formulated as an SMDP in Section 3.2. Performance evaluation is provided in Section 3.3. Finally, conclusions are highlighted in Section 3.4.

- Chapter 4: This chapter introduces our proposed anti-jamming framework with the ambient backscatter communication technology and the deep dueling algorithm. In particular, Section 4.1 discusses the anti-jamming system model together with the ambient backscatter communications. Section 4.2 presents the problem formulation, and Section 4.3 highlights the simulation results. Conclusions are drawn in Section 4.4.

- Chapter 5: This chapter presents our proposed solution to jointly optimize coding and scheduling for distributed learning over wireless edge networks. Specifically, Section 5.1 discusses the system model. Section 5.2 introduces the coded computing for distributed learning formulation. Performance analyses and simulation results are then provided in Section 5.3. Finally, conclusions are highlighted in Section 5.4.

- Chapter 6: This chapter outlines the conclusion and future research direction of this thesis.

Chapter 2

Background

This thesis aims to exploit the current advances in AI including deep learning, reinforcement learning, and deep reinforcement learning to address emerging problems in future communication systems. In the following, the fundamentals of deep learning and reinforcement learning are first provided. Then, deep reinforcement learning, a combination of deep learning and reinforcement learning, is discussed in details.

2.1 Deep Learning

Deep learning is a subset of AI in which a model is used to find the important features of data without requiring the data structure [70, 71]. The key idea that makes deep learning a powerful tool in data science is its deep neural network architecture. The term "deep" represents the number of layers in the deep neural network architecture. The more layers are implemented, the deeper the network is. Deep learning has been successfully applied in many AI applications in our daily life, ranging from face and voice recognition, text translation to intelligent driver assistance systems. Deep learning possesses many advantages compared to traditional AI algorithms [72].

- *No need for feature engineering:* With the deep neural network, deep learning can automatically generate new features from the training dataset without human interventions. In this way, deep learning can handle complex tasks with unknown data structures very well.
- *Supports parallel and distributed algorithms:* Deep learning can be imple-

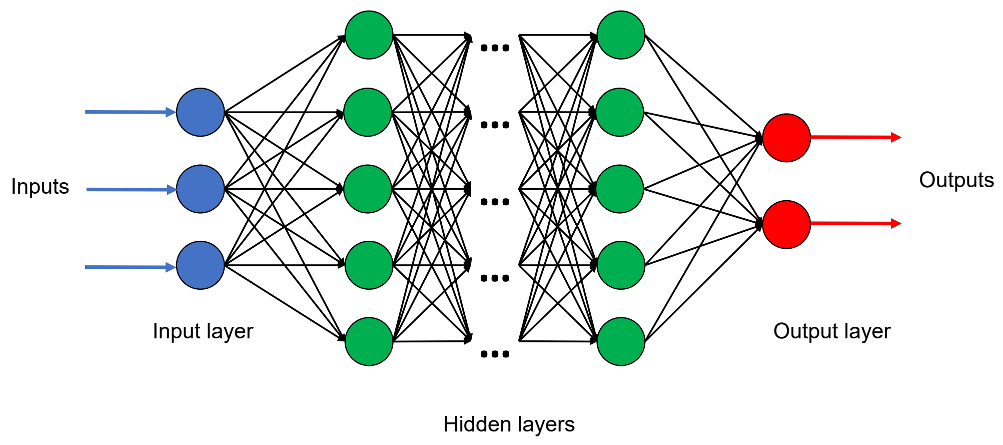


Figure 2.1 : Typical deep neural network architecture.

mented in parallel and distributed systems to accelerate the training process. In particular, instead of training the dataset in a single computer, the model can be trained across multiple computers/systems to leverage their computing power simultaneously.

- *Reusable:* With deep learning, the trained model can be reused in other systems/problems effectively. By using well-trained models built by experts, one can significantly reduce the training time as well as related costs. For example, AlexNet can be reused in new recognition tasks with minimal configurations [72].

The architecture of the deep neural network is inspired by biological nervous systems. A typical deep neural network consists of nonlinear processing layers including an input layer, several hidden layers, and an output layer as shown in Fig. 2.1. These layers are interconnected via nodes, or neurons. A hidden layer uses the outputs of its previous layer as the input. Each neuron has an activation function to compute the output given the weighted inputs, i.e., synapses, and bias [70]. Typically, during the training, synapses are updated by calculating the gradient of the loss function. It is worth mentioning that there are different types of deep neural networks such as

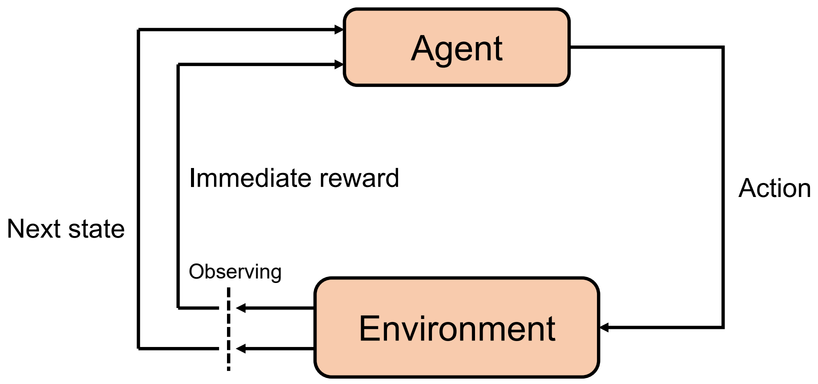


Figure 2.2 : Reinforcement learning.

convolutional neural networks and long short-term memory networks. However, this thesis exploits the typical deep neural network discussed above due to its simplicity and effectiveness in addressing problems in communications.

2.2 Reinforcement Learning

Reinforcement learning is also a type of AI algorithms that can learn by making a sequence of decisions. In particular, reinforcement learning deploys an agent to make actions and interacts with the environment. After making an action, the agent observes the immediate reward and next state of the environment as illustrated in Fig. 2.2. These observations are then learned by the agent to obtain the optimal policy. By doing this, reinforcement learning can deal with the dynamic and uncertainty of the environment, especially in communication systems that consist of a huge number of devices with different configurations and behaviors. In reinforcement learning, the system is first formulated by using the MDP. Then, the Q-learning algorithm is usually used to help the agent learn and obtain the optimal policy.

2.2.1 Markov Decision Process

MDP is a mathematical framework for decision making under the dynamic and uncertainty of the system. MDP is commonly adopted to formulate optimization problems in dynamic programming and reinforcement learning. In particular, an MDP is defined by a tuple $< \mathcal{S}, \mathcal{A}, p, r >$ where $\mathcal{S} \triangleq \{s\}$ is the state space, $\mathcal{A} \triangleq \{a\}$ is the action space, p is the transition probability that the system moves from state s to state s' after action a is performed, and r is the immediate reward of the system after performing action a. We denote π^* as the optimal policy that maximizes the average long-term throughput for the system. Specifically, the optimal policy is a mapping from a state to an action taken by the agent (i.e., decision maker). The main goal of the MDP is to obtain the optimal policy π^* to maximize the expected total reward denoted by $\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t)$, where $\gamma \in [0, 1]$ is the discount factor and $a_t = \pi^*(s_t)$.

Besides MDP, SMDP is also widely adopted in the literature for real-time processes. An SMDP is defined by a tuple $< t_i, \mathcal{S}, \mathcal{A}, \mathcal{L}, r >$ where t_i is a decision epoch and $\mathcal{L}$ captures the state transition probabilities and the state sojourn time. Unlike discrete MDP where decisions are made in every time slot, in an SMDP, we only need to make decisions when an event occurs. This makes the SMDP framework more effective to capture real-time systems in practice.

2.2.2 Q-learning

To obtain the optimal policy π^*, the Q-learning algorithm [73] is the most effective and well-known method in the literature. In particular, as illustrated in Fig. 2.3, the Q-learning algorithm implements a Q-table to store state-action pair values. Given a current state, the algorithm will select an action based on its current strategy. After performing the selected action, the Q-learning algorithm observes the immediate reward and next state, and updates the Q-values based on the Q-

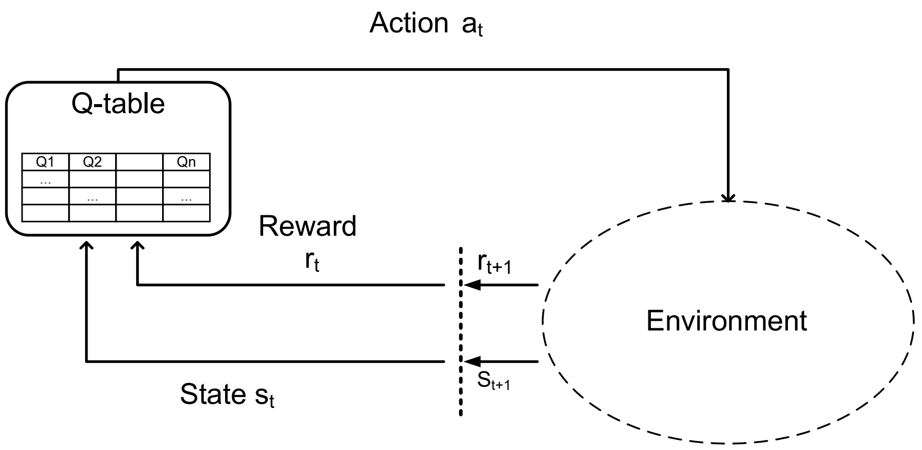


Figure 2.3 : Q-learning model.

value function. In this way, the Q-learning algorithm can learn from its decisions, and it was proved that the Q-learning algorithm will converge to the optimal policy after a finite number of iterations [73].

In an MDP, we aim to find the optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$, i.e., a mapping from states to their corresponding actions, to maximize the average long-term reward. Let's denote $\mathcal{V}^{\pi}(s) : \mathcal{S} \rightarrow \mathbb{R}$ as the expected value function obtained by policy π from a state $s \in \mathcal{S}$, that can be defined as follows:

$$\begin{aligned} \mathcal{V}^{\pi}(s) &= \mathbb{E}_{\pi}\Big[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) | s_0 = s\Big] \\ &= \mathbb{E}_{\pi}\Big[r_t(s_t, a_t) + \gamma \mathcal{V}^{\pi}(s_{t+1}) | s_0 = s\Big], \end{aligned} \tag{2.1}$$

where $0 \leq \gamma \leq 1$ is the discount factor which represents the importance of long-term reward [73]. Specifically, if γ is close to 0, the algorithm is likely to select actions to maximize its short-term reward. In contrast, when γ is close to 1, the algorithm will make actions such that its long-term reward is maximized. $r_t(s_t, a_t)$ is the immediate reward achieved after performing action a_t at state s_t.

To find the optimal policy π^*, at each state, an optimal action has to be found through the following optimal value function.

$$\mathcal{V}^*(s) = \max_a \Big\{ \mathbb{E}_\pi [r_t(s_t, a_t) + \gamma \mathcal{V}^\pi(s_{t+1})] \Big\}, \quad \forall s \in \mathcal{S}. \tag{2.2}$$

For all state-action pairs, the optimal Q-functions are denoted by:

$$\mathcal{Q}^*(s, a) \triangleq r_t(s_t, a_t) + \gamma \mathbb{E}_\pi [\mathcal{V}^\pi(s_{t+1})], \quad \forall s \in \mathcal{S}. \tag{2.3}$$

Then, the optimal value function $\mathcal{V}^*(s)$ can be written as $\mathcal{V}^*(s) = \max_a \{\mathcal{Q}^*(s, a)\}$. By making samples iteratively, the problem is reduced to determining the optimal value of Q-function, i.e., $\mathcal{Q}^*(s, a)$, for all state-action pairs. In particular, the Q-function is updated according to (2.4).

$$\begin{aligned} \mathcal{Q}_{t+1}(s_t, a_t) = \mathcal{Q}_t(s_t, a_t) + \tau_t \Big[& r_t(s_t, a_t) + \\ & \gamma \max_{a_{t+1}} \mathcal{Q}_t(s_{t+1}, a_{t+1}) - \mathcal{Q}_t(s_t, a_t) \Big]. \end{aligned} \tag{2.4}$$

In particular, (2.4) is used to find the temporal difference between the predicted Q-value, i.e., $r_t(s_t, a_t) + \gamma \max_{a_{t+1}} \mathcal{Q}_t(s_{t+1}, a_{t+1})$ and its current value, i.e., $\mathcal{Q}_t(s_t, a_t)$. The learning rate τ_t determines the impact of new information to the existing value. During the learning process, the learning rate can be adjusted dynamically, or it can be chosen to be a constant. However, to guarantee the convergence for the Q-learning algorithm, the learning rate τ_t is deterministic, nonnegative, and satisfies the following conditions [73]:

$$\tau_t \in [0, 1), \sum_{t=1}^{\infty} \tau_t = \infty, \text{ and } \sum_{t=1}^{\infty} (\tau_t)^2 < \infty. \tag{2.5}$$

The details of the Q-learning algorithm are provided in Algorithm 2.1. Specifically, from the current state s_t, the algorithm will choose an action a_t and observe results after performing this action. In practice, to select action a_t, ϵ-greedy algorithm [22] is often adopted. In particular, this technique chooses a random action with probability ϵ, and selects an action that maximizes the $\mathcal{Q}(s, a_s)$ with probability $1 - \epsilon$. After performing the chosen action, the Q-learning algorithm observes

Algorithm 2.1 Q-learning Algorithm

```
1: Inputs: For each state-action pair $(s, a)$, initialize the table entry $\mathcal{Q}(s, a)$ arbitrarily, e.g., to zero. Observe the current state $s$, initialize a value for the learning rate $\tau$ and the discount factor $\gamma$.
2: for $t$=1 to $T$ do
3:     From the current state-action pair $(s_t, a_t)$, execute action $a_t$ and obtain the immediate reward $r_t$
4:     and new state $s_{t+1}$. Select an action $a_{t+1}$ based on the state $s_{t+1}$ and then update the table entry
5:     for $\mathcal{Q}(s_t, a_t)$ as follows:
```

$$\begin{aligned}\mathcal{Q}_{t+1}(s_t, a_t) = \mathcal{Q}_t(s_t, a_t) + \tau_t \Big[& r_t(s_t, a_t) + \\ & \gamma \max_{a_{t+1}} \mathcal{Q}_t(s_{t+1}, a_{t+1}) - \mathcal{Q}_t(s_t, a_t) \Big].\end{aligned} \tag{2.6}$$

```
6:     Replace $s_t \leftarrow s_{t+1}$.
7: end for
8: Outputs: $\pi^*(s) = \arg\max_a \mathcal{Q}^*(s, a)$.
```

the next state and reward, and then updates the table entry for $\mathcal{Q}(s_t, a_t)$ based on Eq. (2.4). When all Q-values converge or a certain number of iterations is reached, the learning process will be terminated. The Q-learning algorithm yields the optimal policy indicating an action to be taken at each state such that $\mathcal{Q}^*(s, a)$ is maximized for all states in the state space, i.e., $\pi^*(s) = \arg\max_a \mathcal{Q}^*(s, a)$. Under the conditions of τ_t stated in Eq. (2.5), in Theorem 2.1, we show that the Q-learning algorithm will converge to the optimum action-values with probability one.

Theorem 2.1. *Under the conditions of* τ_t *in Eq. (2.5), the Q-learning algorithm converges to the optimum action-values with probability one.*

The proof of Theorem 2.1 is provided in Appendix A.1. It is worth noting that the Q-learning algorithm can converge to the optimal policy in a reasonable time when the state space and the action space are small. Nonetheless, for a complicated system with thousands of state-action pairs, the convergence rate of the Q-learning algorithm is usually slow. That makes the Q-learning algorithm practically inapplicable [91]. Thus, in the following, we introduce deep Q-learning, deep double Q-learning, and deep dueling algorithms to quickly obtain the optimal policy thousand times faster than the Q-learning algorithm.

2.3 Deep Reinforcement Learning

2.3.1 Deep Q-learning

In this section, we introduce the deep Q-learning algorithm [92] to cope with the low-convergence problem of the Q-learning algorithm introduced in Section 2.2.2. Intuitively, the deep Q-learning algorithm was introduced by Google DeepMind in 2015 [92] to teach machines to play games without human intervention. The deep Q-learning algorithm implements a deep neural network instead of the Q-table to find the approximated values of $\mathcal{Q}^*(s, a)$ as illustrated in Fig. 2.4.

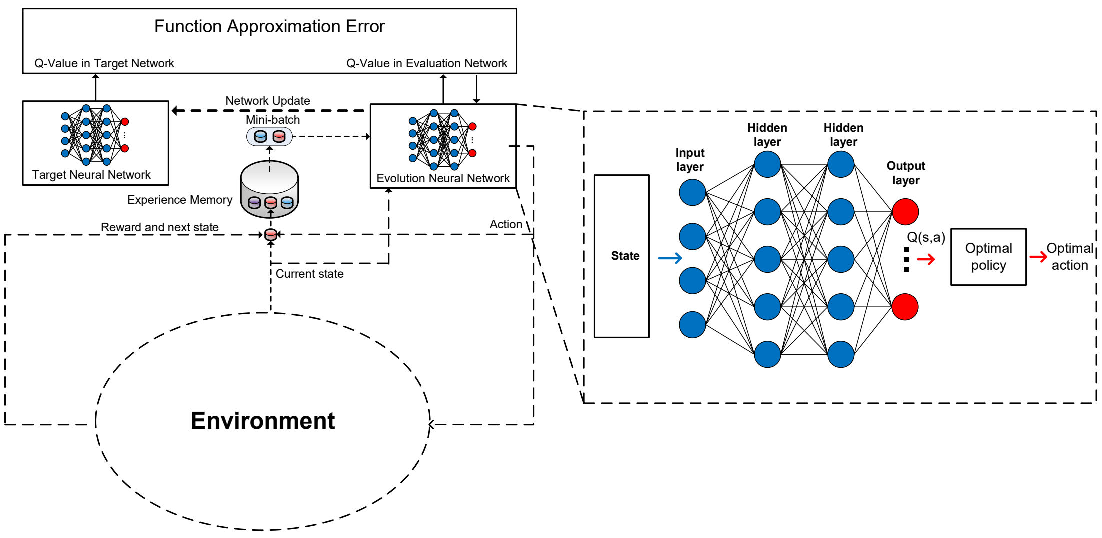


Figure 2.4 : Deep Q-learning model.

According to [92], the performance of reinforcement learning approaches might not be stable or even diverges when using a nonlinear function approximator. The reason is that with a small change of Q-values, the data distribution and correlations between the Q-values and the target values, i.e., $r + \gamma \max_a \mathcal{Q}(s, a)$, are varied, and thus the policy is greatly affected. To address this issue, we use three mechanisms, i.e., experience replay, target Q-network, and feature set.

- *Experience replay mechanism:* The algorithm implements a replay memory **D**, i.e., memory pool, to store transitions (s_t, a_t, r_t, s_{t+1}) instead of running on state-action pairs as they occur during experience. Random samples from the memory pool are then fed to the deep neural network for training. In this way, the algorithm can efficiently learn from previous experiences many times and remove the correlations between observations [92].

- *Target Q-network:* Obviously, the Q-values will be changed during the training process. As a result, the value estimations can be out of control if a constantly shifting set of values is used to update the Q-network resulting in the destabilization of the algorithm. To overcome this issue, the deep Q-learning algorithm implements a target Q-network to frequently but slowly update to the primary Q-network. As such, the correlations between the target and estimated Q-values are significantly eliminated, thereby stabilizing the algorithm.

- *Feature set:* We determine features of the deep neural network as all the aspects of the system state. These features are then fed to the deep neural network to approximate Q-values for each state-action pair. Doing so, all aspects of each state are trained resulting in a high convergence rate.

Algorithm 2.2 Deep Q-learning Algorithm

```
1: Initialize replay memory $\mathbf{D}$ to capacity $\mathcal{D}$.
2: Initialize the Q-network $\mathcal{Q}$ with random weights $\theta$.
3: Initialize the target Q-network $\hat{\mathcal{Q}}$ with weight $\theta^- = \theta$.
4: for episode=1 to $I$ do
5:     With probability $\epsilon$ select a random action $a_t$, otherwise select $a_t = \arg\max \mathcal{Q}^*(s_t, a_t; \theta)$
6:     Perform action $a_t$ and observe reward $r_t$ and next state $s_{t+1}$
7:     Store transition $(s_t, a_t, r_t, s_{t+1})$ in the replay memory $\mathbf{D}$
8:     Sample random mini-batch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $\mathbf{D}$
9:     $y_j = r_j + \gamma \max_{a_{j+1}} \hat{\mathcal{Q}}(s_{j+1}, a_{j+1}; \theta^-)$
10:    Perform a gradient descent step on $(y_j - \mathcal{Q}(s_j, a_j; \theta))^2$ with respect to the network parameter $\theta$.
11:    Every $C$ steps reset $\hat{\mathcal{Q}} = \mathcal{Q}$
12: end for
```

Algorithm 2.2 provides the details of the deep Q-learning algorithm. In particular, as shown in Fig. 2.5, the training phase consists of multiple episodes. In each episode, given the current state, the algorithm chooses an action based on the epsilon greedy algorithm. The algorithm will start with a fairly randomized policy and later slowly move to a deterministic policy. In other words, at the first episode, ϵ is set at a large value, e.g., 0.9, and gradually decayed to a small value, e.g., 0.1. After that, the algorithm performs the selected action and observes results from taking this action, i.e., next state and reward. This transition is then stored in the replay memory for training process at later episodes.

In the learning process, random samples of transitions from the replay memory will be fed into the neural network. The algorithm then updates the neural network by minimizing the following lost function.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(\mathbf{D})}\left[\left(r + \gamma \max_{a'} \hat{\mathcal{Q}}(s', a'; \theta_i^-) - \mathcal{Q}(s, a; \theta_i)\right)^2\right], \tag{2.7}$$

where γ is the discount factor, θ_i are the parameters of the Q-networks at episode i and θ_i^- are the parameters of the target network, i.e., $\hat{\mathcal{Q}}$. Differentiating the loss function in (2.7) with respect to the parameters of the neural networks, we have the following gradient:

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{(s,a,r,s')}\left[\left(r + \gamma \max_{a'} \hat{\mathcal{Q}}(s', a'; \theta_i^-) - \mathcal{Q}(s, a; \theta_i) \nabla_{\theta_i} \mathcal{Q}(s, a; \theta_i)\right)\right]. \tag{2.8}$$

To minimize the loss function in (2.7), one can use the *Stochastic Gradient Descent* algorithm [74], which is a very important algorithm to power nearly all of deep learning algorithms, to calculate the gradient in (2.8). In general, the cost function used by a machine learning algorithm is decayed by a sum over training examples of some per-example loss function. For instance, the negative conditional log-likelihood of the training data can be expressed as:

$$J(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(\mathbf{D})} L\big((s, a, r, s'), \theta\big) = \frac{1}{\mathcal{D}} \sum_{i=1}^{\mathcal{D}} L\big((s, a, r, s')^{(i)}, \theta\big), \tag{2.9}$$

where $\mathcal{D}$ is the size of the memory pool. For this additive cost function, the gradient

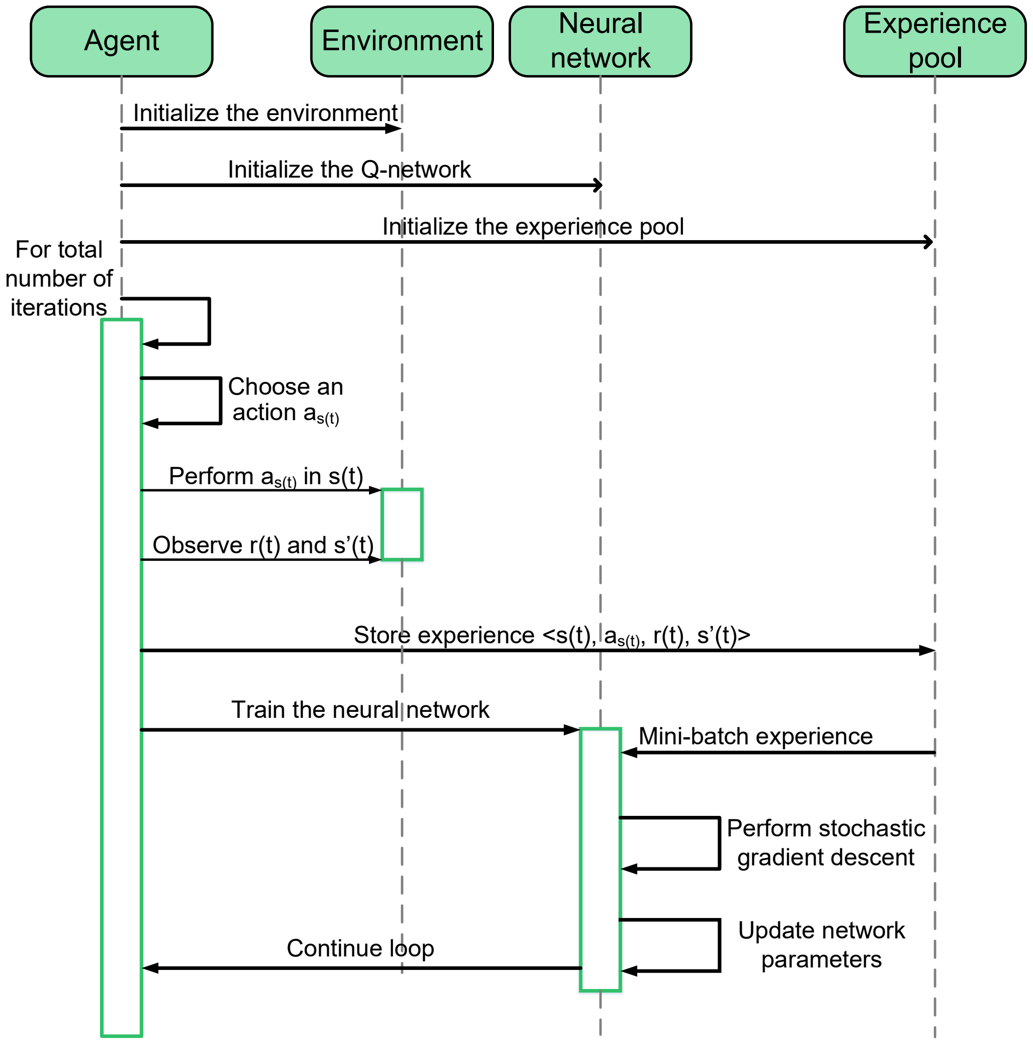

Agent
Environment
Neural network
Experience pool
Initialize the environment
Initialize the Q-network
Initialize the experience pool
For total number of iterations
Choose an action $a_{s(t)}$
Perform $a_{s(t)}$ in s(t)
Observe r(t) and s'(t)
Store experience <s(t), $a_{s(t)}$, r(t), s'(t)>
Train the neural network
Mini-batch experience
Perform stochastic gradient descent
Update network parameters
Continue loop

Figure 2.5 : Flow chart of the deep Q-learning algorithm.

descent requires computing as follows:

$$\nabla_\theta J(\theta) = \frac{1}{\mathcal{D}} \sum_{i=1}^{\mathcal{D}} \nabla_\theta L\big((s,a,r,s')^{(i)}, \theta\big). \tag{2.10}$$

The computational cost for the operation in Eq. (2.10) is $O(\mathcal{D})$. Thus, as the size $\mathcal{D}$ of the replay memory is increased, the time to take a single gradient step becomes prohibitively long. As a result, in this work, we adopt the stochastic gradient descent technique. The key idea of using stochastic gradient descent is that the gradient is an expectation. Clearly, the expectation can be approximately estimated by using a small set of samples. In particular, we can uniformly sample a mini-batch of experiences from the replay memory $\mathbf{D}$ at each step of the algorithm. In general, the mini-batch size can be set to be relatively small number of experiences, e.g., from one to a few hundred. As such, the training time is significantly fast. The estimate of the gradient under the stochastic gradient descent is then formulated as follows:

$$g = \frac{1}{N} \nabla_\theta \sum_{i=1}^{N} L\big((s,a,r,s')^{(i)}, \theta\big), \tag{2.11}$$

where N is the mini-batch size. The stochastic gradient descent algorithm then follows the estimated gradient downhill as in Eq. (2.12).

$$\theta \leftarrow \theta - \nu g, \tag{2.12}$$

where ν is the learning rate of the algorithm. After every C steps, the algorithm updates the target network parameters θ_i^- with the Q-network parameters θ_i. The target network parameters remain unchanged between individual updates. Fig. 2.5 shows the flowchart of the deep Q-learning algorithm.

To further improve the learning rate of the deep Q-learning algorithm, the deep double Q-learning algorithm is proposed recently [75]. The key idea of the deep double Q-learning algorithm is to select an action by using the primary network. It then uses the target network to compute the target Q-value for the action, instead

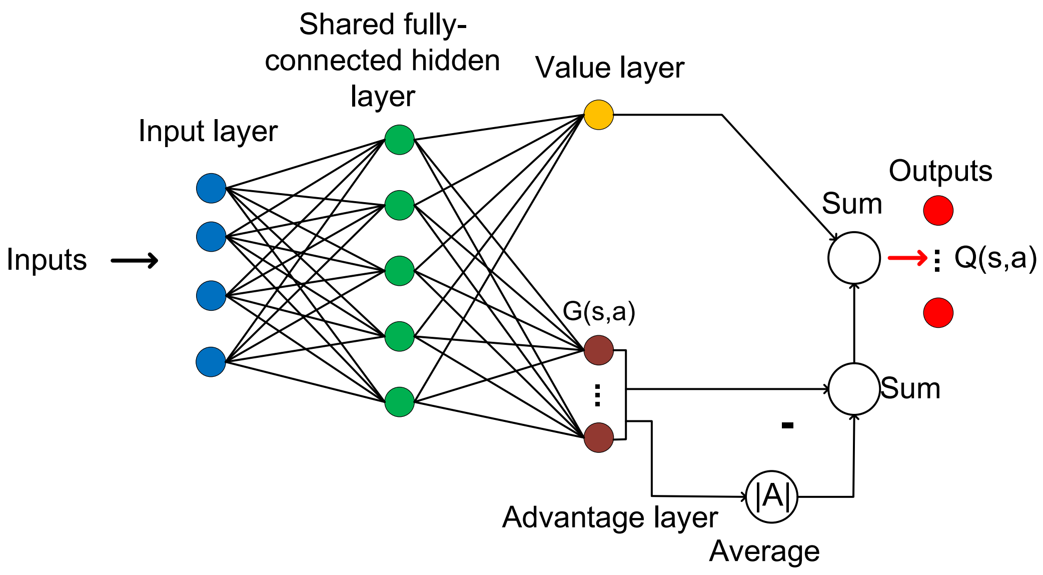


Figure 2.6 : Deep dueling neural network architecture.

of taking the maximum value of all Q-values as in the deep Q-learning algorithm. In this way, the algorithm can be stabilized. The loss function of the deep double Q-learning algorithm is then expressed as follows:

$$L_i(\theta_i) = \mathbb{E}_{(s,a_s,r,s')\sim U(D)}\Big[\Big(r + \gamma\mathcal{Q}(s', \underset{a_{s'}}{\operatorname{argmax}}\ \hat{\mathcal{Q}}(s', a_{s'};\theta);\theta^-)) - \mathcal{Q}(s, a_s;\theta_i)\Big)^2\Big], \tag{2.13}$$

The details of the deep double Q-learning algorithm is provided in Algorithm 2.3. It is worth mentioning that, in our algorithms, one gradient descent step is performed to reduce the computational complexity. In the simulations, we prove that when performing one gradient step at a time only, our proposed algorithms still can achieve the optimal solution quickly. Clearly, our proposed algorithm can perform multiple gradient descent steps by modifying the optimizer operation. Nevertheless, doing this results in high computational complexity for the system.

Algorithm 2.3 Deep Double Q-learning Algorithm

```
1: Initialize replay memory to capacity $D$.
2: Initialize the Q-network $\mathcal{Q}$ with random weights $\theta$.
3: Initialize the target Q-network $\hat{\mathcal{Q}}$ with weight $\theta^- = \theta$.
4: for episode=1 to $T$ do
5:     With probability $\epsilon$ select a random action $a_{s_t}$, otherwise select $a_{s_t} =$ argmax $\mathcal{Q}^*(s_t, a_{s_t}; \theta)$
6:     Perform action $a_{s_t}$ and observe reward $r_t$ and next state $s_{t+1}$
7:     Store transition $(s_t, a_{s_t}, r_t, s_{t+1})$ in the replay memory
8:     Sample random minibatch of transitions $(s_j, a_{s_j}, r_j, s_{j+1})$ from the replay memory
9:     $y_j = r_j + \gamma \mathcal{Q}(s_{j+1}, \text{argmax}_{a_{s_{j+1}}} \hat{\mathcal{Q}}(s_{j+1}, a_{s_{j+1}}; \theta); \theta^-)$
10:    Perform a gradient descent step on $(y_j - \mathcal{Q}(s_j, a_{s_j}; \theta))^2$ with respect to the network parameter $\theta$.
11:    Every $C$ steps reset $\hat{\mathcal{Q}} = \mathcal{Q}$
12: end for
```

2.3.2 Deep Dueling

According to [26], the convergence rate of the deep Q-learning algorithm is still limited due to the overestimation of optimizers, especially in systems with large action and state spaces as considered in this work. Therefore, we propose the deep dueling algorithm [26], which was also originally developed by Google DeepMind in 2016, to further improve the system's convergence speed. The key idea making the deep dueling superior to conventional approaches is its novel neural network architecture. Clearly, in many states, it is unnecessary to estimate the value of corresponding actions as the choice of these actions has no repercussion on what happens [26]. Hence, instead of estimating the action-value function, i.e., Q-function, the algorithm divides the deep neural network into two sequences, i.e., streams, of fully connected layers to separately estimate the values of states and advantages of actions*. The values and advantages are then combined at the output layer as shown in Fig. 2.6. In this way, the deep dueling algorithm can achieve more robust estimates of state value, and thus significantly improving its convergence rate as well as stability. It is worth noting that the flowchart of the deep dueling algorithm is the same as in the deep Q-learning. The main difference between the deep dueling algorithm and other conventional deep reinforcement learning algorithms is the deep dueling neural network. In the following, we present details of separating the Q-value into the value and the advantage functions.

Recall that given a stochastic policy π, the values of state-action pair (s, a) and state s are as follows:

$$\begin{aligned} \mathcal{Q}^{\pi}(s,a) &= \mathbb{E}\big[r_t | s_t = s, a_t = a, \pi\big], \\ \mathcal{V}^{\pi}(s) &= \mathbb{E}_{a\sim\pi(s)}\big[\mathcal{Q}^{\pi}(s,a)\big]. \end{aligned} \tag{2.14}$$

*The value function represents how good it is for the system to be in a given state. The advantage function is used to measure the importance of a certain action compared with others [26].

The advantage function of actions can be expressed as:

$$\mathcal{G}^{\pi}(s,a) = \mathcal{Q}^{\pi}(s,a) - \mathcal{V}^{\pi}(s). \tag{2.15}$$

Specifically, the value function $\mathcal{V}$ corresponds to how *good it is to be in a particular state* s [26]. The state-action pair, i.e., Q-function, calculates the value of performing action a in state s. The advantage function decouples the state value from the Q-function to measure the importance of each action.

To estimate values of $\mathcal{V}$ and $\mathcal{G}$ functions, we use a dueling neural network in which one stream of fully-connected layers outputs a scalar $\mathcal{V}(s;\beta)$ and the other stream estimates an $|\mathcal{A}|$-dimensional vector $\mathcal{G}(s,a;\alpha)$, where α and β are the parameters of fully-connected layers. These two sequences are then combined at the output layer to obtain the Q-function by Eq. (2.16).

$$\mathcal{Q}(s,a;\alpha,\beta) = \mathcal{V}(s;\beta) + \mathcal{G}(s,a;\alpha). \tag{2.16}$$

Note that Eq. (2.16) applies to all (s,a) instances. Thus, to express equation (2.16) in a matrix form, one needs to replicate the scalar, $\mathcal{V}(s;\beta)$, $|\mathcal{A}|$ times. Importantly, $\mathcal{Q}(s,a;\alpha,\beta)$ is a parameterized estimate of the true Q-function, and given $\mathcal{Q}$, we cannot obtain $\mathcal{V}$ and $\mathcal{G}$ uniquely. In other words, adding a constant to $\mathcal{V}(s;\beta)$ and subtracting the same constant from $\mathcal{G}(s,a;\alpha)$ result in the same Q-value. Therefore, Eq. (2.16) is unidentifiable resulting in poor performance. To address this problem, the combining module of the network is implemented the following mapping:

$$\mathcal{Q}(s,a;\alpha,\beta) = \mathcal{V}(s;\beta) + \big(\mathcal{G}(s,a;\alpha) - \max_{a\in\mathcal{A}} \mathcal{G}(s,a;\alpha)\big). \tag{2.17}$$

In this way, the advantage function estimator has a zero advantage when choosing an action. Intuitively, given $a^* = \arg\max_{a\in\mathcal{A}} \mathcal{Q}(s,a;\alpha,\beta) = \arg\max_{a\in\mathcal{A}} \mathcal{G}(s,a;\alpha)$, we have $\mathcal{Q}(s,a^*;\alpha,\beta) = \mathcal{V}(s;\beta)$. Therefore, we can convert (2.17) into a simple form by replacing the max operator with an average as follows:

$$\mathcal{Q}(s,a;\alpha,\beta) = \mathcal{V}(s;\beta) + \big(\mathcal{G}(s,a;\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a} \mathcal{G}(s,a;\alpha)\big). \tag{2.18}$$

Note that subtracting the mean in Eq. (2.18) solves the unidentifiable problem. However, it does not change the relative rank of the advantage function values, and hence the $\mathcal{Q}$ values for actions at each state.

Algorithm 2.4 Deep Dueling Algorithm

```
1:  Initialize replay memory $\mathbf{D}$ to capacity $\mathcal{D}$.
2:  Initialize the primary network $\mathcal{Q}$ including two fully-connected layers with random weights $\alpha$ and $\beta$.
3:  Initialize the target network $\hat{\mathcal{Q}}$ as a copy of the primary Q-network with weights $\alpha^- = \alpha$ and $\beta^- = \beta$.
4:  for episode=1 to I do
5:      Base on the $\epsilon$-greedy algorithm, with probability $\epsilon$ select a random action $a_t$ at state $s_t$. Otherwise,
6:      select $a_t = \arg\max \mathcal{Q}^*(s_t, a_t; \alpha, \beta)$
7:      Perform action $a_t$ and observe reward $r_t$ and next state $s_{t+1}$
8:      Store transition $(s_t, a_t, r_t, s_{t+1})$ in the replay memory
9:      Sample random mini-batch of transitions $(s_j, a_j, r_j, s_{j+1})$ from the replay memory
10:     Combine the value function and advantage functions as follows:
            $$\mathcal{Q}(s_j, a_j; \alpha, \beta) = \mathcal{V}(s_j; \beta) + \Big(\mathcal{G}(s_j, a_j; \alpha) - \frac{1}{|\mathcal{A}|}\sum_{a_j}\mathcal{G}(s_j, a_j; \alpha)\Big). \quad (2.19)$$
11:     $y_j = r_j + \gamma \max_{a_{j+1}} \hat{\mathcal{Q}}(s_{j+1}, a_{j+1}; \alpha^-, \beta^-)$
12:     Perform a gradient descent step on $(y_j - \mathcal{Q}(s_j, a_j; \alpha, \beta))^2$
13:     Every $C$ steps reset $\hat{\mathcal{Q}} = \mathcal{Q}$
14: end for
```

Based on Eq. (2.18) and the advantages of the deep reinforcement learning, we propose the deep dueling algorithm as shown in Algorithm 2.4. It is worth noting that Eq. (2.18) is viewed and implemented as a part of the network and not as a separated algorithmic step [26]. In addition, $\mathcal{V}(s;\beta)$ and $\mathcal{G}(s,a;\alpha)$ are estimated automatically without any extra supervision or modifications in the algorithm. Similar to deep Q-learning and deep double Q-learning algorithms, by performing gradient descent steps, the weights of the deep dueling neural network will be gradually updated to optimal values. However, analyzing the weights of deep neural networks is not the main aim of this thesis. Instead, we focus on evaluating the performance of AI-based solutions in addressing problems in future communication systems. It is worth noting that deep reinforcement learning runs in an online manner without requiring the environment parameters in advance. As such, there is no need for a dataset for training. Deep reinforcement learning learns the optimal policy by interacting with the environment and observing the next state of the system as well as the immediate reward. This information is stored in the memory pool and will be learned multiple times by the deep neural networks.

2.3.3 Complexity Analysis

In this thesis, we implement all the deep reinforcement learning algorithms above to evaluate the system performance. The deep dueling neural network consists of an input layer L_0, a hidden layer L_1, and two streams to estimate the value and the advantage function. The value stream consists of layer L_{value} which is used to estimate the value function. The advantage stream consists of layer $L_{\text{advantage}}$ which is used to estimate the advantage function. Let $|L_{\text{i}}|$ denote the size (i.e., the number of neurons) of layer L_{i}. We then can formulate the complexity of the deep dueling neural network as $|L_0||L_1| + |L_1||L_{\text{value}}| + |L_1||L_{\text{advantage}}|$. At each training step, a number of training samples, i.e., transitions, are randomly taken from the memory pool and fed to the deep dueling neural network for training.

Thus, the total complexity of the training process is $\mathcal{O}\Big(IN_{\mathrm{b}}\Big(|L_0||L_1|+|L_1||L_{\mathrm{value}}|+|L_1||L_{\mathrm{advantage}}|\Big)\Big)$, where N_{b} is the size of the training batch and I is the total number of training iterations. In this thesis, the size of L_0 is the number of state features in the state space. $|L_{\mathrm{value}}| = 1$ as this layer is used to estimate the value of the current state only. The size of $L_{\mathrm{advantage}}$ is the number of actions in the action space. For the deep Q-learning and deep double Q-learning, we deploy two fully connected hidden layers L_1 and L_2. As such, the complexity of these algorithm can be formulated as $\mathcal{O}\Big(IN_{\mathrm{b}}\Big(|L_0||L_1| + |L_1||L_2|\Big)\Big)$.

Clearly, our deep neural network architectures are simple with few hidden layers. In the simulations, we show that with only 16 (for the security and distributed learning problems) and 64 neurons (for the network slicing problem) in the hidden layers, our proposed deep reinforcement learning algorithms can effectively obtain the optimal policy for the system. It is worth noting that with the advanced deep dueling neural network architecture, the proposed deep dueling algorithm can converge to the optimal policy much faster than the Q-learning, deep Q-learning, and deep double Q-learning algorithms.

Chapter 3

Optimal and Fast Real-time Resource Slicing with Deep Dueling Neural Networks

In this chapter, we develop an optimal and fast real-time resource slicing framework that maximizes the long-term return of the network provider while taking into account the uncertainty of resource demand from tenants. Specifically, we first propose a novel system model which enables the network provider to effectively slice various types of resources to different classes of users under separate virtual slices. We then capture the real-time arrival of slice requests by a SMDP. To obtain the optimal resource allocation policy under the dynamics of slicing requests, e.g., uncertain service time and resource demands, a Q-learning algorithm is often adopted in the literature. However, such an algorithm is notorious for its slow convergence, especially for problems with large state/action spaces. This makes Q-learning practically inapplicable to our case in which multiple resources are simultaneously optimized. To tackle it, we propose a novel network slicing approach with an advanced deep learning architecture, called deep dueling that attains the optimal average reward much faster than the conventional Q-learning algorithm. This property is especially desirable to cope with real-time resource requests and the dynamic demands of users. Extensive simulations show that the proposed framework yields up to 40% higher long-term average return while being few thousand times faster, compared with state-of-the-art network slicing approaches.

The rest of this chapter is organized as follows. Section 3.1 and Section 3.2 describe the system model and the problem formulation, respectively. Evaluation results are then discussed in Section 3.3. Finally, conclusions and future works are

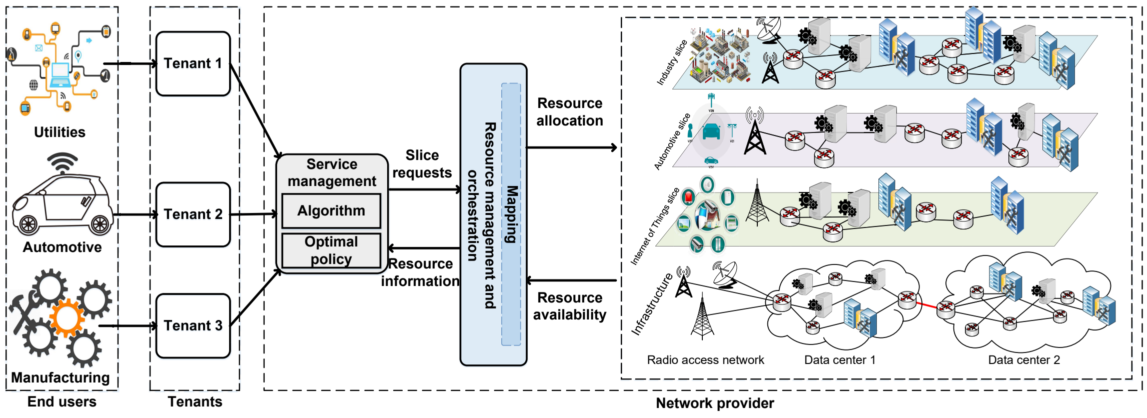


Figure 3.1 : Network resource slicing system model.

given in Section 3.4.

3.1 System Model

In Fig. 3.1, we consider a general network slicing model with three major parties [17], [18], [24]:

- *Network provider:* is the owner of the network infrastructure who provides resource slices including radio, computing, and storage, to the tenants.
- *Tenants:* request and lease resource slices to meet service demands of their subscribers.
- *End users:* run their applications on the slices of the above subscribed tenants.

We consider three tenants corresponding to three popular classes of services, i.e., utilities, automotive, and manufacturing, as shown in Fig. 3.1*. Each class of service possesses some specific features regarding its functional, behavioral perspective, and

*It is worth noting that these services are just three examples considered in this thesis. The proposed model is applicable to any general service that can be represented by a specific resource requirement and service requirement.

requirements. For example, a vehicle may need an ultra-reliable slice for telemetry assisted driving [23]. For slices requested from industry, security, resilience, and reliability of services are of higher priority [76], [77]. Thus, when a tenant sends a network slice request to the network provider, the tenant will specify resources requested and additional service requirements, e.g., security and reliability (defined in the slice blueprint). As a result, tenants may pay different prices for their requests, depending on their service demands. Upon receiving a slice request, the service management component (in Fig. 3.1) analyzes the requirements and makes a decision to accept or reject the request based on its optimal policy.

The service management block consists of two components: (i) the optimal policy and (ii) the algorithm. When a slice request arrives at the system, the optimal policy component will make a decision, i.e., accept or reject, based on the (current) optimal policy obtained by the algorithm component. As the decision can be made immediately, the decision latency is virtually zero. For the algorithm component, the optimal policy is calculated and updated periodically. It is worth noting that our algorithm observes the results after performing the decision, and uses the observations together with the characteristics of slice requests for its training process. By doing so, our algorithm can learn from previous experience and is able to deal with the uncertainty of slice requests. If the request is accepted, the service management will transfer the slice request to the resource management and orchestration (RMO) block to allocate resources. Once a slice request is accepted, the network provider will receive an immediate reward (the amount of money) paid by the tenant for granted resources and services.

In practice, the network provider may possess multiple data centers for network slicing services. Each data center contains a set of servers with diverse resources, e.g., computing and storage, which are used to support VNFs services. Servers in the data center are connected together, and the data centers are connected via backhaul links. Then, the network slicing and resource allocation processes to each slice are taken place as follows.

- A slice request is associated with the *network slice blueprint* (i.e., a template) that describes the structure, configuration, and workflow for instantiating and controlling the network slice instance for the service during its life cycle [24], [78]- [80]. The service/slice instance includes a set of network functions and resources to meet the end-to-end service requirements.
- When a slice request arrives at the system, the orchestrator will interpret the blueprint [78]- [80]. In particular, all information of the infrastructure such as (i) NFV services provided by servers, (ii) resources availability at servers, and (iii) the connectivities among servers are checked.
- Based on the aforementioned information, the orchestrator will find the optimal servers and links to place VNFs to meet the required end-to-end services of the slice (i.e., VNF placement procedure).
- During the life cycle of the slice, the orchestrator can change the allocated computing and storage resources by using the scaling-in and scaling-out mechanisms. In addition, the connectivity among VNFs and their locations can be changed when there is no sufficient resources or the behavior of the slice is changed, e.g., update, migrate, or terminate the network slice.

Note that the VNF placement, routing, and connectivity resource allocation problems have been well investigated in the literature, e.g., [81]- [86]. For example, in [86], the AAP algorithm is introduced to admit and route connection requests by finding possible paths satisfying cost criteria. Instead of focusing on VNF placement, routing, and connectivity resource allocation problems, in this work, we mainly focus on dealing with the uncertainty, dynamics, and heterogeneity of slice requests. Thus, we consider a simplified yet practical model and propose the novel framework using the deep dueling neural network architecture [26] to address the aforementioned problems, which are the main aims and key contributions of this work. Specifically, we assume that there are C classes of slices, denoted by $\mathcal{C} = \{1, \dots, c, \dots, C\}$. Each slice from class c requires r_c^{re}, ω_c^{re}, and δ_c^{re} units of radio, computing, and storage resources, respectively. If a slice request from class c is accepted, the provider will receive an immediate reward r_c. The maximum radio, computing, and storage resources of the network provider are denoted by Θ, Ω, and Δ units, respectively. Let n_c denote the number of slices from class c being simultaneously run/served in the system. At any time, the following resource constraints guarantee that the allocated resources do not exceed the available resources of the infrastructure:

$$\Theta \geq \sum_{c=1}^{C} r_c^{re} n_c, \quad \Omega \geq \sum_{c=1}^{C} \omega_c^{re} n_c, \text{ and } \Delta \geq \sum_{c=1}^{C} \delta_c^{re} n_c. \tag{3.1}$$

3.2 Problem Formulation

To maximize the long-term return for the provider while accounting for the real-time arrivals of slice requests, we recruit the SMDP [25]. An SMDP is defined by a tuple $< t_i, \mathcal{S}, \mathcal{A}, \mathcal{L}, r >$ where t_i is an decision epoch, $\mathcal{S}$ is the system's state space, $\mathcal{A}$ is the action space, $\mathcal{L}$ captures the state transition probabilities and the state sojourn time, and r is the reward function. Unlike discrete MDPs where decisions are made in every time slots, in an SMDP, we only need to make decisions when an event occurs. This makes the SMDP framework more effective to capture real-time network slicing systems.

3.2.1 Decision Epoch

Under our network slicing system model, the provider needs to make a decision upon receiving requests from tenants. Thus, the decision epoch can be defined as the inter-arrival time between two successive slice requests.

3.2.2 State Space

The system state $\mathbf{s}$ of the SMDP at the current decision epoch captures the number of slices n_c from a given class c ($\forall c \in \mathcal{C}$) being simultaneously run/served in the system. Formally, we define $\mathbf{s}$ as an $1 \times C$ vector:

$$\mathbf{s} \triangleq [n_1, \dots, n_c, \dots n_C]. \tag{3.2}$$

Given the network provider's resource constraints in (3.1), the state space $\mathcal{S}$ of all possible states $\mathbf{s}$ is defined as:

$$\begin{aligned} \mathcal{S} \triangleq \Bigg\{ \mathbf{s} = [n_1, \dots, n_c, \dots n_C] : \Theta \geq \sum_{c=1}^{C} r_c^{re} n_c; \\ \Omega \geq \sum_{c=1}^{C} \omega_c^{re} n_c; \Delta \geq \sum_{c=1}^{C} \delta_c^{re} n_c \Bigg\}. \end{aligned} \tag{3.3}$$

At the current system state $\mathbf{s}$, we define the *event vector* $\mathbf{e} \triangleq [e_1, \dots, e_c, \dots, e_C]$ with $e_c \in \{1, -1, 0\}, \forall c \in \mathcal{C}$. e_c equals to "1" if a new slice request from class c arrives, e_c equals to "-1" if a slice's resources are being released (also referred to as a slice completion/departing) to the system's resource, and e_c equals "0" otherwise (i.e., no slice request arrives nor completes/departs from the system). The set $\mathcal{E}$ of all the possible events is then defined as follows:

$$\mathcal{E} \triangleq \Big\{ \mathbf{e} : e_c \in \{-1, 0, 1\}; \sum_{c=1}^{C} |e_c| \leq 1 \Big\}, \tag{3.4}$$

where the *trivial event* $\mathbf{e}^* \triangleq (0, \dots, 0) \in \mathcal{E}$ means no request arrival or completion/departing from all C classes.

3.2.3 Action Space

At state $\mathbf{s}$, if a slice request arrives (i.e., there exists $c \in \mathcal{C}$ such that $e_c = 1$), the network provider can choose either to accept or reject this request to maximize its long-term return. Let $a_{\mathbf{s}}$ denote the action to be taken at state $\mathbf{s}$ where $a_{\mathbf{s}} = 1$ if an arrival slice is accepted and $a_{\mathbf{s}} = 0$ otherwise. The state-dependent action space $\mathcal{A}_{\mathbf{s}}$ can be defined by:

$$\mathcal{A}_{\mathbf{s}} \triangleq \{a_{\mathbf{s}}\} = \{0, 1\}. \tag{3.5}$$

3.2.4 State Transition Probability

As aforementioned, in this work, we propose reinforcement learning approaches which can obtain the optimal policy for the network provider without requiring information from the environment (to cope with the uncertain demands and dynamics of slice requests). However, to lay a theoretical foundation and to evaluate the performance of our proposed solutions, we first assume that the arrival process of slice requests from class c follows the Poisson distribution with mean rate λ_c and its network resource occupation time follows the exponential distribution with mean $1/\mu_c$. The assumptions allow us to analyze the dynamics of the SMDP, which is characterized by the state transition probabilities of the underlying Markov chain. In particular, our SMDP model consists of a renewal process and a continuous-time Markov chain $\{X(t : t \geq 0)\}$ in which the sojourn time in a state is a continuous random variable. We then can adopt the uniformization technique [87] to determine the probabilities for events and derive the transition probabilities $\mathcal{L}$. As shown in Fig. 3.2, the uniformization technique transforms the original continuous-time Markov chain $\{X(t) : t \geq 0\}$ into an equivalent stochastic process $\{\overline{X}(t), t \geq 0\}$ in which the transition epochs are generated by a Poisson process $\{N(t) : t \geq 0\}$ at a uniform rate and the state transitions are governed by the discrete-time Markov chain $\{\overline{X}_n\}$ [88], [89]. The details of the uniformization technique are as the following.

Our Markov chain $\{X(t)\}$ can be considered as a time-homogeneous Markov chain. Suppose that $\{X(t)\}$ is in state $\mathbf{s}$ at the current time t. If the system leaves state $\mathbf{s}$, it transfers to state $\mathbf{s}'$ ($\neq \mathbf{s}$) with probability $p_{\mathbf{s},\mathbf{s}'}(t)$. The probability that

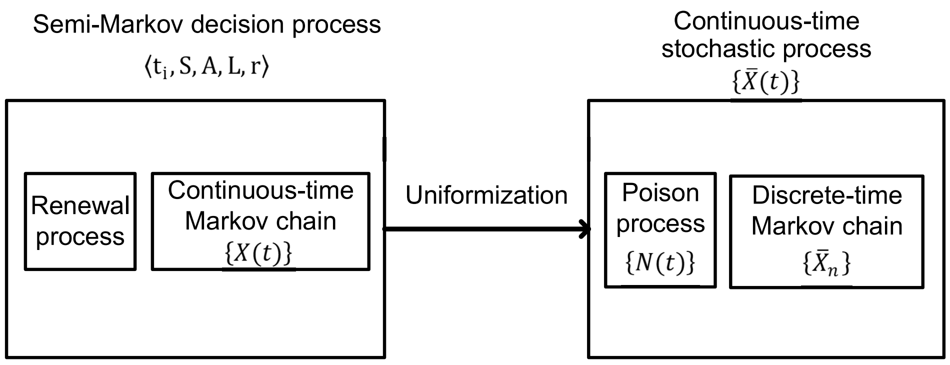

Semi-Markov decision process
$\langle t_i, S, A, L, r\rangle$
Renewal process
Continuous-time Markov chain $\{X(t)\}$
Uniformization
Continuous-time stochastic process $\{\bar{X}(t)\}$
Poison process $\{N(t)\}$
Discrete-time Markov chain $\{\bar{X}_n\}$

Figure 3.2 : Uniformization technique.

the process will leave state $\mathbf{s}$ in the next Δt to state $\mathbf{s}'$ is expressed as follows:

$$P\{X(t+\Delta t)=\mathbf{s}'|X(t)=\mathbf{s}\} \\ = \begin{cases} z_{\mathbf{s}}\Delta t \times p_{\mathbf{s},\mathbf{s}'}(t)+o(\Delta t), & \mathbf{s}'\neq \mathbf{s}, \\ 1-z_{\mathbf{s}}\Delta t + o(\Delta t), & \mathbf{s}'=\mathbf{s}, \end{cases} \tag{3.6}$$

as $\Delta t \to 0$ and $z_{\mathbf{s}}$ is the occurrence rate of the next event expressed as follows:

$$z_{\mathbf{s}} = \sum_{c=1}^{C}(\lambda_c + n_c\mu_c). \tag{3.7}$$

In the uniformization technique, we consider that the occurrence rate $z_{\mathbf{s}}$ of the states are identical, i.e., $z_{\mathbf{s}} = z$ for all $\mathbf{s}$. Thus, the transition epochs can be generated by a Poisson process with rate z. To formulate the uniformization technique, we choose a number z with

$$z = \max_{\mathbf{s}\in\mathcal{S}} z_{\mathbf{s}}. \tag{3.8}$$

Now, we define a discrete-time Markov chain $\{\overline{X}_n\}$ whose one-step transition probabilities $\overline{p}_{\mathbf{s},\mathbf{s}'}(t)$ are given by:

$$\overline{p}_{\mathbf{s},\mathbf{s}'}(t) = \begin{cases} (z_{\mathbf{s}}/z)p_{\mathbf{s},\mathbf{s}'}(t), & \mathbf{s}'\neq\mathbf{s}, \\ 1-z_{\mathbf{s}}/z, & \text{otherwise}, \end{cases} \tag{3.9}$$

for all $\mathbf{s}\in\mathcal{S}$. Let $\{N(t), t\geq 0\}$ be a Poisson process with rate z such that the process is independent of the discrete-time Markov chain $\{\overline{X}_n\}$. We then define the continuous-time stochastic process $\{\overline{X}(t), t \geq 0\}$ as follows:

$$\overline{X}(t) = \overline{X}_{N(t)}, t \geq 0. \tag{3.10}$$

Equation (3.10) represents that the process $\{\overline{X}(t)\}$ makes state transitions at epochs generated by a Poisson process with rate z and the state transitions are governed by the discrete-time Markov chain $\{\overline{X}_n\}$ with one-step transition probabilities $\overline{p}_{\mathbf{s},\mathbf{s}'}(t)$ in (3.9). When the Markov chain $\{\overline{X}_n\}$ is in state $\mathbf{s}$, the system leaves to next state with probability $z_{\mathbf{s}}/z$ and is a self-transition with probability $1 - z_{\mathbf{s}}/z$. In fact, the transitions out of state $\mathbf{s}$ are delayed by a time factor of $z/z_{\mathbf{s}}$, while a factor of $z_{\mathbf{s}}/z$ corresponds to the time until a state transition from state $\mathbf{s}$. In addition, in our system model there is no terminal state, i.e., the discrete-time Markov chain describing the state transitions in the transformed process has to allow for self-transitions leaving the state of the process unchanged. Therefore, the continuous $\{\overline{X}(t)\}$ is probabilistically identical to the original continuous-time Markov chain $\{X(t)\}$. This statement can be expressed as the following equation:

$$\begin{aligned}
&P\{\overline{X}(t+\Delta t) = \mathbf{s}' | \overline{X}(t) = \mathbf{s}\} = z\Delta t \times \overline{p}_{\mathbf{s},\mathbf{s}'} + o(\Delta t) \\
&= z_{\mathbf{s}}\Delta t \times p_{\mathbf{s},\mathbf{s}'} + o(\Delta t) \\
&= q_{\mathbf{s},\mathbf{s}'}\Delta t + o(\Delta t) \\
&= P\{X(t+\Delta t) = \mathbf{s}' | X(t) = \mathbf{s}\} \text{ for } \Delta t \to 0, \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S} \\
&\text{and } \mathbf{s} \neq \mathbf{s}',
\end{aligned} \tag{3.11}$$

where $q_{\mathbf{s},\mathbf{s}'}$ is the infinitesimal transition rate of the continuous-time Markov chain $\{X(t)\}$ and is expressed as follows:

$$q_{\mathbf{s},\mathbf{s}'} = z_{\mathbf{s}} p_{\mathbf{s},\mathbf{s}'}, \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S} \quad \text{and} \quad \mathbf{s}' \neq \mathbf{s}. \tag{3.12}$$

Clearly, in our system, the occurrence rate of the next event $z_{\mathbf{s}} = \sum_{c=1}^{C}(\lambda_c + n_c\mu_c)$ are positive and bounded in $\mathbf{s} \in \mathcal{S}$. Thus, it is proved that the infinitesimal transition rates determine a unique continuous-time Markov chain $\{X(t)\}$ [88]. We then make a necessary corollary as follows:

Corollary 3.1. *The probabilities $p_{\mathbf{s},\mathbf{s}'}(t)$ are given by:*

$$p_{\mathbf{s},\mathbf{s}'}(t) = \sum_{n=0}^{\infty} e^{-zt} \frac{zt^n}{n!} \overline{p}_{\mathbf{s},\mathbf{s}'}^{(n)}, \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S} \text{ and } t \geq 0, \quad (3.13)$$

where the probabilities $\overline{p}_{\mathbf{s},\mathbf{s}'}^{(n)}$ can be recursively computed from

$$\overline{p}_{\mathbf{s},\mathbf{s}'}^{(n)} = \sum_{k \in \mathcal{S}} \overline{p}_{\mathbf{s},k}^{(n-1)} \overline{p}_{k,\mathbf{s}'}, n = 1, 2, \ldots \quad (3.14)$$

starting with $\overline{p}_{\mathbf{s},\mathbf{s}}^{(0)} = 1$ and $\overline{p}_{\mathbf{s},\mathbf{s}'}^{0} = 0$ $\forall \mathbf{s}' \neq \mathbf{s}$.

In the next theorem, we prove that two processes $\{\overline{X}(t)\}$ and $\{X(t)\}$ are probabilistically equivalent.

Theorem 3.1. *$\{\overline{X}(t)\}$ and $\{X(t)\}$ are probabilistically equivalent as*

$$p_{\mathbf{s},\mathbf{s}'}(t) = P\{\overline{X}(t) = \mathbf{s}' | X(0) = \mathbf{s}\}, \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S} \text{ and } t \leq 0. \quad (3.15)$$

The proof of Theorem 3.1 is given in Appendix B.1. ■

From (3.13) and (3.14), the computational complexity of the uniformization method is derived as $O(vt|\mathcal{S}|^2)$, where $|\mathcal{S}|$ is the number of states of the system. Based on z and $z_\mathbf{s}$, we can determine the probabilities for events as follows. The probability for an arrival slice from class c occurring in the next event $\mathbf{e}$ equals λ_c/z. The probability for a departure slice from class c occurring in the next event $\mathbf{e}$ equals $n_c\mu_c/z$, and the probability for a trivial event occurring in the next event $\mathbf{e}$ is $1 - z_\mathbf{s}/z$. Hence, we can derive the transition probability $\mathcal{L}$.

3.2.5 Reward Function

The immediate reward after action $a_\mathbf{s}$ is executed at state $\mathbf{s} \in \mathcal{S}$ is defined as follows:

$$r(\mathbf{s}, a_\mathbf{s}) = \begin{cases} r_c, & \text{if } e_c = 1,\ a_\mathbf{s} = 1, \text{ and } \mathbf{s}' \in \mathcal{S}, \\ 0, & \text{otherwise.} \end{cases} \quad (3.16)$$

At state $\mathbf{s}$, if an arrival slice is accepted, i.e., $a_{\mathbf{s}} = 1$, the system will move to next state $\mathbf{s}'$ and the network provider receives a reward of r_c. In contrast, the immediate reward is equal to 0 if an arrival slice is rejected or there is no slice request arriving at the system. It is worth mentioning that the value of r_c represents the amount of money paid by the tenant based on resources and additional services required.

As our system's statistical properties are time-invariant, i.e., stationary, the decision policy π of the SMDP model, which is a pure strategy, i.e., accept or reject an arrival request, can be defined as a time-invariant mapping from the state space to the action space: $\mathcal{S} \rightarrow \mathcal{A}_{\mathbf{s}}$. Thus, the long-term average reward starting from a state $\mathbf{s}$ can be formulated as follows:

$$\mathcal{R}_\pi(\mathbf{s}) = \lim_{K\rightarrow\infty} \frac{\mathbb{E}\{\sum_{k=0}^{K} r(\mathbf{s}_k, \pi(\mathbf{s}_k))|\mathbf{s}_0 = \mathbf{s}\}}{\mathbb{E}\{\sum_{k=0}^{K} \tau_k|\mathbf{s}_0 = \mathbf{s}\}}, \forall \mathbf{s} \in \mathcal{S}, \tag{3.17}$$

where τ_k is the time interval between the k-th and $(k+1)$-th decision epoch, r is the immediate reward of the system, and $\pi(\mathbf{s})$ is the action corresponding to the policy π at state $\mathbf{s}$.

In the following theorem, we will prove that the limit in Equation (3.17) exists.

Theorem 3.2. *Given the state space $\mathcal{S}$ is countable and there is a finite number of decision epochs within a certain considered finite time, we have:*

$$\begin{aligned}\mathcal{R}_\pi(\mathbf{s}) &= \lim_{K\rightarrow\infty} \frac{\mathbb{E}\{\sum_{k=0}^{K} r(\mathbf{s}_k, \pi(\mathbf{s}_k))|\mathbf{s}_0 = \mathbf{s}\}}{\mathbb{E}\{\sum_{k=0}^{K} \tau_k|\mathbf{s}_0 = \mathbf{s}\}} \\ &= \frac{\overline{\mathcal{L}}_\pi r(\mathbf{s}, \pi(\mathbf{s}))}{\overline{\mathcal{L}}_\pi y(\mathbf{s}, \pi(\mathbf{s}))}, \forall \mathbf{s} \in \mathcal{S},\end{aligned} \tag{3.18}$$

where $y(\mathbf{s}, \pi(\mathbf{s}))$ is the expected time interval between adjacent decision epochs when action $\pi(\mathbf{s})$ is taken under state $\mathbf{s}$, and

$$\overline{\mathcal{L}}_\pi = \lim_{K\rightarrow\infty} \frac{1}{K} \sum_{k=0}^{K-1} \mathcal{L}_\pi^k, \tag{3.19}$$

where $\mathcal{L}_\pi^k$ and $\overline{\mathcal{L}}_\pi$ are the transition probability matrix and the limiting matrix of the embedded Markov chain for policy π, respectively.

Proof. We first have the following lemma.

Lemma 3.1. *Given the the transition probability matrix $\mathcal{L}_\pi$, the limiting matrix $\overline{\mathcal{L}}_\pi$ exists.*

The proof of Lemma 3.1 is given in Appendix B.2.

Since $\mathcal{L}_\pi$ is the transition probability matrix, $\overline{\mathcal{L}}_\pi$ exists as stated in Lemma 3.1. As the total of probabilities that the system transform from state $\mathbf{s}$ to other states is equal to 1, we have:

$$\sum_{\mathbf{s}'\in\mathcal{S}} \overline{\mathcal{L}}_\pi(\mathbf{s}'|\mathbf{s}) = 1. \tag{3.20}$$

From (3.20), we derive $\mathcal{L}_\pi^n$ and $\overline{\mathcal{L}}_\pi$ as follows:

$$\begin{aligned} \overline{\mathcal{L}}_\pi r(\mathbf{s},\pi(\mathbf{s})) &= \lim_{K\to\infty}\frac{1}{K+1}\mathbb{E}\{\sum_{k=0}^{K} r(\mathbf{s}_k,\pi(\mathbf{s}_k))\}, \forall \mathbf{s}\in\mathcal{S}, \\ \overline{\mathcal{L}}_\pi y(\mathbf{s},\pi(\mathbf{s})) &= \lim_{N\to\infty}\frac{1}{N+1}\mathbb{E}\{\sum_{n=0}^{N} \tau_n\}, \forall \mathbf{s}\in\mathcal{S}. \end{aligned} \tag{3.21}$$

Therefore, (3.18) is obtained by taking ratios of these two quantities. Note that the limit of the ratios equals to the ratio of the limits, and that, when taking the limit of the ratios, the factor $\frac{1}{K+1}$ can be removed from the numerator and denominator. □

Note that in our SMDP model, the embedded Markov chain is unichain including a single recurrent class and a set of transient states for all pure policies π [25]. Hence, the average reward $\mathcal{R}_\pi(\mathbf{s})$ is independent to the initial state, i.e., $\mathcal{R}_\pi(\mathbf{s}) = \mathcal{R}_\pi, \forall \mathbf{s}\in\mathcal{S}$. The average reward maximization problem is then written as:

$$\begin{aligned} \max_\pi \quad & \mathcal{R}_\pi = \frac{\overline{\mathcal{L}}_\pi r(\mathbf{s},\pi(\mathbf{s}))}{\overline{\mathcal{L}}_\pi y(\mathbf{s},\pi(\mathbf{s}))} \\ \text{s.t.} \quad & \sum_{\mathbf{s}'\in\mathcal{S}} \overline{\mathcal{L}}_\pi(\mathbf{s}'|\mathbf{s}) = 1, \forall \mathbf{s}\in\mathcal{S}. \end{aligned} \tag{3.22}$$

Our objective is to find the optimal admission policy that maximizes the average reward of the network provider, i.e.,

$$\pi^* = \operatorname*{argmax}_\pi \mathcal{R}_\pi. \tag{3.23}$$

As aforementioned, the network resources may come from multiple data centers with diverse connectivity among servers and data centers. In such a case, the above formulation can be straightforwardly extended by accommodating additional states to the system state space. Specifically, one can define the system state space that includes (i) services of requests together with their corresponding resources and order, i.e., the network slice blueprint, (ii) available resources and services at the servers, and (iii) connectivity among servers and data centers. This means that we just need to increase the state space, compared with the current state space (with three types of resources, as an example) in the current formulation, to capture additional resources and options. Then, the proposed admission/rejection framework can be implemented at the orchestrator to allocate the available resources to requested slices. Specifically, based on this state space, when a slice request arrives, the orchestrator is able to check whether to allocate an optimal possible link to the request (using existing network slicing mechanisms) and then makes a decision to accept or reject the request. In addition, after making a decision to allocate the resources for a slice request (i.e., after the initial deployment of VNFs), if the running slice requires to add more resources or remove some resources (i.e., scaling out or scaling in, respectively), we can consider some new events (i.e., requests to add or remove resources from running slices) to the system state space. Again, this implies that we only need to add more states to the system state space of the current model. Note that, the action space will be kept the same, i.e., only two actions (accept or reject), and we just need to set new rewards for accepting/rejecting requests from running slices in the problem formulation.

Note that the problem (3.22) requires environment information, i.e., arrival and completion rates of slice requests, to construct the transition probability matrix $\mathcal{L}$. Nevertheless, due to the uncertain demands and the dynamics of slice requests from tenants, these environment parameters may not be available and can be time-varying. To cope with the demand uncertainty, we consider deep double Q-learning and deep dueling algorithms proposed in Section 2.3 to find the optimal admission policy at the RMO to maximize the long-term average reward.

3.3 Performance Evaluation

3.3.1 Parameter Setting

We perform the simulations using TensorFlow [93] to evaluate the performance of the proposed solutions under different parameter settings. We consider three common classes of slices, i.e., utilities (class-1), automotive (class-2), and manufacturing (class-3). Unless otherwise stated, the arrival rates λ_c of requests from class-1, class-2, and class-3 are set at 12 requests/hour, 8 requests/hour, and 10 requests/hour, respectively. The completion rates μ_c of requests from class-1, class-2, and class-3 are set at 3 requests/hour. The immediate reward r_c for each accepted request from class-1, class-2, and class-3 are 1, 2, and 4, respectively. These parameters will be varied later to evaluate the impacts of the immediate reward on the decisions of the RMO. Each slice request requires 1 GB of storage resources, 2 CPUs for computing, and 100 Mbps of radio resources [94]. Importantly, the architecture of the deep neural network requires thoughtful design as it greatly affects the performance of the algorithm. Intuitively, increasing the number of hidden layers will increase the complexity of the algorithm. However, when the number of hidden layers is very small, the algorithm may not converge to the optimal policy. Similarly, when the size of hidden layers and mini-batch size are large, the algorithm will need more time to estimate the Q-function. In our experiment, we choose these parameters based on common settings in the literature [26, 92]. In particular, for the deep Q-learning and deep double Q-learning algorithms, two fully-connected hidden layers are implemented together with input and output layers. For the deep dueling algorithm, the neural network is divided into two streams [26]. Each stream connects to a shared hidden layer as shown in Fig. 2.6. The size of the hidden layers is 64. The mini-batch size is set at 64. Both the Q-learning algorithm and the deep reinforcement learning algorithms use ϵ-greedy algorithm with the initial value of ϵ is 1, and its final value is 0.1 [73, 95]. The maximum size of the experience replay buffer is 10,000, and the target Q-network is updated every 1,000 iterations [92], [74].

3.3.2 Simulation Results

3.3.2.1 Performance Evaluation

Comparison to Existing Network Slicing Solutions As mentioned, most existing works, e.g., [10, 11, 17–19] optimized slicing for only the radio resource. In practice, besides the radio resource, both computing and storage resource should also be accounted for while orchestrating slices. This makes existing solutions sub-optimal. In this section, we set the maximum radio resources at 500 Mbps. Each request requires 50 Mbps for radio access, 2 CPUs for computing, and 2 GB of storage resources. The computing and storage resources are then varied from 1 CPU to 9 CPUs and 1 GB to 9 GB, respectively. Fig. 3.3 shows the average reward of the system obtained by the Q-learning algorithm for the case with three resources are taken into account (as in our considered system model) and for the case with only radio resource as considered in [10, 11, 17–19]. As can be observed, when the computing and storage resources increase, the average reward is increased as more slice requests are accepted. However, the average reward of our approach (taking all radio, computing, and storage resources into account) is significantly higher than those of other solutions in the literature, especially when the amount of computing and storage resources are small. This is due to the fact that slices not only request radio resources to ensure the bandwidth for connections but also computing and storage resources to fulfill the requirements of different services.

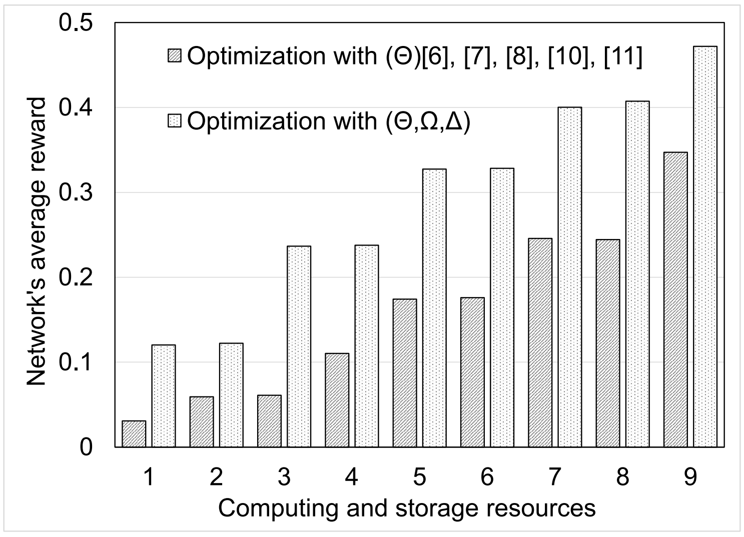


Figure 3.3 : The average reward when optimizing with one resource and three resources.

Average Reward and Network Performance Next, we compare the performance of the proposed solution, i.e., deep dueling algorithm, with other methods, i.e., Q-learning [18] and greedy algorithms [19], [96], in terms of average reward and the number of requests running in the system. For a small-size system (the maximum radio, computing, and storage resources are set at 400 Mbps, 8 CPUs, and 4 GB, respectively), Fig. 3.4 shows the average reward of the system obtained by three algorithms while varying the reward of slices from class-3 from 1 to 6. As can be seen, with the increasing of the reward of slices from class-3, the average reward of the system is increased. However, the average reward obtained by the reinforcement learning algorithms, i.e., deep dueling and Q-learning, is significantly higher than that of the greedy algorithm. This is due to the fact that the proposed reinforcement learning approaches reserve resources for coming requests that may have high rewards, while the greedy algorithm accepts slices based on the available resource of the system as shown in Fig. 3.5. It is worth noting that the achieved reward of the Q-learning algorithm is not as good as the reward obtained by the deep dueling algorithm even with small-size scenarios. This is because that the Q-learning algorithm has a slow convergence rate due to the curse-of-dimensionality problem. This observation is more pronounced when we later increase the size of the system.

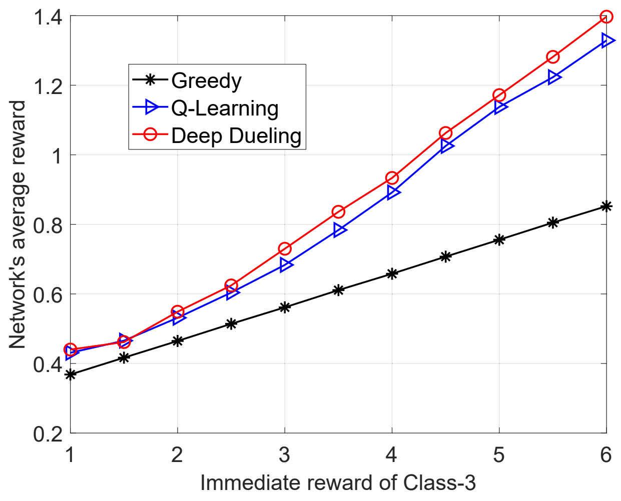


Figure 3.4 : The average reward of the system when the immediate reward of class-3 is varied.

As observed in Fig. 3.5, the number of requests running in the systems under the greedy algorithm remains the same when the immediate reward of slices from class-3 is varied. The reason is that the greedy algorithm does not consider the immediate reward of slice requests into account. In other words, upon receiving a slice request, the greedy algorithm will accept this request if the available resources of the infrastructure satisfy the slice service demands. In contrast, for the reinforcement learning algorithms, the immediate reward is also an essential factor to make optimal decisions. In particular, when the immediate reward of slice requests from class-3 increases, the algorithms are likely to reject the slice requests from classes which have lower immediate rewards, i.e., slice requests from class-1. For example, when the immediate reward of slice request from class-3 is 6, the number of requests from class-1, whose immediate reward is 1, approaches 0.

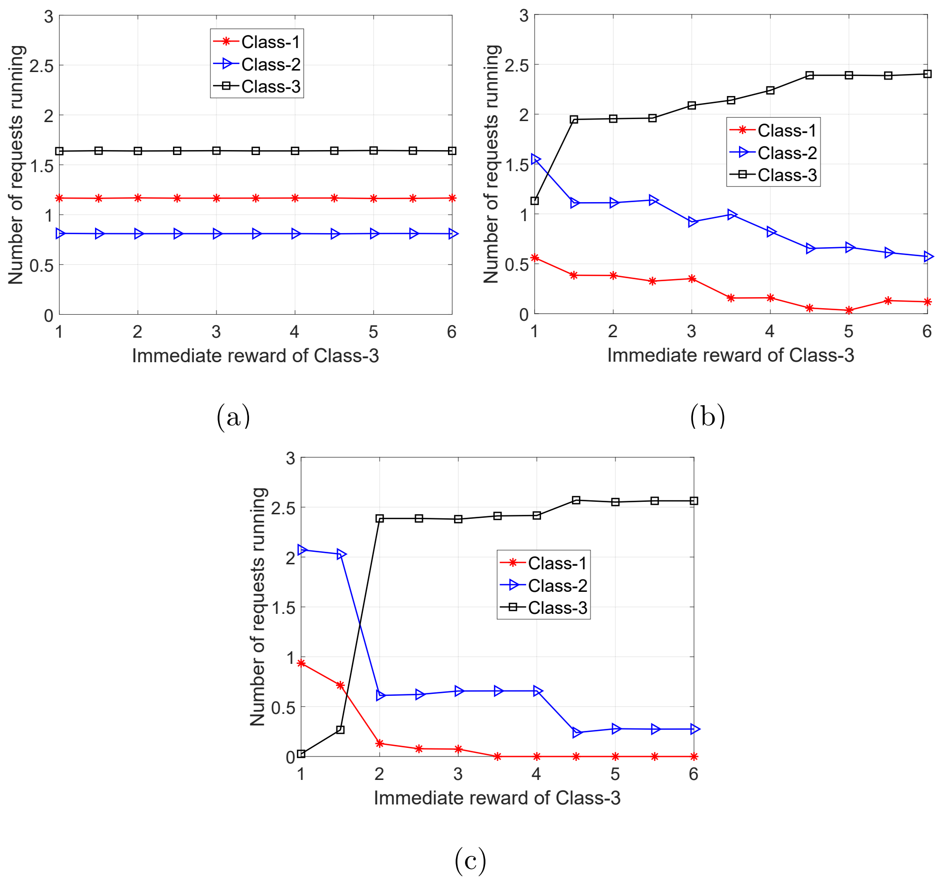


Figure 3.5 : The number of request running in the system of (a) greedy algorithm, (b) Q-learning algorithm, and (c) deep dueling algorithm when the immediate reward of class-3 is varied.

To observe the performance of the proposed solutions when the state space of the system is large, we increase the radio, computing, and storage resources to 2 Gbps, 40 CPUs, and 20 GB, respectively. The arrival rate of requests from class-1 is 48 requests/hour, from class-2 is 32 requests/hour, and from class-3 is 40 requests/hour. The completion rates from all classes are set at 2 requests/hour. Fig. 3.6 shows that the average reward obtained by the deep dueling algorithm is much higher than those of the greedy and Q-learning algorithms. This is because of the slow convergence of the Q-learning algorithm to optimality. Specifically, with 10^6 iterations, the performance of the Q-learning algorithm is just the same as that of the greedy algorithm. The performance of the Q-learning algorithm is improved with 10^7 iterations, but it is still way inferior to that of the deep dueling algorithm. For this large system scenario with over 74,000 state-action pairs, on a laptop with Intel Core i7-7600U and 16GB RAM, the deep dueling algorithm takes only about 2 hours to finish 15,000 iterations and obtain the optimal policy. This is a very practical number compared with the Q-learning algorithm that cannot obtain the optimal policy within 10^7 iterations (more than 15 hours). In practice, with specialized hardware and much more powerful computing resource (compared with our laptop) at the network provider (e.g., GPU cards from NVIDIA), the deep dueling algorithm should take much shorter time than 2 hours to finish 15,000 iterations [97]. These results confirm that the Q-learning algorithm, despite its optimality, requires a much longer time to converge, compared with the deep dueling algorithm.

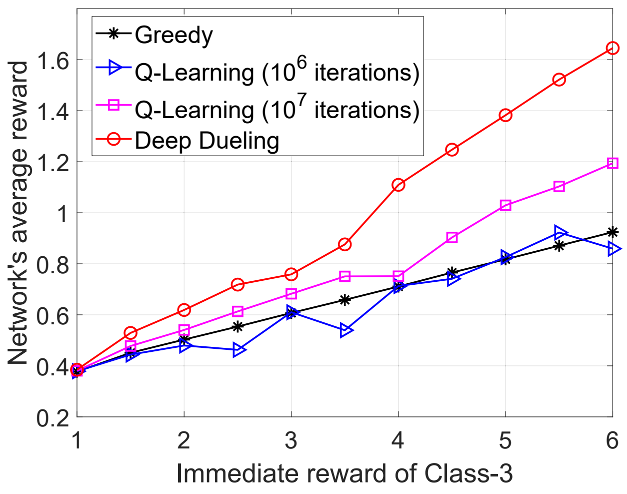


Figure 3.6 : The average reward of the system when the immediate reward of class-3 is varied.

Similar to the case in Fig. 3.5, as shown in Fig. 3.7, the deep dueling and Q-learning algorithms reserve resources for slices from classes which have high immediate rewards. However, the deep dueling algorithm achieves better performance compared to the Q-learning algorithm. For example, when the immediate reward of slices from class-3 is 6, the number of requests running in the systems is about 16 requests and 11 requests for the deep dueling and the Q-learning algorithms, respectively.

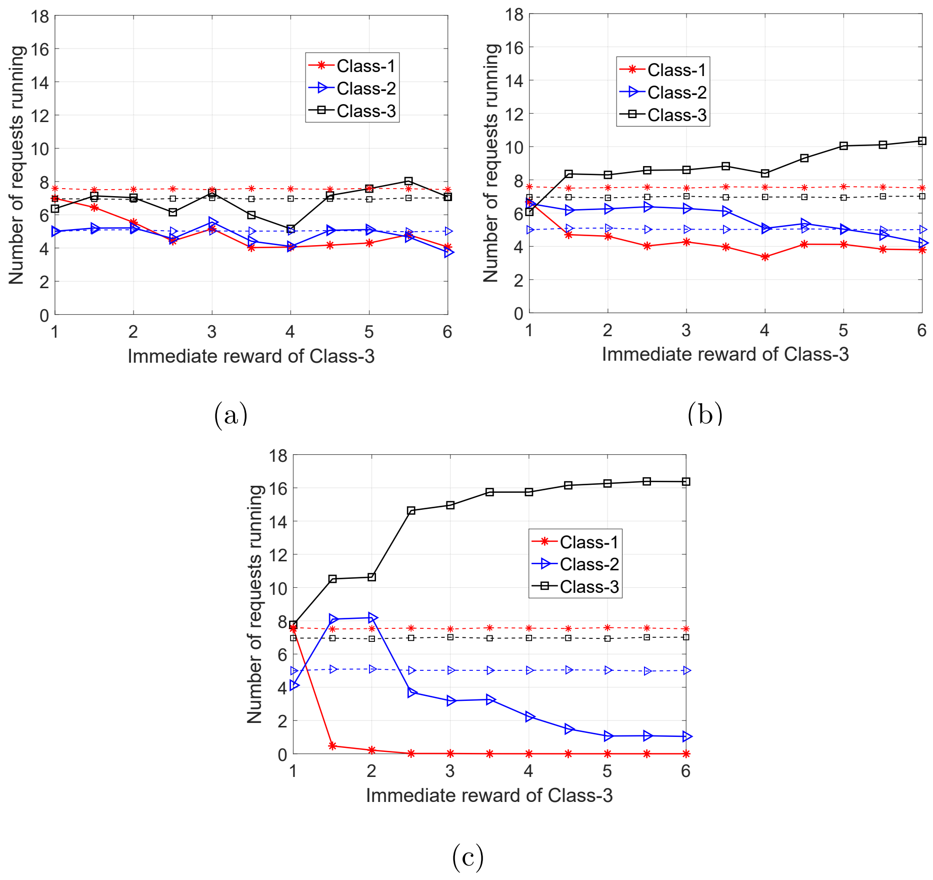


Figure 3.7 : The number of request running in the system of (a) Q-learning algorithm (10^6 iterations), (b) Q-learning algorithm (10^7 iterations), and (c) deep dueling algorithm (20,000 iterations) when the immediate reward of class-3 is varied. The dash lines are results of the greedy algorithm.

In summary, in all the cases, the deep dueling algorithm always achieves the best performance in terms of the average reward and network performance.

Optimal Policy In Fig. 3.8, we examine the optimal policy of the deep dueling and Q-learning algorithms. Specifically, we set the maximum resources of the system at 4 times, 10 times, and 20 times of resources requested by a slice and evaluate the policy of the algorithms with different available resources in the system as shown in Fig. 3.8(a), Fig. 3.8(b), and Fig. 3.8(c), respectively. Note that the lines Q-learning{1,2,3} and Deep Dueling{1,2,3} represent the probabilities of accepting requests from class-{1,2,3} by using the Q-learning and deep dueling algorithms, respectively. Clearly, in three cases, the deep dueling always obtains the best policy. In particular, it will reject almost all requests from class-1 (lowest immediate reward) when there are few available resources in the system. When the available resources in the system increase, the probability of accepting a request from class-1 is also increased. Note that, when the maximum system resource capacity is large, i.e., 20 times of resources requested by a slice, the performance of the Q-learning is fluctuated as it cannot converge to the optimal policy event with 10^7 iterations.

3.3.2.2 Convergence of Deep Reinforcement Learning Approaches

Next, we show the learning process and the convergence of the deep reinforcement learning approaches, i.e., deep Q-learning, deep double Q-learning, and deep dueling, in different scenarios. As shown in Fig. 3.9(a), when the maximum radio, computing, storage resources are 400 Mbps, 8 CPUs, and 4 GB, respectively, the convergence rates of the three deep Q-learning algorithms are considerably higher than that of the Q-learning algorithm. Specifically, while the deep reinforcement learning approaches converge to the optimal value within 10,000 iterations, the Q-learning need more than 10^6 iterations to obtain the optimal policy. This is stemmed from the fact that in the system under consideration, the state space is dimensional and the system dynamically changes over time. In Fig. 3.9(b), we show the convergence of the Q-learning and deep dueling algorithms in the first 20,000 iterations to clearly verify this observation. On the contrary, by implementing the neural network with fully-connected layers, the deep reinforcement algorithms can efficiently reduce the

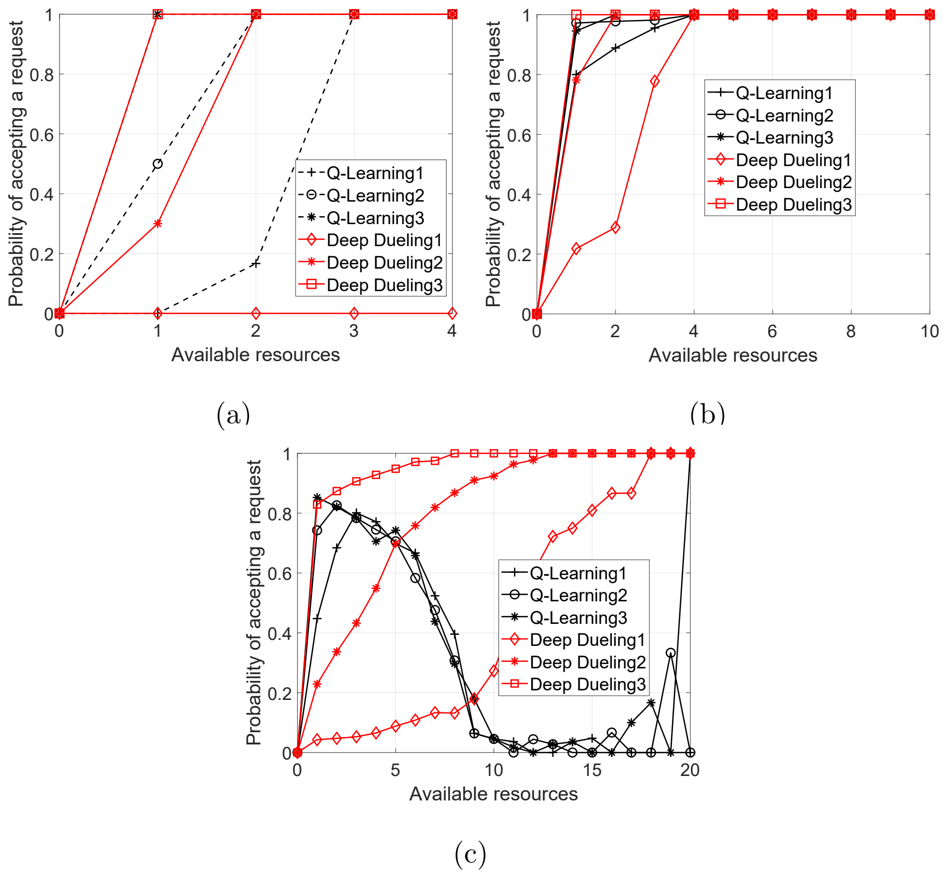
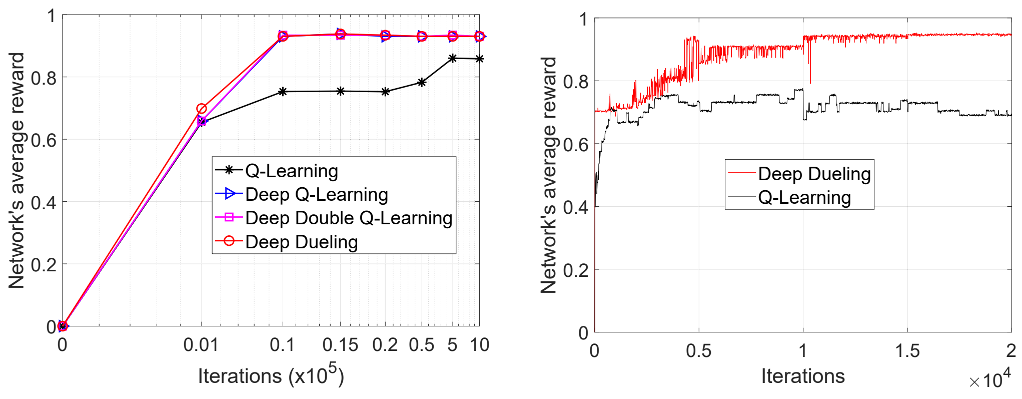


Figure 3.8 : The probabilities of accepting a request from classes when the maximum available resources of the system is (a) 4 times, (b) 10 times, and (c) 20 times of resources requested by a slice.

curse of dimensionality, thereby improving the convergence rate.

We continue to increase the radio, storage, computing resources to 1 Gbps, 10 GB, and 20 CPUs, respectively. The arrival rates of classes are increased by 4 times, i.e., $\lambda_1 = 48$, $\lambda_2 = 32$, and $\lambda_3 = 40$ requests/hour, while the completion rates are equal to 2 requests/hour for all classes. As shown in Fig. 3.10(a), the performance of the deep reinforcement algorithms is significantly higher than that of the Q-learning algorithm. It is important to note that as the state space now is more complicated than in the previous case, the deep dueling algorithm obtains the optimal policy within 15,000 iterations, while the other two deep reinforcement



(a) (b)

Figure 3.9 : The convergence of reinforcement learning algorithms when the radio, computing, storage resources are 400 Mbps, 8 CPUs, and 4 GB, respectively with (a) 10^6 iteration and (b) 20,000 iterations.

learning approaches require more time to converge to the optimal policy. We keep increasing the radio, storage, computing resources to 2 Gbps, 20 GB, and 40 CPUs, respectively and observe the convergence rate of the deep reinforcement algorithms as shown in Fig. 3.10(b). Clearly, as now the system is very complicated, the deep dueling can achieve the optimal policy within 20,000 iterations while the deep Q-learning and deep double Q-learning algorithms cannot converge to the optimal policy after 100,000 iterations. This is due to the fact that by decoupling the neural network into two streams, the deep dueling algorithm can significantly reduce the overestimation of the optimizer, i.e., stochastic gradient descent.

Next, we show the effects of the learning rate on the performance of the deep dueling algorithm. The learning rate is the most critical hyper-parameters to tune for training deep neural networks. If the learning rate is too slow, the training process is more reliable but requires a long time to converge to the optimal policy. In contrast, if the learning rate is too high, the algorithm may not converge to the optimal policy or even diverge. This is stemmed from the fact that the deep

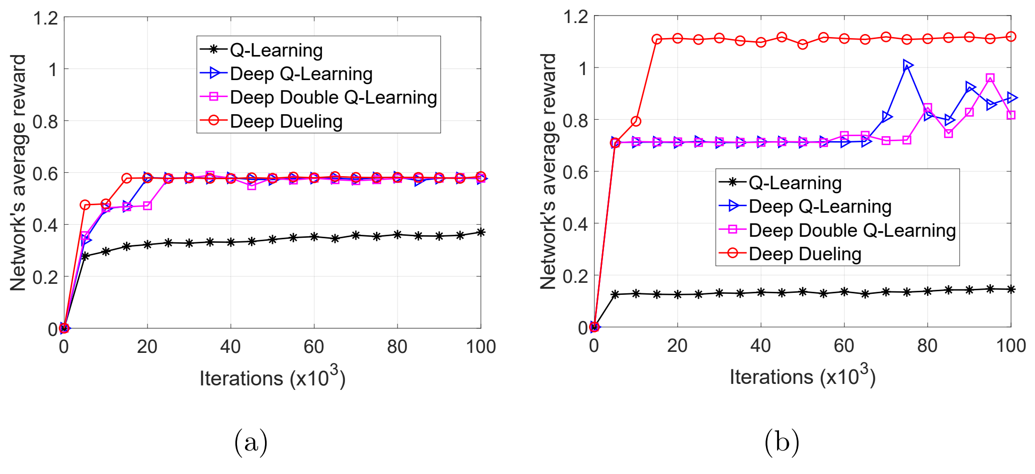


Figure 3.10 : The convergence of reinforcement learning algorithms when (a) the radio, computing, storage resources are 1 Gbps, 20 CPUs, and 10 GB, respectively and (b) the radio, computing, storage resources are 2 Gbps, 20 GB, and 40 CPUs, respectively.

dueling algorithm uses the gradient descent method. If the learning rate is too large, gradient descent may overshoot the optimal point, and thus resulting in poor performance. This observation is proved by simulation results as shown in Fig. 3.11. Specifically, with the learning rate of 0.01, the deep dueling algorithm achieves the best performance in terms of the average reward and the convergence rate compared to other learning rates.

3.4 Conclusion

In this chapter, we have developed an optimal and fast network resource management framework which allows the network provider to jointly allocate multiple combinatorial resources (i.e., computing, storage, and radio) to different slice requests in a real-time manner. To deal with the dynamic and uncertainty of slice requests, we have adopted the SMDP. Then, the reinforcement learning algorithms, i.e., Q-learning, deep Q-learning, deep double Q-learning, and deep dueling, have

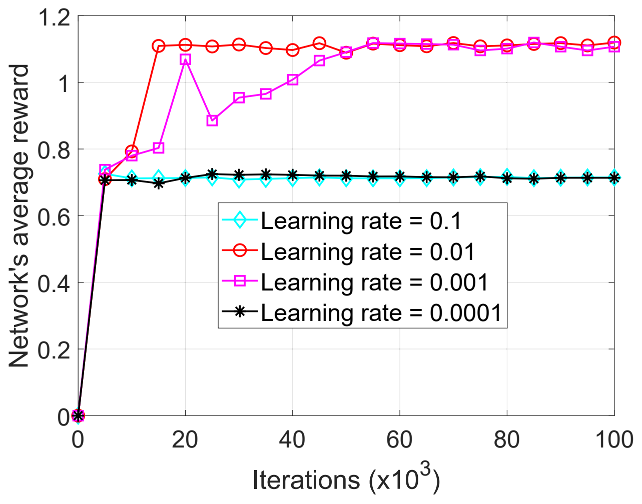


Figure 3.11 : The performance of deep dueling algorithm with different learning rates.

been employed to maximize the long-term average reward for the network provider. The key idea of the deep dueling is using two streams of fully connected hidden layers to concurrently train the value and advantage functions, thereby improving the training process and achieving the outstanding performance for the system. Extensive simulations have shown that the proposed framework using deep dueling can yield up to 40% higher long-term average reward with few thousand times faster compared with those of other network slicing approaches. Future works comprise considering the connectivity resources and the existence of multiple data centers in complex network slicing models by accommodating more states to the system state space. The performance of the proposed solution will be evaluated in terms of complexity and scalability. Moreover, the convergence rate and stability of the deep dueling algorithm will be improved by using the state-of-the-art deep neural networks.

Chapter 4

“Jam Me If You Can”: Defeating Jammers with Deep Dueling Neural Network Architecture and Ambient Backscattering Augmented Communications

With conventional anti-jamming solutions like frequency hopping or spread spectrum, legitimate transceivers often tend to “escape” or “hide” themselves from jammers. These anti-jamming approaches are constrained by the lack of timely knowledge of jamming attacks (especially from smart jammers). Bringing together the latest advances in neural network architectures and ambient backscattering communications, this thesis allows wireless nodes to effectively “face” the jammer (instead of escaping) by first learning its jamming strategy, then adapting the rate or transmitting information right on the jamming signals (i.e., backscattering modulated information on the jamming signals). Specifically, to deal with unknown jamming attacks (e.g., jamming strategies, jamming power levels, and jamming capability), existing work often relies on reinforcement learning algorithms, e.g., Q-learning. However, the Q-learning algorithm is notorious for its slow convergence to the optimal policy, especially when the system state and action spaces are large. This makes the Q-learning algorithm pragmatically inapplicable. To overcome this problem, we design a novel deep reinforcement learning algorithm using the recent dueling neural network architecture. Our proposed algorithm allows the transmitter to effectively learn about the jammer and attain the optimal countermeasures (e.g., adapt the transmission rate or backscatter or harvest energy or stay idle) thousand times faster than that of the conventional Q-learning algorithm. Through extensive simulation results, we show that our design (using ambient backscattering and the deep dueling neural network architecture) can improve the average throughput by up to 426% and reduce the packet loss by 24%. By augmenting the ambient backscattering capability on devices and using our algorithm, it is interesting to observe that the (successful) transmission rate increases with the jamming power. Our proposed solution can find its applications in both civil (e.g., ultra-reliable and low-latency communications or URLLC) and military scenarios (to combat both inadvertent and deliberate jamming).

The rest of this chapter is organized as follows. Section 4.1 and Section 4.2 describe the system model and the problem formulation, respectively. After that, the evaluation results are discussed in Section 4.3. Finally, conclusions are drawn in Section 4.4.

4.1 System Model

We consider a wireless system consisting of a gateway and a transmitter as illustrated in Fig. 4.1. The transmitter is equipped with a data buffer to store data before transmitting to the gateway. In addition, we assume that the transmitter is equipped with an energy harvesting circuit and an energy storage. The energy harvesting circuit is used to harvest energy from surrounding signals, and then the harvested energy will be stored in the energy storage for future use. We consider an ambient RF source, e.g., an FM radio tower, that is located near the system, and thus the transmitter can harvest energy from the RF energy source when the source is active, i.e., broadcasts signals. Then, the transmitter can use the harvested energy to transmit data to the gateway when the RF energy source becomes idle. This transmission scheme is also known as the harvest-then-transmit protocol that is well known in the literature [40].

Note that although we consider a single transmitter in this work, the proposed

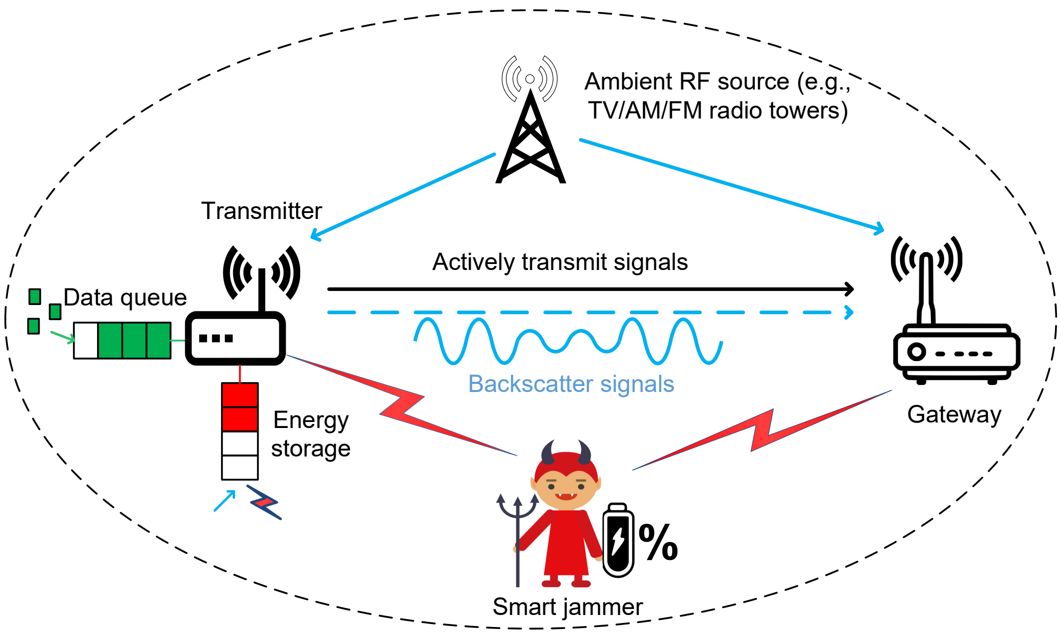


Figure 4.1 : System model

model and analysis can be extended to the case with multiple transmitters. In such a case, one can adopt popular scheduling mechanisms to avoid the collision between transmitters. Another approach is that transmitters backscatter data at different rates [39], [40]. In this way, the gateway can decode the information in the backscattered signals by leveraging the difference in communication rates.

4.1.1 Smart Jammer with Self-Interference Suppression Capability

We consider a* smart jammer with self-interference suppression (SiS) capability. With the latest advances in SiS [30], the jammer can "listen" to the channel while jamming. That allows the jammer to instantaneously discern its jamming outcome and optimize its jamming strategy to maximize the disruption of the victims. The

*Note that our system model can be extended straightforwardly to the case with multiple jammers who perform attacks to the channel cooperatively, i.e., only one jammer attacks the channel at a time.

SINR at the gateway is formally calculated as follows [3], [36]:

$$\theta = \frac{P^{\mathrm{R}}}{\phi P^{\mathrm{J}} + \rho^2}, \tag{4.1}$$

where P^{R} is the received power from the transmitter at the gateway, P^{J} is the jamming power transmitted by the jammer, ρ^2 is the variance of additive white Gaussian noise, and ϕP^{J} expresses the jamming power received at the gateway in which $0 \leq \phi \leq 1$ is an attenuation factor. Note that we consider the case in which the transmitter operates as a secondary user that shares the spectrum with the ambient RF source and only actively transmits data when the ambient RF source is inactive. As such, Eq. (4.1) above does not need to account for the ambient RF signals. Our work can be extended to the case in which both the transmitter and the ambient RF source operate on (different) licensed frequencies.

In practice, the jammer can adjust its pulse duty cycle factor to achieve the maximum degradation on the target channel while maintaining a time-average power constraint P_{avg}. Note that the average power P_{avg} should be less than the peak jamming power P_{max}, i.e., $P_{\mathrm{avg}} \leq P_{\mathrm{max}}$ [36]. Specifically, let $\mathbf{P}_{\mathrm{J}} = \{P^{\mathrm{J}}_0, \ldots, P^{\mathrm{J}}_n, \ldots, P^{\mathrm{J}}_N\}$ denote the vector of discrete jamming power levels. In each time slot, the jammer can select any transmit power level P^{J}_n as long as its average power constraint is satisfied. If we denote $\mathbf{x} \triangleq \{x_0, \ldots, x_n, \ldots, x_N\}$ as a probability vector, then the strategy space of the jammer, denoted by $\mathbf{J}_{\mathrm{s}}$, can be defined as follows:

$$\begin{aligned} \mathbf{J}_s \triangleq \Big\{ &(x_0, \ldots, x_n, \ldots, x_N), \sum_{n=0}^{N} x_n = 1, \\ &x_n \in [0,1], \forall n \in \{0, \ldots, N\}, \mathbf{x}\mathbf{P}_{\mathrm{J}}^{\top} \leq P_{\mathrm{avg}} \Big\}. \end{aligned} \tag{4.2}$$

To find the defense policy in the worst case, as mentioned above, we consider a smart jammer that would know the information of the transmitter, e.g., how many packets the transmitter can transmit/backscatter and how many packets it can bring down if the jamming is successful (thanks to the SiS capability). In such a case, based on this information and its given average power constraint P_{avg}, the jammer will find an optimal strategy to attack the channel in order to maximize the disruption. In particular, we assume that the jammer receives a reward w_n^{J} if it attacks the channel with power level P_n^{J}. w_n^{J} can be referred as the number of packets that have been completely corrupted (i.e., not being successfully received/decoded, hence not ACKed by the receiver) if the jamming power is P_n^{J}. Let $\mathbf{w}_{\text{J}} = \{w_0^{\text{J}}, \dots, w_n^{\text{J}}, \dots, w_N^{\text{J}}\}$ denote the reward vector of the jammer. Thus, the objective function of the jammer can be defined as follows:

$$\begin{aligned} &\max_{\mathbf{x}} \quad \mathbf{x}\mathbf{w}_{\text{J}}^{\top}, \\ &\text{s.t.} \begin{cases} \sum_{n=0}^{N} x_n = 1, \\ x_n \in [0,1], \forall n \in \{0, \dots, N\}, \\ \mathbf{x}\mathbf{P}_{\text{J}}^{\top} \leq P_{\text{avg}}. \end{cases} \end{aligned} \tag{4.3}$$

4.1.2 Ambient Backscattering-Augmented Communications

To defeat the above smart jamming attack, we propose a novel communications scheme, namely "ambient backscatter-augmented communications". Our high-level circuit architecture is shown in Fig. 4.2.

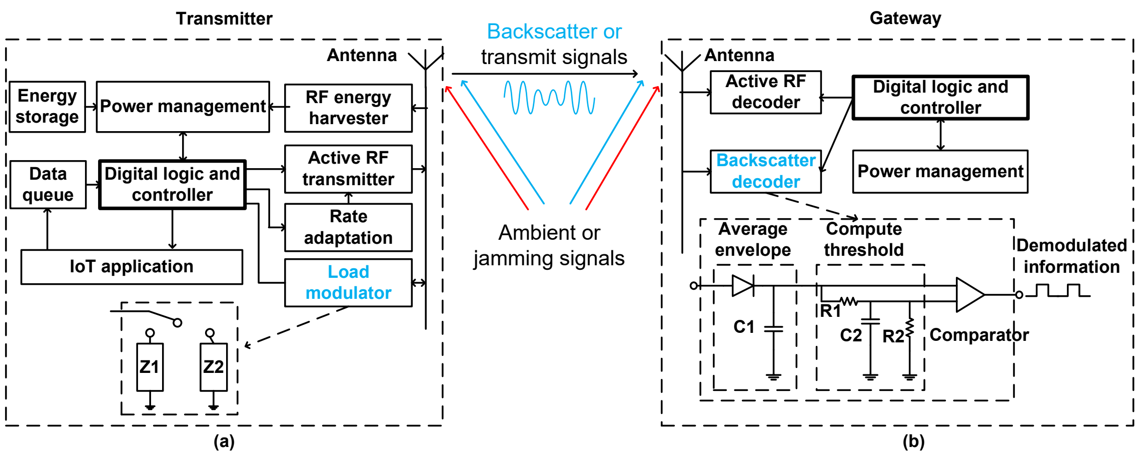


Figure 4.2 : Function and circuit diagram of the proposed anti-jamming system

In Fig. 4.2(a), we show the proposed circuit diagram that allows the transmitter to be able to harvest energy, perform active RF transmission, and backscatter information. The circuit design for the integration of energy harvesting and backscattering communication has also been considered in several research works in the literature such as [39] and [98]. The architecture of the transmitter consists of an antenna, a controller, an energy harvester, an RF transmitter for active transmissions, an energy storage, a data buffer, and a load modulator together with a backscatter decoder for ambient backscatter communications. The controller takes responsibilities to make decisions, e.g., stay idle, transmit data, backscatter data, and harvest energy, for the transmitter. When the ambient RF source is active and/or the jammer attacks the channel, if the transmitter chooses to harvest energy, it will use the energy harvester to harvest energy from the ambient signals or the jamming signals. The harvested energy is then stored in the energy storage and used when the transmitter decides to actively transmit data to the gateway. In contrast, if the transmitter chooses to backscatter to immediately transmit data to the gateway, the transmitter will modulate and reflect the ambient RF signals or the jamming signals by using the load modulator [39]. In particular, the load modulator consists of an RF switch, e.g., ADG902, directly connected to the antenna. The input of the load modulator is a stream of one and zero bits which is generated by the controller depending on the application. When the input bit is zero, the load modulator switches to load Z_1, and thus the transmitter is in the non-reflecting state. Otherwise, when the input bit is one, the load modulator turns to load Z_2, and thus the transmitter is in the reflecting state. By doing so, the transmitter can backscatter its data to the gateway. It is worth noting that while operating in the backscatter mode, the transmitter still can harvest energy (in the non-reflecting state), but the amount of the harvested energy is relatively small and only suitable for operations in the backscatter mode [39], [40].

To allow the gateway to decode backscattered signals, the transmitter backscatters information at a lower rate than the ambient signals, i.e., jamming signals and ambient signals. For completeness, we formally describe the principle as follows. We assume that we have a digital receiver that samples the received signals at the Nyquist-information rate, e.g., using ADC [39]. The received signals at the gateway is sampled to $y[n]$ as:

$$y[n] = x[n] + \zeta B[n]x[n] + l[n], \tag{4.4}$$

where $x[n]$s are the samples of the ambient signals (i.e., the jamming signals, the ambient RF signals, or both of them) received at the gateway, $l[n]$ is the noise, ζ is the complex attenuation of the backscattered signals, and $B[n]$s are the bits transmitted by the transmitter (through the load modulator). If the transmitter sends information at a fraction of the rate, say $\frac{1}{N}$, then $B[Ni + j]$ are all equal for $j = 1$ to N [39]. Then, the gateway averages powers of N received samples as follows:

$$\frac{1}{N}\sum_{i=1}^{N} |y[n]|^2 = \frac{1}{N}\sum_{i=1}^{N} |x[n] + \zeta B x[n] + w[n]|^2, \tag{4.5}$$

where B takes a value of '0' or '1' depending on the non-reflecting and reflecting states, respectively. As $x[n]$ is uncorrelated with the noise $w[n]$, (4.5) can be expressed as follows:

$$\frac{1}{N}\sum_{i=1}^{N} |y[n]|^2 = \frac{|1+\zeta B|^2}{N}\sum_{i=1}^{N} |x[n]|^2 + \frac{1}{N}\sum_{i=1}^{N} w[n]^2. \tag{4.6}$$

Denote $P = \frac{1}{N}\sum_{i=1}^{N} |x[n]|^2$ as the average power of the received jamming signals (or ambient signals). Ignoring the noise, the average power at the receiver is $|1+\zeta|^2 P$ and P when the backscatter transmitter is at the reflecting ($B = 1$) state and the non-reflecting ($B = 0$) state, respectively. Based on the differences between $|1+\zeta|^2 P$ and P, the backscatter receiver can decode the data from the backscattered signals with a conventional digital receiver [39].

However, the ADC consumes a significantly amount of power to sample the received signals. For a low-power design, in Fig. 4.2(b), we describe a circuit diagram using only analog components to decode the backscattered signals. In particular, the gateway is equipped with an antenna to receive information (either active transmission or backscatter information) from the transmitter. Based on the received signals, the gateway will decide to use a suitable mode to decode information from the transmitter. In particular, the active RF decoder is used to decode active transmission signals from the transmitter. If the transmitter transmits data by using the backscatter mode, the gateway will use the backscatter decoder to extract the information from the backscattered signals. Specifically, at the backscatter decoder, the backscattered signals are first smoothed by the envelope-averaging circuit. After that, the compute-threshold circuit produces an output voltage between low and high levels of the smoothed signals. Then, the comparator compares the signals with a predefined threshold to derive output bits zero and one properly. The more detailed information about hardware designs as well as decoding algorithms at the receiver can be found in [39].

4.1.3 System Operation

We denote the probability of the ambient RF source being idle in each time slot by η. Due to the constraints on average power P_{avg} and maximum transmit power $P_{\max}$, the jammer may attack the channel with different power levels at different times. When the jammer attacks the channel and the ambient RF source is idle, the transmitter can choose one of the following actions (i) go to sleep mode, i.e., stay idle, (ii) harvest energy from the jamming signals, (iii) backscatter information based on the jamming signals, or (iv) adapt its transmission rate by using rate adaption (RA) techniques [3, 36]. Depending on the transmit power level P_n^{J} of the jammer, the transmitter can harvest e_n^{J} units of energy and backscatter maximum $\widehat{d}_n^{\mathrm{J}}$

packets through the jamming signals. In practice, the more power the jammer uses to attack the channel, the more energy the transmitter can successfully harvest from the jamming signals [†]. In addition, through many real experiments and analysis on backscatter communication systems in the literature [39, 41, 98], it can be observed that the more power the jammer uses to attack the channel, the more energy per information bit the transmitter can backscatter to the gateway, and thus the less Bit Error Rate (BER) of backscatter communication is. This also implies that more packets the transmitter can successfully transmit to the gateway by backscattering the jamming signals when the jammer uses higher power levels to attack. We denote $\mathbf{e} = \{e^{\mathrm{J}}_0, \dots, e^{\mathrm{J}}_n, \dots, e^{\mathrm{J}}_N\}$ as the amount of energy that the transmitter can successfully harvest from the jamming signals when the jammer attacks the channel with power level $\mathbf{P}_{\mathrm{J}} = \{P^{\mathrm{J}}_0, \dots, P^{\mathrm{J}}_n, \dots, P^{\mathrm{J}}_N\}$, respectively. Similarly we denote $\widehat{\mathbf{d}} = \{\widehat{d}^{\mathrm{J}}_0, \dots, \widehat{d}^{\mathrm{J}}_n, \dots, \widehat{d}^{\mathrm{J}}_N\}$ as the number of packets that the transmitter can successfully transmit to the gateway when the jammer attacks the channel with power level $\mathbf{P}_{\mathrm{J}} = \{P^{\mathrm{J}}_0, \dots, P^{\mathrm{J}}_n, \dots, P^{\mathrm{J}}_N\}$, respectively.

In practice, when the jammer attacks the channel and the ambient RF source does not transmit data, the transmitter still can transmit its data by reducing its data rate. Specifically, based on jamming power $P^{\mathrm{J}}_{\mathrm{n}}$, the transmitter can actively transmit data at maximum rate r_m. We then denote $\mathbf{r} = \{r_1, \dots, r_m, \dots, r_M\}$ as the set of available transmission rates that the transmitter can choose to transmit data when the jammer attacks the channel. At each rate r_m, the transmitter can transmit maximum $\widehat{d}^{\mathrm{r}}_m$ packets. Note that, for $m = 1, \dots, M$, when $\gamma_{m-1} \leq \theta < \gamma_m$ with γ_m is the value of SINR, the gateway can only decode packets sent at rates $r_0, r_1, \dots, r_{m-1}$, and the packets sent at rate r_m or higher will be completely lost [3]. To detect the states of the ambient RF source and the jammer, several detection techniques can

[†]Based on the Friis equation [99], we also can obverse the proportional relationship between the amount of harvested energy and the transmission power of an energy source, i.e., the jammer.

be adopted, e.g., energy detection [100], [101]. Note that there are miss detection and false alarm probabilities when detecting the states of channels. However, our proposed algorithm can learn these probabilities and dynamically adjust its optimal policy.

In this work, we define the packet delivery ratio (PDR) as the ratio of packets that are successfully delivered to the gateway over the total number of packets arriving at the system. The arrival data process follows the Poisson distribution with mean rate λ. The maximum data queue size and energy storage capacity are denoted by D and E, respectively. If a packet arrives at the system when the data queue is full, it will be dropped. To consider a low-latency system, if a packet stays in the queue longer than a latency threshold, i.e., t_{th}, it will be discarded.

If at least one of the sources (i.e., either the ambient RF source, or the jammer, or both of them) is active, the transmitter can choose to backscatter data or harvest energy. The transmitter then observes the results of the taken action, i.e., the total number of packets backscattered or the total amount of harvested energy, and update the learning function. Based on the states of the ambient RF source and the jammer, the operations of our system can be expressed as follows:

- *When the ambient RF source is idle and the jammer does not attack the channel*: the transmitter can (i) transmit maximum $\widehat{d}_{\mathrm{t}}$ packets if it has enough energy (each packet requires e_{t} units of energy to be successfully transmitted) or (ii) stay idle.
- *When the ambient RF source is idle and the jammer attacks the channel with power level P_n^{J}*: the transmitter can (i) use the RA technique to transmit maximum $\widehat{d}_m^{\mathrm{r}}$ packets if it has enough energy, (ii) backscatter maximum $\widehat{d}_n^{\mathrm{J}}$ packets, (iii) harvest e_n^{J} units of energy, or (iv) stay idle.
- *When the ambient RF source is active and the jammer does not attack the*

channel: the transmitter can choose to (i) backscatter maximum $\widehat{d}_{\mathrm{b}}$ packets, (ii) harvest e_{h} units of energy, or (iii) stay idle.

- *When the ambient RF source is active and the jammer attacks the channel with the power level P_n^{J}*: the transmitter can choose to (i) backscatter d_{sum} packets with $d_{\mathrm{min}} \leq d_{\mathrm{sum}} \leq d_{\mathrm{max}}$ where $d_{\mathrm{min}} = \min(\widehat{d}_{\mathrm{b}}, \widehat{d}_n^{\mathrm{J}})$ and $d_{\mathrm{max}} = \widehat{d}_{\mathrm{b}} + \widehat{d}_n^{\mathrm{J}}$[‡], (ii) harvest e_{sum} units of energy with $e_{\mathrm{min}} \leq e_{\mathrm{sum}} \leq e_{\mathrm{max}}$ where $e_{\mathrm{min}} = \max(e_{\mathrm{h}}, e_n^{\mathrm{J}})$ and $e_{\mathrm{max}} = e_{\mathrm{h}} + e_n^{\mathrm{J}}$ [102][§], or (iii) stay idle.

In this work, time is slotted. In each time slot, given a particular channel condition and states of the ambient RF source and the jammer, the amount of harvested energy and the number of backscattered/transmitted packets, i.e., $\widehat{d}_{\mathrm{t}}$, $\widehat{d}_m^{\mathrm{r}}$, $\widehat{d}_n^{\mathrm{J}}$, e_n^{J}, $\widehat{d}_{\mathrm{b}}$, e_{h}, d_{sum}, e_{sum}, can be observed after interacting with the environment. Our proposed deep dueling algorithm does not require this explicit information in advance. Instead, the algorithm learns these values and converges to the optimal policy for the transmitter.

4.2 Problem Formulation

To deal with the uncertainty of jamming attacks and ambient RF signals, we adopt the MDP framework to formulate the optimization problem of the system. This framework allows the transmitter to dynamically make optimal actions based

[‡]The backscatter rate when both sources are active, d_{sum}, should be in between the minimum of the backscatter rates of individual sources and the summation of them. In general, we assume that d_{sum} is unknown and captured by a random variable with a given distribution in the above range.

[§]The harvested energy when both sources are active, e_{sum}, should be in between the maximum of the energy harvested from each individual source and the summation of them. In general, we assume that e_{sum} is unknown and captured by a random variable with a given distribution in the above range.

on its observations to maximize its average long-term reward. The MDP is defined by a tuple $< \mathcal{S}, \mathcal{A}, r >$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, and r is the immediate reward of the system.

4.2.1 State Space

We define the state space of the system as follows:

$$\begin{aligned} \mathcal{S} \triangleq \Big\{(c, j, d, e) :& c \in \{0, 1\}; j \in \{0, 1\}; \\ & d \in \{0, \dots, D\}; e \in \{0, \dots, E\}\Big\}, \end{aligned} \tag{4.7}$$

where c represents the state of the ambient RF channel, i.e., $c = 1$ when the ambient RF channel is busy and $c = 0$ otherwise. j represents the state of the jammer, i.e., $j = 1$ when the jammer is active and $j = 0$ otherwise. d and e represent the number of packets in the data queue and the energy units in the energy storage of the transmitter, respectively. D and E are the maximum data queue size and energy storage capacity, respectively. The system state is then defined as a composite variable $s = (c, j, d, e) \in \mathcal{S}$. Note that the transmitter can always obtain the system state. In particular, the states of the data queue and energy queue are always available to the transmitter. Moreover, the transmitter can detect the activity of the jammer and the ambient RF source by determining their signals strengths through common signal detection techniques, e.g., energy detection.

4.2.2 Action Space

The transmitter can perform one of the $(M + 4)$ actions, i.e., stay idle, actively transmit data, harvest energy, backscatter data, or actively transmit data when then channel is attacked with one of M transmission rates by using the RA technique. Then, the action space of the transmitter can be defined by $\mathcal{A} \triangleq \{a : a \in \{1, \dots, M+$

4}}, where

$$a = \begin{cases} 1, & \text{the transmitter stays idle,} \\ 2, & \text{the transmitter transmits data,} \\ 3, & \text{the transmitter harvests energy,} \\ 4, & \text{the transmitter backscatters data,} \\ 4+m, & \text{the transmitter adapts its transmission} \\ & \text{to rate } r_m \text{ with } m \in \{1, \dots, M\}. \end{cases} \tag{4.8}$$

4.2.3 Immediate Reward

We define the reward for the system as the number of packets that are successfully transmitted to the gateway. Thus, the immediate reward of the system after the transmitter makes an action a_t at state s_t can be defined as follows:

$$r_t(s_t, a_t) = \begin{cases} d_{\mathrm{t}}, & \text{if } c = 0, j = 0, d > 0, e \geq e_{\mathrm{t}}, \\ & \text{and } a = 2, \\ d_{\mathrm{b}}, & \text{if } c = 1, j = 0, d > 0, \text{and } a = 4, \\ d_n^{\mathrm{J}}, & \text{if } j = 1, c = 0, d > 0, \text{and } a = 4, \\ d_{\mathrm{sum}}, & \text{if } j = 1, c = 1, d > 0, \text{and } a = 4 \\ d_m^{\mathrm{r}}, & \text{if } c = 0, j = 1, d > 0, e > 0, \\ & \text{and } a = 4+m, \\ 0, & \text{otherwise.} \end{cases} \tag{4.9}$$

In the above, when the ambient RF source is idle, the jammer does not attack the channel, and the number of data and energy units are sufficient for active transmission, the transmitter can actively transmit $0 < d_{\mathrm{t}} \leq \widehat{d_{\mathrm{t}}}$ packets to the gateway (i.e, $a_t = 2$). When the ambient RF source is active, the jammer is idle, and the transmitter has data to transmit, it can choose to backscatter $0 < d_{\mathrm{b}} \leq \widehat{d_{\mathrm{b}}}$ packets(i.e., $a_t = 4$). Similarly, when the jammer attacks the channel, the RF source is idle, and the transmitter has data to transmit, if it chooses to backscatter, it can transmit maximum $0 < d_n^{\mathrm{J}} \leq \widehat{d_n^{\mathrm{J}}}$ packets(i.e., $a_t = 4$). If the ambient RF source is idle, the jammer attacks the channel, and the transmitter has enough energy and data in the queues, it can choose to adapt its rate (i.e., $a_t = 4+m; m \in \{1, \dots, M\}$) and actively transmit $0 < d_m^{\mathrm{r}} \leq \widehat{d_m^{\mathrm{r}}}$ packets to the gateway. If both the jammer and the RF source are active, and the transmitter has data to transmit, if it chooses to backscatter data, it can transmit $d_{\mathrm{min}} \leq d_{\mathrm{sum}} \leq d_{\mathrm{max}}$ to the gateway [102]. Finally, the immediate reward is equal to 0 if the transmitter cannot successfully transmit any packet to the gateway.

Note that after performing an action, the transmitter will observe the results from the environment including reward, i.e., number of packets that are successfully transmitted based on ACK messages sent from the gateway. In other words, d_{t}, d_{b}, d_n^{J}, d_{sum}, and d_m^{r} are the actually received packet at the gateway, i.e., successfully-ACKed packets. For that, the reward function captures the overall path between the source and the tag-receiver, e.g., fading, end-to-end SNR, BER, or the packet error rate.

4.2.4 Optimization Formulation

We formulate an optimization problem to obtain the optimal policy, denoted by π^*, that maximizes the average long-term throughput for the system. Specifically, the optimal policy is a mapping from a state to an action taken by the transmitter. In other words, given the current system state, i.e., data queue, energy level, jammer, and channel states, the policy determines an optimal action to maximize the average long-term reward for the system. The optimization problem is then expressed as follows:

$$\max_{\pi} \quad \mathcal{R}(\pi) = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left(r_t(s_t, \pi(s_t))\right), \tag{4.10}$$

where $\mathcal{R}(\pi)$ is the average reward of the transmitter under the policy π and $r_t(s_t, \pi(s_t))$ is the immediate reward under policy π at time step t. Clearly, the state space $\mathcal{S}$

contains only one communicating class, i.e., from a given state the process can go to any other states after a finite number of steps. In other words, the MDP with states in $\mathcal{S}$ is irreducible. Thus, for every π, the average throughput $\mathcal{R}(\pi)$ is well defined and does not depend on the initial state [103]. To obtain the optimal defense policy for the transmitter, we use the deep Q-learning and deep dueling algorithms proposed in Section 2.3.

4.3 Performance Evaluation

4.3.1 Parameter Setting

In our system, the data queue of the transmitter can store up to 20 packets. The energy storage capacity is set to be 20 units. When the ambient RF source/channel is active, the transmitter can either harvest two units of energy or backscatter one packet to the gateway. When the channel is idle and the jammer does not attack the channel, if the transmitter performs active transmission, it can successfully transmit 4 packets. Each transmitted packet requires one unit of energy. The jammer has four transmit power levels, i.e., $\mathbf{P}_{\mathrm{J}} = \{0\mathrm{W}, 7\mathrm{W}, 15\mathrm{W}, 21\mathrm{W}\}$, with $P_{\max}$= 21W [106]. As explained in Section 4.1, as the jamming power increases, the transmitter can successfully harvest more energy or transmit more packets by backscattering jamming signals, and thus we set $\mathbf{e} = \{0, 2, 3, 4\}$ and $\widehat{\mathbf{d}} = \{0, 1, 2, 3\}$. In addition, when the jammer attacks the channel and the rate adaption technique is implemented, the transmitter can transmit d_m^{r} =$\{2, 1, 0\}$ packets when the jammer attacks under power levels P_n^{J}=$\{7\mathrm{W}, 15\mathrm{W}, 21\mathrm{W}\}$, respectively. In the case both the jammer and the ambient RF source are active, the total number of backscattered packets d_{sum} and the total amount of harvested energy e_{sum} follow the Poisson distribution with the means of $(d_{\min} + d_{\max})/2$ and $(e_{\min} + e_{\max})/2$, respectively. The latency threshold t_{th} is set at 3 time units. Unless otherwise stated, the idle channel probability is 0.5 and P_{avg} = 7W. Note that the reinforcement learning algorithms, i.e., Q-learning, deep Q-learning, and deep dueling, do not require the information about the jammer, e.g., jamming strategy, and the channel activity, i.e., the idle channel probability, in advance. Instead, these information is learned through the real-time learning process.

The architecture of the deep neural network significantly affects the performance of the deep reinforcement learning algorithms, and thus it requires a thoughtful design. In particular, the complexity of the algorithms increases when the number of hidden layers increases. Nevertheless, if the number of hidden layers is small, the algorithm requires a long time to converge to the optimal policy. Similarly, if the size of hidden layers, i.e., numbers of neurons, and the mini-batch size N are large, the algorithm will need more time to estimate the Q-function. For deep Q-learning algorithms, we adopt parameters based on the common settings for designing neural networks [26,92]. Specifically, for the deep Q-learning algorithm, two fully-connected hidden layers are implemented together with input and output layers. For the deep dueling algorithm, the neural network is divided into two streams. Each stream connects to a shared hidden layer as shown in Fig. 2.6. The size of the hidden layers is 16. The mini-batch size is set at 16. The maximum size of the experience replay buffer is 10,000, and the target Q-network is updated every 1,000 iterations [74,92]. All learning algorithms use the ϵ-greedy scheme with the initial value of ϵ set at 1 and its final value set at 0.1 [73].

To evaluate the proposed solutions, we compare their performance with two other schemes, i.e., HTT and WTJ. For the HTT scheme, the transmitter only implements the harvest-then-transmit protocol without considering ambient backscatter communication technology. This scheme is to evaluate the impact of ambient backscatter communications to the system performance. For the WTJ, the transmitter can implement both harvest-then-transmit protocol and ambient backscatter communication technology only for the ambient RF signals. This scheme evaluates

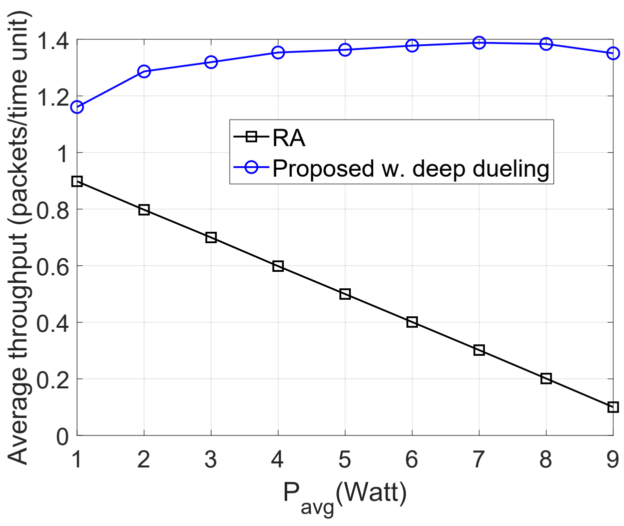


Figure 4.3 : Average throughputs of the proposed solution and the RA technique vs. P_{avg}.

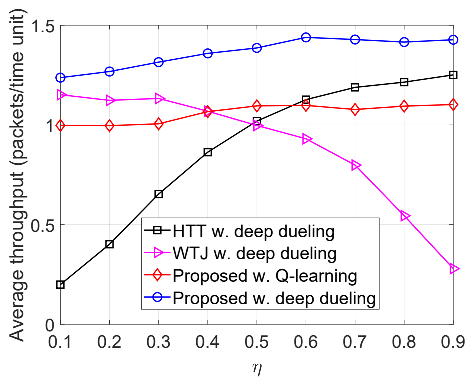
the system performance without leveraging the jamming signals. It is important to note that the optimal policies of both HTT and WTJ are also obtained by the deep dueling algorithm, i.e., Algorithm 2.4, presented in Section 2.3.

4.3.2 Simulation Results

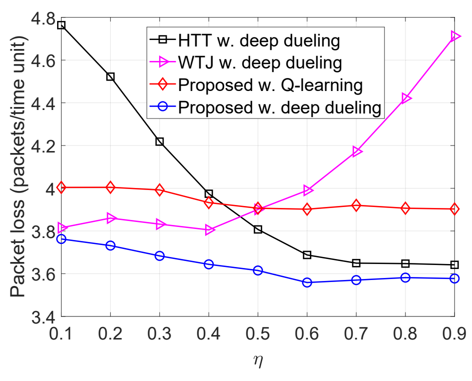
Compare with Non-machine Learning Rate Adaptation Technique First, we compare our proposed solution with a non-machine learning technique, i.e., RA. With the RA technique, the transmitter has a fixed defend policy as follows: (i) when the jammer attacks the channel, the transmitter adapts its rate based on the power level of the jammer and (ii) otherwise, the transmitter harvests energy from the ambient RF signals. As shown in Fig. 4.3, the average throughput achieved by our proposed solution is much higher than that of the RA technique. The reason is that with the RA technique, the transmitter transmits data at a low rate when the jammer attacks the channel with high power levels. Moreover, as stated in [36], with the RA technique, the jammer can force the transmitter to always operate at the lowest rate by merely randomizing its power levels, provided that the average jamming power is above a given threshold. When P_{avg} increases, i.e., the jammer has more opportunities to attack the channel with high power levels, the throughput achieved by the RA technique is significantly decreased. Our proposed solution, in contrast, can allow the transmitter to much more effectively adapt its defense strategy based on the environment condition and the attack strategy of the jammer. This is possible by implicitly learning the strategy of the jammer as well as unknown parameters that are often assumed to be available in the literature (e.g., the power constraint of the jammer and the jamming strategy). Additionally, with the ambient backscatter capability, the transmitter can always transmit its data to the gateway by leveraging the strong jamming signals.

Performance Evaluation Next, we perform simulations to evaluate and compare the performance of the proposed solution with those of the HTT and WTJ schemes in terms of average throughput, packet loss, delay, and PDR. For the HTT and WTJ schemes, we adopt the deep dueling algorithm (with 4×10^4 iterations) to obtain the optimal policy for the transmitter. For the proposed solutions, we recruit both the deep dueling (with 4×10^4 iterations) and Q-learning algorithms (with 10^6 iterations).

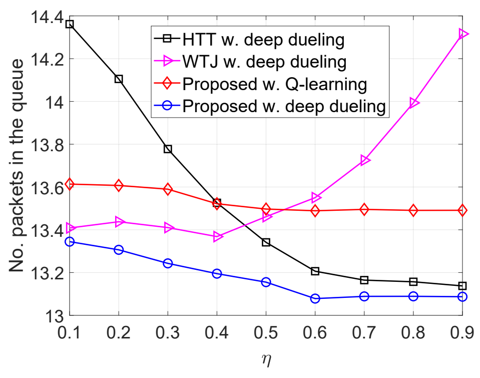
In Fig. 4.4, we vary the idle channel probability of the ambient RF source η and observe the performance of the system. Clearly, the throughput of the WTJ policy decreases when η increases as shown in Fig. 4.4(a). This is stemmed from the fact that when the ambient RF source is likely to be idle, the WTJ has less opportunities to harvest energy and backscatter data from the ambient signals. This also leads to the increase of packet loss and number of packets in the data queue as shown in Fig. 4.4(b) and Fig. 4.4(c), respectively. In contrast, as the idle channel probability increases, the average throughputs obtained by the HTT policy and the proposed solution increase, and their packet loss and number of packets in the data queue will be reduced. The reason is that the transmitter has more opportunities to



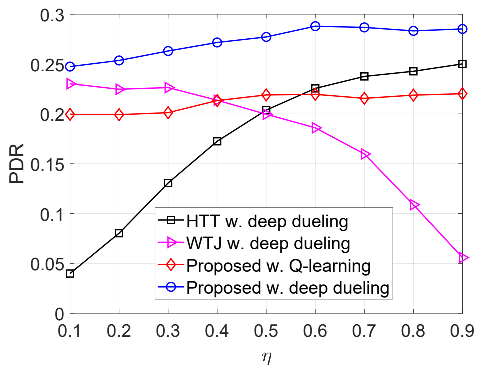
(a)



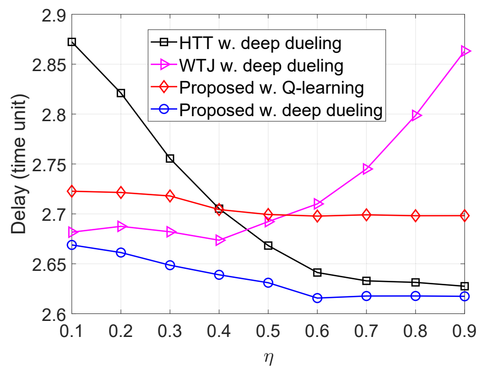
(b)



(c)



(d)



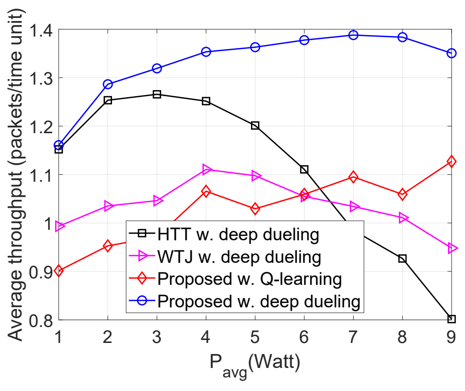
(e)

Figure 4.4 : (a) Average throughput (packets/time unit), (b) Packet loss (packets/time unit), (c) Average number of packets in the data queue, (d) PDR, (e) Delay (time/units) vs. η.

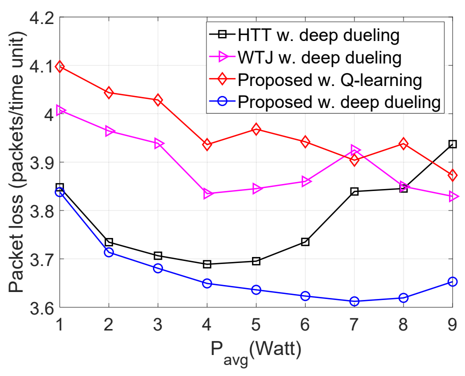
harvest energy from the jamming signals and use the harvested energy to actively transmit data when the channel is idle. Additionally, the proposed solution can also backscatter data through both the jamming and ambient signals, thereby its throughput is considerably higher than that of the HTT scheme. In Fig. 4.4(d), we observe the PDR of the system. Clearly, the proposed solution achieves the best PDR compared to the other schemes. It is worth noting that, the Q-learning algorithm cannot obtain the optimal policy in the first 10^6 iterations, thereby the performance derived by the Q-learning algorithm is much lower than that of the deep dueling algorithm.

In Fig. 4.5, we vary P_{avg} to evaluate the average throughput, packet loss, number of packets in the data queue, and PDR of the system. Obviously, when P_{avg} increases from 1W to 3W, the throughput of the HTT and WTJ policies increases. The reason is that the transmitter has more chances to harvest energy from the strong jamming signals and uses the harvested energy to transmit data when the jammer and the ambient RF source are idle. However, when P_{avg} is large (e.g., higher than 3W), i.e., the jammer is more likely to attack the channel, the throughput of these policies decreases as the transmitter has less chance to actively transmit data to the gateway. In contrast, the throughput achieved by the proposed solution increases. The reason is that the proposed solution allows the transmitter to switch to the backscatter mode when the jammer is likely to attack the channel. Consequently, the proposed solutions achieve the best performance in terms of packet loss, number of packets in the queue, PDR, and delay as shown in Fig. 4.5(b), Fig. 4.5(c), Fig. 4.5(d), and Fig. 4.5(e), respectively. Again, the performance of the Q-learning algorithm is not as good as the deep dueling algorithm due to the slow-convergence problem.

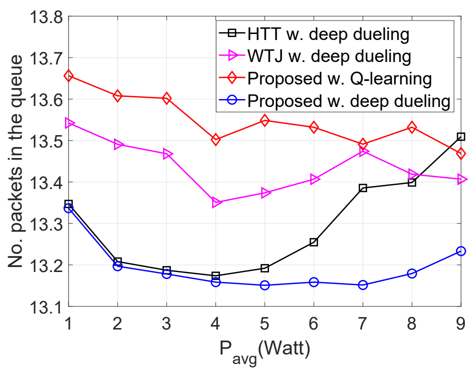
Next, we vary the maximum number of packets $\widehat{d_{\text{t}}}$ that the transmitter can actively transmit to the gateway and evaluate the performance of the proposed solution as shown in Fig. 4.6. As shown in Fig. 4.6(a), as $\widehat{d_{\text{t}}}$ increases, the throughput



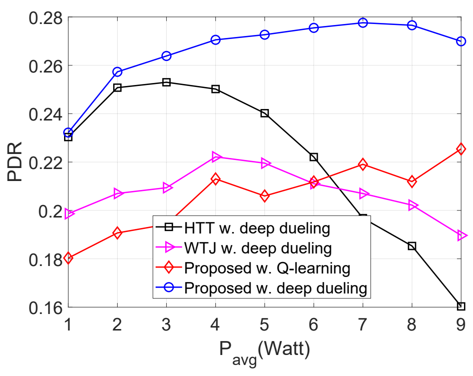
(a)



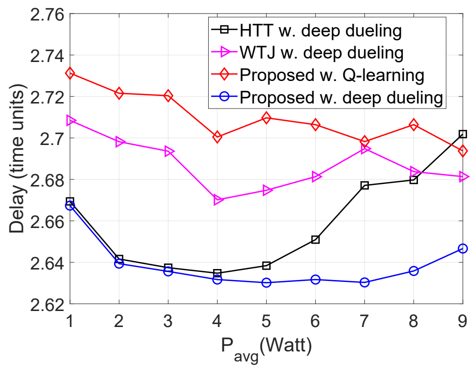
(b)



(c)



(d)



(e)

Figure 4.5 : (a) Average throughput (packets/time unit), (b) Packet loss (packets/time unit), (c) Average number of packets in the data queue, (d) PDR, (e) Delay (time/units) vs. P_{avg}.

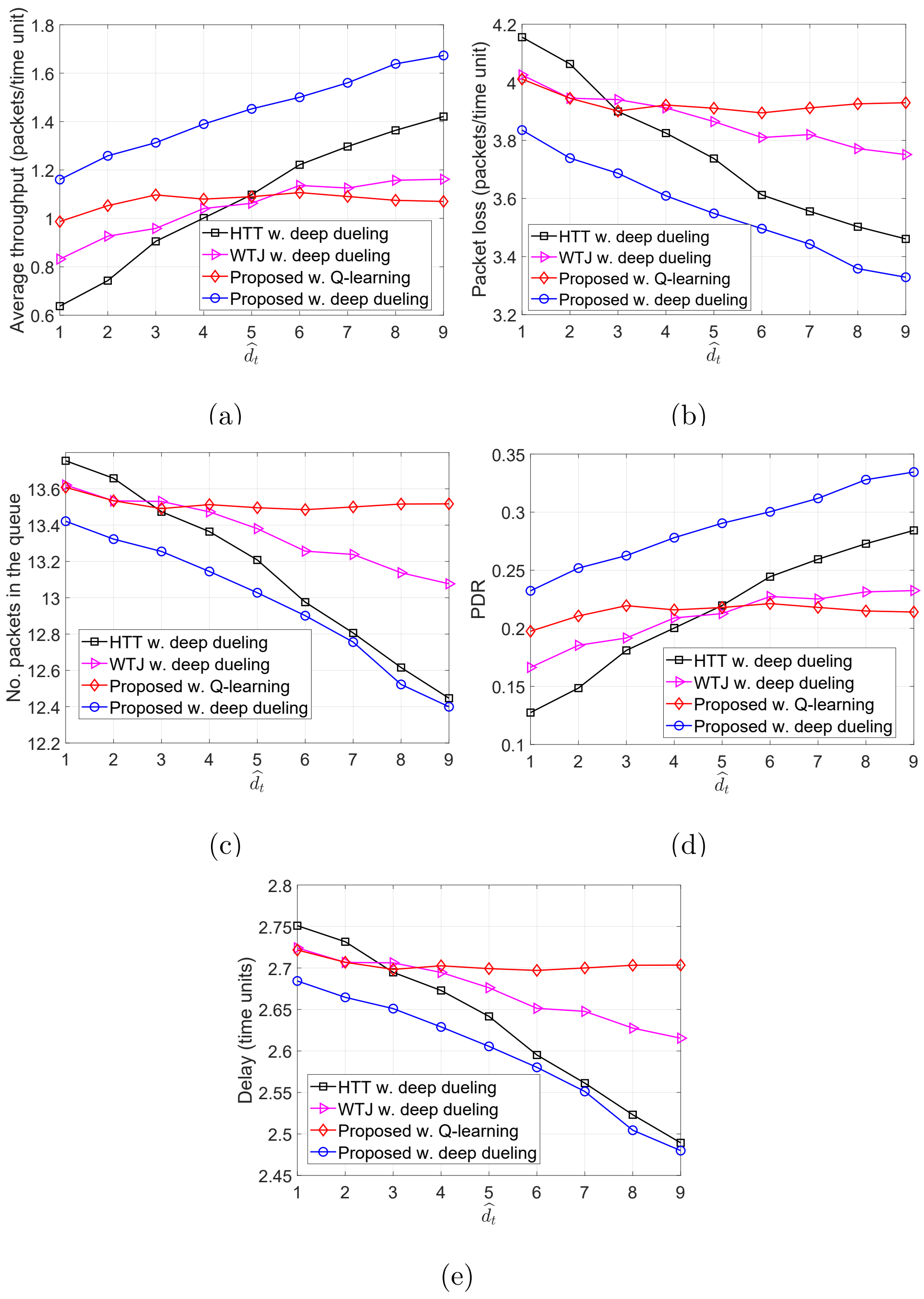
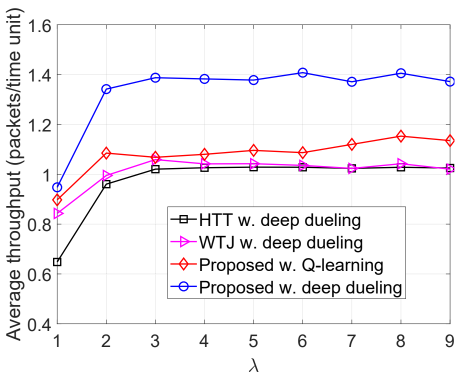


Figure 4.6 : (a) Average throughput (packets/time unit), (b) Packet loss (packets/time unit), (c) Average number of packets in the data queue, (d) PDR, (e) Delay (time/units) vs. $\widehat{d_t}$.

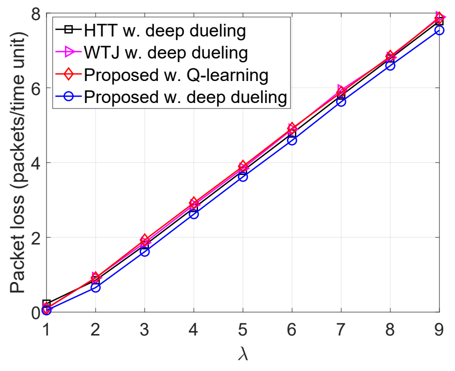
of WTJ scheme also increases and remains the same when $\widehat{d}_{\mathrm{t}} \geq 6$. This is due to the fact that the transmitter does not leverage the strong jamming signals (except when both the source are active), and thus the amount of harvested energy is limited. In contrary, the HTT policy allows the transmitter to harvest energy from both the ambient and jamming signals. As a result, its throughput increases and is higher than that of the WTJ policy. Importantly, by balancing the time for backscattering data and harvesting energy, the throughput achieved by the proposed solution is significantly higher than that of the HTT and WTJ schemes. This also leads to the reductions of the packet loss and number of packets waiting in the data queue as shown in Fig. 4.6(b) and Fig. 4.6(c), respectively. In Fig. 4.6, we observe the PDR of obtained by the three schemes. Clearly, the proposed solution continues to achieve the best PDR compared to other schemes.

In Fig. 4.7, we vary the packet arrival rate λ to evaluate the performance of the proposed solution. Clearly, when λ increases to 2 packets/time slot, the throughputs of all three schemes are increased as the transmitter can transmit more packets. However, when $\lambda > 2$ packets/time slot, the throughputs remain the same as the transmitter obtains the optimal policy. Note that with energy harvesting and backscattering capabilities, the proposed solution can achieve the highest throughput among three schemes. As the transmitter cannot transmit all the arrival packets, the packet loss and the number of packets waiting in the data queue increase when λ increases as shown in Fig. 4.7(b) and Fig. 4.7(c), respectively. With the total number of arrival packets increases, the PDRs of all the three schemes are reduced as shown in Fig. 4.7(d). It is worth noting that in all the cases the performance of the Q-learning algorithm is not as high as the deep dueling algorithm as it can not converge to the optimal policy within 10^6 iterations.

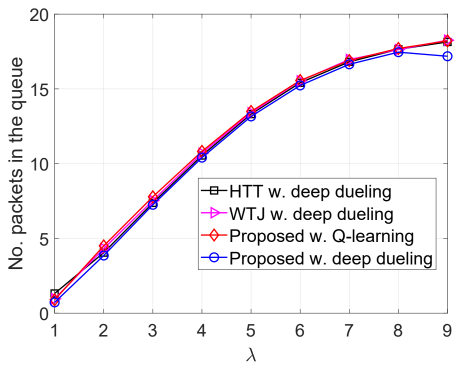
Finally, we vary the latency threshold and investigate the performance of the proposed solution as shown in Fig. 4.8. Obviously, when the latency threshold



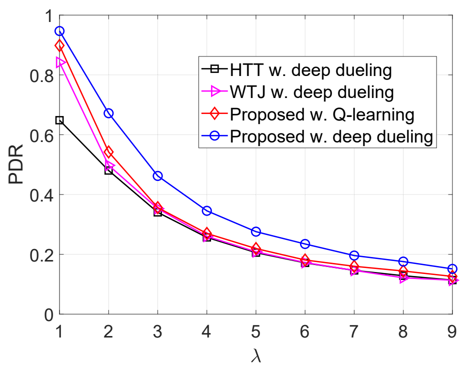
(a)



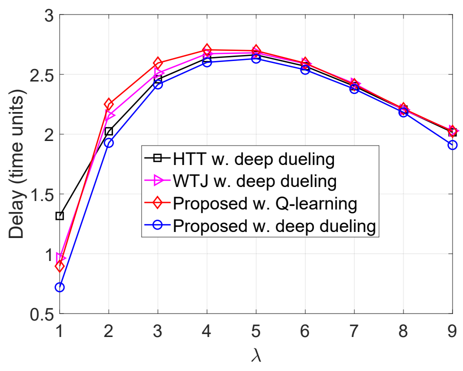
(b)



(c)

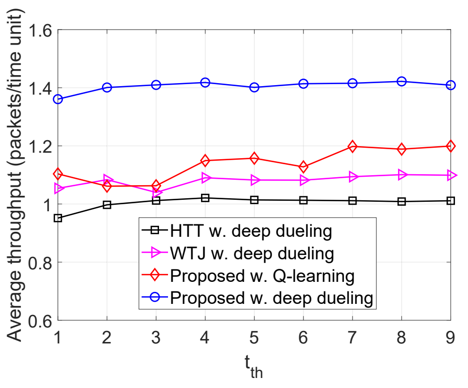


(d)

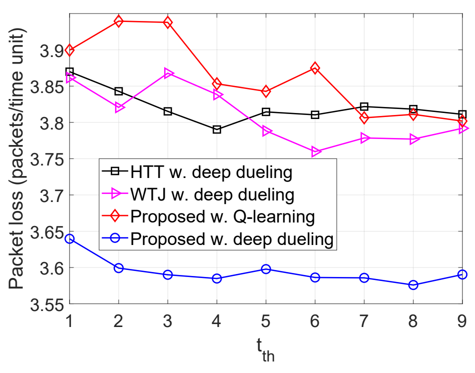


(e)

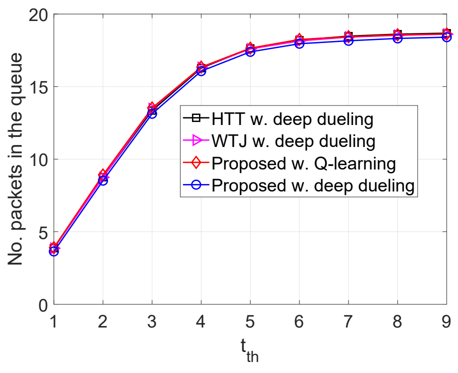
Figure 4.7 : (a) Average throughput (packets/time unit), (b) Packet loss (packets/time unit), (c) Average number of packets in the data queue, (d) PDR, (e) Delay (time/units) vs. λ.



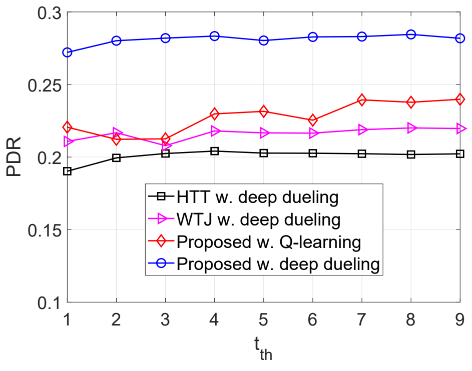
(a)



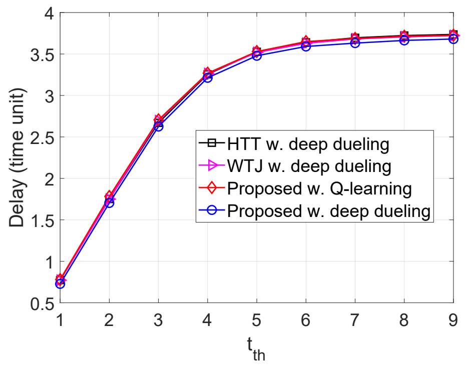
(b)



(c)

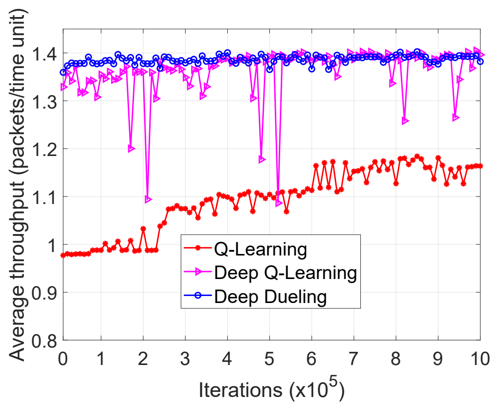


(d)

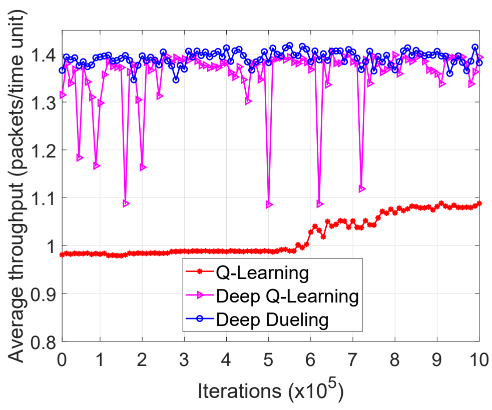


(e)

Figure 4.8 : (a) Average throughput (packets/time unit), (b) Packet loss (packets/time unit), (c) Average number of packets in the data queue, (d) PDR, (e) Delay (time/units) vs. t_{th}.



(a) $D = E = 10$



(b) $D = E = 20$

Figure 4.9 : Convergence rates when (a) $D = E = 10$ and (b) $D = E = 20$.

increases from 1 to 4 time units, the throughputs and the PDRs obtained by all the schemes increase as shown in Fig. 4.8(a) and Fig. 4.8(d), respectively, and the packet losses decreases as shown in Fig. 4.8(b). This is stemmed from the fact that with a very short period of latency, more packets will be discarded from the data queue resulting in lower throughputs. When the latency threshold is large, the throughputs remain the same as the deep dueling algorithm obtains the optimal solution to effectively utilize the ambient signals as well as the jamming signals. As the latency threshold increases, the arrival packets have more time to stay in the data queue. As such, the number of packets in the data queue increases as shown in Fig. 4.8(c). Note that the deep dueling algorithm always achieves the best performance in all the cases. In contrast, the Q-learning algorithm cannot achieve the optimal policy for the transmitter due to the slow-convergence problem.

Convergence of Deep Reinforcement Learning Approaches We first show the learning process and the convergence of the proposed deep reinforcement learning algorithms, i.e., deep Q-learning and deep dueling, in several scenarios. As shown in Fig. 4.9(a) and Fig. 4.9(b), when the maximum sizes of data and energy queues are set at 10 and 20, respectively, after 10^6 iteration, the average throughput obtained by the Q-learning algorithm is much lower than those of the deep reinforcement learning algorithms, especially in the first 10^5 iterations. This implies that as the system state space increases, the Q-learning algorithm requires more time to be converged, and thus given a fixed short time period, the system performance obtained by the Q-learning algorithm cannot achieve the results as great as those of deep reinforcement algorithms (i.e., deep Q-learning and deep dueling algorithms). Note that in Fig. 4.9, the performance obtained by deep Q-learning algorithm is as close as that of the deep dueling algorithm, however the average throughput obtained by the deep Q-learning algorithm is very fluctuated compared with that of the deep dueling algorithm. This implies that the deep Q-learning algorithm requires more time to be converged compared with that of the deep dueling algorithm.

4.4 Conclusion

In this chapter, we have developed an optimal anti-jamming framework which allows the wireless transceivers to effectively defeat jamming attacks. In particular, with the ambient backscatter capability, while being attacked, the device can either adapt its transmission rate or backscatter its data to the gateway through the jamming signals or harvest energy from the jamming signals to support its operations. To effectively learn about the jamming attacks as well as the channel activities, we have proposed an optimal anti-jamming strategy based on MDP to obtain the optimal defend policy for the transmitter. Then, the reinforcement learning algorithms, i.e., Q-learning, deep Q-learning, and deep dueling, have been developed to maximize the long-term average throughput and minimize the packet loss. Extensive simulations have demonstrated that by using two streams of fully-connected hidden layers, the proposed framework using the deep dueling algorithm can improve the average throughput up to 426% and reduce the packet loss by 24%. Importantly, with ambient backscatter and energy harvesting technology, jamming signals can be leveraged by the transmitter as the ambient RF signals, thereby effectively eliminating jamming attacks. To the best of our knowledge, this is the first anti-jamming solution that allows wireless transceivers to not only survive jamming attacks without requiring additional resources but also leverage the jamming signals to improve their transmission rate. The proposed ambient backscattering augmented communications framework can be applicable to both civil (e.g., ultra-reliable and low-latency communications or URLLC) and military scenarios (to combat both inadvertent and deliberate jamming).

Chapter 5

Joint Coding and Scheduling Optimization for Distributed Learning over Wireless Edge Networks

Unlike theoretical analysis of distributed learning (DL) in the literature, DL over wireless edge networks faces the inherent dynamics/uncertainty of wireless connections and edge nodes, making DL less efficient or even inapplicable under the highly dynamic wireless edge networks (e.g., using mmW interfaces). This thesis addresses these problems by leveraging recent advances in coded computing and the deep dueling neural network architecture. By introducing coded structures/redundancy, a distributed learning task can be completed without waiting for straggling nodes. Unlike conventional coded computing that only optimizes the code structure, coded distributed learning over the wireless edge also requires to optimize the selection/scheduling of wireless edge nodes with heterogeneous connections, computing capability, and straggling effects. However, even neglecting the aforementioned dynamics/uncertainty, the resulting joint optimization of coding and scheduling to minimize the distributed learning time turns out to be NP-hard. To tackle this and to account for the dynamics and uncertainty of wireless connections and edge nodes, we reformulate the problem as an MDP and then design a novel deep reinforcement learning algorithm that employs the deep dueling neural network architecture to find the jointly optimal coding scheme and the best set of edge nodes for different learning tasks without explicit information about the wireless environment and edge nodes' straggling parameters. Simulations show that the proposed framework reduces the average learning delay in wireless edge computing up to 66%

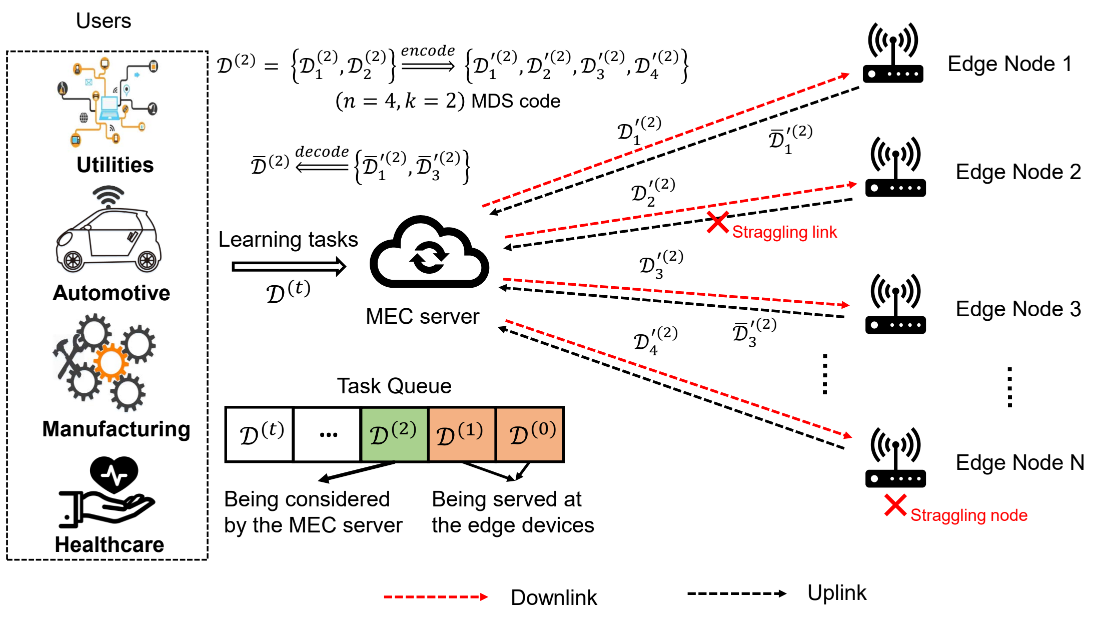

Users
Utilities
Automotive
Manufacturing
Healthcare
$\mathcal{D}^{(2)} = \{\mathcal{D}_1^{(2)}, \mathcal{D}_2^{(2)}\} \xrightarrow{encode} \{\mathcal{D}_1'^{(2)}, \mathcal{D}_2'^{(2)}, \mathcal{D}_3'^{(2)}, \mathcal{D}_4'^{(2)}\}$
$(n = 4, k = 2)$ MDS code
$\bar{\mathcal{D}}^{(2)} \xleftarrow{decode} \{\bar{\mathcal{D}}_1'^{(2)}, \bar{\mathcal{D}}_3'^{(2)}\}$
Learning tasks
$\mathcal{D}^{(t)}$
MEC server
Task Queue
$\mathcal{D}^{(t)}$ ... $\mathcal{D}^{(2)}$ $\mathcal{D}^{(1)}$ $\mathcal{D}^{(0)}$
Being considered by the MEC server
Being served at the edge devices
$\mathcal{D}_1'^{(2)}$
$\bar{\mathcal{D}}_1'^{(2)}$
$\mathcal{D}_2'^{(2)}$
Straggling link
$\mathcal{D}_3'^{(2)}$
$\mathcal{D}_4'^{(2)}$
$\bar{\mathcal{D}}_3'^{(2)}$
Edge Node 1
Edge Node 2
Edge Node 3
Edge Node N
Straggling node
Downlink
Uplink

Figure 5.1 : System model for coded distributed learning over wireless edge network. Here, we illustrate the case when learning task $\mathcal{D}^{(2)}$ is processed with $(n = 4, k = 2)$ MDS code. The sub-learning tasks are sent to edge nodes $1, 2, 3,$ and N to process. Then, when edge node 2 is disconnected, and edge node N is straggling, the learning task $\mathcal{D}^{(2)}$ still can be completed by using computed results from edge nodes 1 and 3.

compared with other DL approaches. The jointly optimal framework in this work is also applicable to any distributed learning scheme with heterogeneous and uncertain computing nodes.

The rest of this chapter is organized as follows. Section 5.1 presents the system model and the computing and communication models. The MDP framework and the problem formulation are provided in Section 5.2. Simulation results are discussed in Section 5.3. Finally, conclusions are highlighted in Section 5.4.

5.1 System Model

We consider a distributed edge learning system that consists of a mobile edge computing (MEC) server and a set of N edge nodes denoted by $\mathbf{E} = \{E_1, \dots, E_j, \dots, E_N\}$. The edge nodes communicate with the MEC server through wireless links as illustrated in Fig. 4.1. Let C_j denote the wireless link that connects the MEC server and edge node E_j. Practically, different links may be allocated on different channels, and thus their properties, e.g., fading conditions, interference, and disconnection probability, may be different. In this work, we deploy a task queue at the MEC server to maximize the utilization of the system. In particular, a new learning task can be stored in the queue when the system is busy. This learning task will be served as soon as there are enough resources available at the edge nodes. Note that, our learning task queue has a limited capacity of M. If the queue is full, the task waiting the longest in the queue will be dropped, and thus the user (owning this task) can resend the learning task or a new task to the system to process. In this way, we can not only take the advantage of employing the task queue but also can enhance information freshness for the learning tasks. We assume that time is slotted. In each time slot, a learning task arrives at the system with probability μ. Note that, our proposed solution still can work well with other packet arrival processes as they will be learned by the proposed algorithm and are not required to be available in advance. Different learning tasks, e.g., matrix multiplication, data shuffling, or gradient descent for linear regression problems [52], may have different data sizes. We denote $f(\mathcal{D}^{(t)})$ as the data size of learning task $\mathcal{D}^{(t)}$.

In our system, learning tasks in the queue are served in a first-come-first-served manner. At each time slot, if the computing resources at the edge nodes are available, the MEC server will look at the queue and consider to serve a learning task which comes the earliest in the queue but not yet served by any edge nodes (e.g., $\mathcal{D}^{(2)}$

as illustrated in Fig. 4.1). By using the optimal (n, k) MDS code and the optimal set of edge nodes obtained by our proposed algorithm, this learning task is then encoded into n sub-learning tasks, and these sub-learning tasks are offloaded to edge nodes in the optimal set to execute. These devices then serve the assigned sub-learning tasks and return the results to the MEC server. Note that the edge nodes have dissimilar configurations and communication links that greatly affect the performance of distributed learning over wireless edge networks. Choosing the best set of edge nodes for each learning task at different times can have critical impact on the system's performance (e.g., the number of tasks waiting in the queue, the average task dropping probability, and the average delay of learning tasks in the system). For example, with the same (n, k) MDS code, different sets of n edge nodes may require different computation times for a given task. In particular, selecting an edge node with high processing power and an unstable wireless connection may be worse than selecting an edge node with average processing power and a stable wireless connection. As shown in our simulations, by obtaining the optimal scheduling policy, our proposed solution achieves better performance compared to other approaches.

In this work, the learning task still remains in the queue until the MEC server receives k results returned from the edge nodes and successfully decodes them. For example, as illustrated in Fig. 4.1, with $(n = 4, k = 2)$ MDS code, the MEC server does not need to wait for results from edge node 2 and edge node N, which are delayed by the straggling problems. Instead, the MEC server can decode the final result by using computed results returned from edge node 1 and edge node 3 which have better wireless connections and computing power. In contrast, conventional distributed learning models need to wait for computed results from all the assigned edge nodes to obtain the final result, and thus dramatically increasing the computing delay of the whole system. For the ease of notation, we assume that when an edge node receives a sub-learning task, it will use its all computing resource to execute this task. This is also stemmed from the fact that edge nodes (e.g., IoT gateways) are usually equipped with limited resources, and thus they may not be able to serve multiple learning tasks simultaneously. In addition, if an edge node has to process multiple sub-learning tasks at the same time, the straggling problem may be more serious as its computing resources have to share to execute multiple tasks simultaneously. We denote e_j as the state of edge node E_j. Specifically, $e_j = 0$ if the edge node is currently busy, i.e., serving one sub-learning task. $e_j = 1$ if the edge node is available, i.e., there is no learning task executing at the edge node. Then, the set of available edge nodes can be denoted as $\mathbf{E}_{\text{av}} \overset{\text{def}}{=} \{E_j : \forall E_j \in \mathbf{E}$ and $e_j = 1\}$. It is worth noting that our proposed solution can be extended to the case if one edge node can handle multiple tasks at the same time by implementing multiple virtual machines (VMs). Then, each VM can be reserved to execute one learning task. Thus, the MEC just needs to take the available VMs of each edge node into account when it assigns learning tasks to them.

5.1.1 Coded Computing for Distributed Learning over Wireless Edge Networks

The key idea of coded computing techniques is to leverage coding theoretic mechanisms to add structured computing redundancy into learning tasks to mitigate the effects of straggling edge nodes and wireless communication links [54]. One of the most effective coding techniques used in coded computing is the maximum distance separable (MDS) code [52]. The fundamentals of the MDS code are illustrated in Fig. 4.1. In particular, with the (n, k) MDS code $(1 \leq k \leq n)$, a learning task $\mathcal{D}^{(t)}$ can be first divided into k equal-sized sub-learning tasks $\{\mathcal{D}_1^{(t)}, \mathcal{D}_2^{(t)}, \ldots, \mathcal{D}_k^{(t)}\}$. Then, these sub-learning tasks are encoded by the (n, k) MDS code. After encoding, we get n encoded sub-learning tasks $\{\mathcal{D}_1^{'(t)}, \mathcal{D}_2^{'(t)}, \ldots, \mathcal{D}_n^{'(t)}\}$. These sub-learning tasks are then sent to n edge nodes to execute. Upon receiving any k results from any k

edge nodes, the MEC server can decode them to obtain the result. When a learning task is completed, it will be removed from the task queue. After that, the MEC server will inform edge nodes that are still working on the remaining sub-learning tasks to stop performing these sub-learning tasks and make them free.

It is worth noting that choosing the values of n and k to maximize the system performance in terms of serving time, delay, and task drop probability is very challenging under the dynamics and uncertainty of the wireless environment as well as straggling problems at the edge computing devices. Currently, an optimal MDS code setting (with optimal values of n and k) can be determined based on static optimization methods, e.g., [54, 56, 59, 107]. However, these methods require prior information about the straggling parameters at edge nodes and wireless links. In practice, these parameters usually are not available in advance. Thus, they are not applicable to wireless edge computing as they do not account for the inherent dynamics of wireless channels and edge nodes, leading to uncertainty of straggling problems. Moreover, it is even more challenging when choosing the best edge nodes to execute different learning tasks. To the best of our knowledge, all current existing works cannot address all these problems. Thus, in this work, we propose an intelligent approach which allows the MEC server to dynamically select the optimal MDS code together with the best edge nodes based on the current status of the whole system. Note that this approach can not only select the optimal values of n and k, but also find the best edge nodes to serve each learning task.

5.1.2 Communication and Computation Models

Recall that with (n, k) MDS code, learning task $\mathcal{D}^{(t)}$ is first divided into k equal-sized sub-learning tasks $\{\mathcal{D}_1^{(t)}, \mathcal{D}_2^{(t)}, \dots, \mathcal{D}_k^{(t)}\}$. These sub-learning tasks are then encoded into n encoded sub-learning tasks $\{\mathcal{D}_1^{'(t)}, \mathcal{D}_2^{'(t)}, \dots, \mathcal{D}_n^{'(t)}\}$. The encoded sub-learning tasks are finally sent to n edge nodes for processing. In this section, we formulate the serving time for encoded sub-learning task $\mathcal{D}_i^{'(t)}$ $(1 \leq i \leq n)$ at a given edge node $E_j \in \mathbf{E}$. For the ease of notation, we denote $T_{\text{serve}}^{(t,i)}$ as the total serving time of sub-learning task $\mathcal{D}_i^{'(t)}$. Thus, $T_{\text{serve}}^{(t,i)}$ can be written as:

$$T_{\text{serve}}^{(t,i)} = T_{\text{se}}^{(t,i)} + T_{\text{cmp}}^{(t,i)} + T_{\text{es}}^{(t,i)}, \tag{5.1}$$

where $T_{\text{se}}^{(t,i)}$ and $T_{\text{es}}^{(t,i)}$ are the communication time for sending $\mathcal{D}_i^{'(t)}$ from the MEC server to edge node E_j and the time for sending back its computed result from edge node E_j to the MEC server through wireless link C_j, respectively. Note that both the uplink (from edge node E_j to the MEC server) and the downlink (from MEC server to edge node E_j) can share the same channel as the MEC server and edge node E_j do not need to transmit data at the same time. $T_{\text{cmp}}^{(t,i)}$ is the time that edge node E_j requires to complete the sub-learning task. With the high-speed backhaul connections from the edge node to the server (e.g., via mmWave), a sub-learning task or its result can be transmitted over the wireless link within one time slot. To capture the dynamics of the wireless link C_j from the edge node E_j to the server, e.g., due to fading or moving obstacles and/or interfere with surrounding RF signals, let's define p_j as the disconnection probability of the wireless link from the MEC server to edge node E_j over a given time slot. We then denote $\mathbf{p} = \{p_1, \ldots, p_j, \ldots, p_N\}$ as the set of disconnection probabilities corresponding to wireless channels $\{C_1, \ldots, C_j, \ldots, C_N\}$. Using a disconnection probability to capture the quality of a wireless link is widely used in many wireless systems in the literature to evaluate their performance [109,110]. In particular, if the wireless channel is likely to be unstable (due to fading, noise, or interference from surrounding devices), the disconnection probability of this link will be high. Moreover, it is worth noting that, the disconnection probability is not the input of our proposed algorithm. Instead, our algorithm can learn the disconnection probabilities of wireless links through interacting with the environment. As such, our proposed solution does not require this information in advance.

In the case wireless link C_j is disconnected, the MEC server or edge node E_j needs to resend its data in the next time slot. As such, the communication delay increases, especially when the disconnection probability is high. We thus can formulate $T_{\text{se}}^{(t,i)}$ and $T_{\text{es}}^{(t,i)}$ as follows:

$$T_{\text{se}}^{(t,i)} = T_{\text{es}}^{(t,i)} = H_j \xi, \tag{5.2}$$

where ξ is the duration of a time slot and H_j is the number of time slots needed to successfully transmit data on wireless channel C_j. H_j is i.i.d based on the *Geometric* distribution with the successful probability $p_{\text{success}} = 1 - p_j$ as follows [54]:

$$Pr(H_j = x) = p_j^{x-1}(1 - p_j), x = 1, 2, 3, \dots \tag{5.3}$$

According to the *Geometric* distribution, a higher value of p_j, i.e., disconnection probability, results in a higher value of H_j. Thus, in many scenarios, the delay caused by unstable connections is even more serious than that of straggling devices, especially when the link disconnection probability is very high [111]. To deal with this issue, in the sequel, we propose an effective learning solution that can learn the disconnection probabilities to avoid bad connections when serving learning tasks (e.g., assigning tasks to edge nodes with more favorable connections). Hence, the straggling effects caused by unstable links can be significantly mitigated.

The computing time $T_{\text{cmp}}^{(t,i)}$ of sub-learning task $\mathcal{D}_i^{'(t)}$ $(1 \leq i \leq n)$ at edge node E_j is the sum of the stochastic time for random memory access during read/write cycles and the deterministic time for processing data [52, 54, 112]. Thus, $T_{\text{cmp}}^{(t,i)}$ can be written as follows:

$$T_{\text{cmp}}^{(t,i)} = \underbrace{f(\lambda_j)}_{\text{stochastic time}} + \underbrace{\eta_j |\mathcal{D}_i^{'(t)}|}_{\text{deterministic time}}, \tag{5.4}$$

where $f(\lambda_j)$ is a random variable denoting the stochastic component of the computing time caused by the straggling problem at edge node E_j. $f(\lambda_j)$ follows an exponential distribution with rate λ_j [54, 112], i.e., $p_{f(\lambda_j)}(x) = \lambda_j e^{-\lambda_j x}, x \geq 0$. η_j

is the deterministic time for edge node E_j to process one data point (e.g., one row in the matrix). $|\mathcal{D}_i^{'(t)}|$ is the data size of sub-learning task $\mathcal{D}_i^{'(t)}$. We denote $\boldsymbol{\lambda} = \{\lambda_1, \dots, \lambda_j, \dots, \lambda_N\}$ as the set of rate parameters determining the stochastic time at edge nodes. Specifically, $\frac{1}{\lambda_j}$ is the average stochastic time that can be considered as the straggling parameter of edge node E_j. The straggling parameter depends on many factors such as shared resources, maintenance activities, power limitation, and random memory access [57, 68]. With high straggling parameters, edge nodes need more time for processing learning tasks. Therefore, avoiding straggling edge nodes is crucial in distributed learning as they can significantly slow down the learning process. In practice, the straggling problems at edge nodes may occur randomly and cannot be effectively predicted. To tackle this problem, our framework below aims to learn the straggling parameters of edge nodes to find the best edge nodes for each learning task, and thus remarkably mitigating the impacts of straggling devices.

5.1.3 Learning-Task Delay Minimization Problem

From (5.1), (5.2), and (5.4), the total serving time of a sub-learning task $\mathcal{D}_i^{'(t)}$ can be expressed as:

$$\begin{aligned} T_{\text{serve}}^{(t,i)} = \Big(2H_j\xi + f(\lambda_j) + \eta_j|\mathcal{D}_i^{'(t)}|\Big)c_{i,j}, \\ \forall i \in \{1, \dots, n\}, \forall j \in \{1, 2, \dots, N\}, \end{aligned} \tag{5.5}$$

where $c_{i,j}$ is a scheduling binary decision. $c_{i,j} = 1$ if sub-learning task $\mathcal{D}_i^{'(t)}$ is served by edge node E_j, and $c_{i,j} = 0$, otherwise. Note that each sub-learning task is only severed by one edge node, and thus $\sum_{j=1}^{N} c_{i,j} = 1, \forall i \in \{1, \dots, n\}$. With the (n, k) MDS code, the MEC server only needs k results from any k (out of n) edge nodes to successfully decode the result for learning task $\mathcal{D}^{(t)}$. Thus, the total serving time for a learning task $\mathcal{D}^{(t)}$ is the serving time of the k-th completed sub-learning task, which can be expressed as follows:

$$T^{(t)}_{\text{serve}} = \min_{k-th}\left(\left\{T^{(t,i)}_{\text{serve}} : \forall i \in \{1,2,\dots,n\}\right\}\right), \tag{5.6}$$

where $\min\limits_{k-th}(.)$ returns the k-th minimum value of a set. For example, $\min\limits_{3-th}\left(\{1,5,10,4,6\}\right) = 5$. We then can formulate the serving time minimization problem as follows:

$$\begin{aligned}
\min_{n,k,\{c_{i,j}\}} \quad & T^{(t)}_{\text{serve}}, \\
\text{s.t.} \quad & 1 \leq k \leq n, \forall n \in \{1,2,\dots,|\mathbf{E}_{\text{av}}|\}, \\
& c_{i,j} \in \{0,1\}, \forall i \in \{1,\dots,n\} \text{ and } \forall j \in \{1,\dots,N\}, \\
& \sum_{j=1}^{N} c_{i,j} = 1, \forall i \in \{1,\dots,n\} \text{ and } \forall j \in \{1,\dots,N\}, \\
& c_{i,j} = 0, \text{ if } e_j = 0, \forall j \in \{1,\dots,N\}.
\end{aligned} \tag{5.7}$$

In Theorem 5.1, we show that the delay minimization problem in (5.7) is an NP-hard problem.

Theorem 5.1. *The joint coding and scheduling optimization problem (5.7) is NP-hard.*

Proof. It can be observed that the optimization problem in (5.7) is a form of the Knapsack problem [113]. In particular, (5.7) aims to find the optimal MDS code (i.e., optimal values of n and k) and the optimal scheduling policy (i.e., the best set of $\{c_{i,j}\}$) to minimize the serving time for each learning task. It is worth noting that the problem in (5.7) is much more complicated than the Knapsack problem as the serving time of each edge node is a stochastic value and changes over time as shown in (5.5). As a result, solving (5.7) is more difficult than solving the Knapsack problem. As shown in [113], the Knapsack problem is an NP-hard problem. As such, the optimization problem in (5.7) is also an NP-hard problem. □

It is worth noting that if the environment parameters can be correctly estimated, we can design a more appropriate baseline method to compare with our proposed approach. However, as shown in Theorem 5.1, even if these parameters are known and fixed, the delay minimization problem is NP-hard, hence usually intractable to be solved. Moreover, it is worth mentioning that under our system model, these parameters are not fixed due to the dynamics and uncertainty of wireless links as well as communications/computing resources at edge nodes. In practice, it is not trivial to estimate these parameters [114–116]. Even if we could do so, these parameters can change over time due to the straggling problems at the edge nodes (caused by random hardware error, maintenance activities, and power outage) and wireless links (caused by fading, noise, and interference from nearby devices). In such cases, the server needs to reestimate and obtain the optimal codes together with best nodes. This process, as aforementioned, can be costly in time, communications overhead, and computing resources. Moreover, this work aims to minimizing the average delay for all learning tasks, which is much more challenging than that for a single learning task as in (5.7). This is stemmed from the fact that learning tasks are sharing the same computing resources from edge nodes, and thus the optimal coding and scheduling for a learning task will have significant impacts on all next arrival learning tasks. Consequently, all current static optimization techniques (even for suboptimal solutions) in existing works in the literature [52–54] may not be effective in dealing with these practical issues. To tackle this and to account for the dynamics and uncertainty of wireless connections and edge nodes, in the following, we reformulate the problem as a Markov decision process and then design a novel deep reinforcement learning algorithm that employs the deep dueling neural network architecture to find the jointly optimal coding and scheduling policy for different learning tasks without explicit information about the wireless environment and edge nodes' straggling parameters.

5.2 Coded Computing for Distributed Learning Formulation

We first adopt the Markov decision process (MDP) framework to formulate the system delay minimization problem for distributed learning over wireless edge networks. Generally, the MDP is defined by a tuple $< \mathcal{S}, \mathcal{A}, r >$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, and r is the immediate reward function of the system. With the MDP framework, the MEC server can dynamically make optimal actions, i.e., select optimal MDS codes as well as the best edge nodes for serving sub-learning tasks, based on its current states, e.g., task queue and available edge nodes' resources, to maximize its long-term average reward. Thus, this framework can significantly reduce the average delay of learning tasks.

5.2.1 State Space

As stated above, a learning task stored in the learning task queue will be served in a first-come-first-served manner. In particular, the MEC first selects n available edge nodes and chooses (n, k) MDS code to encode the learning task. After that, the encoded sub-learning tasks are transmitted to a set of optimal edge nodes. The learning task will not be removed from the queue until the server receives any k results from the edge nodes. As such, the queue size, the available edge nodes, and the size of the considered learning task are important factors that should be captured by the system state s. We then define the state space $\mathcal{S}$ of the system as follows:

$$\begin{aligned}\mathcal{S} \triangleq \Big\{(m, f, \{e_1, \dots, e_j, \dots, e_N\}) : m \in \{0, \dots, M\};\\ f \geq 0; e_j \in \{0, 1\}, \forall j \in \{1, \dots, N\}\Big\},\end{aligned} \tag{5.8}$$

where m represents the number of learning tasks currently waiting in the queue, f is the size of the current considered learning task. Note that the current task size equals 0 only when: (i) the task queue is empty or (ii) all current tasks in the queue are being served and there is no new task arriving. The system state is then defined as a composite variable $s = (m, f, \{e_1, \dots, e_j, \dots, e_N\}) \in \mathcal{S}$. Note that the environment parameters such as straggling parameters of edge nodes and wireless links as well as the channel quality cannot obtain in advance as discussed in the previous sections. Thus, our system state space does not take these parameters into account. Instead, these parameters are implicitly captured in the immediate function defined below and then learned by our proposed learning algorithm to simultaneously obtain the optimal coding and scheduling policy.

5.2.2 Action Space

As mentioned, most of existing works only focus on optimizing the optimal code, i.e., the optimal values of n and k [52]. Nevertheless, the straggling problems at wireless links and edge nodes are randomly and unpredictable. Consequently, optimizing only the values of n and k often leads to sub-optimal solutions in terms of the average delay. To tackle this issue, this work aims to find not only the optimal code but also the best set of edge nodes for each learning task. As such, we define an action a as the combination of of n, k, and the set of edge nodes to serve the current learning task. Denote $\mathbf{E}_{\text{av}}$ as the set of all available edge nodes ($\mathbf{E}_{\text{av}} \subseteq \mathbf{E}$), we have the action space of the system as follows:

$$\begin{aligned} \mathcal{A} \triangleq \{a\} = \{(0,0,\emptyset),(n,k,\mathbf{E}_{\text{b}})\}, \forall n \in \{1, \dots, |\mathbf{E}_{\text{av}}|\}, \\ \forall k \in \{1, \dots, n\}, \forall \mathbf{E}_{\text{b}} \in \binom{\mathbf{E}_{\text{av}}}{n}, \end{aligned} \tag{5.9}$$

where $\mathbf{E}_{\text{b}}$ is the set of n-best edge nodes to serve the current learning task. $|\mathbf{E}_{\text{av}}|$ is the total number of available edge nodes. $\binom{\mathbf{E}_{\text{av}}}{n}$ is the combination operation that returns all size-n subsets of $\mathbf{E}_{\text{av}}$. For example, if $\mathbf{E}_{\text{av}} = \{E_1, E_2, E_3\}$ and $n = 2$, we have $\binom{\mathbf{E}_{\text{av}}}{2} = \Big\{\{E_1, E_2\}, \{E_1, E_3\}, \{E_2, E_3\}\Big\}$. From this set, the MEC server can select any size-2 subset of edge nodes to serve the learning task, i.e., $\mathbf{E}_{\text{b}} = \{E_1, E_2\}$, $\mathbf{E}_{\text{b}} = \{E_1, E_3\}$ or $\mathbf{E}_{\text{b}} = \{E_2, E_3\}$. In general, $a = (n, k, \mathbf{E}_{\text{b}})$ if the MEC server chooses (n, k) MDS code to encode the learning task and the edge nodes in set $\mathbf{E}_\mathrm{b}$ to execute the encoded sub-learning tasks. $a = (0, 0, \emptyset)$ if the MEC server chooses to stay idle, i.e., not select any code nor edge nodes to execute the task.

5.2.3 Immediate Reward

In this work, we aim to minimize the average long-term delay of learning tasks. In general, the delay of a learning task is determined as the time it stays in the system, including the waiting/queuing time and the serving time. However, in our system, a learning task will remain in the queue until the MEC server successfully decodes the results sent back from the assigned edge nodes. Therefore, the average delay of a learning task can be calculated from the time it arrives at the system until the MEC server successfully decodes its result. It is worth noting that at time slot t when the MEC server performs action a_t to serve a learning task at state s_t, it may not know exactly when the learning task is completed. This is stemmed from the fact that the time to complete this task is determined by the communication time and the computing time as expressed in (2) and (4), respectively. However, the communication time and the computing time are not deterministic due to the random straggling problems in both the edge nodes and wireless links. As a result, to determine the immediate reward when an action is made, we can observe the number of learning tasks in the queue. The reason is that we can implicitly capture the delay of all learning tasks through the length/size of the task queue according to the Little theorem. Thus, we define an immediate reward function for action a_t at state s_t using the instantaneous size of the queue as follows:

$$r_t(s_t, a_t) = -m, \tag{5.10}$$

where $m \in \{0, \dots, M\}$ is the number of learning tasks waiting in the queue after performing action a_t at state s_t. By maximizing the immediate reward function, we can minimize the number of learning tasks in the queue, and thus minimizing the average latency of the whole system.

5.2.4 Long-Term Delay Minimization Formulation

This work aims to obtain the optimal coding and scheduling policy which is a mapping from a given state s to an optimal action to maximize the average long-term reward of the system. In other words, we aim to minimize the average number of learning tasks waiting in the queue. Thus, the optimal coding and scheduling policy can be denoted by $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$, which is then expressed as follows:

$$\max_{\pi} \qquad \mathcal{R}(\pi) = \lim_{T\rightarrow\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left(r_t(s_t, \pi(s_t))\right), \tag{5.11}$$

where $r_t(s_t, \pi(s_t))$ is the immediate reward under policy π at time step t and $\mathcal{R}(\pi)$ is the average long-term reward of the system under policy π. To guarantee that the optimal coding and scheduling policy exists and can be obtained, we prove that the average reward $\mathcal{R}(\pi)$ is well defined and does not depend on the initial state as stated in Theorem 5.2.

Theorem 5.2. *The average reward does not depend on the initial state and is well defined.*

The proof of Theorem 5.2 is provided in Appendix C.1. To obtain the optimal coding and scheduling policy, we use the deep Q-learning and deep dueling algorithms proposed in Section 2.3. It is worth mentioning that our proposed joint coding and scheduling solution does not require any feedback from edge devices. In particular, the system state space consists of the size of the current learning task, the number of learning tasks waiting in the queue, and the states of edge devices. The MEC server always has the information about its queue as well as any learning task arrived at the system. The states of edge devices can be determined by the MEC server as soon as a learning task is successfully served, i.e., all assigned edge nodes become available. Note that, if a straggling node still serves this learning task, it will automatically terminate its computation for this learning task and serve the new learning task sent from the MEC server. As a result, the proposed solution does not add any communication overhead to the system.

5.3 Performance Analysis and Simulation Results

5.3.1 Parameter Setting

In this work, we consider that the task queue at the MEC server can store up to 10 learning tasks. There are five edge nodes in the system to serve learning tasks. Unless otherwise stated, the task arrival probability is set at 0.7. The time for serving one data point is set at 5 milliseconds for all edge nodes [112]. The size (i.e., number of data points) of each learning task is randomly taken from the set of $\{100, 200, 300\}$. We set $\mathbf{p} = \{0.1, 0.5, 0.2, 0.3, 0.9\}$ and $\boldsymbol{\lambda} = \{0.1, 1, 0.5, 0.2, 2\}$. All the above parameters are then varied in the next section to evaluate the performance of our proposed algorithm in various scenarios. It is worth noting that our proposed deep dueling algorithm does not require to know these parameters in advance. Instead, it can interact with the environment, observe results, and then learn these parameters to obtain the optimal coding and scheduling policy in a real-time manner.

The architecture of the deep neural network plays an important role in the learning process, and thus it needs to be carefully designed [74]. In particular, with more hidden layers, the algorithm can learn the problem better. However, the complexity of the algorithm will increase, resulting in a long training period. Moreover, a high number of hidden layers does not always guarantee a good learning performance due to the overestimation problems of optimizers. Similarly, the number of neurons in each layer as well as the mini-batch size are also required a thoughtful design. For the deep Q-learning, we deploy two fully-connected hidden layers connected to the input layer and the output layer. For the deep dueling algorithm, the neural network consists of two streams to separately estimate the value function and the advantage function. These two streams are connected to a shared hidden layer (after the input layer) as illustrated in Fig. 2.6. The sizes of all the hidden layers are set at 16. The mini-batch size is set at 16. The maximum size of the experience replay buffer is $10,000$. The target Q-network is updated after every $1,000$ learning steps. We use the ϵ-greedy scheme with the initial value of ϵ is 1, and its final value is 0.01. The decay factor is set at 0.9999. The learning rate and the discount factor of the deep dueling and the deep Q-learning algorithms are set at 0.0001 and 0.99, respectively. For the Q-learning, the learning rate and the discount factor are set at 0.1 and 0.9, respectively.

To evaluate the proposed solution, we compare its performance with three other approaches: (i) *Greedy*, (ii) *OneNode*, and (iii) *StaticOptimalCode*.

- *Greedy:* For this policy, the MEC server selects all available edge nodes $\mathbf{E_{av}}$ to serve a learning task. The task is then coded with $(n = |\mathbf{E_{av}}|, k = |\mathbf{E_{av}}|)$ MDS code and equally distributed to all the available edge nodes. The MEC server then requires results from all the assigned edge nodes to successfully decode the final result. This policy is used to evaluate the straggling impact of edge nodes and wireless links.

- *OneNode:* In this policy, the MEC server randomly selects one available edge node to serve a learning task. This policy is used to evaluate the typical uncoded and non-distributed learning approaches.

- *StaticOptimalCode*: This policy is based on the optimal MDS code proposed in [52]. In particular, given $\mathbf{E}_{\text{av}}$ edge nodes, the optimal value of k is derived as follows:
$$k^{\dagger} = \left[1 + \frac{1}{W_{-1}(-e^{-\hat{\lambda}-1})}\right] |\mathbf{E}_{\text{av}}|, \tag{5.12}$$
where $W_{-1}(.)$ is the lower branch of Lambert W function and $\hat{\lambda}$ presents the average straggling parameter of all edge nodes. By using this equation, the

MEC server can select $(n = |\mathbf{E}_{\text{av}}|, k = k^{\dagger})$ MDS code for each learning task. It is worth noting that in [52], the authors consider that all edge nodes are identical, i.e., all edge nodes have the same straggling parameter. Moreover, the authors obtain the optimal value of k in the case all edge nodes are used to process a learning task. However, in our work, we consider that all edge nodes are heterogeneous with different straggling parameters. As a result, in (5.12), we define $\hat{\lambda}$ as the average of the straggling parameters of all edge nodes. The *StaticOptimalCode* policy is used to show the performance of a static optimal code that does not consider the heterogeneity of both edge nodes and wireless links, e.g., in channel/backhaul link quality to/from the MEC server, straggling effects and computing capabilities of edge nodes. Moreover, this policy cannot deal with the dynamics and uncertainty of the environment, resulting in poor performance in our considered system as shown in Section 5.3.2.

We also obtain the policy of the proposed solution by running the conventional Q-learning algorithm [73] to compare the effectiveness of the proposed deep dueling algorithm. In this work, we aim to obtain the joint optimal coding and scheduling policy to minimize the average delay for the whole system. Thus, the performance metrics for evaluating the proposed approach are the average number of learning tasks in the queue per time slot, the average task dropping probability, and the average delay in the system of a learning task. The average task dropping probability corresponds to the average number of dropped learning tasks in each time slot when the task queue is full. The average delay of a learning task in the system is calculated from the time a learning task arrives at the system until the task leaves the system (i.e., the MEC server finishes serving the task). The metrics can reveal different aspects of the system which are very useful to help the service provider control Quality-of-Service to the users.

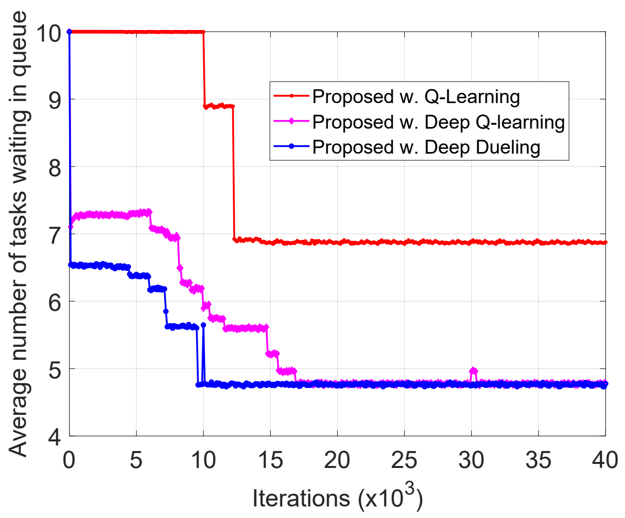


Figure 5.2 : Convergence rates of learning algorithms.

5.3.2 Simulation Results

5.3.2.1 Convergence of Learning Algorithms

In Fig. 5.2, we show the learning processes of the Q-learning, deep Q-learning, and deep dueling algorithms. As can be observed, the convergence rate of the Q-learning algorithm is much slower than those of the deep Q-learning and deep dueling algorithms. This is stemmed from the fact that the Q-learning algorithm has a very slow-convergence due to the curse-of-dimensionality problem, especially in complicated systems as considered in our work. By using the novel deep dueling neural network architecture, our deep dueling algorithm achieves the best convergence rate. In particular, the deep dueling algorithm can converge to the optimal coding and scheduling policy within $10,000$ iterations, while the deep Q-learning algorithm needs more than $15,000$ iterations to converge to the optimal coding and scheduling policy. In the next section, all results of the deep dueling algorithm are obtained at 4×10^4 iterations, while those of the Q-learning algorithm are obtained at 10^6 iterations. Note that the Q-learning algorithm is used as a benchmark to demonstrate the effectiveness of the proposed algorithm.

5.3.2.2 System Performance

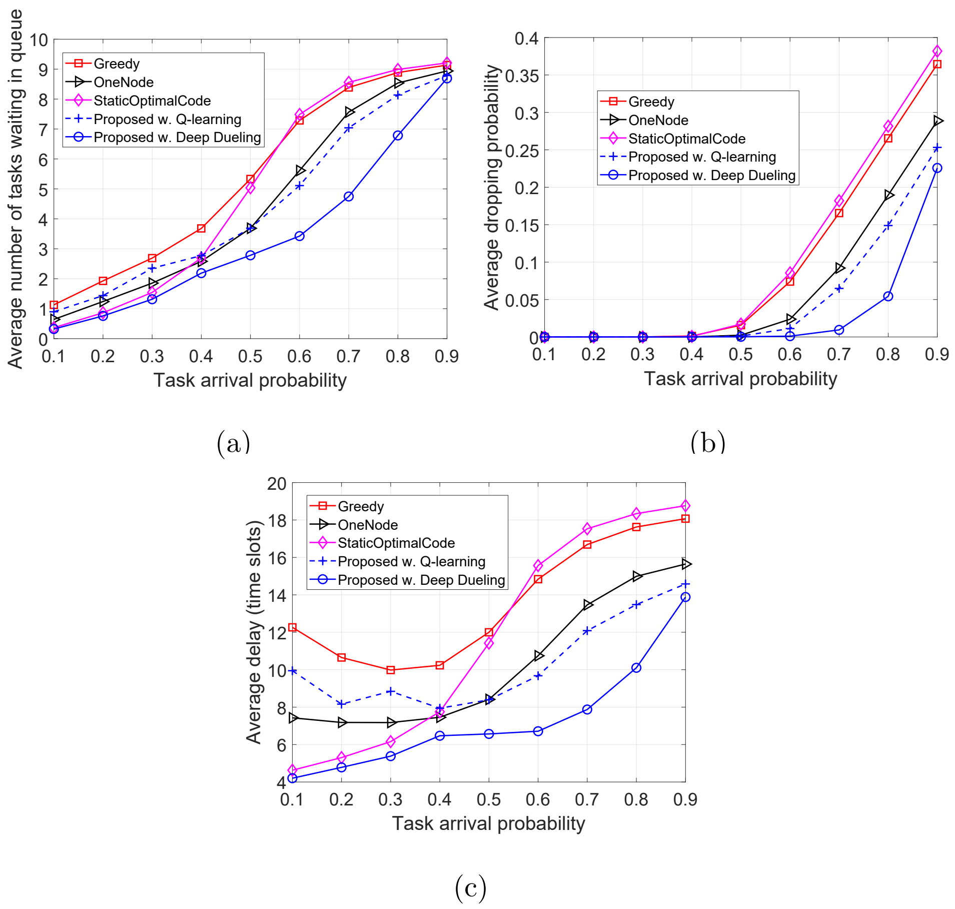


Figure 5.3 : (a) Average number of tasks waiting in the queue, (b) task dropping probability, and (c) average delay of learning tasks in the system vs. task arrival probability.

In this section, we perform simulations to evaluate the performance of the proposed solution in terms of the number of tasks waiting in the queue, the average task dropping probability, and the average delay of learning tasks in the system in various scenarios. First, we vary the arrival probability of learning tasks and compare the performance of the proposed solution with those of *Greedy*, *OneNode*, and *StaticOptimalCode* policies as shown in Fig. 5.3. Clearly, when the task arrival probability increases, the average number of learning tasks in the queue increases as there are more learning tasks arriving at the system as shown in Fig. 5.3(a). With the proposed deep dueling algorithm, our solution can reduce the number of tasks in the queue by up to 71%, 50%, and 54% compared to the *Greedy*, *OneNode*, and *StaticOptimalCode*, respectively. The reason is that the deep dueling algorithm can learn and maximize the number of learning tasks served by the edge nodes by determining the optimal MDS code as well as the best edge nodes with stable wireless links. Fig. 5.3(b) also confirms the outperformance of our proposed solution in terms of task dropping probability. Interestingly, in Fig. 5.3(c), when the task arrival probability increases from 0.1 to 0.3, the average delay of learning tasks in the system under the *Greedy* and *OneNode* schemes decrease by nearly 20% and 3%, respectively. The reason is that under these policies, the MEC server randomly chooses edge nodes to serve learning tasks. As such, there are cases in which learning tasks are severed by highly-straggling edge nodes and/or unstable wireless links. However, when there are more learning tasks arriving at the system, these learning tasks are likely severed by good edge nodes and stable links as the straggling devices may take longer time to serve other learning tasks, and thus they may not be often available. As a result, the average waiting time in the system of a learning task reduces when the arrival probability increases from 0.1 to 0.3. The *StaticOptimalCode* does not encounter this trend and achieves a better performance compared to the *Greedy* and *OneNode* policies, thanks to the use of MDS code. However, when the task arrival probability is higher than 0.3, the performance of the *StaticOptimalCode* is similar to that of the *Greedy* policy and lower than that of the *OneNode* policy. The reason is that when there are many learning tasks arriving at the system, the computing resources of edge nodes are mostly utilized. Thus, at each time slot, the available edge nodes are likely the nodes with good computing power (as they can finish their assigned task quickly and become available for new tasks). Thus, sending a task to a single edge node with high computing power for processing is better than distributing it to several edge nodes with different wireless connections. Note that the *StaticOptimalCode* policy does not account for the effects of unstable wireless links. Nevertheless, by learning and avoiding choosing the slow edge nodes and unstable wireless links, our proposed solution always achieves the best performance. In particular, the average delay of learning tasks obtained by our solution is much lower than those of the *Greedy*, *OneNode*, and *StaticOptimalCode* policies, reduced by up to 66% as shown in Fig. 5.3(c).

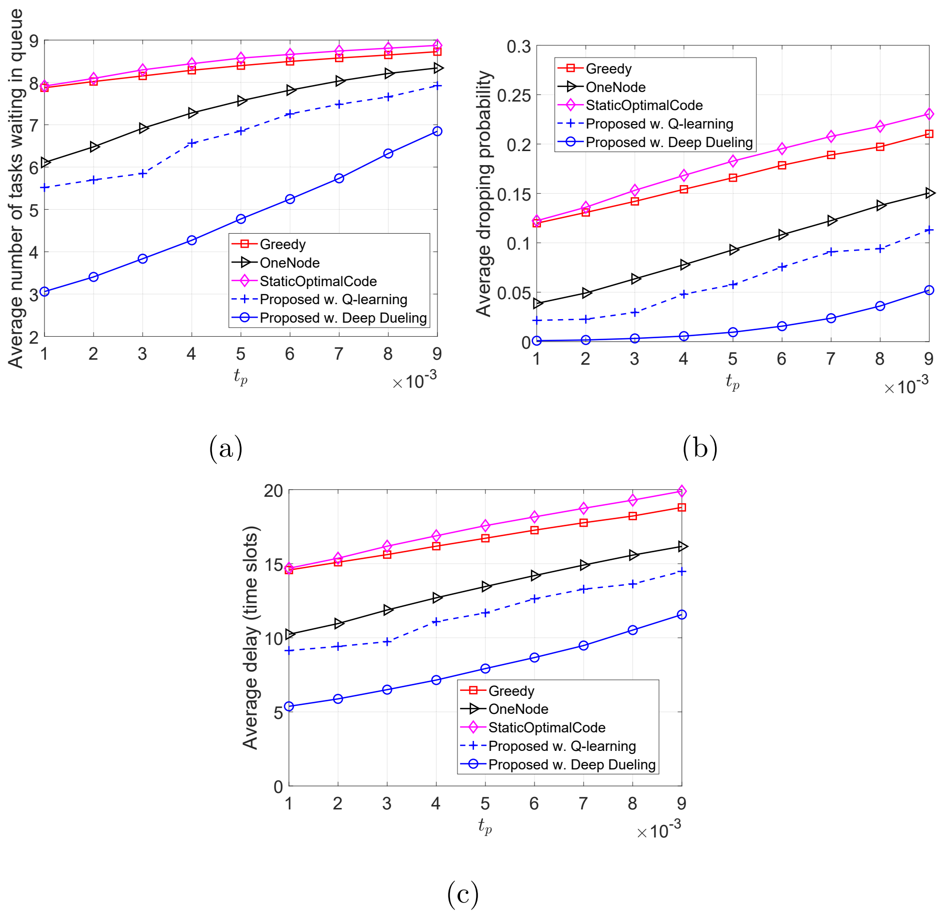
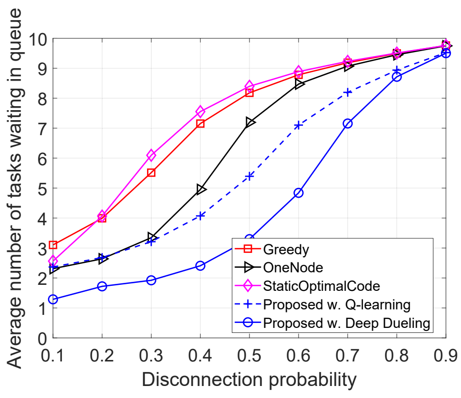


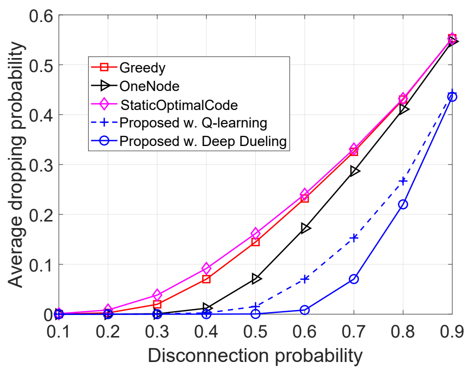
Figure 5.4 : (a) Average number of tasks waiting in the queue, (b) task dropping probability, and (c) average delay of learning tasks in the system vs. processing time.

Next, we vary the time for serving one data point (e.g., one matrix row) t_p

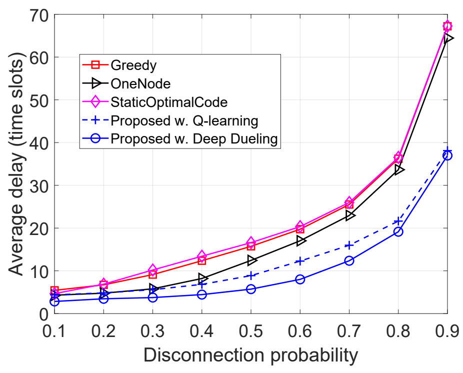
and evaluate the system performance as shown in Fig. 5.4. It can be observed that when the processing time increases, the system performances obtained by all methods significantly decrease. The reason is that, with higher processing time, the edge nodes need more time to serve learning tasks, and thus increasing the serving time of learning tasks. Consequently, learning tasks need to wait longer in the queue. Nevertheless, in all the scenarios, the proposed solution always achieves the best performance and can reduce the average delay of learning tasks by 63%, 47%, and 63% compared to the *Greedy*, *OneNode*, and *StaticOptimalCode*, as shown in Fig. 5.4(c) respectively. The reason is that the deep dueling algorithm can learn and adapt with the environment parameters in order to select the optimal MDS code for each learning task as well as avoid unstable wireless links. It is worth mentioning that the performance achieved by the *OneNode* policy is better than those of the *Greedy* and *StaticOptimalCode* policies. The reason is that we set the disconnection probability of wireless links from the server to edge nodes 2 and 5 at high values (i.e., 0.5 and 0.9, respectively) to clearly see the effect of unstable wireless links. Under the *OneNode* policy, each learning task is served by a single edge node. As such, it can reduce the effect of the unstable wireless links. Among all policies, the *StaticOptimalCode* possesses the worst performance as this policy obtains the optimal MDS code (see (5.12)) without considering the heterogeneity of edge nodes and wireless links. This also confirms our analyses on the drawback of existing static coding mechanisms, i.e., they are only effective under specific scenarios and assumptions. Our proposed solution, otherwise, can learn all the unpredictable parameters of the edge nodes and wireless links to jointly optimize coding and scheduling policies for learning tasks. It is worth noting that the Q-learning algorithm cannot obtain the optimal coding and scheduling policy at 10^6 iterations as discussed above. Thus, the results obtained by the Q-learning algorithm are not as good as those of the proposed deep dueling algorithm.



(a)



(b)



(c)

Figure 5.5 : (a) Average number of tasks waiting in the queue, (b) task dropping probability, and (c) average delay of learning tasks in the system vs. disconnection probability of links.

In Fig. 5.5, we vary the disconnection probability of wireless links and observe the system performance in terms of the number of tasks waiting in the queue, task dropping probability, and the average delay of learning tasks in the system. Clearly, when the disconnection probability increases, system performances obtained by all the policies drop dramatically. This is stemmed from the fact that when the wireless links are more unstable, the MEC server and the edge results need more time to resend the sub-learning task and results, respectively. Consequently, the serving time of learning tasks increases, resulting in a higher delay for learning tasks. It is worth noting that when the disconnection probability increases from 0.1 to 0.6, the performance obtained by the *OneNode* policy is much better than that of the *Greedy* policy. The reason is that under the *Greedy* policy, the MEC server and edge nodes need to resend data when the wireless links are disconnected. Differently, with the *OneNode* policy, the MEC server assigns only one edge node for each learning task, and thus the frequency of resending data is lower than those of the *Greedy* and *StaticOptimalCode* policies, resulting in a better performance. However, the performance gaps between these policies are very small when the disconnection probability is high, i.e., higher 0.7, as all links are likely disconnected. Nevertheless, our proposed solution always achieves the best performance in all the cases compared to those of the *Greedy* and *OneNode* policies. The reason is that the proposed deep dueling can learn from the environment and avoid choosing highly-straggling edge nodes as well as adapt its optimal coding and scheduling policy when the disconnection probability changes. Again, the *StaticOptimalCode* achieves the worst performance as this policy does not account for the effects of unstable wireless links when obtaining the optimal MDS code for each learning task as expressed in (5.12).

Next, we vary the rate parameter λ (in the exponential distribution) of the stochastic computing time of edge nodes and observe the system performance in Fig. 5.6. Recall that, a lower value of λ results in a longer time for stochastic computing. As such, when λ increases, system performances obtained by all the policies will be decreased. Moreover, when λ is small, the gap between the solutions is small. Nevertheless, when λ is increased, the gap will be enlarged. The reason is that, with lower values of λ, the edge nodes require more time to execute learning tasks. As such, the resources of the system are likely to be fully utilized. In these cases, there are not many options for the MEC server to serve learning tasks, resulting in a small performance gap between solutions. However, in all the scenarios, our proposed solution can achieve the best performance as the optimal policy can avoid unstable

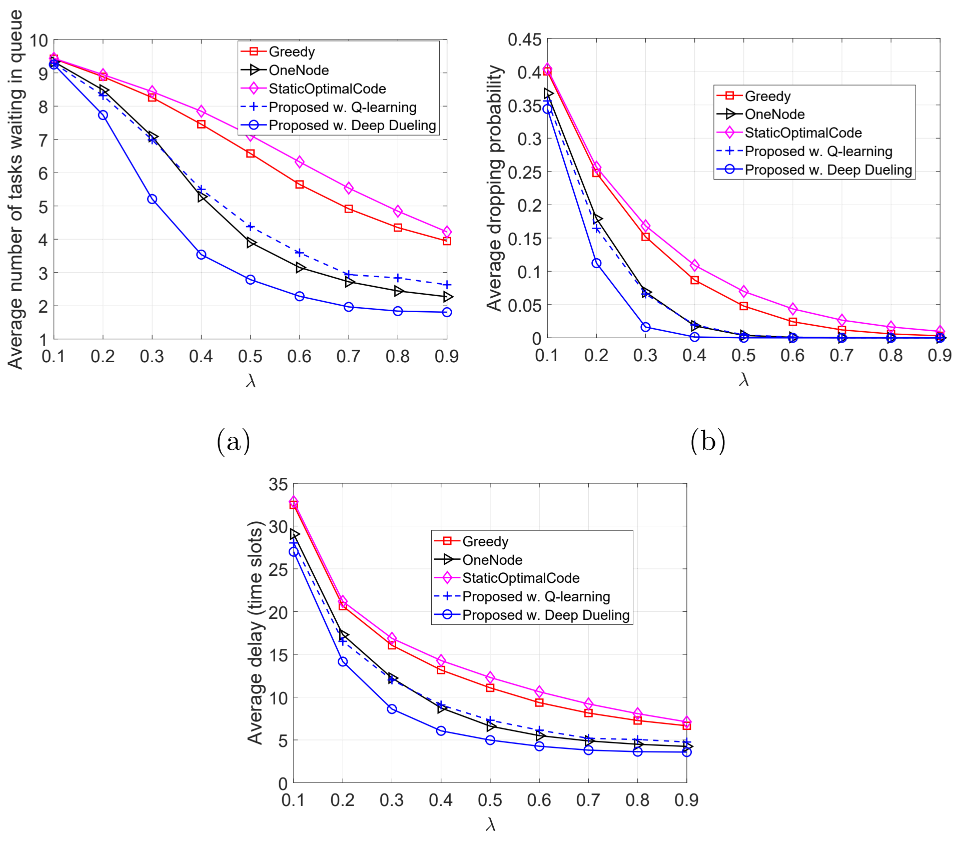


Figure 5.6 : (a) Average number of tasks waiting in the queue, (b) task dropping probability, and (c) average delay of learning tasks in the system vs. the rate parameter λ (in the exponential distribution) of the stochastic computing time of edge nodes.

wireless links and select the optimal MDS code for each learning task.

Finally, we vary the data size of learning tasks and observe the system performances under different policies as shown in Fig. 5.7. Clearly, when the task size increases, the system performances obtained by all the policies will be dropped. It is stemmed from the fact that with a larger task size, the edge nodes need more time to serve learning tasks. However, our proposed solution can always achieve the best performance compared to the other policies because it can learn and select the best

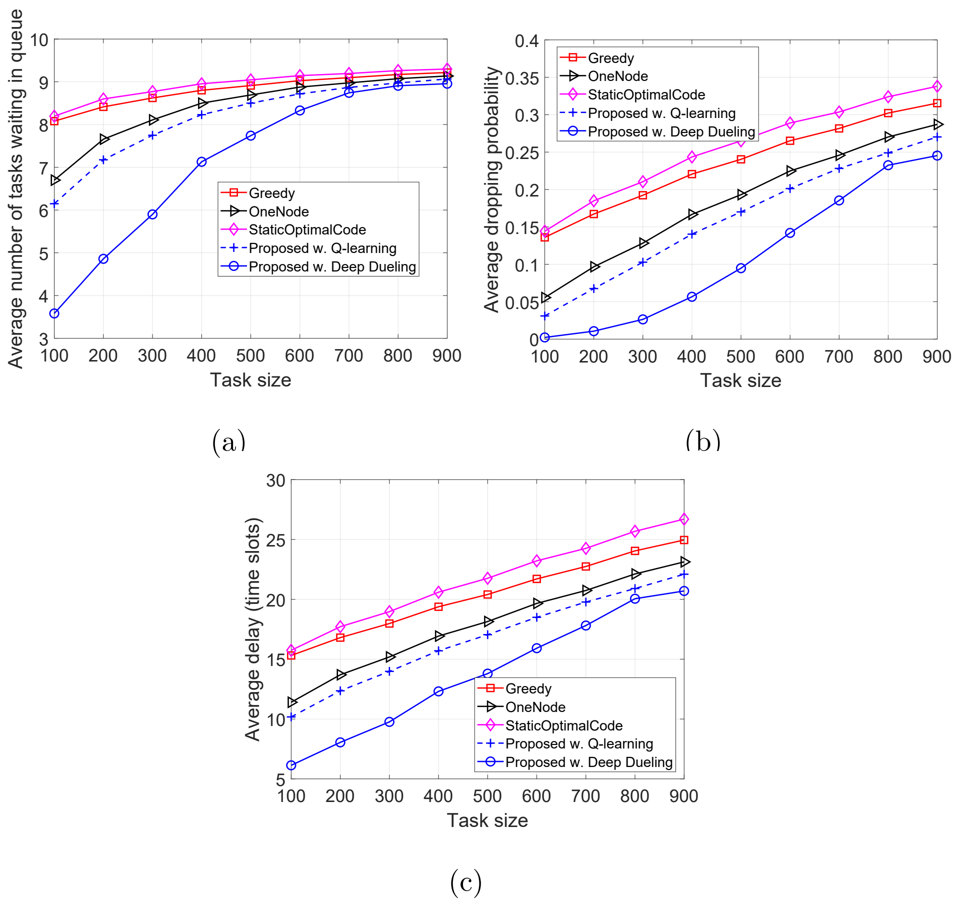


Figure 5.7 : (a) Average number of tasks waiting in the queue, (b) task dropping probability, and (c) average delay of learning tasks in the system vs. task size.

MDS code as well as the best edge nodes for each learning task. For example, when the task size is small, the deep dueling algorithm can select a small number of devices with more stable wireless links to serve a learning task. In contrast, when the task size is large, more edge nodes will be selected to reduce the average serving time of learning tasks. It can be observed that our proposed solution can reduce the average number of tasks waiting in the queue by 60%, 46%, and 61% compared to those of the *Greedy*, *OneNode*, and *StaticOptimalCode* policies, respectively. Again, the *StaticOptimalCode* achieves the worst performance as this policy does not consider the learning task size and the effects of unstable wireless links.

5.4 Conclusion

In this chapter, we have proposed a novel framework that can effectively address key challenges for the development of distributed learning in wireless edge networks. Specifically, we have first introduced a distributed learning model utilizing the recent advances in coded computing to mitigate the straggling problems on both the wireless links and the edge nodes. With the proposed distributed learning model, a learning task is first encoded into sub-learning tasks, and the sub-learning tasks are then transmitted to edge nodes for executing. This solution allows to significantly mitigate straggling problems caused by straggling edge nodes as well as unstable links between the MEC server and edge nodes. Furthermore, to deal with the dynamics and uncertainty of wireless links and straggling edge nodes, we have proposed a novel deep reinforcement learning, called deep dueling, to obtain the optimal code and scheduling policy for each learning task. Extensive simulation results have then demonstrated that our proposed solution can significantly improve the system performance by not only obtaining the optimal MDS code but also finding the best edge nodes to serve each learning task. One of the potential research directions from this work is to deploy multiple virtual machines at each edge node to serve various learning tasks simultaneously.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

In this thesis, we have presented our works in addressing several problems of communications. In particular, our first work concerned with the resource allocation problem in network slicing. In particular, we have developed an optimal and fast network resource management framework which allows the network provider to jointly allocate multiple combinatorial resources (i.e., computing, storage, and radio) to different slice requests in a real-time manner. To deal with the dynamic and uncertainty of slice requests, we have adopted the SMDP. Then, the reinforcement learning algorithms, i.e., Q-learning, deep Q-learning, deep double Q-learning, and deep dueling, have been employed to maximize the long-term average reward for the network provider. The key idea of the deep dueling is using two streams of fully connected hidden layers to concurrently train the value and advantage functions, thereby improving the training process and achieving the outstanding performance for the system. Extensive simulations have shown that the proposed framework using deep dueling can yield up to 40% higher long-term average reward with few thousand times faster compared with those of other network slicing approaches. Future works comprise considering the connectivity resources and the existence of multiple data centers in complex network slicing models by accommodating more states to the system state space. The performance of the proposed solution will be evaluated in terms of complexity and scalability. Moreover, the convergence rate and stability of the deep dueling algorithm will be improved by using the state-of-the-art deep neural networks.

In the second work, we have developed the optimal anti-jamming framework which allows the wireless transceivers to effectively defeat jamming attacks. In particular, with the ambient backscatter capability, while being attacked, the device can either adapt its transmission rate or backscatter its data to the gateway through the jamming signal or harvest energy from the jamming signal to support its operations. To effectively learn about the jamming attacks as well as the channel activities, we have proposed an optimal anti-jamming strategy based on MDP to obtain the optimal defend policy for the transmitter. Then, the reinforcement learning algorithms, i.e., Q-learning, deep Q-learning, and deep dueling, have been developed to maximize the long-term average throughput and minimize the packet loss. Extensive simulations have demonstrated that by using two streams of fully-connected hidden layers, the proposed framework using deep dueling algorithm can improve the average throughput up to 426% and reduce the packet loss by 24%. Importantly, with ambient backscatter and energy harvesting technology, jamming signal can be leveraged by the transmitter as the ambient RF signal, thereby effectively eliminating jamming attacks. To the best of our knowledge, this is the first anti-jamming solution that allows wireless transceivers to not only survive jamming attacks without requiring additional resources but also leverage the jamming signal to improve their transmission rate. The proposed ambient backscattering augmented communications framework can be applicable to both civil (e.g., ultra-reliable and low-latency communications or URLLC) and military scenarios (to combat both inadvertent and deliberate jamming).

In the third work, we have proposed a novel framework which can effectively address key challenges for the development of distributed learning in wireless edge networks. Specifically, we have first introduced a distributed learning model utilizing the recent advances in coded computing to mitigate the straggling problems on both the wireless links and the edge nodes. With the proposed distributed learning model, a learning task is first encoded into sub-learning tasks, and the sub-learning tasks are then transmitted to edge nodes for executing. This solution allows to significantly mitigate straggling problems caused by straggling edge nodes as well as unstable links between the MEC server and edge nodes. Furthermore, to deal with the dynamics and uncertainty of wireless links and straggling edge nodes, we have proposed a novel deep reinforcement learning, called deep dueling, to obtain the optimal code and scheduling policy for each learning task. Extensive simulation results have then demonstrated that our proposed solution can significantly improve the system performance by not only obtaining the optimal MDS code but also finding the best edge nodes to serve each learning task. One of the potential research directions from this work is to deploy multiple virtual machines at each edge node to serve various learning tasks simultaneously.

6.2 Future Works

AI has great potential in addressing emerging problems in future communication systems that conventional approaches cannot effectively handle. However, there is still room to improve the performance of current AI-based solutions as well as to explorer new advanced AI techniques.

- *AI for Heterogeneous Communication Networks:* With the emergence of new services from different types of users and environments, heterogeneous infrastructures are expected to be deployed in future communication systems. However, the curse-of-dimensionality and uncertainty of network parameters introduce new challenges to service providers. Conventional solutions with static approaches may not be feasible to deal with these complex and dynamic problems. To that end, several AI algorithms can be adopted, such as deep learning, transfer learning, and federated learning. Firstly, deep learning can efficiently handle massive amounts of data with dissimilar properties from various services and users. Secondly, transfer learning can be used to leverage knowledge from one environment for other environments. This is partially beneficial for heterogeneous communication networks. Finally, federated learning can be deployed on a huge number of devices to significantly improve the system performance. In particular, each device has its own deep learning model to learn its local data. Once trained, the local model will be sent to the server to compute the global model. This global model then is sent back to the devices for further training. In this way, the overall learning process can be enhanced.

- *AI for Spectrum Management:* The next generation of wireless communications will experience a massive number of devices connected to the Internet, e.g., IoT networks. This can significantly increase the density of the network and reduce the overall throughput. As such, how to smartly and intelligently access the shared spectrum is an important challenge for future wireless communications networks. Moreover, wireless communications are highly dynamic and uncertain. Consequently, existing static approaches in the literature may not be feasible. To address these problems, deep reinforcement learning can be adopted as it can effectively deal with the dynamic and uncertainty of the environment. The reason is that deep reinforcement learning does not require complete information about the environment in advance. When the environment changes, deep reinforcement learning can learn and adjust its optimal policy. Moreover, with our proposed deep dueling algorithm, one can quickly obtain the optimal policy. This is very beneficial in dynamic spectrum access problems of future wireless communication systems, in which wireless users are very dynamic and unpredictable. Given the above, deep reinforcement learning is a promising tool for spectrum management.

- *AI for Edge Networks:* Federated deep learning can be adopted to effectively and securely optimize the performance of edge networks. Nevertheless, exchanging data effectively between collaborators (e.g., wireless edge nodes) is a challenging task. There is a demand for low-complexity techniques that can minimize or tradeoff overhead in computations and communications costs to: (i) reduce the effect of missing/straggling computations during collaboration and (ii) minimize the cost of exchanging data and model parameters. The joint coding and scheduling framework in this thesis can be further extended to different types of coding techniques as well as wireless channel models. Moreover, incentive mechanisms can be proposed to encourage users to contribute to federated learning processes. This will significantly increase the training time and reliability of federated learning. Finally, to minimize the communication and computation cost at the centralized server, another layer of edge device can be used to aggregate trained models from a group of users with related properties. As a result, the communication overhead and computation load at the server can be greatly reduced.

Appendix A

Proofs in Chapter 2

A.1 The proof of Theorem 2.1

Here, we prove that Q-learning converges to the optimum action-values with probability one, i.e., $\mathcal{Q}_t(s,a) \to \mathcal{Q}^*(s,a)$ as $n \to \infty$. The main idea of the convergence proof is an artificial controlled Markov process called the action-replay process (ARP) [73], which is formulated from the learning rate τ_t and the episode sequence. In particular, the state space of the ARP is $\{\langle s,t\rangle\}$ together with a special absorbing state, where s is a state of the real process and $t \geq 1$ is the level of the ARP. The action space is $\{a\}$ where a is an action from the real process.

The stochastic reward and state transition consequence when action a is taken at state s are as follows:

$$\mathbf{i}_* = \begin{cases} argmax_i\{t^i \leq t\}, & \text{if } (s,a) \text{ is perfomed} \\ & \text{before episode t}, \\ 0, & \text{otherwise}, \end{cases} \tag{A.1}$$

where i is the index of the i^{th} time action a is taken at state s. In this way, t^{i_*} is the last time before episode t where (s,a) is executed in the real process. When $i_* = 0$, the reward is set as $\mathcal{Q}_0(s,a)$, and the ARP absorbs. Otherwise, let denote

$$\mathbf{i_e} = \begin{cases} i_*, & \text{with probability } \tau_{t^{i_*}}, \\ i_*-1, & \text{with probability } (1-\tau_{t^{i_*}})\tau_{t^{i_*-1}}, \\ i_*-2, & \text{with probability } (1-\tau_{t^{i_*}})(1-\tau_{t^{i_*-1}})\tau_{t^{i_*-2}}, \\ & \vdots \\ 0, & \text{with probability } \prod_{i=1}^{i_*}(1-\tau_{t^i}), \end{cases} \tag{A.2}$$

as the index of the episode that is replayed or taken. If $i_e = 0$, as above, the reward is set at $\mathcal{Q}_0(s,a)$, and the ARP absorbs. In contrary, when $i_e \# 0$, the reward is $r_{t^{i_e}}$, and a state transition is formed as $\langle s'_{t^{i_e}}, t^{i_e} - 1\rangle$.

As stated in [73], the ARP tends towards the real process. Thus, $\mathcal{Q}_t(s,a)$ tends to $\mathcal{Q}^*(s,a)$, where $\mathcal{Q}_t(s,a) = \mathcal{Q}^*_{ARP}(\langle s,t\rangle, a), \forall a, s$, and $t \geq 0$, is the optimal Q-value for the $t^t h$ level of the ARP [73, Lemma A]. Without loss of generality, we assume that $\mathcal{Q}_t(s,a) < \frac{r^*}{(1-\gamma)}$, where $r* \geq |r_t|$ is the bound of the reward. Given $\chi > 0$, choose ξ such that

$$\gamma^{\xi} \frac{r*}{1-\gamma} < \frac{\chi}{6}. \tag{A.3}$$

Based on Lemmas B.2, B.3, B.4 in [73], we can compare the value $\bar{\mathcal{Q}}_{ARP}(\langle s,t\rangle, a_1, \ldots, a_\xi)$ of taking actions $a_1, \ldots, a_\xi$ in the ARP with $\bar{\mathcal{Q}}(s, a_1, \ldots, a_\xi)$ of taking them in the real process as follows:

$$\begin{aligned} &|\bar{\mathcal{Q}}_{ARP}(< s,t >, a_1, \ldots, a_\xi) - \bar{\mathcal{Q}}(s, a_1, \ldots, a_\xi)| < \\ &\frac{\chi(1-\gamma)}{6\xi r*}\frac{2\xi r*}{1-\gamma} + \frac{2\chi}{3\xi(\xi+1)}\frac{\xi(\xi+1)}{2} = \frac{2\chi}{3}. \end{aligned} \tag{A.4}$$

Clearly, the effect of taking only ξ actions makes a difference of less than $\frac{\chi}{6}$ for both the ARP and the real processes. As Eq. (A.4) can be applied to any set of actions, it applies perforce to a set of actions optimal for either the real process or the ARP. Thus, we have

$$|\mathcal{Q}^*_{ARP}(\langle s,t\rangle, a) - \mathcal{Q}^*(s,a)| < \chi. \tag{A.5}$$

As a result, with probability 1, $\mathcal{Q}_t(s,a) \to \mathcal{Q}^*(s,a)$ as $n \to \infty$.

Appendix B

Proofs in Chapter 3

B.1 The proof of Theorem 3.1

For any $t \geq 0$, define the matrix $\mathbf{P}(t)$ by $\mathbf{P}(t) = (p_{\mathbf{s},\mathbf{s}'}(t)), \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S}$. Denote by $\mathbf{Q}$, the matrix $\mathbf{Q} = q_{\mathbf{s},\mathbf{s}'}, \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S}$, where the diagonal elements $q_{\mathbf{s},\mathbf{s}}$ are defined by:

$$q_{\mathbf{s},\mathbf{s}} = -z_{\mathbf{s}}. \tag{B.1}$$

After that Kolmogoroff's forward differential equations can be written as $\mathbf{P}'(t) = \mathbf{P}(t)\mathbf{Q}$ for any $t \geq 0$. Hence, the solution of this system of differential equations is given by:

$$\mathbf{P}(t) = e^{t\mathbf{Q}} = \sum_{n=0}^{\infty} \frac{t^n}{n!}\mathbf{Q}^n, t \geq 0. \tag{B.2}$$

The matrix $\overline{\mathbf{P}} = \overline{p}_{\mathbf{s},\mathbf{s}'}, \forall \mathbf{s}, \mathbf{s}' \in \mathcal{S}$ can be reformulated as $\overline{\mathbf{P}} = \mathbf{Q}/z + \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. Therefor, we have

$$\begin{aligned}\mathbf{P}(t) = e^{t\mathbf{Q}} = e^{zt(\overline{\mathbf{P}}-\mathbf{I})} &= e^{zt\overline{\mathbf{P}}}e^{-zt\mathbf{I}} = e^{-zt}e^{zt\overline{\mathbf{P}}} \\ &= \sum_{n=0}^{\infty} e^{-zt}\frac{(zt)^n}{n!}\overline{\mathbf{P}}^n.\end{aligned} \tag{B.3}$$

Based on conditioning on the number of Poisson events up to time t in the $\{\overline{X}(t)\}$ process, we have

$$P\{\overline{X}(t) = \mathbf{s}' | \overline{X}(0) = \mathbf{s}\} = \sum_{n=0}^{\infty} e^{-zt}\frac{(zt)^n}{n!}\overline{p}^{(n)}_{\mathbf{s},\mathbf{s}'}, \tag{B.4}$$

where $\overline{p}^{(n)}_{\mathbf{s},\mathbf{s}'}$ is the n-step transition probability of the discrete-time Markov chain $\overline{X}_n$. By recalling the Corollary 3.1, the proof is completed.

B.2 The proof of Lemma 3.1

Let $\{A_n : n \geq 0\}$ be a sequence of matrices. We have $\lim\limits_{n\to\infty} A_n = A$ if $\lim\limits_{n\to\infty} A_n(\mathbf{s}'|\mathbf{s}) =$ $(\mathbf{s}'|\mathbf{s})$ for each $(\mathbf{s}, \mathbf{s}') \in \mathcal{S} \times \mathcal{S}$. We now consider the Cesaro limit which is defined as follows. We say that A is the Cesaro limit (of order one) of $\{A_n : n \geq 0\}$ if

$$\lim_{n\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} A_n = A, \tag{B.5}$$

and write

$$C\text{-}\lim_{N\to\infty} = A \tag{B.6}$$

to distinguish this as a Cesaro limit. We then define the limiting matrix $\overline{P}$ by

$$\overline{P} = C\text{-}\lim_{N\to\infty} P^N. \tag{B.7}$$

In component notation, where $\overline{p}(\mathbf{s}'|\mathbf{s})$ denotes the $(\mathbf{s}'|\mathbf{s})$-th element of $\overline{P}$, this means that, for each $\mathbf{s}$ and $\mathbf{s}'$, we have

$$\overline{p}(\mathbf{s}'|\mathbf{s}) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} p^{n-1}(\mathbf{s}'|\mathbf{s}), \tag{B.8}$$

where p^{n-1} denotes a component of P^{n-1} and $p^0(s'|s)$ is a component of an $\mathcal{S} \times \mathcal{S}$ identity matrix. As P is aperiodic, $\lim_{N\to\infty}$ exists and equals to $\overline{P}$.

Appendix C

Proofs in Chapter 5

C.1 The proof of Theorem 5.2

We first prove that the underlying Markov chain in this work is irreducible. In other words, from any state, the process can always move to any other states after a finite number of steps. Recall that the system state is defined as the state of the queue m, the task size f, and the state of all edge nodes in the system $\{e_1, \dots, e_j, \dots, e_N\}$. At each time slot, a learning task arrives at the system with probability μ. Thus, there always exists a probability that the queue state moves from m to $m' = m+1$. Moreover, a learning task will be removed from the queue if it is successfully served. In this case, the queue state moves from m to $m' = m-1$. The task size is a random value. As such, it can take any positive values. Alternatively, edge node E_j is available (i.e., $e_j = 1$) when it does not serve any learning task. In contrast, edge node is unavailable (i.e., $e_j = 0$) if it is serving another learning task. As a result, edge nodes can always move from the available state to the unavailable state. Thus, the underlying Markov chain can move from a given state to any other states after a finite number of steps. As such, the average long-term reward $\mathcal{R}(\pi)$ is well defined and does not depend on the initial state for every π [103].

Bibliography

[1] Y. Sun, M. Peng, Y. Zhou, Y. Huang, and S. Mao, "Application of Machine Learning in Wireless Networks: Key Techniques and Open Issues," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3072-3108, Fourth Quarter 2019.

[2] Cisco Annual Internet Report (2018–2023) White Paper. Available Online: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[3] M. K. Hanawal, M. J. Abdel-Rahman, and M. Krunz, "Joint adaptation of frequency hopping and transmission rate for anti-jamming wireless systems," *IEEE Transactions on Mobile Computing*, vol. 15, no. 9, Sept. 2016, pp. 2247-2259.

[4] W. Xu *et al.*, "The feasibility of launching and detecting jamming attacks in wireless networks," In *Proc. ACM MobiHoc*, pp. 46-57, Urbana-Champaign, IL, USA, May 2005.

[5] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-Edge Ai: Intelligentizing Mobile Edge Computing, Caching and Communication by Federated Learning," *IEEE Network*, vol. 33, no. 5, pp. 156-165, Sept/Oct. 2019.

[6] Q. V. Pham, F. Fang, V. N. Ha, M. J. Piran, M. Le, L. B. Le, W.-J. Hwang, and Z. Ding, "A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art," IEEE Access, vol. 8, pp. 116974-117017, Jun. 2020.

[7] "NGMN 5G White Paper," *Next Generation Mobile Networks*, White Paper, 2015.

[8] A. Manzalini, C. Buyukkoc, P. Chemouil, S. Kuklinski, F. Callegati, A. Galis, M. -P. Odini, C. -L. I, J. Huang, M. Bursell, N. Crespi, E. Healy, and S. Sharrock, "Towards 5g software-defined ecosystems," *IEEE SDN White Paper*, 2016.

[9] H. Zhang, N. Liu, X. Chu, K. Long, A. -H. Aghvami, and V. C. Leung, "Network slicing based 5G and future mobile networks: mobility, resource management, and challenges," *IEEE Communications Magazine*, vol. 55, no. 8, Aug. 2017, pp. 138-145.

[10] M. Jiang, M. Condoluci, and T. Mahmoodi, "Network slicing management & prioritization in 5G mobile systems," *22th European Wireless Conference*, Oulu, Finland, Finland, May 2016.

[11] H. M. Soliman, and A. Leon-Garcia, "QoS-aware frequency-space network slicing and admission control for virtual wireless networks," *IEEE Global Communications Conference (GLOBECOM)*, Washington, DC, USA, Dec. 2016.

[12] J. Zheng, P. Caballero, G. D. Veciana, S. J. Baek, and A. Banchs, "Statistical multiplexing and traffic shaping games for network slicing," *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 6, Dec. 2018, pp. 2528-2541.

[13] S. D'Oro, F. Restuccia, T. Melodia, and S. Palazzo, "Low-complexity distributed radio access network slicing: Algorithms and experimental results," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, Dec. 2018, pp. 2815-2828.

[14] P. L. Vo, M. N. H. Nguyen, T. A. Le, and N. H. Tran, "Slicing the edge: Resource allocation for RAN network slicing," *IEEE Wireless Communications Letters*, vol. 7, no. 6, Dec. 2018, pp. 970-973.

[15] Y. Sun, M. Peng, S. Mao, and S. Yan, "Hierarchical Radio Resource Allocation for Network Slicing in Fog Radio Access Networks," *IEEE Transactions on Vehicular Technology*, Early Access, Jan. 2019.

[16] J. Ni, X. Lin, and X. S. Shen, "Efficient and secure service-oriented authentication supporting network slicing for 5G-enabled IoT," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, Mar. 2018, pp. 644-657.

[17] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs, "Mobile traffic forecasting for maximizing 5G network slicing resource utilization," *IEEE Conference on Computer Communications (INFOCOM)*, Atlanta, GA, USA, May 2017.

[18] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez, "Optimising 5G infrastructure markets: The business of network slicing," *IEEE Conference on Computer Communications (INFOCOM)*, Atlanta, GA, USA, May 2017.

[19] A. Aijaz, "Hap-SliceR: A Radio Resource Slicing Framework for 5G Networks With Haptic Communications," *IEEE Systems Journal*, no. 99, Jan. 2017, pp. 1-12.

[20] A. Aijaz, "Radio resource slicing in a radio access network," *U.S. Patent Application 15/441,564*, filed November 2, 2017.

[21] A. B. Koehler, R. D. Snyder, J. K. Ord, "Forecasting models and prediction intervals for the multiplicative Holt–Winters method," *International Journal of Forecasting*, vol. 17, no. 2, pp. 269 - 286, Jun. 2001.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: MIT Press, 1998.

[23] An Introduction to Network Slicing, *The GSM Association,* [Online]. Available: `https://www.gsma.com/futurenetworks/wp-content/uploads/2017/11/GSMA-An-Introduction-to-Network-Slicing.pdf`

[24] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, May 2017, pp. 94-100.

[25] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Hoboken, NJ: Wiley, 1994.

[26] Z. Wang, T. Schaul, M. Hessel, H. V. Hasselt, M. Lanctot, and N. D. Freitas, "Dueling network architectures for deep reinforcement learning," [Online]. Available: arXiv:1511.06581.

[27] R. T. Yazicigil, P. Nadeau, D. Richman, C. Juvekar, K. Vaidya, and A. P. Chandrakasan, "Ultra-fast bit-level frequency-hopping transmitter for securing low-power wireless devices," *2018 IEEE Radio Frequency Integrated Circuits Symposium (RFIC)*, PA, USA, Aug. 2018.

[28] A. Mpitziopoulos, D. Gavalas, G. Pantziou, and C. Konstantopoulos, "Defending wireless sensor networks from jamming attacks," *IEEE PIMRC*, Athens, Greece, Dec. 2007.

[29] A. Sabharwal, P. Schniter, D. Guo, D. W. Bliss, S. Rangarajan, and R. Wichman, "In-band full-duplex wireless: Challenges and opportunities," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 9, Jun. 2014, pp. 1637-1652.

[30] B. Wang, Y. Wu, K. R. Liu, and T. C. Clancy, "An anti-jamming stochastic game for cognitive radio networks," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 4, Apr. 2011, pp. 877-889.

[31] Y. Gao, Y. Xiao, M. Wu, M. Xiao, and J. Shao, “Game theory-based anti-jamming strategies for frequency hopping wireless communications,” *IEEE Trans. Wireless Commun.*, vol. 17, no. 8, Aug. 2018, pp. 5314-5326.

[32] L. Kong, Y. Xu, Y. Zhang, X. Pei, M. Ke, X. Wang, W. Bai, and Z. Feng, “A reinforcement learning approach for dynamic spectrum anti-jamming in fading environment,” *IEEE International Conference on Communication Technology (ICCT)*, Chongqing, China, Oct. 2018.

[33] X. Liu, Y. Xu, L. Jia, Q. Wu and A. Anpalagan, “Anti-Jamming Communications Using Spectrum Waterfall: A Deep Reinforcement Learning Approach,” IEEE Commun. Letters, vol. 22, no. 5, pp. 998-1001, May 2018.

[34] W. Cheng, Z. Li, F. Gao, L. Liang, and H. Zhang, “Mode Hopping for Anti-Jamming in Cognitive Radio Networks,” *IEEE/CIC International Conference on Communications in China (ICCC)*, Beijing, China, Aug. 2018.

[35] K. Pelechrinis, I. Broustis, S. V. Krishnamurthy, and C. Gkantsidis, “Ares: An anti-jamming reinforcement system for 802.11 networks,” *ACM CoNEXT*, Rome, Italy, Dec. 2009.

[36] K. Firouzbakht, G. Noubir, and M. Salehi,“On the capacity of rate-adaptive packetized wireless communication links under jamming,” in*Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, Tucson, AZ, USA, 2012, pp. 3-14.

[37] G. Noubir, R. Rajaraman, B. Sheng, and B. Thapa, “On the robustness of IEEE 802.11 rate adaptation algorithms against smart jamming,” in *Proceedings of the fourth ACM conference on Wireless network security*, Hamburg, Germany, Jun. 2011, pp. 97-108.

[38] A. Mpitziopoulos, D. Gavalas, C. Konstantopoulos, and G. Pantziou, "A survey on jamming attacks and countermeasures in WSNs," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 4, Fourth Quarter 2009, pp. 42-56.

[39] V. Liu, A. Parks, V. Talla, S. Gollakota, D. Wetherall, and J. R. Smith, "Ambient backscatter: Wireless communication out of thin air," *ACM SIGCOMM*, Hong Kong, China, Aug. 2013.

[40] N. V. Huynh, D. T. Hoang, X. Lu, D. Niyato, P. Wang, and D. I. Kim, "Ambient Backscatter Communications: A Contemporary Survey," *IEEE Communications Surveys & Tutorials*, vol. 20 , no. 4 , Fourthquarter 2018, pp. 2889-2922.

[41] J. Kimionis, A. Bletsas, and J. N. Sahalos, "Bistatic backscatter radio for tag read-range extension," in *Proceedings of IEEE International Conference on RFID-Technologies and Applications (RFID-TA)*, pp. 356-361, Nice, France, Nov. 2012.

[42] J. Kimionis, A. Bletsas, and J. N. Sahalos, "Increased range bistatic scatter radio," *IEEE Transactions on Communications*, vol. 62, no. 3, Mar. 2014, pp. 1091-1104.

[43] P. N. Alevizos, N. F. Hilliard, K. Tountas, N. Agadakos, N. Kargas, and A. Bletsas, "Channel coding for increased range bistatic backscatter radio: Experimental results," in *Proc. of IEEE RFID Technology and Applications Conference (RFID-TA)*, Tampere, Finland, Sept. 2014, pp. 38-43.

[44] S. N. Daskalakis, S. D. Assimonis, E. Kampianakis, and A. Bletsas, "Soil Moisture Scatter Radio Networking With Low Power," *IEEE Transactions on Microwave Theory and Techniques*, vol. 64, no. 7, Jul. 2016, pp. 2338-2346.

[45] U. Olgun, C. C. Chen, and J. L. Volakis, "Low-profile planar rectenna for batteryless RFID sensors," in *Proc. IEEE APSURSI*, Toronto, ON, USA, Jun.

2010, pp. 1-4.

[46] U. Olgun, C. C. Chen, and J. L. Volakis, “Wireless power harvesting with planar rectennas for 2.45 GHz RFIDs,” in *Proc. IEEE EMTS*, Berlin, Germany, Aug. 2010, pp. 329-331.

[47] S. Scorcioni, L. Larcher, and A. Bertacchini, “Optimized CMOS RF-DC converters for remote wireless powering of RFID applications,” in *Proc. IEEE Int. Conf. RFID*, Orlando, FL, USA, Apr. 2012, pp. 47-53.

[48] C. Zhang, H. Wang, and M. Yen, “Low power analog circuit design for RFID sensing circuits,” in *Proc. IEEE Int. Conf. RFID*, Orlando, FL, USA, Apr. 2010, pp. 16-21.

[49] Z. Hu, N. Wei, and Z. Zhang, “Optimal resource allocation for harvested energy maximization in wideband cognitive radio network with SWIPT,” *IEEE Access*, vol. 5, 2017, pp. 23383-23394.

[50] A. E. Shafie, N. Al-Dhahir, and R. Hamila, “Cooperative access schemes for efficient SWIPT transmissions in cognitive radio networks,” in *Proc. IEEE Globecom Workshops (GC Wkshps)*, San Diego, CA, USA, 2015, pp. 1-6.

[51] X. Gong, “Delay-Optimal Distributed Edge Computing in Wireless Edge Networks,” *IEEE INFOCOM*, Toronto, ON, Canada, 6-9 Jul. 2020.

[52] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, “Speeding Up Distributed Machine Learning Using Codes,” *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514-1529, Mar. 2018.

[53] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, “Speeding-up Distributed Machine Learning Using Codes”, *NIPS: Workshop on Machine Learning Systems*, Montreal, Canada, December, 2015.

[54] S. Prakash, S. Dhakal, M. R. Akdeniz, Y. Yona, S. Talwar, S. Avestimehr, and N. Himayat, “Coded Computing for Low-Latency Federated Learning Over Wireless Edge Networks,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 233-250, Nov. 2020.

[55] S. Dhakal, S. Prakash, Y. Yona, S. Talwar, and N. Himayat, “Coded computing for distributed machine learning in wireless edge network,” *IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, Honolulu, HI, USA, 22-25 Sept. 2019.

[56] A. Severinson, A. G. Amat, and E. Rosnes, “Block-diagonal coding for distributed computing with straggling servers,” *IEEE Information Theory Workshop (ITW)*, Kaohsiung, Taiwan, 6-10 Nov. 2017.

[57] S. Dutta, V. Cadambe, and P. Grover, ““Short-Dot”: Computing Large Linear Transforms Distributedly Using Coded Short Dot Products,” *IEEE Transactions on Information Theory*, vol. 65, no. 10, pp. 6171-6193, Jul. 2019.

[58] R. Bitar, P. Parag, and S. E. Rouayheb, “Minimizing latency for secure coded computing using secret sharing via staircase codes,” *IEEE Transactions on Communications*, vol. 68, no. 8, pp. 4609-4619, Apr. 2020.

[59] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, “Gradient coding: Avoiding stragglers in distributed learning,” *International Conference on Machine Learning (ICML)*, Sydney, Australia, 06-11 Aug. 2017.

[60] C. Karakus, Y. Sun, S. N. Diggavi, and W. Yin, “Redundancy Techniques for Straggler Mitigation in Distributed Optimization and Learning,” *Journal of Machine Learning Research*, vol. 20, no. 72, pp. 1-47, Apr. 2019.

[61] H. Park, K. Lee, J. -y. Sohn, C. Suh, and J. Moon, “Hierarchical coding for distributed computing,” *IEEE International Symposium on Information Theory (ISIT)*, Vail, CO, USA, 17-22 Jun. 2018.

[62] F. Haddadpour and V. R. Cadambe, “Codes for distributed finite alphabet matrix-vector multiplication,” *IEEE International Symposium on Information Theory (ISIT)*, Vail, CO, USA, 17-22 Jun. 2018.

[63] N. Ferdinand and S. C. Draper, “Hierarchical coded computation,” *IEEE International Symposium on Information Theory (ISIT)*, Vail, CO, USA, 17-22 Jun. 2018.

[64] S. Prakash, A. Reisizadeh, R. Pedarsani, and A. S. Avestimehr, “Hierarchical coded gradient aggregation for learning at the edge,” *IEEE International Symposium on Information Theory*, Los Angeles, CA, USA, 21-26 Jun. 2020.

[65] D. J. Han, J.-Y. Sohn, and J. Moon, “Hierarchical Broadcast Coding: Expediting Distributed Learning at the Wireless Edge,” *IEEE Transactions on Wireless Communications*, vol. 20, no. 4, pp. 2266-2281, Apr. 2021.

[66] D. J. Han, J.-Y. Sohn, and J. Moon, “Coded distributed computing over packet erasure channels,” *IEEE International Symposium on Information Theory*, Paris, France, 7-12 Jul. 2019.

[67] F. Maturana and K. V. Rashmi, “Convertible codes: new class of codes for efficient conversion of coded data in distributed storage,” *Innovations in Theoretical Computer Science Conference (ITCS 2020)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[68] K. T. Kim, C. J.-Wong, and M. Chiang, “Coded Edge Computing,” *IEEE INFOCOM*, Toronto, ON, Canada, 6-9 Jul. 2020.

[69] S. Ha, J. Zhang, O. Simeone, and J. Kang, “Coded federated computing in wireless networks with straggling devices and imperfect CSI,” *IEEE International Symposium on Information Theory (ISIT)*, Paris, France, 7-12 Jul. 2019.

[70] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133-3174, Fourthquarter 2019.

[71] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224-2287, Thirdquarter 2019.

[72] Introducing Deep Learning with MATLAB, [Online]. Available: `https://www.mathworks.com/campaigns/offers/next/deep-learning-ebook.html`

[73] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, 1992.

[74] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[75] H. V. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *AAAI*, 2016.

[76] N. Bihannic, T. Lejkin, I. Finkler, and A. Frerejean, "Network Slicing and Blockchain to Support the Transformation of Connectivity Services in the Manufacturing Industry," *IEEE Softwarization*, Mar. 2018.

[77] 5G Network Slicing for Vertical Industries, *Global mobile Suppliers Association.* Available Online: `https://www.huawei.com/minisite/5g/img/5g-network-slicing-for-vertical-industries-en.pdf`

[78] ETSI, "Network Functions Virtualisation (NFV); Use Cases". [Online]. Available: `https://www.etsi.org/deliver/etsi_gr/NFV/001_099/001/01.02.01_60/gr_nfv001v010201p.pdf`

[79] ETSI, "Network Functions Virtualisation (NFV) Release 3; Evolution and Ecosystem; Report on Network Slicing Support with ETSI NFV Architecture Framework". [Online]. Available: `https://www.etsi.org/deliver/etsi_gr/NFV-EVE/001_099/012/03.01.01_60/gr_NFV-EVE012v030101p.pdf`

[80] ETSI, "Network Functions Virtualisation (NFV); NFV Performance & Portability Best Practises". [Online]. Available: `https://www.etsi.org/deliver/etsi_gs/NFV-PER/001_099/001/01.01.01_60/gs_nfv-per001v010101p.pdf`

[81] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, Feb. 2013, pp. 1888-1906.

[82] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *CloudNet*, Niagara Falls, ON, Canada, Oct. 2015.

[83] A. Gupta, M. F. Habib, P. Chowdhury, M. Tornatore, and B. Mukherjee, "Joint virtual network function placement and routing of traffic in operator networks," in *NetSoft*, 2015.

[84] M. Mechtri, C. Ghribi, and D. Zeghlache, "A scalable algorithm for the placement of service function chains," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, Aug. 2016, pp. 533-546.

[85] M. Leconte, G. Paschos, P. Mertikopoulos, and U. Kozat, "A resource allocation framework for network slicing", IEEE INFOCOM 2018, Honolulu, HI, USA, 15-19 Apr. 2018.

[86] B. Awerbuch, Y. Azar, and S. Plotkin, "Throughput Competitive on-line routing," in *Proc. 34th IEEE Ann. Svmp. Foundations Comput. Sci.*, Nov. 1993, pp. 3240.

[87] R. G. Gallager, *Discrete stochastic processes*, Kluwer Academic Publishers, London, 1995.

[88] H. C. Tijms, *A first course in stochastic models,* John Wiley and Sons, 2003.

[89] L. Kallenberg, *Markov Decision Process,* [Online]. Available: `http://www.math.leidenuniv.nl/%7Ekallenberg/Lecture-notes-MDP.pdf`.

[90] A. Geramifard, T. J. Walsh, S. Tellex, G. Chowdhary, N. Roy, and J. P. How, "A tutorial on linear function approximators for dynamic programming and reinforcement learning," *Found. Trends Mach. Learn.*, vol. 6, no. 4, pp. 375–451, 2013.

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

[92] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, Feb. 2015, pp. 529-533.

[93] M. Abadi et al., "Tensorflow: Large-Scale Machine Learning on Heterogeneous Systems," [Online]. Available: arXiv:1603.04467.

[94] D. Sattar and A. Matrawy, "Optimal Slice Allocation in 5G Core Networks," [Online]. Available: arXiv:1802.04655.

[95] X. Chen, Z. Li, Y. Zhang, R. Long, H. Yu, X. Du, and M. Guizani, "Reinforcement Learning based QoS/QoE-aware Service Function Chaining in Software-Driven 5G Slices," [Online]. Available: arXiv:1804.02099.

[96] B. Han, J. Lianghai, and H. D. Schotten, "Slice as an Evolutionary Service: Genetic Optimization for Inter-Slice Resource Management in 5G Networks,"

IEEE Access, Jun. 2018.

[97] F. B. Uz, "GPUs vs CPUs for deployment of deep learning models", [Online]. Available: `https://azure.microsoft.com/en-au/blog/gpus-vs-cpus-for-deployment-of-deep-learning-models/`

[98] C. Boyer and S. Roy, "Backscatter communication and RFID: Coding, energy, and MIMO analysis, *IEEE Transactions on Communications*, vol. 62, no. 3, pp. 770-785, Mar. 2014.

[99] C. A. Balanis,*Antenna Theory: Analysis and Design.* New York, NY: Wiley, 2012.

[100] F. F. Digham, M.-S. Alouini, and M. K. Simon, "On the energy detection of unknown signals over fading channels," *IEEE Transactions on Communications*, vol. 55, no. 1, Jan. 2007, pp. 21-24.

[101] W. Zhang, R. K. Mallik, and K. B. Letaief, "Optimization of cooperative spectrum sensing with energy detection in cognitive radio networks," *IEEE Transactions on Wireless Communications*, vol. 8, no. 12, Dec. 2009, pp. 5761-5766.

[102] C. Yang, J. Gummeson, and A. Sample, "Riding the airways: Ultra-wideband ambient backscatter via commercial broadcast systems," *IEEE INFOCOM*, Atlanta, GA, USA, May 2017.

[103] J. Filar and K. Vrieze, *Competitive Markov Decision Processes.* Springer Press, 1997.

[104] Dan Sullivan, "Bringing deep learning to IoT devices", [Online]. Available: https://samsungnext.com/whats-next/deep-learning-iot/

[105] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proceedings of the 28th International Con-*

ference on Neural Information Processing Systems (NIPS), Montreal, Canada, Dec. 2015.

[106] 21W jammer [Online]. Available: http://drone-jammers.com/shop/21w-jammer-with-8-antennas-for-blocking-cdma-2g-3g-4g-lte-wimax-wifi-2-4ghz-uhf-vhf-rc-gps-lojack/

[107] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Transactions on Information Theory*, vol. 65, no. 7, pp. 4227-4242, Mar. 2019.

[108] K. Li, M. Tao, J. Zhang, and O. Simeone, "Multi-cell mobile edge coded computing: Trading communication and computing for distributed matrix multiplication," *IEEE International Symposium on Information Theory (ISIT)*, Los Angeles, CA, USA, 21-26 Jun. 2020.

[109] H. Sawada, S. Takahashi, and S. Kato, "Disconnection probability improvement by using artificial multi reflectors for millimeter-wave indoor wireless communications," *IEEE Transactions on Antennas and Propagation*, vol. 61, no. 4, pp. 1868-1875, Apr. 2013.

[110] S. Takabe and T. Wadayama, "Approximation theory for connectivity of ad hoc wireless networks with node faults," *IEEE Wireless Communications Letters*, vol. 8, no. 4, pp. 1240-1243, Aug. 2019.

[111] F. Wu and L. Chen, "Latency Optimization for Coded Computation Straggled by Wireless Transmission," *IEEE Wireless Communications Letters*, vol. 9, no. 7, pp. 1124-1128, Jul. 2020.

[112] J. Zhang and O. Simeone, "On model coding for distributed inference and transmission in mobile edge computing systems," *IEEE Communications Letters*, vol. 23, no. 6, pp. 1065-1068, Apr. 2019.

[113] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Wiley-Interscience series in discrete mathematics and optimization, 1990.

[114] G. Cerar, H. Yetgin, M. Mohorcic, and C. Fortuna, "Machine learning for wireless link quality estimation: A survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 696-728, Second Quarter 2021.

[115] Z. Huang, L. Yee, T. F. Ang, M. H. Anisi, and M. S. Adam, "Improving the accuracy rate of link quality estimation using fuzzy logic in mobile wireless sensor network," *Advances in Fuzzy Systems*, 2019.

[116] N. Baccour, A. Koubaa, L. Mottola, M. A. Zuniga, H. Youssef, C. A. Boano, and M. Alves, "Radio link quality estimation in wireless sensor networks: A survey," *ACM Transactions on Sensor Networks*, vol. 8, no. 4, pp. 1-33, Sep. 2012.

[117] N. V. Huynh, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz, "Jam Me If You Can: Defeating Jammer with Deep Dueling Neural Network Architecture and Ambient Backscattering Augmented Communications", *IEEE Journal on Selected Areas in Communications*, vol. 37 , no. 11, pp. 2603-2620, Nov. 2019.

[118] N. V. Huynh, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "Optimal and fast real-time resource slicing with deep dueling neural networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1455-1470, Mar. 2019.

[119] N. V. Huynh, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "Deep-Fake: Deep Dueling-based Deception Strategy to Defeat Reactive Jammers," arXiv:2005.07034.