TECHNISCHE UNIVERSITÄT MÜNCHEN
TUM School of Computation, Information and Technology

Network Slicing with Reinforcement Learning and Transfer Learning

Tianlun Hu

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz:		Prof. Chunyang Chen, Ph.D.
Prüfer der Dissertation:	1.	Prof. Dr.-Ing. Georg Carle
	2.	Assistant Prof. Qiang Liu

Die Dissertation wurde am 24.06.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 30.10.2024 angenommen

Abstract:

Network slicing, a pivotal aspect of 5G and beyond, allows operators to configure virtual network instances tailored to diverse services with specific requirements. However, achieving efficient slice-aware radio resource scheduling poses challenges due to complex inter-cell dependencies, inter-slice resource constraints, and service-specific needs.

Reinforcement Learning (RL) opens a novel avenue for addressing dynamic optimization challenges, particularly in network management. As model-free solutions, RL approaches are committed to dynamically providing optimal solutions by formulating real systems as Markov Decision Process (MDP) and addressing the associated problems through RL algorithms. Recent advancements in Deep Reinforcement Learning (DRL) further expand their capabilities, enabling the resolution of more complex scenarios. This dissertation proposes innovative solutions leveraging multi-agent DRL to enhance slice performance while adhering to resource capacity constraints. This research also discusses the approaches employing multiple DRL agents to cooperatively optimize resource partition in individual cells for diverse slices, for escalating the densification of network deployment introduces non-trivial inter-cell interference.

Recently, in several classical machine learning fields, Transfer Learning (TL) techniques are also recommended to improve the model reproducibility and sample efficiency via transferring prior knowledge. Recognizing the limitations of DRL approaches tied to specific network environments, this dissertation also investigates the TL-aided solutions for network slicing. Experimental results indicate that integrating TL significantly improves service performance, reduces exploration costs, accelerates convergence rates, and improves model reproducibility of RL approaches, thereby outperforming existing baseline approaches. Lastly, to overcome the limitations of deep learning models in managing dynamic slicing configurations, the dissertation introduces a novel framework integrating constrained optimization methods and deep learning models, which exhibits high scalability, accommodating varying numbers of slices and configurations.

This comprehensive dissertation contributes to advancing network slicing optimization through innovative applications of RL and TL techniques, specifically addressing the critical issue of slicing resource allocation. As closing remarks, we reiterate the caveats of using RL and TL techniques in next-generation network management and argue for broader research and practical usage of advanced machine learning methods in network optimization.

Kurzfassung:

Network Slicing, ein Schlüsselelement der 5G-Technologie und darüber hinaus, erlaubt Betreibern, virtuelle Netzwerkinstanzen zu erstellen, die für verschiedene Dienste mit spezifischen Anforderungen maßgeschneidert sind. Allerdings ist die effiziente, slice-bewusste Planung von Funkressourcen aufgrund komplexer, zellübergreifender Abhängigkeiten, Inter-Slice-Ressourcenbeschränkungen und dienstspezifischer Anforderungen eine Herausforderung.

Reinforcement Learning (RL) bietet neue Möglichkeiten, um dynamische Optimierungsprobleme zu lösen, insbesondere im Bereich des Netzwerkmanagements. Als modellfreie Ansätze formulieren RL Methoden reale Systeme als Markov-Entscheidungsprozesse (MDP) und lösen die damit verbundenen Probleme dynamisch durch RL Algorithmen. Fortschritte im Bereich des Deep Reinforcement Learning (DRL) haben dessen Fähigkeiten erweitert, um noch komplexere Szenarien zu bewältigen. Diese Dissertation präsentiert innovative Lösungen, die Multi-Agent-DRL verwenden, um die Leistung von Network Slices zu verbessern und gleichzeitig Ressourcenbeschränkungen einzuhalten. Sie diskutiert auch Ansätze, bei denen mehrere DRL Agenten zusammenarbeiten, um die Ressourcenaufteilung innerhalb einzelner Zellen für verschiedene Slices zu optimieren, wobei zunehmende Netzverdichtung erhebliche zellübergreifende Interferenzen verursacht.

Darüber hinaus werden in dieser Arbeit Transfer-Learning (TL) Techniken vorgeschlagen, um in traditionellen Bereichen des maschinellen Lernens die Modellreproduzierbarkeit und Sample-Effizienz durch die Übertragung von Vorwissen zu verbessern. Angesichts der Einschränkungen von DRL, die oft an spezifische Netzwerkumgebungen gebunden sind, untersucht diese Dissertation auch TL-unterstützte Lösungen für das Network Slicing. Experimentelle Ergebnisse zeigen, dass die Integration von TL die Dienstleistung erheblich verbessert, die Erkundungskosten senkt, die Konvergenzraten beschleunigt und die Reproduzierbarkeit der Modelle von RL verbessert, wodurch bestehende Basisansätze übertroffen werden. Abschließend wird ein neuartiges Framework vorgestellt, das die Einschränkungen von Deep-Learning-Modellen bei der Verwaltung dynamischer Slicing-Konfigurationen überwindet, indem es beschränkte Optimierungsmethoden und Deep-Learning-Modelle integriert und eine hohe Skalierbarkeit aufweist, die unterschiedliche Anzahlen von Slices und Konfigurationen unterstützt.

Diese umfassende Dissertation leistet einen wesentlichen Beitrag zur Optimierung des Network Slicings durch die innovative Anwendung von Transfer Learning und Reinforcement Learning Techniken und adressiert insbesondere das kritische Problem der Ressourcenaufteilung. Abschließend betone ich die Notwendigkeit weiterer Forschung und praktischer Anwendung fortschrittlicher maschineller Lernmethoden zur Optimierung von Netzwerken der nächsten Generation.

Acknowledgments:

This thesis would not have been possible without the extensive support I received from many individuals. Although it is impossible to list everyone, I would like to highlight some who have played an important role in my journey.

First and foremost, I would like to express my special thanks to Prof. Dr.-Ing. Georg Carle for giving me the opportunity to join the Department of Network Architecture and Services and for his invaluable guidance throughout my thesis. Each interaction with him has greatly enriched my understanding and shaped my future plans and research work. Secondly, I would like to thank Prof. Liu Qiang for serving as the second examiner. His incisive questions during our discussions greatly broadened my understanding of the research field and methodologies. Furthermore, I would also like to extend my gratitude to my industrial mentor, Dr.-Ing. Liao Qi from Nokia Bell Labs, for her generous mentorship, rigorous standards, and patience. Her guidance transformed me from a novice in scientific research into a doctoral candidate.

I am deeply appreciative of the team at the network automation department of Nokia Bell Labs in Stuttgart—Markus, Ilaria, Alessandro, and Lutz—for their selfless assistance and the exceptional industrial training opportunities they provided. My heartfelt thanks also go to my fellow colleagues Yanni, Changran, Haotian, Ruichuan, Marco, Naveenta, Markus, Junqing, Lelio, Vladislav, and Emre, who infused my PhD journey with boundless energy and joy. Additionally, I am grateful for the contributions of my co-authors Dan, Antonio, Stanislaw, Patrick, and Nikolaj, whose collaborative efforts significantly elevated the quality of our publications.

Lastly, I must express my profound appreciation to my parents, Weihou and Zhaofen. Their constant support and inspiration have been fundamental to my academic success. Thank you both for your love and encouragement, which propelled me on this path. I could not have reached this point without your enduring support.

Garching bei München, 08.01.2025

Tianlun Hu

Contents

1. Introduction

As the integration of mobile networks into our daily lives and production processes has evolved into a fundamental aspect, the expectation of internet access at all times, regardless of location, has become a universal consensus and a habitual necessity for human society. In every instance of human interaction, whether with others or devices, the mobile network serves as an essential element, which underscores its vital position in facilitating connectivity within the context of wireless communication.

Envisioning next-generation communication networks, where individuals, devices, and systems communicate seamlessly and in real-time, establishes a framework where high-speed connectivity, low latency, and reliability form the foundational elements of this visionary landscape. In 5^{th} Generation Mobile Network (5G) [1] and beyond, networks are expected to meet a wide array of advanced requirements, far surpassing the capabilities of previous generations of mobile communications (Fig. 1.1). In general, a network with 5G standard [2] should have a high data rate of up to $20Gbps$ for downlink and $10Gbps$ for uplink transition, ultra-low latency as low as $1ms$ and massive connectivity to support Internet of Things (IoT). Besides, new network management should guarantee improved usage of available spectrum to increase overall network performance, with efficient allocation and utilization of network resources to optimize performance and reduce operational costs.

Network slicing technology occupies a pivotal position in the landscape of next-generation networks to enable those characteristics. It offers a paradigm shift in how communication services are delivered. By allowing the creation of isolated and customized virtual networks tailored to specific applications, industries, or user requirements, network slicing introduces unparalleled flexibility and efficiency. This innovative approach enables diverse services to coexist on a shared physical infrastructure. Advanced network slicing technique optimizes resource utilization and facilitates dynamic adaptation to varying traffic patterns and evolving service demands. In 5G and beyond, network slicing is also regarded as a cornerstone leading wireless communication towards a more adaptive, responsive, and application-aware network framework, catalyzing transformative capabilities of next-generation communication networks. To achieve service-specific use cases, 5G network operators defined several commonly defined slice types, such as Extreme Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communication (URLLC), and Massive Machine-Type Communications (mMTC), to meet the diverse requirements of various applications and services.

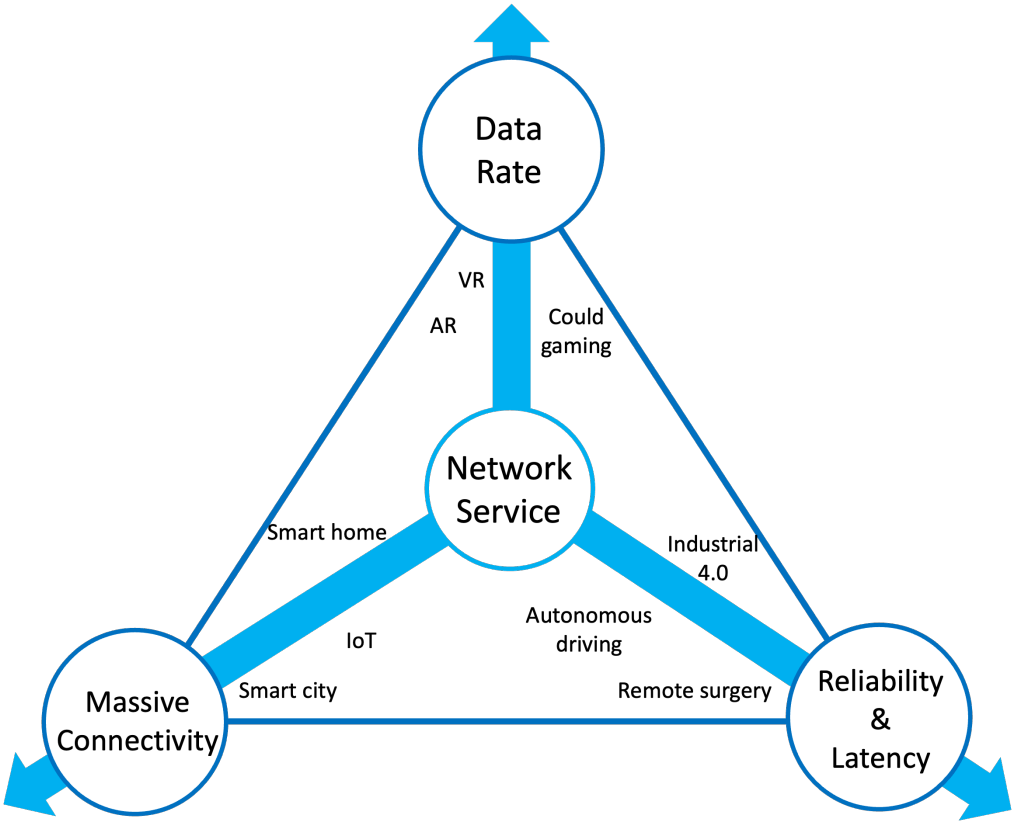

Figure 1.1: Evolution of network services in 5G

- **eMBB**: eMBB is designed to deliver significantly faster data rates and lower latency compared to previous generations of mobile networks. eMBB slices prioritize high data throughput and low latency to ensure a seamless user experience for bandwidth-intensive applications, which is the type of network slicing designed for the 5G applications that generate the most traffic on mobile networks. It aims to support applications such as high-quality video streaming, Virtual Reality (VR), Augmented Reality (AR), and cloud gaming running on remote hardware.

- **URLLC**: URLLC focuses on providing ultra-reliable and low-latency communication services, particularly for mission-critical applications where reliability and responsiveness are paramount. URLLC slices prioritize extremely low latency, high reliability, and bounded fluctuation of connectivity to meet the stringent requirements of mission-critical applications. The solution leverages state-of-the-art mobile edge computing technology to provide dependable, low-latency guarantees to priority applications. It targets supporting industrial manufacturing, real-time control systems, remote surgery, autonomous vehicles, and critical infrastructure monitoring applications.

- **mMTC**: mMTC is tailored to support numerous low-power devices that transmit small amounts of data sporadically. mMTC slices prioritize scalability, energy efficiency, and coverage extension to accommodate devices with diverse traffic patterns. Its applications are mainly under large-scale systems, such as smart cities, smart homes, industrial IoT, and environmental monitoring. Network operators use mMTC for applications where vast quantities of interconnected devices are under the scope of 5G IoT.

The above-mentioned slice types enable 5G network to efficiently allocate resources and meet the diverse requirements of different applications and services, ranging from high-speed multimedia streaming to ultra-reliable mission-critical communications and massive

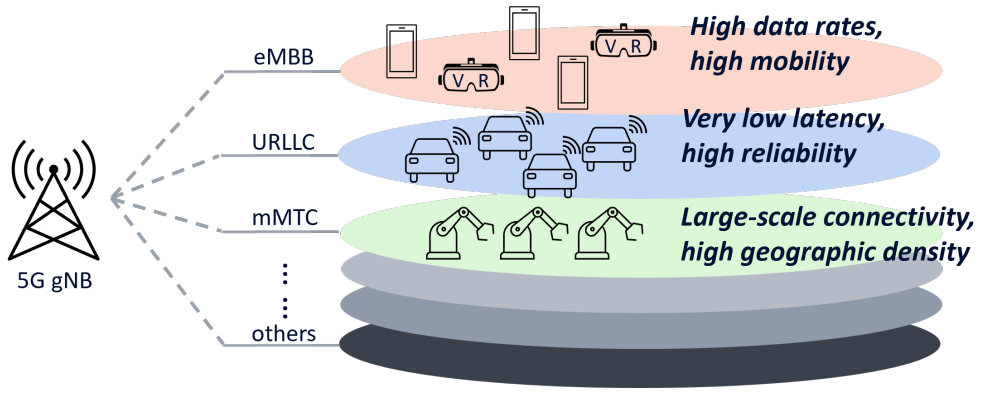

Figure 1.2: Services and slices with different requirements in 5G networks

IoT deployments. However, with the emerging evolution of network services recalled by technical advancements in other domains, the types of slices in next-generation networks are not static or predefined. In our research, we are not only considering slicing scenarios involving eMBB, URLLC, and mMTC, but also exploring a broader spectrum of slice types defined by diverse QoS requirements.

As breakthroughs continuously emerge across various fields such as Computer Vision (CV), Large Language Model (LLM), intelligent vehicles, and space technology, there is a corresponding push for mobile networks to support a broad spectrum of network services, including Extended Reality (XR), vehicular communication, online streaming, and even satellite communication. The critical role of network slicing technology in new network services, particularly for those advanced merging applications, can not be overstated. However, integrating network slicing in next-generation networks presents multifaceted challenges that demand highly efficient solutions for seamless operation. Efficient end-to-end orchestration across different domains, dynamic adaptation to changing service demands, and ensuring cross-domain interoperability further contribute to the complexity.

In the design process of network slicing techniques, the following principles should be considered as outlined in [1]:

- **Flexibility:** The invention of implementing slicing techniques in networks is to ensure case-specific traffic treatment while avoiding unnecessary functionalities at each slice. This remarks the importance of flexibility, enabling the enhancement of existing services and the development of new ones.

- **Requirements Satisfaction:** In the context of automotive network applications, the slicing solutions should ensure exceptional slice security, reliability, and low latency.

- **Function Granularity:** Determining function granularity is critical. Although a finer granularity can provide higher flexibility, it also introduces complexity. The challenges associated with testing different function combinations, implementing slice configurations, and interference issues.

Addressing these issues effectively is crucial for successfully integrating network slicing into next-generation networks, offering substantial benefits. Therefore, research into slicing techniques is vital for advancing next-generation networks. This study proceeded under consideration of comprehensive exploration of network slicing in 5G and beyond, emphasizing the need for innovative solutions to navigate the evolving challenges of resource allocation, interference coordination, and real-time adaptability in the face of increasingly diverse and dynamic network requirements.

1.1 Network Slicing in Next-Generation Networks

The advent of 5G and beyond demands a sophisticated approach to network management, while network slicing empowers operators to craft isolated virtual networks tailored for specific services on shared physical infrastructures. These slices, addressing diverse needs such as enhanced mobile broadband, ultra-reliable low-latency communications, and emerging technologies like autonomous driving, are characterized by varying performance requirements, including throughput and latency. The basic implementation of network slicing is to partition radio resources dynamically across multiple base stations, such as gNodeBs (gNBs), to maximize efficiency while minimizing inter-cell resource usage. As technologies like high-load online streaming and emerging Artificial Intelligence (AI) models further intensify network dynamics, the challenge for communications service providers lies in efficiently managing the dynamic and complex interplay of traffic, mobility, and demand within network slices.

Efforts to realize dynamic network slicing under changing traffic conditions have led to the development of slice-aware scheduling algorithms in RAN. However, the proliferation of base station deployments, including small cells, adds a layer of complexity, especially in multi-cell scenarios where the lack of interference coordination can degrade slice performance. Traditional model-based solutions face limitations in this landscape, prompting exploring novel approaches such as ML to address the ever-complicating resource management challenges. This dynamic environment also necessitates flexible and real-time network monitoring and configuration efforts, exemplified by initiatives like Open Radio Access Network (O-RAN).

1.1.1 Network Slicing Technique

Regarding the frequency of slicing orchestration changes, in the beginning, slicing techniques are proposed as "static network slicing" that strictly isolated difference network services for specific use cases [3]. Static network solutions are usually pre-planned by experts. This slicing method ensures that network resources, such as bandwidth, allocated to network slice instances are exclusively dedicated to the instances. Under complex network slicing scenarios, specific network resources are designated and reserved for corresponding slices, thereby ensuring that the network Service Level Agreement (SLA) is maintained at a satisfactory and robust level. Compared to static slicing methods, "dynamic network slicing" offers a more flexible and operational approach to distributing network resources, partitioning them among different slices based on real-time network demands and changing service behaviors. The primary advantage of dynamic slicing over static solutions is its ability to enable more efficient resource utilization across various use cases. Considering the growing variety of services in the next-generation networks, the slicing methods are expected to be embedded with dynamical optimization ability. Thus, in this work, we mainly focus on exploring the methods of solving slicing resource allocation problems in dynamic manners under the assumption of frequently changing network environments in terms of network architecture, configuration, and user behavior.

Dynamically slicing a network system to accommodate various services is complicated with the conventional network basis; thus, several techniques are designed to realize network slicing on shared physical network architectures. Among these methods, Software-Defined Networking (SDN) and Network Function Virtualization (NFV) are commonly known as the key techniques for facilitating diverse, slice-specific services.

- **Network Function Virtualization (NFV):** NFV plays an important role in enabling network slicing in 5G, which proposes to decouple network functions from proprietary hardware devices, allowing them to operate as software on virtual machines

or containers. This decoupling facilitates the creation and management of network slices by enabling flexible and dynamic slicing resource allocation. NFV abstracts physical resources into a pool that can be dynamically allocated to software-defined network functions, such as firewalls, routers, and load-balancers. This abstraction allows for the flexible creation and management of individual network slices with specific resource allocations. Besides, virtual network functions can be scaled up and down in real time to adapt to varying network load conditions and slice requirements. These features of NFV build the fundamentals of slicing applications by providing flexibility, scalability, and automation needed to create and manage multiple network slices.

- **Software-Defined Networking (SDN)**: SDN enables network administrators to manage network services by abstracting lower-level functionality. It facilitates dynamic, programmable, and centralized control of network resources by separating the control plane from the data plane. As its name suggests, SDN programs and customizes network policies and configurations according to slice requirements, simplifying the creation and management of slices tailored to diverse use cases. It enables the dynamic allocation of network resources based on slice demands, ensuring optimal performance and efficient resource utilization with appropriate methods. Performance assessment of slice resource allocation methods is typically based on slice QoS, measured by evaluating whether slice users receive the necessary bandwidth, latency, and reliability to meet their performance requirements.

Implementing acNFV and SDN in network slicing offers a flexible and efficient approach to network slicing management. It enables operators to create customized, isolated, and optimized virtual networks tailored to the diverse needs of modern applications and services.

1.1.2 Challenges

Despite network slicing technology holds great promise in tailoring network services to different needs, it faces many challenges that require careful consideration and innovative solutions[4]. As wireless communication meshes continue to be deployed with expanded coverage, the increasing scale of network systems introduces more difficulties for network management and slicing methods in terms of complexity and efficiency. The discussion of this work is based on the premise of large-scale network systems that comprise multiple base stations and slices. Therefore, the following concerns were taken into consideration:

a) **Resource Allocation:** Efficiently allocating resources across network slices presents a critical challenge for network slicing, as each slice has unique bandwidth, latency, and computing requirements, which demands precise resource management. Balancing these diverse needs to ensure optimal performance without resource wastage poses a significant hurdle. Addressing this challenge requires advanced algorithms and dynamic resource orchestration mechanisms to adapt to real-time fluctuations in demand.

b) **Inter-Cell Interference**: The coexistence of multiple base stations within the same physical infrastructure introduces the potential for inter-cell interference [5], impacting overall network performance. Intelligent interference coordination strategies, such as frequency planning and spectrum management [6], are commonly used to mitigate conflicts and ensure smooth and reliable operation. Resolving inter-cell interference is essential for achieving the envisioned seamless and interference-free

connectivity across different slices. Under multi-cell and multi-slice network scenarios, different network slices sharing the same physical infrastructure may experience interference, affecting performance and reliability.

c) **Slice Specific Requirements**: Varying Slice-specific requirements pose significant challenges for slicing solutions, including allocating resources effectively to meet diverse demands. Additionally, the dynamic nature of slice requirements, which may change based on varying traffic patterns and user demands, complicates resource allocation and configuration management. Scalability issues arise when attempting to accommodate multiple slices while maintaining performance and efficiency, further exacerbating the challenge. Coordinating slice management while ensuring network stability and interoperability with existing infrastructure requires sophisticated algorithms and frameworks.

Considering these challenges, successfully implementing network slicing requires a holistic approach that can address resource allocation, inter-cell interference, and domain discrepancies well. Tackling these challenges is essential for unlocking the full potential of network slicing in 5G and beyond, enabling efficient communication, and evolving modern applications and services.

1.1.3 Key Aspects of Solutions

In addition to addressing the challenges mentioned earlier, network slicing solutions must also align with practical considerations for real-time implementation. Therefore, an ideal slicing solution should account for both the effectiveness of the methods and the feasibility of implementing these novel methods in real-world scenarios. Based on my knowledge of network management, an effective dynamic slicing method should support rapid deployment, high scalability, and high reproducibility. This insight comes from the understanding that deriving an optimal network solution is both computationally and temporally intensive. Particularly during periods of network architecture expansion and frequent replanning, developing a new solution from scratch for a large-scale system is inefficient. Therefore, the following aspects should be considered when developing an effective slicing solution:

a) **Processing Time:** Efficient network slicing requires swift processing times to meet the real-time demands of modern applications and services. Time-sensitive applications, such as augmented reality and autonomous systems, necessitate minimal processing delays. Therefore, an effective slicing solution must have the ability of rapid resource allocation, orchestration, and decision-making to ensure low-latency communication and a seamless user experience.

b) **Model Accuracy:** The accuracy of underlying models used in network slicing is crucial for effective resource allocation and service customization. Precision in predicting and understanding network conditions, traffic patterns, and service requirements directly impacts the efficiency of the slices. Ensuring high model accuracy through continuous refinement and validation is essential for meeting the diverse needs of applications and users.

c) **Reproducibility:** An essential requirement for the viability of network slicing is the capacity to replicate solutions across diverse network scenarios and environments. Reproducibility ensures that successful slicing strategies can be consistently applied, promoting scalability and adaptability. Standardized approaches, well-documented methodologies, and open interfaces contribute significantly to ensuring the reproducibility of slicing solutions across a range of deployment scenarios.

d) **Various Granularity:** Having network slicing solutions with different granularity is crucial for effectively enabling diverse network management levels. Fine-grained slicing allows for precise customization of resources, facilitating a higher degree of service differentiation. However, balancing accuracy and granularity is essential, as excessively granular slicing can introduce complexity and overhead. Defining the optimal level of granularity ensures that the network can adapt to specific application needs without sacrificing overall efficiency.

e) **Flexibility and Adaptability:** The dynamic nature of communication networks necessitates flexibility and adaptability in network slicing solutions. As networks evolve over time, the slicing framework must accommodate changes in traffic patterns, user demands, and technological advancements. A solution capable of real-time adaptation to shifting requirements ensures the longevity and relevance of network slicing in an ever-changing technological landscape.

1.2 Methods of Slice Resource Allocation

In the context of network slicing, resource allocation and optimization are developed to address the problem of efficiently distributing network resources to different network slices while ensuring individual requirements are fulfilled for each slice. Generally, the primary goal of slicing resource optimization is to maximize the overall network performance and resource utilization with respect to the QoS and Quality of Experience (QoE) of each slice. For specific network slicing scenarios, the optimization objectives are detailed by certain requests, such as enhancing network efficiency, ensuring slice performance, reducing operational costs, or improving scalability and flexibility. However, those objectives are challenging because of the dynamic nature of network demands, the heterogeneity of services, and the constraints imposed by physical resources.

In wireless communication, network resources refer to a wide range of elements and capacities within a network system that are necessary to provide communication services, including the frequency spectrum, bandwidth, and transmission power. For Long-Term Evolution (LTE) and 5G networks, Physical Resource Block (PRB) presents the finest unit of resource allocation in the time-frequency grid, consisting of a specific number of frequency subcarriers over a certain time interval. The resource allocation methods on PRB provide a fine-grained level of granularity, allowing for precise control over resource distribution and efficient adaptation to diverse service requirements. PRBs can be dynamically allocated and managed based on the changing demands of network traffic and QoS requirements of different slices. This approach scales effectively to handle the growing complexity and demand of modern wireless communication systems, making PRB-based resource allocation methods a practical and effective solution for network slicing.

Fig. 1.3 presents an instance of PRB pairs in the time-frequency domain of a typical LTE network, where the concepts can be easily extended to 5G and beyond. Each block in the grid represents a Resource Element (RE), which corresponds to a time symbol in the time domain and a small subcarrier in the frequency domain, typically with $15kHz$ in LTE. Information is encoded and mapped onto symbols through modulation techniques determined by the channel quality. RE is recognized as the smallest identifiable resource unit, which comprises a subcarrier and an Orthogonal Frequency-Division Multiplexing (OFDM) symbol, serving as the fundamental unit for data modulation. In both LTE and 5G networks, a Resource Block (RB) is a virtually defined unit of resource allocation, typically comprising 12 subcarriers and 7 symbols, which represents the resource unit that can be directly assigned to users, this resource portion amounts to $180kHz$ of bandwidth within a slot with $0.5ms$. Despite the number of subcarriers in an RB remains fixed at

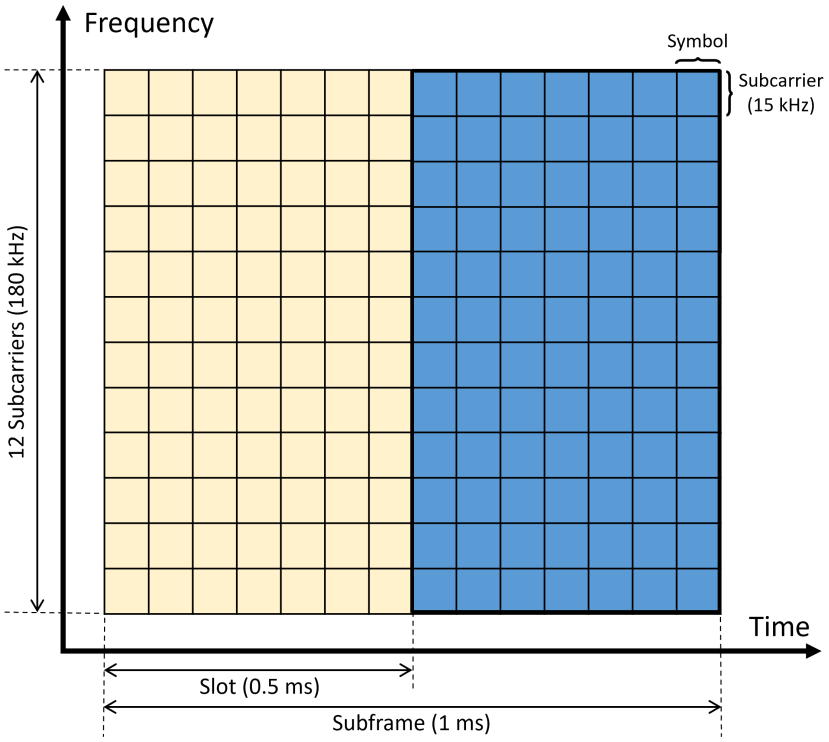

Figure 1.3: Time-frequency grid of PRB in LTE

12 in both LTE and 5G, the bandwidth of these subcarriers varies in 5G, allowing for greater flexibility and optimization in resource allocation. Above RB, PRB defines the smallest unit of resource managed by the network resource scheduler and represents the actual physical allocation of resources in the time and frequency domain grid. PRBs are typically allocated in pairs forming as a Transmission Time Interval (TTI) or a subframe, which consists of two RB adjacent in time and with the same subcarriers, serving as the smallest downlink unit that can be scheduled to a User Equipment (UE).

The significance of network resource scheduler is evident in both LTE and 5G systems. It manages the allocation of PRBs, determining the distribution ratio of PRBs for the successive transmission of various slices to each user across different network services. Thus, a sufficient partitioning strategy for the resource scheduler is vital for ensuring the overall performance and efficiency of the network.

1.2.1 State-of-the-art

As network slicing technology becomes increasingly deployed in services, there is already a sufficient number of research and successful applications focusing on optimizing slicing resource partitioning despite encountering numerous challenges and critical requirements. Recent advancements in slice resource partitioning methods can be broadly categorized into two major groups: mathematical optimization and ML approaches.

a) **Mathematical Optimization:** Mathematical optimization methods, which refer to linear programming, mixed-integer linear programming, and convex optimization, provide the essential solution framework for network slicing. These methods can precisely formulate the optimization problems and yield optimal or near-optimal solutions under the given constraints. Despite their effectiveness, these methods often rely on simplified or approximated models of network systems based on certain assumptions, which can not fully represent the complexities of real-life network systems. Moreover, the computational complexity of these solutions can be prohibitive,

especially in large-scale networks, acquiring significant computational resources for solution derivation and implementation.

b) **Machine Learning (ML):** ML, particularly in subsets of Deep Learning (DL) and RL, has been increasingly recognized for its potential in solving network slicing optimization. These techniques are proven to benefit in predicting traffic patterns, identifying effective slicing strategies, and accommodating dynamic network changes in real time. Specifically, RL has been applied to dynamically adjust resource allocation in response to fluctuation in network demand, thereby enhancing the efficiency and flexibility of network slices. However, traditional RL approaches are not without their challenges, primarily the exploration-exploitation dilemma, which requires a lengthy process of searching through the action space to identify optimal strategies and thus delays the decision-making and adaptation processes.

More importantly, from my literature research experience, it appears that the current proposed methodologies in network slicing resource allocation focus more on the potential impacts and performance improvement of novel approaches within experimental environments. However, there lack of discussions regarding the efficiency of deriving slicing solutions, as well as improving sample efficiency, model reusability, and the generality of solutions that lean toward a one-size-fits-all approach. While many studies claim significant enhancements in network performance, particularly in terms of QoS within virtual or manipulated network environments, they often overlook the challenges associated with complex real systems. Despite the difficulties and limitations in applying immature solutions in under-operational networks, the new proposed approaches should consider the trade-off between novelty and practicality. As an application-oriented research field, works within communication society are expected to bridge the gap between academic study and industrial application, suggesting that proposed methods for network slicing should be designed with real-world implementation in mind.

In the following chapters, from Chapter 5 to Chapter 9, we will conduct a detailed literature review of the proposed solutions. This will facilitate convenient and relevant comparisons of specific resource allocation methods across various slicing scenarios.

1.2.2 Advantages of Reinforcement Learning

RL is a subset of ML, which is dedicated to learning an agent to make sequential decisions by performing dynamic actions under an exclusive environment to maximize defined cumulative rewards. Unlike supervised ML, which relies on labeled data, RL learns and adapts from interactions with the real environment. The advantages of RL over conventional ML approaches can be summarized as following perspectives:

- **Continuous Learning**: RL agents continuously update their policies based on newly collected samples, making them highly adaptable to changing conditions and dynamic environments. RL algorithms can balance exploration by trying new actions to discover their effects on environments and exploitation using known actions that yield high rewards. These features are essential in dynamic environments where optimal solutions are not static and need continuous adjustment and outline the high adaptability of RL in dynamic optimization problems, where the system states change frequently.
- **Sequential Decision Making**: RL is good at solving optimization problems where decisions need to be made sequentially over time. This is particularly useful in dynamic optimization scenarios, where actions taken at a certain time step can affect future states and rewards.

- **Model-free Approach**: As RL learns system dynamics through online interactions, it can operate without a precise environment model, which is advantageous when the environment is complex or not fully understood.

By incorporating neural networks into RL, DRL demonstrates an excellent capacity for managing high system complexity. It uses function approximation with DNN models to handle high-dimensional state and action spaces, thereby facilitating the management of complex systems. Furthermore, Multi-Agent Reinforcement Learning (MARL) distributes the complexity across multiple agents, with each agent learning and optimizing a specific part of the overall problem. Some modern RL implementations employ parallel computing to process multiple interactions simultaneously, accelerating the learning process and efficiently managing large datasets. With these advantages, RL approaches have been applied in the wireless communication field to address several critical aspects such as network traffic prediction, load balancing, energy efficiency, and anomaly detection. Under slicing scenarios, RL can dynamically allocate network resources to different slices based on real-time demands, ensuring optimal resource utilization and maintaining QoS across diverse applications. In this dissertation, we intend to explore RL's capability and drawbacks in solving slice resource allocation problems.

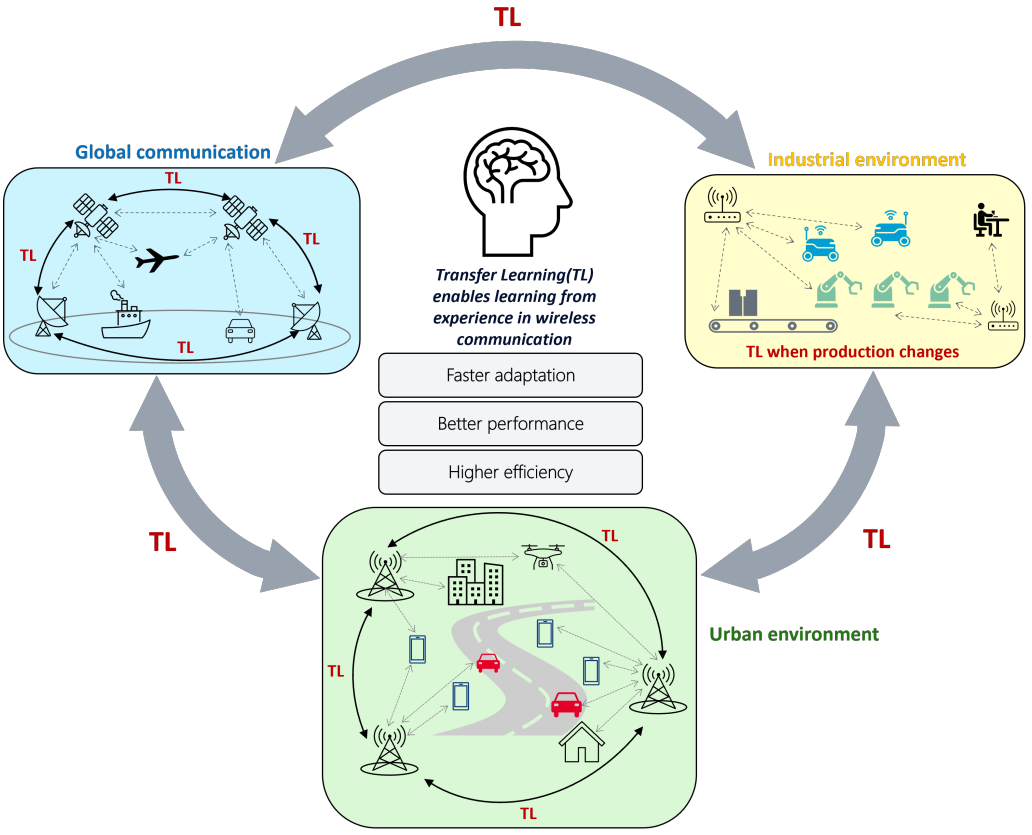

Figure 1.4: Potential application of TL in the field of wireless communications

1.2.3 Advances of Transfer Learning

Considering the limited generalization capability of existing solutions, TL has emerged as a paradigm in communication networks with the advent of 5G. Unlike the traditional methods listed above, TL is dedicated to improving the adaptation of models trained on one domain for application in another, thereby reducing the time and computational consumption required to develop new solutions from scratch. For instance, a model successfully implemented in one network segment can be adapted for use in another, leading to faster and more efficient network management and troubleshooting [7]. Specifically,

in the context of network slicing, TL becomes increasingly important as it broadcasts knowledge between different service scenarios, each with unique bandwidth, latency, and security requirements. This adaptability of TL significantly streamlines the slicing process. When combined with RL approaches, TL allows pre-trained models from one network scenario to be adapted to new scenarios, thereby reducing training time and enhancing performance. This is particularly beneficial in network slicing, where different network configurations necessitate rapid adaptation. In 5G and beyond networking, both network solutions and physical architectures are expected to evolve, further enhancing support for various network services and leveraging network slicing techniques more effectively.

Despite the advantages, TL in communication networks faces challenges such as data limitation, cross-domain applicability, and model generalization. Besides, the discussion of TL methods always appears along with the DL because of the natural generalization ability of neural network models, while for conventional mathematical models, TL plays a rather less important role due to case-specific model derivation. Future research of TL is directed towards overcoming these hurdles by enhancing model transferability, exploring unsupervised or semi-supervised TL approaches to reduce the reliance on labeled application domain data [8].

1.3 Research Objectives and Thesis Outline

1.3.1 Research Objectives

Under the context of network slicing, the motivation of this dissertation is to investigate potential methodologies to generalize network slicing solutions to reduce resource consumption, accelerate deployment progress in real-world applications, and ultimately reduce reliance on human experts when network configuration changes (Fig. 1.5). This research work aims to address the resource allocation challenge within 5G and beyond network environments, evaluating the advantages of TL and examining resource allocation strategies preliminary within the DRL paradigm. Specifically, our objective is to solve the slice allocation problem at the Media Access Control (MAC) layer, providing guidance on resource budgets of every slice for network Operation, Administration, and Maintenance (OAM) in a medium or long term, as illustrated in Fig. 1.6. The RAN resource scheduler will then allocate the PRB resources to each slice accordingly at each snapshot time interval.

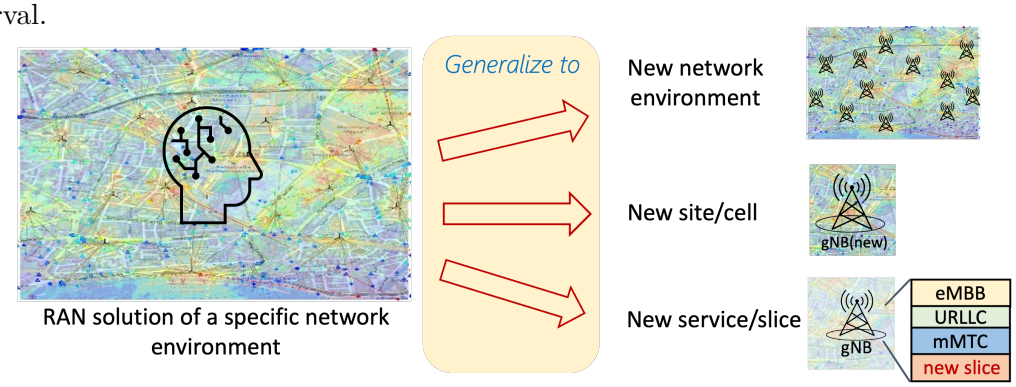

Figure 1.5: Generalization of network solutions

As one of the most classical and crucial challenges in network slicing, solving optimal resource partitioning has high research value and practical significance. Our consideration is that if this problem is well addressed with the successful implementation of TL, the same progress or pipeline could be easily extended and replicated to another application in network slicing, communication networks, or even broader application fields. Secondly, although RL, particularly DRL, has gained great success in the communication society

and other research fields, only a few works discuss applying TL on DRL, neither looks into its generalization ability. Despite the advantages of DRL, such as capturing dynamic environments more effectively than traditional models, it suffers from notable drawbacks, including slow convergence and dependency on large data samples. Thus, this thesis also aims to tackle the inherent challenge of improving the transferability of DRL models, a key obstacle due to the nature of DRL methodologies.

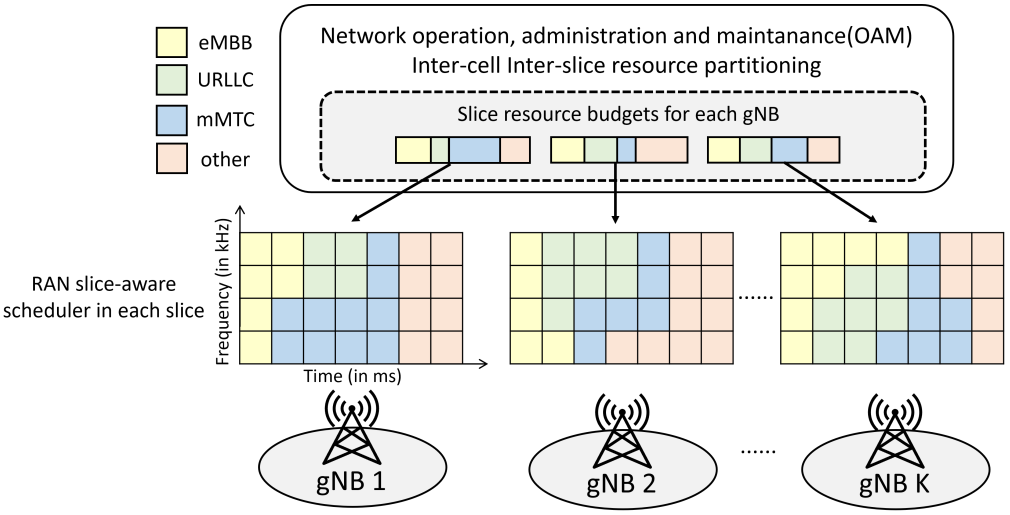

Figure 1.6: Dynamic slicing resource allocation for multi-cell multi-slice network

To fulfill these objectives, the thesis initiates by investigating the feasibility of employing sophisticated DRL algorithms in multi-cell multi-slice network slicing scenarios, solving optimal per-slice source partitioning at medium time scale (e.g., minutes or a quarter of hour) on MAC layer, and facilitating PRB resource in time-frequency domain with networkOAM. This exploration delves into DRL with various solution granularity levels, ranging from centralized approaches to slice-wise methodologies as illustrated by Fig. 1.7. Subsequently, the discussion of the examination of TL techniques is proceeded with the incorporation of DRL. My plan is to assess the DRL and TL on the different function granularity levels:

1. **Centralized Scheme**: The first scheme is solving resource partition optimization in a centralized manner. The centralized solutions leverage a comprehensive view of network systems, facilitating optimal resource distribution and coherent management across the network. It simplifies the task of OAM with coherence and consistency. However, these centralized approaches face scalability challenges as the network expands, becoming potential bottlenecks, particularly when applying TL. Furthermore, the use of larger models in centralized systems may result in increased latency in decision-making and thus impair the system ability to respond to specific local network conditions and demands effectively.

2. **Distributed Scheme:**

   - **Per-cell Scheme**: The distributed approach applies on site-wise or cell-wise granularity, enhancing scalability over the centralized scheme by letting each gNB manage its slices based on local network states. The distributed manner leads to more efficient use of local resources and faster response to local network dynamics. More importantly, it provides higher scalability in adapting solutions among gNBs, which builds a wider stage for applying TL methods. However, this approach might lead to sub-optimal global resource distribution as each gNB operates independently without a comprehensive network perspective, potentially overlooking inter-cell interference effects. Additionally, developing and

maintaining distinct solutions for various gNBs can be complex and resource-intensive, which may cause inconsistencies in resource in resource allocation across different gNBs.

- **Per-slice Scheme**: The distributed approach can be further refined into more detailed granularity, i.e., slice-wise manner. This scheme enables highly customized and flexible slicing orchestration, meeting the diverse requirements of different network slices. It provides the highest generalization ability for TL application. It can be seen as a "one-size-fits-all" approach since its derivation and deployment do not depend on slice configurations within gNBs. It is also ideal for facing dynamic network service setups for 5G and O-RAN. Unfortunately, deriving per-slice solutions is challenging based on limited slice information and dynamics. Coordinating between slices without coupling, such as inter-slice resource constraints and inter-cell dependencies, makes it impossible to derive sufficient solutions under the scope of the entire network system.

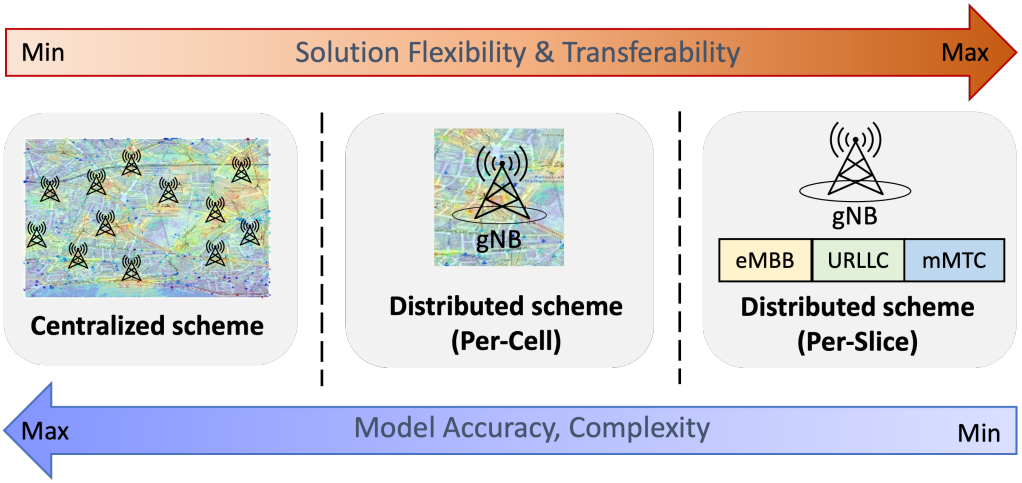

Figure 1.7: Slicing resource allocation on different granularity levels

Additionally, in the context of TL, we assess 2 knowledge transfer manners, aligned with respective DRL application granularity levels:

- **Generalist-to-Specialist**: This method involves transferring knowledge from a general model, trained on samples from a wide range of scenarios or tasks, to a specific model tailored to a particular scenario or task within the network slicing domain.
- **Specialist-to-Specialist**: For the scenarios that are difficult to derive a general model, this alternative approach entails transferring knowledge between models that are each tailored to specific but different scenarios or tasks, enhancing the adaptability of DRL models across similar yet distinct network environments or slice configurations.

In this thesis, we will investigate those topics and methods followed by associated assessments and try to tackle the abovementioned challenges. This thesis is organized in detail by answering the following scheme-aligned questions step-by-step.

Question 1: How to convert slice resource allocation into the form that RL approaches can handle?

Before diving into the research on implementing RL on resource allocation problems under slicing networks, it is necessary to formulate the dynamic optimization under the scope of Markov Decision Process (MDP). It is also vital to properly design the RL rewards that can both facilitate network QoS and resource constraints. Besides, for solving dynamic optimization problems, we need to determine the definition of the utility function considering fairness among slices and cells.

Question 2: How to distribute the centralized resource allocation problem into the per-cell scheme while maintaining the correlation of inter-cell dependency?

Intuitively, RL can handle the resource allocation optimization problem by deriving a global agent that takes care of all network states and slice partitioning actions under the interaction of every cell. While the distributed per-cell scheme is dedicated to resolving this problem within the scope of local observations, despite decreasing the complexity of RL agents, losing inter-cell dependencies may cause severe interference under multi-cell scenarios. Thus, we need to find a way of distributing RL solutions to local cells and recover the inter-dependencies with limited observations.

Question 3: How to proceed TL methods on the top of RL solutions?

For TL methods, it is essential to address the problems of how knowledge transfer proceeds under the given conditions. Different from TL implementation under supervised ML approaches, RL methods have more dynamic characteristics. Thus, in this work, we need to explain knowledge transferring in terms of knowledge type of transferring manners.

Question 4: How to determine the source of knowledge transfer given a target slicing scenario?

Specifically, for "Specialist-to-Specialist" TL, in addition to addressing the ways of transferring, it is also necessary to determine the source domains before knowledge transferring. This is because choosing improper source knowledge may cause negative effects on target domains. A more detailed introduction about negative transfer will be provided in 3.2. In combining with Question 3, these two questions address three fundamental questions for TL implementation, namely, we need to answer the TL queries as what to transfer, how to transfer, and when to transfer.

Question 5: Is TL helpful in enhancing the generality of RL solutions?

After addressing Question 3 and Question 4, it is essential to evaluate the improvement that TL can provide to RL methods for slicing resource allocation to address its efficiency. Therefore, during the experiment, we propose to design the comparison experiments properly to compare its performance with "training from scratch" under the same network settings.

Question 6: How to address the resource constraints under different granularity of solutions?

The inter-slice resource constraints are always retrievable under the fixed slice combinations and correlations for centralized and distributed per-cell schemes. However, under the scope of O-RAN, solutions on the slice level only rely on slice-wise network observations and are required to provide slice partitions accordingly without the information of resource budgets of other slices. After integrating them among slices, this self-interested progress on each slice will lead to invalid resource partitions.

Question 7: How to facilitate slicing solutions under dynamic slice configurations?

Considering the requirements of O-RAN, the slice configurations may vary in real time. Both centralized and per-cell distributed schemes are derived for fixed slicing scenarios with poor adaptability, while theoretically, the per-slice can be applied to arbitrary combinations of slice configurations. Thus, in this work, we aim to implement slice-wise resource allocation solutions under dynamic slicing scenarios.

Question 8: How to improve model generality under domain shift caused by slice configuration changing?

In addition to the dynamic slice configurations under O-RAN, there are cases that face new slice types or network environments where the data or features shift from the original space from which we derived the slice-wise resource allocation solution. To further extend their effectiveness, in this work, we aim to extend the generality of per-slice resource allocation solutions, including investigating the methods of covering features of unseen data and enhancing the model adaptability.

Corresponding answers to these questions will be given along with the discussions in the rest of the dissertation. Specifically, we listed the positions of the answers in the following table:

Table 1.1: Research questions waited to be answered throughout the dissertation

Assessment I: DRL-based Slice Resource Allocation	Chapter 4
Q1: How to convert slice resource allocation into the form that RL can handle?	
Assessment II: Distributed DRL Methods	Chapter 5
Q2: How to distribute the centralized resource allocation problem into the per-cell scheme while maintaining the correlation of inter-cell dependency?	
Assessment III: Generalist-to-Specialist TL-aided RL	Chapter 6
Q3: How to proceed TL methods to the top of RL solutions?	
Q5: Is TL helpful in enhancing the RL methods?	
Assessment IV: Specialist-to-Specialist TL-aided RL	Chapter 7
Q4: How to determine the source of knowledge transfer given a target slicing scenario?	
Assessment V: Per-slice Resource Allocation	Chapter 8
Q6: How to address resource constraints under different granularity of solutions?	
Q7: How to facilitate slicing solutions under dynamic slice configurations?	
Assessment VI: DA of Per-slice Resource Allocation	Chapter 9
Q8: How to improve the model generality under domain shifts caused by slice configuration changing?	

1.3.2 Thesis Outline

I hereby present the dissertation outline and list all the published works associated with this thesis. To give a cohesive discussion with more detailed background information on basic techniques used in this thesis. First, the overviews of RL and TL are given in Chapter 2 and Chapter 3 respectively. At the end of each chapter, literature research on the corresponding technique is provided, addressing the current challenges and future impacts.

In Chapter 4, we first formulate the network slicing resource portion task of network OAM into MDP and state the resource allocation optimization problem. Then, we implement a series of trial experiments in a system-level virtual environment for sanity checks.

The rest of the thesis is organized into 5 main chapters, each corresponding to the discussion of applying RL and TL on one granularity level of resource allocation solution mentioned above.

Specifically, in Chapter 5, we discuss applying a distributed DRL algorithm to multi-cell, multi-slice RAN slicing scenarios to address the dynamic inter-cell slicing resource partitioning problem. This chapter explores transforming a centralized RL approach into a distributed manner while preserving inter-cell dependency information. We introduce a multi-agent DRL approach, exploring two coordination schemes, with and without inter-agent coordination, and evaluate two methods to ensure DRL agents respect resource constraints: reward shaping and decoupled softmax layer embedding.

Then, in Chapter 6, we integrate the proposed distributed DRL approach with TL techniques to achieve higher model reproducibility and sample efficiency. We investigate methods for transferring a general distributed DRL model trained on a large dataset to a specific domain. This chapter also explores two additional objectives: maximizing the minimum service quality across all slices and cells and maximizing the average logarithmic utility over all slices by developing a novel TL-based DIRP algorithm.

We discuss another TL scenario in Chapter 7, which focuses on transferring knowledge from one specific domain to another and requires additional domain similarity analysis. We propose a TL-aided MADRL approach with domain similarity analysis for inter-slice resource partitioning. This includes designing a coordinated MADRL method for inter-cell resource partitioning, where DRL agents share local information to mitigate inter-cell interference, and integrating a TL method to accelerate policy deployment among different agents.

Down to the per-slice scheme, Chapter 8 discusses an integrated method by combining deep learning models with constrained optimization methods. We propose the IDLA algorithm to address resource partitioning challenges in network slicing with assured inter-slice resource constraints. This chapter demonstrates the near-optimal QoS satisfaction and generalization performance of the IDLA algorithm through evaluations in a system-level network simulator.

Finally, on top of the proposed per-slice solution, in Chapter 9, we investigate TL methods for improved model reproducibility and sample efficiency, enhancing the generality of the slice-wise estimator through TL techniques. We implement a DA-based model to replace the traditional Multi-Layer Perceptron (MLP)-based regression, demonstrating its effectiveness in mitigating domain discrepancies and enhancing the accuracy of slicing resource partitioning in diverse network scenarios.

Chapter 10 concludes the thesis along with the potential future research directions by summarizing the key findings and contributions made in the thesis, highlighting the efficacy of RL and TL applications in solving network resource allocation tasks, and discussing their limitations and future impact for further exploration.

1.3.3 Publications in the Context of this Thesis

This thesis is based on the following peer-reviewed publications:

> Inter-Cell Slicing Resource Partitioning via Coordinated Multi-Agent Deep Reinforcement Learning
> *Tianlun Hu, Qi Liao, Qiang Liu, Dan Wellington, Georg Carle*
> ICC 2022 - IEEE International Conference on Communications, 2022, pp. 1-6,
> doi: 10.1109/ICC45855.2022.9838518.

> Knowledge Transfer in Deep Reinforcement Learning for Slice-Aware Mobility Robustness Optimization
> *Qi Liao, Tianlun Hu, Dan Wellington*
> ICC 2022 - IEEE International Conference on Communications, 2022, pp. 1-6,
> doi: 10.1109/ICC45855.2022.9838657.

> Network Slicing via Transfer Learning aided Distributed Deep Reinforcement Learning
> *Tianlun Hu, Qi Liao, Qiang Liu, Georg Carle*
> GLOBECOM 2022 - 2022 IEEE Global Communications Conference, 2022, pp. 1-6,
> doi: 10.1109/GLOBECOM48099.2022.10000763.

> A Joint Industrial-Network Simulator for Leveraging Automation in 5G Private Networks
> *Alessandro Lieto, Stanislaw Strzyz, Patrick Agostini, Tianlun Hu*
> European Wireless 2022; 27th European Wireless Conference, 2022, pp. 1-5.

> Inter-Cell Network Slicing with Transfer Learning Empowered Multi-Agent Deep Reinforcement Learning
> *Tianlun Hu, Qi Liao, Qiang Liu, Georg Carle*
> IEEE Open Journal of the Communications Society, Volume. 4, 2023, pp. 1-15,
> doi: 10.1109/OJCOMS.2023.3273310

> Fast and Scalable Network Slicing by Integrating Deep Learning with Lagrangian Methods
> *Tianlun Hu, Qi Liao, Qiang Liu, Antonio Massaro, Georg Carle*
> GLOBECOM 2023 - 2023 IEEE Global Communications Conference, 2023, pp. 1-6,
> doi: 10.1109/GLOBECOM54140.2023.10436849.

2. Reinforcement Learning

In this dissertation, the proposed methods are preliminarily built under the context of RL. Thus, the detailed introduction from RL basic to recent research is necessary to address the challenges and advantages of applying RL in communication networks. In communication society, most of the audiences possess expert knowledge of basic ML and DL methods but may not be familiar with RL. This chapter is organized especially for readers with ML background but not versed RL experience.

The rest of the chapter is organized as follows: In Section 2.1, we introduce the basic concept of RL with precise experiments, followed by discussions on how it differs from traditional ML methods. Section 2.2 covers the detailed foundational knowledge and key elements in RL, from MDP formulations to the Bellman equation, and extends to DRL. Policy-oriented and value-oriented RL are discussed independently in separate sections. Starting from Section 2.3, we introduce successfully implemented classic RL and DRL algorithms. Specifically, in Section 2.4, we outline the Policy Gradient (PG)-based DRL approaches and their advancements. Finally, in Section 2.5, we discuss the DDPG and TD3 algorithms, which provide the fundamental pipeline for the proposed methods in this dissertation.

2.1 Introdution to RL

In 1904, Russian physiologist Ivan Pavlov conducted an experiment to study the concept of the conditioned reflex. In the experiment, Pavlov used a bell as the neutral stimulus. Whenever he gave food to his dogs, he also rang the bell. Initially, the dogs did not respond to the bell since it was irrelevant to them. However, dogs salivate when they see or smell food, a natural response to food, known as an unconditioned response to an unconditioned stimulus. After repeating this procedure multiple times, the dogs started associating the sound of the bell with the food. Eventually, Pavlov observed that the dogs began to salivate merely at the sound of the bell, even when no food was presented. This salivation in response to the bell is conditioned, demonstrating that a new behavior could be learned through association. This famous experiment was initially designed to illustrate the process of classical conditioning and is regarded as the kick-off research in behaviorism and learning theories in psychology.

About 120 years later, Neuralink, a company started by Elon Mask, processed another experiment involving a monkey named Pager. The company implanted a chip into the monkey's brain as a brain-computer interface, allowing the monkey to control a computer

game, specifically Pong, through Bluetooth without needing a physical controller. This no-touching interaction was enabled by decoding the monkey's neuronal activity and predicting the signals the brain would typically send to the hands to move them, essentially allowing the monkey to play the game with its mind. As a reward for playing, Pager received sips of a milkshake. The experiment aims to demonstrate the potential of enabling direct brain-to-computer communication.

Despite the above two cross-century experiments having different research goals, we can still see a consistent framework when studying an "intelligent agent," namely conditioning response and real-time interaction.

2.1.1 Concepts in RL

RL plays an important role in the field of ML, where an "agent" learns to make decisions by interacting with its environment to achieve a particular goal. The principles of RL can also be described by classical conditioning, as exemplified by the above two experiments, which underscore the foundational elements of RL, "state" ($s \in \mathcal{S}$), "action" ($a \in \mathcal{A}$), "reward" ($r \in \mathbb{R}$), and "policy" ($\pi \in \Pi$). As Fig. 2.1 illustrates, all these elements are intricately linked to the concept of an intelligent agent, which can be regarded as the dog in Pavlov's experiments or the monkey in Neuralink scenario.

- **State:** In RL, a state represents the current situation within an environment. It represents a snapshot of all the information necessary to describe the context in which decisions are made. The state space encompasses all possible scenarios that might occur. Specifically, in Pavlov's experiment, the state is defined by environmental stimuli, such as the presence or absence of food and the sound of a bell. Similarly, in Neuralink experiment, a state represents the current condition within the game, such as the position of the paddle, the ball, and their velocities. From Pager's perspective, it could also stand for the neural signals representing these game elements.

- **Action:** Actions are the set of all possible moves or decisions the agent can make. In each state, the agent selects an action based on a policy. The action space can vary dramatically depending on the problem - from moving in a direction in a maze to choosing a stock to invest in a financial application. In Pavlov's setup, the dog's actions are physiological responses, like salivating in reaction to stimuli. In the Neuralink scenario, the action takes the form of imagined paddle movements, interpreted from the monkey brain signals. Despite the physical inactivity, these mental decisions align with the RL concept of actions, as they directly influence the outcomes in the experimental environments.

- **Reward:** After taking an action, the agent receives a reward from the environment, which is a signal reflecting the immediate benefit of the previous action. Rewards are crucial as they guide the learning process. The goal of the agent is typically to maximize the total amount of reward it receives over time. In RL, rewards guide the agent by indicating the desirability of an action taken in a particular state. Pavlov's dogs received physical food as a reward, reinforcing the connection between the bell and the expected outcome. In the Neuralink experiment, rewards are provided through positive reinforcement, such as successfully controlling the game or receiving a milkshake, guiding the agent (in this case, Pager) towards beneficial behaviors. This reward structure is foundational in RL, shaping the agent strategy to maximize these positive outcomes.

- **Policy:** RL policy refers to the strategy used by the agent to decide the actions to take based on the current state. It can be seen as a mapping from states to actions. The

objective of RL is to find an optimal policy that maximizes the cumulative reward of the agent over time. Through repeated trials, the dog in Pavlov's experiment developed a conditioned reflex, effectively a policy linking the sound of the bell with the anticipation of food. In the Neuralink experiment, the policy that Pager follows evolves through interaction and feedback, learning to manipulate the game effectively with brain activity alone. This adaptation showcases the learning and optimization at the heart of RL, where the objective is continually refining the policy to achieve maximal rewards over time.

Above all these elements, the "agent" stands as the central concept of RL that represents the entity or "intelligence" that learns and makes decisions based on its interactions with the environment. The role of the agent is multifaceted: First, the agent interacts with the environment, which can be anything from a physical space, such as a maze or a real-world environment. The agent gathers information through these interactions and experiences different states and outcomes. Then, based on the RL state, the agent makes decisions based on its current policy. It chooses the optimal action based on its current knowledge in response to the state and attempts to achieve its goals by maximizing the long-term cumulative reward. The agent continuously learns from the outcomes of its actions. Through a process of trial and error, it updates its knowledge or strategy, referred to as the policy, based on the feedback received in the form of rewards or penalties. Besides, the agent also evaluates the consequences of its actions based on the rewards received from the environment. This evaluation helps the agent refine its policy to make better decisions in the future.

The interaction between the agent and these elements underpins the learning process in RL. In general, the objective of RL is to develop an optimal policy that allows the agent to maximize the total reward it receives over the long run. The agent learns from the consequences of its actions, which are evaluated by rewards, and adjusts its policy accordingly to improve future decisions and outcomes. This dynamic process of action, observation, and adjustment continues until the agent achieves its goals or until the policy converges to the optimal strategy. A typical RL process can be described as a step-by-step process in a loop based on common principles (Fig. 2.1):

1. **Initialization:** The RL process begins with the initialization of the environment and the agent. The agent is typically unaware of the environment's dynamics and has no predefined policy. In practical implementation, the agent always starts with a random or selected policy.
2. **Interaction:**
   (1) **Observation:** The agent observes the current state of the environment. This state could represent various elements depending on the problem, such as the position of a player in a game, the current market conditions in trading, or the readings from sensors in robotics.
   (2) **Taking action:** Based on its current policy (which could initially be random), the agent selects an action to perform in the environment. The policy is a function that maps states to actions.
   (3) **State transition:** The agent executes the chosen action in the environment, which leads to a change or transition to a new state.
   (4) **Receiving reward:** After the action is performed, the agent receives a reward from the environment. This reward is a signal that indicates the success or failure of the action with respect to the objective.

(5) **Policy updating:** The agent uses the reward it received and the transition information (from the old state to the new state) to update its policy. This learning process is aimed at improving the policy over time.

3. **Repetition:** The steps in 2 are repeated for many episodes or until a termination condition is met, e.g., a certain level of performance is achieved, or a maximum number of steps is reached.

4. **Policy Improvement:** Through repeated interactions with the environment and learning from the outcomes, the agent policy improves over time. Ideally, the agent learns to make better decisions that maximize the cumulative rewards it receives.

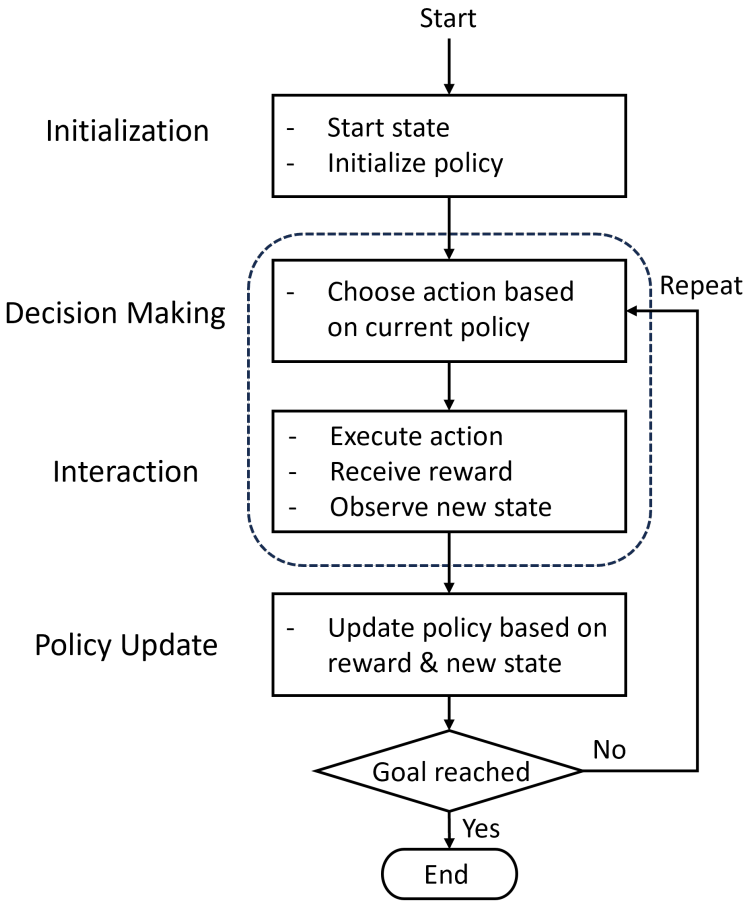

Figure 2.1: General process of RL

2.1.2 Comparing with Conventional ML

From my experience, most applications of ML methods in communication fields are primarily built on traditional ML, especially Supervised Learning (SL) approaches. Thus, it is necessary to address the differences between RL and traditional ML methods, discuss their advantages and weaknesses, and explain why RL is chosen as the basic pipeline in this thesis. This comparison also provides a more intuitive understanding of RL for readers familiar with ML.

Overall, traditional ML is primarily concerned with learning from a dataset to make predictions, while RL focuses on learning an optimal strategy through interactions and feedback. RL is suited for problems where an agent must make a series of decisions to achieve a goal, especially in uncertain and dynamically changing environments. Meanwhile, SL is more applicable when there is a clear set of instances and associated labels to learn from. The main differences between RL and SL can be addressed from the following perspectives:

- **Training Data:**
  - **SL:** The training data consists of input-output pairs, where the input data are always assumed to be Independent and Identically Distributed (i.i.d). The data is labeled, meaning that each input is associated with the correct output. The learning algorithm uses this dataset to learn a model that can make predictions for new, unseen data.
  - **RL:** There is no predefined dataset. Instead, the data is generated through the interactions between the agent and the environment. The data may have strong coherent relations, as previous outputs can affect future inputs. The agent learns from the consequences of its actions, guided by a reward system rather than direct instruction.
- **Decision Making Process:**
  - **SL:** The decision-making process involves making predictions or classifications based on the learned model. Once trained, the model applies what it has learned to new data to predict an outcome.
  - **RL:** The decision-making process is sequential, based on the selection of actions that maximize cumulative reward as interactions continue. The agent decides on actions based on a policy, which evolves through trial and error and feedback from the environment.
- **Ground Truth:**
  - **SL:** Operates on the premise of ground truth, meaning the correct answers (labels) are provided during training, and the performance of the model can be evaluated against this ground truth.
  - **RL:** Does not have ground truth in the conventional sense. Instead, the "correctness" of an action is determined by the rewards received and how they align with the long-term goal of maximizing cumulative reward.
- **Feedback Mechanism:**
  - **SL:** Feedback comes in the form of the accuracy of the predictions compared to the known labels in the training data. This feedback is direct and immediate.
  - **RL:** Feedback is given through rewards and penalties. It is more indirect as the agent learns which actions yield the highest total reward over time.

Unlike traditional ML, RL steps out of the "data collection-model training" loop, introducing more dynamic characteristics in deriving ML models. Therefore, RL is distinctively suited for tasks that require sequential decision-making, learning from interaction, and adaptation in dynamic environments. These tasks include game playing, robotics, finance, healthcare, and resource management. Mobile networks are complex systems with numerous dynamic features, and the slicing resource management task requires continuous and coherent resource partitions across different time intervals. Thus, RL is inherently suitable for solving slicing resource optimization problems in communication networks.

2.2 RL Basics

In this section, we formally introduce the foundational methods and algorithms of RL and explain its basic principles from a mathematical perspective. Central to RL is the concept of the MDP, which serves as the standard framework for formulating decision-making agent

models. This section starts with an introduction to the fundamental aspects of MDP, moves to discussions on the Bellman equation, a critical component in understanding the dynamics of RL, and finally explores two principal iterative approaches employed in RL: value-based iteration and policy-based iteration, which provide mechanisms for optimizing the decision-making process.

2.2.1 Markov Decision Process (MDP)

MDP is a mathematical framework used for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker. MDPs are extensively used in RL by providing a formalization of agent behavior in dynamic systems to help agents learn how to behave in an environment to achieve a goal.

An MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$, each element representing a fundamental component of the decision-making model:

- **States** ($\mathcal{S}$): The set of all possible states in which the agent can exist. A state $s \in \mathcal{S}$ encapsulates all the information the agent needs to make decisions. In an MDP, it is assumed that the future depends only on the current state and the action taken, not on the sequence of events preceding it. This property is known as the Markov property.
- **Actions** ($\mathcal{A}$): The set of all actions available to the agent. In any given state s, the agent can choose from a set of possible actions. The action taken by the agent can lead to a change in the state, transitioning the agent from one state to another.
- **Transition Probability** (P): The probability that taking action a at state s will lead to the next state $s' \in \mathcal{S}$. This is often denoted as $P(s'|s, a)$, where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. The transition probability reflects the dynamics of the environment, indicating how likely it is to move between states under specific actions.
- **Reward Function** (r): The reward received after transitioning from state s to the next state s' by executing action a, denoted as $r(s, a)$ with $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. It measures the immediate benefit to the agent of taking action a at state s. The reward function guides the agent by indicating which actions are beneficial and which are not.
- **Discount Factor** (γ): A factor with $\gamma \in [0, 1]$ that represents the difference in importance between future rewards and present rewards. A discount factor of 0 makes the agent "short-sighted" by considering only immediate rewards, while a discount factor near 1 makes it "far-sighted" by valuing future rewards almost as highly as immediate rewards.

The goal within an MDP framework is typically to find a policy π, a strategy defining the action that the agent should choose when in a specific state to maximize some notion of cumulative reward. The policy maps states to actions and is central to the behavior of agent.

An Example

Imagine a robot trying to find its way out of a maze, similar to what Fig. 2.1 illustrates. Each position in the maze can be regarded as a state s, and at each position, the robot can choose to move in different directions: up, down, left, or right (as the choice of actions, a). Some moves take the robot closer to the exit or lead to dead ends, corresponding to reward or penalty, measured by r. The objective of RL agent is to maximize the total

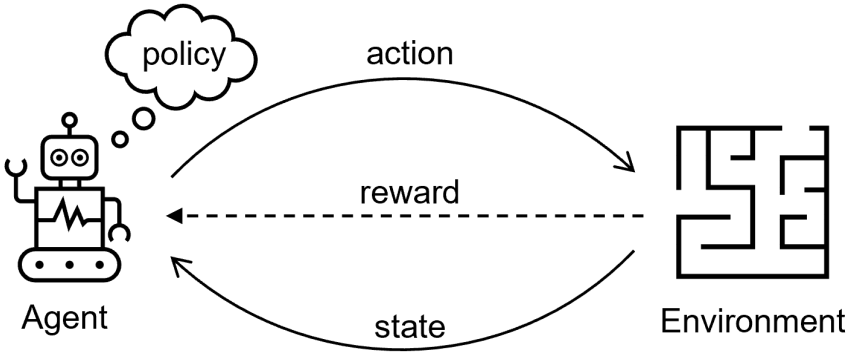

Figure 2.2: A RL example: robot navigation in maze

reward the robot receives, which in this case could be finding the fastest route out, i.e., the optimal policy π.

Before converting this procedure into an MDP process, let us focus on the factors determining whether the robot can eventually find the most effective path out of the maze. The most critical factors are the state s_t (position) and the choice of action a_t (the forward direction) at each step t. The state-action process of the robot in the maze can be described as $s_0, a_0, s_1, a_1, \ldots, s_{t-1}, a_{t-1}, s_t$. In this process, we can see two types of transitions: one from state to action and the other from action to state.

- **State-Action Transition:** The transition from state to action reflects the decision-making function of the agent, which is to select the best action (forward direction) based on the current location of the robot. This is the agent policy π, that is, the mapping from the state domain to the action domain or to the probability distribution of the action. In the transition from state to action, the agent weighs the consequences of each possible action in the current state based on the learned policy π and finally selects the action with the highest evaluation. Mathematically, a policy can be expressed in several ways, depending on whether it is deterministic or stochastic.

  - **Deterministic Policy:** A deterministic policy is a direct mapping from states to actions, i.e., under he current state s_t, the policy π selects a specific action a_t with:

    $$a_t = \pi(\{s_0, ..., s_{t-1}, a_{t-1}, s_t\}). \tag{2.1}$$

    For the current state $s_t \in \mathcal{S}$, the policy π dictates exactly one action $a_t \in \mathcal{A}$ to be taken.

  - **Stochastic Policy:** A stochastic policy, on the other hand, provides probabilities for taking each possible action in a given state. It can be represented as:

    $$a_t = \arg\max_{a_t^i} \; p_\pi(a_t^i | \{s_0, ..., s_{t-1}, a_{t-1}, s_t\}). \tag{2.2}$$

    This means that for state s_t, the policy π gives the probability of choosing action a_t from the action space.

  We can see that the description of the current state includes all state-action transitions before reaching the current state s_t. But, according to the Markov property, for a MDP process, the action a_t to be taken is only related to the state s_t at the current step t, and the previous conversion process can be represented by state s_t. Therefore, Eq. 2.1 and Eq. 2.2 can be simplified as $\pi(s_t)$ for deterministic policies and $\arg\max a_t^i \; p\pi(a_t^i | s_t)$ for stochastic policies.

- **Action-State Transition**: The environment generally determines the transition from action to state, i.e., the transition probability $p(s_{t+1}|s_t, a_t)$. The environment objectively determines the results of the actions the agent decides. The best policy obtained by the agent can generally only be applied in a specific or the same environment. In subsequent sections of this dissertation, we will provide a more comprehensive discussion of this limitation and explore the strategies we have implemented to address this challenge.

An MDP process can be regarded as alternating these two transitions. For RL, the difference is that it needs to explore and learn the best policy π through interactions, usually termed as exploration and exploitation phases. This process is essential for steering the MDP towards desirable outcomes, underscoring the necessity for rewards. Mathematically, the reward r of transitions defined with a reward function R can be expressed as:

$$r_t = R(s_t, a_t, s_{t+1}), \tag{2.3}$$

which signifies the reward received after transitioning from state s_t to state s_{t+1} as a consequence of executing action a_t.

The reward function serves as a signal that reinforces desirable actions while discouraging undesirable ones. By associating positive rewards with beneficial actions and negative rewards with harmful ones, the agent learns to navigate the environment to maximize cumulative rewards. It defines the goal within the MDP framework. The objective of RL is to maximize the total accumulated reward over time, aligning its behavior with the overarching goals of the task. Besides the reward that implies the immediate benefit, the discount factor γ plays a crucial role in MDPs by affecting how future rewards are valued. This ensures practical and realistic decision-making in uncertain environments. The rationales for applying the discount factor γ are multifaceted:

- **Present Value of Future Rewards:** The discount factor ensures that immediate rewards are generally valued more than distant future rewards, modeling the uncertainty and diminishing value of future outcomes in decision-making processes.
- **Convergence and Stability:** Incorporating a discount factor helps ensure the convergence of value functions and policies, preventing the total accumulated reward from becoming infinitely large, which can destabilize learning algorithms.
- **Temporal Preferences:** The discount factor allows for modeling preferences over time. A lower γ makes the agent short-sighted, prioritizing immediate rewards, whereas a higher γ makes the agent more far-sighted, considering the long-term consequences of its actions.
- **Decision Making in Uncertain Environments:** In many real-world scenarios, the future is uncertain, and immediate rewards are a more reliable indicator of sound policy. The discount factor helps balance exploring new strategies with exploiting known rewarding strategies, reflecting real-world decision-making challenges.

Based on the definitions above, this robot-maze process illustrated by Fig. 2.2 can be exemplified by repeating the following steps under the framework of an MDP:

1. **Defining the MDP Components:**

- $\mathcal{S}$: Each position in the maze represents a state. For simplicity, imagine that the maze is on a grid, and each cell on the grid is a potential state the robot can occupy. Special states might include the starting position and the goal position.
- $\mathcal{A}$: At each position (state), the robot has a set of possible actions. Typically, these actions would be to move north, south, east, or west to adjacent cells.
- P: It describes the likelihood of moving from one state to another under the given action. In a perfect world, if the robot decides to move north, it will always move north. However, in a more realistic scenario, there might be a probability distribution due to slippery floors or obstacles, e.g., there is a 90% chance that moving north will succeed and a 10% chance the robot stays in place.
- r: After each move, the robot receives a reward. Moving closer to the exit yields positive rewards, while hitting the wall or moving away from the exit incurs negative rewards. Reaching the exit could give a significant positive reward and end a transition epoch.
- γ: This factor determines the importance of future rewards. A value close to 1 would make future rewards very significant, while a value close to 0 would make the robot short-sighted, caring only about immediate rewards.

2. **Navigating the Maze:** The robot starts in the initial state, the starting point of the maze, and decides the best action to take based on its policy. Initially, if the robot has no prior knowledge, it will choose actions randomly. As the robot moves through the maze, it accumulates rewards based on the distances moved toward or away from the goal. It begins to learn which actions in which states lead to higher rewards.

3. **Learning and Policy Improvement:** Using algorithms such as value iteration or policy iteration, the robot updates its policy, i.e., the strategy of choosing actions in different states to maximize total future rewards. The policy of robot gets refined through iteration:

   - **Value Iteration:** The robot iteratively updates the expected cumulative reward of each state under the current policy until the values stabilize. This helps the robot indirectly improve its policy.
   - **Policy Iteration:** The robot evaluates the current policy by implementing it during interactions, then improves the policy by choosing actions that lead to the best subsequent states, and repeats this process.

4. **Reaching the Goal:** After sufficient learning, the robot eventually develops an optimal policy indicating the best action from any state. Following this policy, the robot navigates the maze efficiently, reaching the goal while maximizing its accumulated reward.

This toy scenario exemplifies how MDPs can be applied to real-life problems where decision-making is sequential and the environment is uncertain. The MDP components: states, actions, transitions, rewards, and the discount factor, all play crucial roles in this learning and decision-making process.

2.2.2 Value Function

In the example above, we can find that in order to achieve the final goal of getting out of the maze, the robot needs to consider choosing actions that are more consistent with long-term returns for subsequent state actions when making every action decision. In RL,

for any action in each state, there is a corresponding value to quantify the contribution of this action to the final goal. Based on this value, the RL agent can make optimal action decisions under different states. This is called the value function V, which reflects the total amount of reward an agent can expect to accumulate over time, starting from that state.

In fact, in RL definitions, the instant reward r can already represent the simultaneous value brought by taking an action a in a particular state s. However, this r only reflects the return in the local state and is not the final goal of RL. For example, for the robot in the maze, if only the local reward of each position is considered, the final result obtained after a limited training round may only be a policy that can avoid hitting the wall. Such a policy makes it difficult to guarantee that the robot can be guided out of the maze in every instance. Therefore, we need to introduce the concept of long-term reward, which is, in addition to considering the simultaneous return of the action, we must also consider the value that can be obtained in the next state after the action occurs. The objective of RL to derive the optimal policy π^* is identical to maximizing $\sum_t r_t$.

However, if we simply add the possible value in the future state to the local value, the accumulated reward may lead to divergent results. Therefore, RL agents utilize the discount factor γ to mitigate the effects of the future return in the current state, and the revised return is given as $\sum_t \gamma^t r_t$. In 2.2.1, we have provided the rationales of using discount factor in MDP such as avoiding infinite returns in cyclic MDP and uncertainty about the future rewards. Here, we complete the definition of long-term RL reward (also refer as return) concerning discount factor γ as:

$$G_t := r_{t+1} + \gamma r_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.4}$$

In the context of RL, the value function is a fundamental concept used to estimate how good it is for an agent to be in a given state or perform a specific action in a given state. Essentially, the value function measures the expected long-term reward for different states and actions, guiding the agent's decision-making process. There are primarily two types of value functions: the state value function and the action value function.

State Value Function The state value function, denoted as $v(s)$, represents the expected total reward an agent can expect to accumulate over the future, starting from state s and following a particular policy π. The policy π dictates the behavior of the agent by defining which action to take in each state. The state value function is expressed as:

$$v_\pi(s) = E_\pi[G_t|S_t = s], \tag{2.5}$$

where E_π denotes the expected value when the agent follows policy π.

Action Value Function The action value function, denoted as $q(s, a)$, estimates the expected return of taking action a in state s and following policy π. It tells us how good it is to perform a particular action from a particular state. The action value function is defined as:

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a], \tag{2.6}$$

In comparison with the state value function $v_\pi(s)$, the action value function $q_\pi(s, a)$, also known as the state-action value, reflects both the contributions of current state s and action a to the RL agent.

Both value functions are crucial in RL as they encode the knowledge about the best ways to act in different situations. They guide the agent towards more profitable behaviors.

The objective of RL can also be expressed mathematically through these value functions, which is to find the optimal policy π^* that maximizes the value function of certain states as:

$$\pi^* = \arg\max_{\pi} v_\pi(s), \forall s, \tag{2.7}$$

and for any action at state S, find the optimal action that can maximize its action value function with

$$a^* = \arg\max_{a} q_{\pi^*}(s, a). \tag{2.8}$$

With these objectives, an optimal policy can often be derived directly from the value functions by choosing the action with the highest expected return in each state. Over time, as the agent learns more about the environment, the value functions converge towards the actual values, allowing the agent to make increasingly better decisions and ideally reach optimal behavior.

2.2.3 Bellman equation

The Bellman equation plays an important role in calculating the best policy in RL based on value functions. It provides a recursive solution to find the value of each state under a particular policy. The value function V reflects the total amount of reward an agent can expect to accumulate over time, starting from that state.

Given $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, which denotes the total expected return from time step t where r_{t+k+1} represents the reward received at time step $t + k + 1$, the value function Eq. 2.5 can break down the total expected return G_t into immediate reward plus future rewards:

$$\begin{aligned} v_\pi(s) =& E_\pi[G_t | S_t = s] && (2.9)\\ =& E_\pi[r_{t+1} + \gamma\ v_\pi(S_{t+1}) | S_t = s]. && (2.10) \end{aligned}$$

Expanding based on the policy π, which gives the probability of taking action a at s, and the Markov property (i.e., the future is independent of the past given the present), we obtain:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a)[r + \gamma\ v_\pi(s')], \tag{2.11}$$

where s' represents the potential next state resulting from taking action a at the current state s.

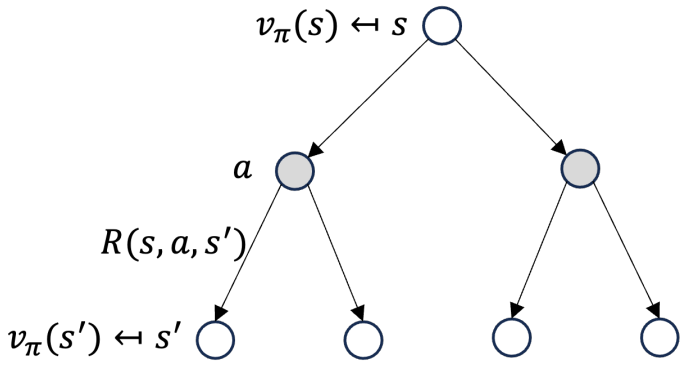

Figure 2.3: MDP with state value function

This is called the Bellman expectation equation for $v_\pi(s)$, encapsulating the recursive relationship between the value of a state and the expected values of the next states, weighted by the transition probabilities and policy.

Similarly, the action value $q_\pi(s, a)$ of taking action a in state s under policy π follows:

$$q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a] \tag{2.12}$$
$$= E_\pi[r_{t+1} + \gamma\ q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \tag{2.13}$$

Expanding this, considering the outcomes of action a, yields:

$$q_\pi(s, a) = \sum_{s'} p(s'|s, a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')]. \tag{2.14}$$

This is the Bellman expectation equation for $q_\pi(s, a)$, illustrating how the action-value function depends on the expected reward for the action and the value of the subsequent state-action pairs.

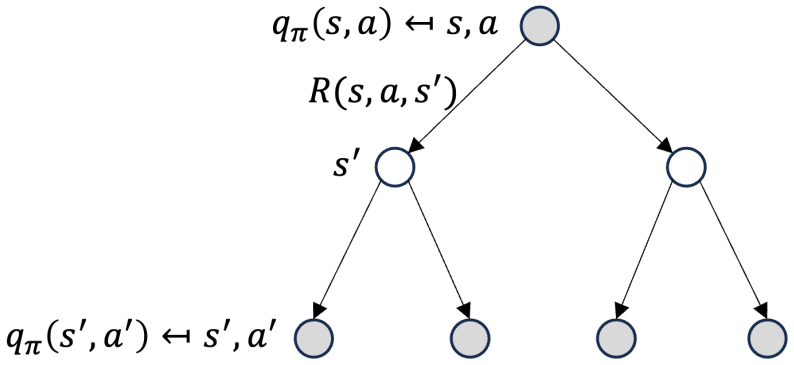

Figure 2.4: MDP process with action value function

Fig. 2.3 shows the MDP process starting from state s and transitioning to the next state s' under the guidance of policy π, as well as the value function of the corresponding state node. Fig. 2.4 represents the state-action function similarly.

In RL, we often seek the optimal value function, which gives the maximum value attainable under any policy:

- **Optimal state value function:**

$$v^*(s) = \max_\pi v_\pi(s) \tag{2.15}$$

- **Optimal action value function:**

$$q^*(s, a) = \max_\pi q_\pi(s, a) \tag{2.16}$$

Using the Bellman equations, we can express these optimal functions without reference to a specific policy:

$$v^*(s) = \max_a \left[\sum_{s'} p(s'|s, a)[r + \gamma\ v^*(s')] \right], \tag{2.17}$$

$$q^*(s, a) = \sum_{s'} p(s'|s, a)[r + \gamma\ \max_{a'} q^*(s', a')]. \tag{2.18}$$

These equations provide a way to compute the optimal strategies for decision-making. Based on the recursive nature of the Bellman equation, it might appear straightforward to solve the value function through iterative methods. However, in practice, we initially lack value functions for states or actions following initialization. Furthermore, solving the value function for a specific state requires knowledge of other value functions. This poses the question: how do we effectively address this challenge? In the rest part of this section, we will cover the development of algorithms such as Value Iteration and Policy Iteration that approximate the optimal value functions.

2.2.4 Policy-based Iteration

Policy-based iteration, or policy iteration, is a fundamental method in RL for finding an optimal policy. This technique iteratively evaluates and improves a policy until it converges to the optimal policy evaluated by value functions. As Eq. 2.7 and Eq. 2.8 illustrate, finding the optimal policy requires simultaneous updates of the policy and value function, which are closely interdependent. This interdependence creates a circular causation dilemma: obtaining the optimal strategy requires precise state and action value functions, yet accurately estimating these value functions necessitates a sufficiently effective strategy. To address this challenge, iterative optimization is typically employed to progressively converge toward the optimal outcome. The policy-based iteration method can be briefly summarized as follows:

1. **Initialization:** Start from the initialized policy $\pi^{(0)}$ and calculate value functions under the current policy.
2. **Policy Evaluation:** Assess the value functions under the current policy.
3. **Policy Improvement:** Improve the policy based on the evaluated value function.
4. **Iteration:** Repeat steps 2 and 3 until convergence or the iteration limit is reached.

Specifically, policy iteration uses the Bellman equation Eq. 2.11 to solve the state value function based on the current policy under the assumption that the value functions on both sides of the equation are at the same time index. In the policy iteration process, new value functions are obtained iteratively with two steps, termed as policy evaluation Eq. 2.19 and policy improvement Eq. 2.21:

- **Policy Evaluation**: Assess the value functions under the current policy;
- **Policy Improvement**: Improve the policy based on the evaluated value function.

At iteration step k, the Bellman equation of the value function $v^{(k)}$ for evaluating current policy $\pi^{(k)}$ is

$$v^{(k+1)} = \sum_a \pi^{(k)}(a|s) \sum_{s'} p(s'|s,a)[r + \gamma\ v^{(k)}(s')], \tag{2.19}$$

while the action-state value functions are calculated with

$$q^{(k)} = \sum_{s'} p(s'|s,a)[r + \gamma v^{(k)(s')}], \tag{2.20}$$

and $q^{(k+1)}$ can be determined respect to Eq. 2.14. Finally, the policy is updated based on new action-state values:

$$\pi^{(k+1)}(a|s) = \arg\max_a q^{(k+1)}(s,a). \tag{2.21}$$

Due to the discount factor γ, the state values tend to converge. The convergence properties of this procedure have been demonstrated in several studies [9, 10]. Each iteration of the policy results in an improvement, implying that a specific policy is encountered only once throughout the process. Given the finite number of possible policies, which is equal to the product of the number of actions and the number of states, the process is bound to converge after iterating through all potential policies. By the very definition of convergence, the policy remains unchanged between successive iterations at convergence, such that $\pi_{k+1}(s) = \pi_k(s)$ for all states s. This stability implies that the policy satisfies the Bellman equation, thereby confirming that it corresponds to the optimal value function.

2.2.5 Value-based Iteration

From the policy above iteration process, we can see that the current policy is evaluated by solving the value function, and subsequently, the policy is updated based on this evaluation. This process is relatively quick for problems with small state and action spaces. However, as the complexity of the problem increases, policy iteration becomes less efficient. Therefore, the value-based iteration process, also known as value iteration, focuses directly on finding the optimal value function from which the optimal policy can be straightforwardly determined.

The value iteration process is built upon the principle of dynamically updating the value function v until it converges to the optimal v^*. Similarly, this process iteratively applies the Bellman equation like policy iteration but aims to find the maximum value achievable for each state given optimal actions, while the optimal policy π^* is determined after deriving the optimal state values. The procedure of value iteration can be illustrated as follows:

1. **Initialization:** Begin with an arbitrary value function $v^{(0)}(s)$ for all states s.
2. **Value Function Update:** Update the value function for each state using the Bellman equation:
$$v^{(k+1)} = \max_a \sum_{s'} p(s'|s,a)[r + \gamma\ v^{(k)}(s')], \tag{2.22}$$
which refers to the action value function as $v(s) \leftarrow \max_a q(s,a)$.
3. **Convergence Check:** Repeat step 2 until the changes in the value function between iterations are below a predefined threshold, indicating convergence.
4. **Policy Extraction:** Once $v^*(s)$ is determined, the optimal policy $\pi^*(s)$ can be extracted by choosing the action that maximizes the expected return for each state with
$$\pi^*(s) = \arg\max_a \sum_{s'} p(s'|s,a)[r + \gamma\ v^*(s)]. \tag{2.23}$$

In comparison to the policy iteration process, value iteration is typically more direct and computationally efficient in terms of convergence speed, as it updates values for all states in a sweeping manner using the Bellman equation. It requires no separate policy evaluation and improvement phases. The algorithm converges directly towards the optimal value function without needing multiple policy evaluations, making it generally faster, especially when the state and action spaces are large. Nevertheless, policy iteration often requires fewer iterations to converge compared to value iteration, as each iteration typically results in significant policy improvement. However, each iteration is computationally costly since it involves solving the entire system of equations to find the correct value function for the current policy.

Both value-based and policy-based iterations are powerful methods for determining optimal policies in RL. The choice between these methods often depends on the specific characteristics of the problem domain, such as the size of the state and action spaces and the computational resources available. Starting from the next section, we will introduce classical RL algorithms built upon policy and value iteration, respectively, and cover the state-of-the-art advancements to further illustrate the effectiveness of these methods.

2.3 Fundamental RL Approaches

In this section, we will delve deeper into RL in more practical contexts. Previously, it was assumed that the transition probabilities $p(s'|s,a)$ were known at the beginning. For

instance, in the robot and maze example, we operated under a "god view" of the maze, precisely knowing the outcome of each action taken by the robot, referring to $p(s'|s,a)$, which can be regarded as a model-based RL problem. In reality, however, this scenario is often referred to as a black box, where prior knowledge of $p(s'|s,a)$ is absent, typically referred to as a model-free problem. In RL, the key distinction between model-based and model-free approaches lies in whether an environmental model that includes assumptions or knowledge about the dynamics, such as transition probabilities and rewards, is used [11].

- **Model-based RL**: Model-based RL algorithms utilize an explicit environmental model that predicts the next state and reward following a specific action in a given state. These methods necessitate understanding environment dynamics, which can be previously known or learned through interaction. They often involve a planning phase to simulate the outcomes of various actions to determine the most advantageous approach for environment system modeling. Model-based approaches are typically more sample-efficient, maximizing the utility of limited data by learning a model and leveraging it to derive optimal policies, potentially without additional real-world interaction. However, the requirement to develop or understand the model adds complexity, particularly in environments with complex dynamics.
- **Model-free RL**: In contrast, model-free RL methods do not rely on environmental models. Instead, they learn policies or value functions directly from trial-and-error interactions with the environment without presupposing knowledge of its dynamics. These methods operate independently of any prior knowledge of transition probabilities or rewards. Learning is based solely on data gathered through interactions. Model-free approaches are key in developing DRL algorithms and involve directly estimating value functions or learning policies that map states to actions. Although generally less sample-efficient than model-based methods, model-free approaches require extensive interaction with the environment to perform effectively. Their independence from a predefined model simplifies implementation and enhances flexibility, making them suitable for a broad spectrum of applications where environmental modeling is unfeasible.

From the discussion on policy iteration in the previous section, we understand that the RL agent generates experience by interacting with the environment under a certain policy and then learns the policy from those experiences and solving for the optimal policy π^* requires iterative execution of policy evaluation Eq. 2.19 and policy improvement Eq. 2.21, assuming knowledge of the transition probabilities. However, deriving the optimal policy using the Bellman equation is not feasible in scenarios where the transition probabilities $p(s'|s,a)$ are unknown. This brings us back to a fundamental characteristic of RL: learning through interaction with the environment. A general approach to learning a model-free solution in RL can be outlined as follows:

1. **Initialization:** Initialize the RL agent with an initial policy $\pi^{(0)}$.
2. **Interaction:** Allow the RL agent to interact with the environment using the current policy, thereby collecting a sequence of state-action pairs $\{s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1}, s_t\}$.
3. **Value Function Calculation:** Calculate the value functions after accumulating sufficient state-action sequences to facilitate policy evaluation.
4. **Policy Improvement:** Upon obtaining the value function, proceed with the policy iteration steps to perform policy improvement. Repeat step 2 continuously until the policy converges.

This exploration into model-free RL strategies provides a foundation for discussing more nuanced strategies within RL, specifically on-policy and off-policy learning. While model-free approaches focus on direct interaction without a predefined model of the environment, those learning methods delve deeper into how policies are updated and utilized.

- **On-policy RL**: On-policy methods learn the value of the policy currently used for decision-making, essentially evaluating or enhancing the policy that the agent actively employs. The agent knows the value function of this policy and accumulates experiences directly under it. The learning process is intrinsically linked with exploration; the policy must balance exploring the environment to discover new strategies and exploiting known strategies to maximize rewards. These methods focus on directly improving the policy responsible for generating the data. A commonly used on-policy algorithm is SARSA [12], where the agent updates its policy based on the actions taken and the rewards received, thus aligning policy learning closely with the chosen action-selection strategy, such as epsilon-greedy.
- **Off-policy RL**: Off-policy methods learn the value of a policy different from the one used to generate the interaction data. This separation allows for evaluating and improving a target policy using data collected from a distinct behavior policy. Learning about the optimal policy occurs independently of the agent's current actions. The behavior policy, which is used for generating behavior, may differ from the target policy that is being evaluated and refined. These methods are highly adaptable as they can utilize experiences from past interactions stored in memory, data from previous policies, or hypothetical scenarios. Q-learning [13] is a typical example of off-policy methods where the updates are made based on selecting the best possible action, regardless of the actual action taken.

These differences between on-policy and off-policy methods highlight their distinct characteristics. Off-policy methods are typically more data-efficient because they can learn from data generated by any policy, including older versions. This flexibility allows off-policy learning to effectively utilize data from various policies, making it well-suited for learning from observational data or in environments where experimentation is expensive. However, off-policy methods exhibit more complex theoretical convergence properties due to the differences between the target and behavior policies. In contrast, on-policy methods usually exhibit simpler convergence properties, as they consistently evaluate and refine the same policy.

In the rest of this section, we will delve into the fundamental aspects of RL solutions, outlining the core principles necessary for deriving optimal RL strategies. Additionally, we will explore basic RL algorithms across various categories, establishing a solid foundation for understanding more advanced algorithms. This comprehensive overview aims to equip readers with a thorough grasp of both the theoretical underpinnings and practical applications of RL, serving as a stepping stone towards mastering complex RL systems.

2.3.1 Monte Carlo Method

The Monte Carlo method offers a powerful set of techniques in RL for solving problems where the environment dynamics are partially or completely unknown. Unlike Dynamic Programming (DP) approaches, Monte Carlo methods do not require knowledge of the environment transition probabilities and rewards. Instead, they estimate the value functions on experience—specifically and derive policies based on samples of complete sequences of states, actions, and rewards. These methods are particularly applicable in episodic tasks, where interaction sequences naturally terminate.

Monte Carlo methods require the completion of episodes to update the value estimates. Each episode provides a sample of the rewards that follow from states and actions, which is used to approximate the value functions. They do not require a model of the environment, as they learn directly from the experiences generated by interacting with the environment. Value function updates are performed at the end of each episode, not after each step. The goal of Monte Carlo methods in RL is to estimate the value function, which can be either state value function $v(s)$ or action value function $q(s, a)$:

- **Estimation of $v(s)$:** Under the given policy π, the state value function $v_\pi(s)$ is estimated by averaging the returns following visits to state s in multiple episodes, the return for state s at time t in one episode is given by its long-term reward G_t, as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-t-1} r_T, \tag{2.24}$$

  where r_{t+k} is the reward received in k time steps after time t, and T is is the terminal time step of the episode. The value of state s is then updated as:

$$v(s) = \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_t^{(i)}, \tag{2.25}$$

  where $N(s)$ is the counts of how many times state s has been visited across all episode, and $G_t^{(i)}$ is the long-term return at the i $- th$ visit to state s.

- **Estimation of $q(s, a)$:** Action values are estimated similarly, but they require tracking returns for state-action pairs rather than just states:

$$q(s, a) = \frac{1}{N(s, a)} \sum_{i=1}^{N(s,a)} G_t^{(i)}. \tag{2.26}$$

  Similarly, $N(s, a)$ denotes the count of how many times action a is chosen at state s.

After obtaining the estimation of value functions, the RL policy can then be updated using a greedy approach:

- Policy update using state value function $v(s)$:

$$\pi^{(k+1)}(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma\ v^{(k)}(s')] \tag{2.27}$$

- Policy update using action value function $q(s, a)$:

$$\pi^{(k+1)}(s) = \arg\max_a q^{(k)}(s, a) \tag{2.28}$$

where $p(s', r|s, a)$ denotes the probabilities of transitioning to state s' and receiving reward r when taking action a in state s, estimated implicitly by the Monte Carlo method. Building upon the approach of estimating value functions using current policies, the distinction between on-policy and off-policy Monte Carlo methods in RL can be addressed as follows:

- **On-policy Monte Carlo Method**: The on-policy Monte Carlo method estimates the value function for the current policy using episodes generated from that same policy. This method often applies an ϵ-soft policy, where each action has a small, finite probability of being chosen to ensure sufficient exploration of alternative actions. After each episode, the returns are used to learn the action-value function, and the policy is then improved based on the updated value function for all the states visited during the episode;

- **Off-policy Monte Carlo Method**: The off-policy Monte Carlo method estimates the value function for a target policy using data generated from different policies. This method evaluates the target policy independently by applying importance sampling to adjust for differences between the target and behavior policies. Although data is collected under the behavior policy, importance sampling enables the estimation of expected returns under the target policy.

In on-policy Monte Carlo methods, the policy serves dual purposes: it generates state-action sequences through exploration and simultaneously learns the optimal policy. This approach is inherently a compromise because it learns action values not from the optimal policy but from a near-optimal policy that continues to explore. In contrast, off-policy Monte Carlo methods utilize two distinct policies: the behavior policy for exploration and generating trajectories and the target policy for learning and improvement. This method decouples the exploration process from the policy improvement process, following the behavior policy while learning about and refining the target policy.

A common challenge with Monte Carlo methods is that updates to value estimations occur only at the end of each episode, which can slow down the learning process, especially in environments with lengthy episodes. This distinction highlights how off-policy approaches can offer more flexibility and efficiency in environments where separate exploration and exploitation are beneficial.

2.3.2 Temporal Difference

Temporal Difference (TD) is another method to minimize the estimation error in policy evaluation. The TD method merges aspects of Monte Carlo and DP. Like the Monte Carlo method, it determines the value function by acquiring sequences through interaction and employs the Bellman equation for iterative updates. Yet TD learning stands out by leveraging the strengths of both techniques, allowing for efficient policy evaluation and control without requiring a model of the environment's dynamics. This method's unique capability to learn directly from raw experience, updated incrementally after each step, makes it especially powerful in environments where obtaining complete interaction sequences (as required by Monte Carlo methods) is impractical.

TD methods are based on the principle that estimates are updated partly based on other learned estimates without waiting for a final outcome (as is typical in Monte Carlo methods). This approach uses the Bellman equation not to compute exact updates, as in DP, but to perform bootstrap updates—partial updates that improve estimates based on other current estimates. TD methods update estimates based on existing estimates (bootstrap from current estimates), allowing learning to occur before knowing the final outcome. Unlike Monte Carlo methods, which wait until the end of an episode, TD methods update value estimates incrementally at every step.

The simplest and most common TD method is called TD(0), or one-step TD learning, which updates value estimates based on the value of the succeeding state. The update rule for TD(0) for the state value function $V(S_t)$ respect to learning rate $\alpha \in (0, 1]$ is given by:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]. \tag{2.29}$$

This equation shows how the new estimate for $V(S_t)$ is adjusted towards the estimated return, $R_{t+1} + \gamma V(S_{t+1})$, a target that includes the reward received for moving to the next state plus the discounted value of the next state. This is clearly different from the value function update in the Monte Carlo methods, which use the actual return G_t, with:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]. \tag{2.30}$$

TD methods refine their estimates by incorporating updates based on previous estimates, effectively learning from a series of educated guesses. This iterative refinement process has been demonstrated to converge asymptotically to accurate predictions.

2.3.3 SARSA

Moving beyond simple value estimation, TD methods are readily extended to learn optimal policies directly. An example of this application is SARSA [11], an on-policy TD-based RL algorithm, originally proposed as Modified Connectionist Q-Learning (MCQ-L) [12]. As a direct extension of the TD approach, SARSA uses the same principle as TD(0) but updates the action-value function Q, considering the action taken and the next action to be taken, which is determined by the current policy. The learning process in SARSA is continuous throughout the interaction between the agent and environment, updating its estimates of the action values based on the transitions that occur from one state-action pair to the next. The SARSA algorithm can be outlined as follows:

Algorithm 1 SARSA Algorithm

Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
Initialize policy π to be ϵ-greedy relative to Q (typically all a equal probability)
repeat
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    repeatfor each step of the episode:
        Take action A, observe R, S'
        Choose A' for S' using policy derived from Q (e.g., ϵ-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S'$; $A \leftarrow A'$
    until S is terminal
until convergence of Q or after a certain number of episodes

The SARSA update rule at each time step t can be regarded as a typical TD learning, given by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma\, Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]. \tag{2.31}$$

In SARSA, the policy used to determine A_{t+1} from S_{t+1} is the same as the policy being evaluated and improved, making SARSA an on-policy method. Typically, policies such as ϵ-greedy [11] are used to ensure adequate exploration of the action space. This approach allows the agent to occasionally diverge from the greedy choice, thereby exploring new actions that might lead to higher long-term rewards.

The ϵ-greedy algorithm selects the best-known action (greedy action) most of the time but explores the environment by choosing a random action with a small probability $\epsilon \in [0, 1]$, where the agent is purely greedy if $\epsilon = 0$ and selects actions completely at random if $\epsilon = 1$, with:

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} & \text{if } a = \arg\max_{a'} Q(s, a') \\ \frac{\epsilon}{|\mathcal{A}(s)|} & \text{otherwise,} \end{cases} \tag{2.32}$$

where $|\mathcal{A}(s)|$ is the number of available actions under state s.

The complete process of SARSA is given in Algorithm 1. Since SARSA learns the value of the policy it follows and incorporates exploration into its updates, it tends to be safer and more stable than off-policy methods like Q-learning, which will be introduced in the next section. SARSA seamlessly integrates varying degrees of exploratory behavior directly into the policy improvement step, making it adaptable to various environmental dynamics

and complexities. This method is beneficial in tasks where the safety and stability of the learning process are paramount. The subsequent discussion on Q-learning will further illuminate the distinctions between on-policy and off-policy methods, providing a broader perspective on choosing appropriate strategies for different challenges of RL.

2.3.4 Q-Learning

Unlike SARSA, Q-learning [13] is an off-policy RL algorithm that decouples the learning policy from the behavior policy. This separation allows Q-learning to learn the optimal policy, regardless of agent actions dictated by the current policy. This characteristic makes Q-learning particularly effective for situations where a robust optimal policy is required without the restrictions imposed by the adherence to a specific exploratory strategy, as is the case with on-policy methods like SARSA.

Algorithm 2 Q-learning Algorithm

Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Initialize learning rate α, discount factor γ, and ϵ for ϵ-greedy policy
repeat
    Initialize state S
    repeat(for each step of the episode):
        Choose action A from state S using ϵ-greedy policy derived from Q
        Take action A, observe reward R, and next state S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
        $S \leftarrow S'$
    until S is terminal
until convergence of Q or after a certain number of episodes

Q-learning aims to find the best strategy for action selection that maximizes the total reward over successive steps, starting from the current state. It does this by updating its estimates of the action values (in Q-learning also termed Q-values) based on the Bellman equation, which provides a recursive definition for the optimal policy given by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]. \tag{2.33}$$

The policy π is then implicitly defined as the greedy policy concerning the Q-values. In Algorithm 2, we can intuitively find the difference of Q-learning compared to SARSA: Q-learning directly approximates the optimal action-value function, independent of the policy followed by the agent. This leads to learning the optimal policy even when the actions are sub-optimal. The algorithm is simple to implement and robust across various environments, particularly those with well-defined and discrete action spaces.

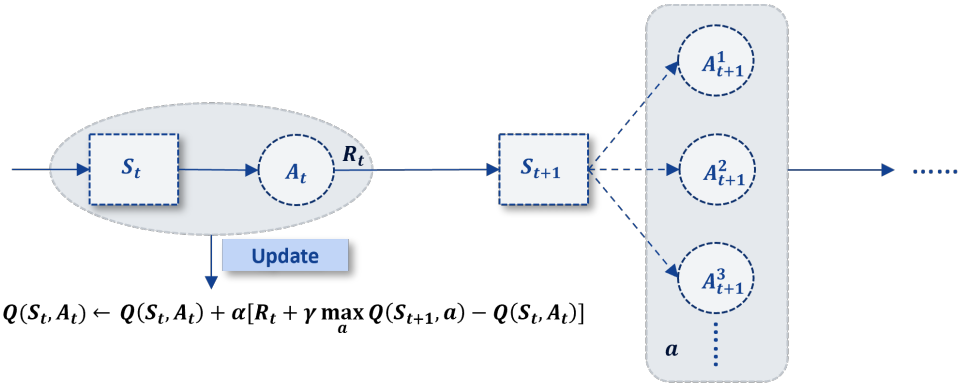

Figure 2.5: Update of Q-value in Q-learning algorithm

While Q-learning theoretically converges to the optimal action values with sufficient training under certain conditions (like all state-action pairs being visited infinitely often), in

practice, especially in complex environments, finding the optimal policy can be computationally demanding and slow. Additionally, due to its maximization step, Q-learning can suffer from overestimation of action values, especially in noisy environments. This overestimation bias can lead to sub-optimal policies, making it essential to carefully tune the learning parameters and explore strategies to mitigate this issue, such as using Double Q-learning [14] methods.

2.3.5 Deep Q-Learning Network

Through tabular methods, Q-learning provides a robust framework for learning optimal policies in discrete action spaces. However, as the complexity of the environment increases, the table storing Q-values can become infeasibly large, making classical methods computationally expensive and less effective. Building upon the foundation set by Q-learning, DQN [15] extends its principles by addressing dimensional limitation by using a deep neural network to approximate the Q-value function, allowing for generalization across similar states without explicitly storing values for every possible state-action pair. This integration addresses the scalability issues associated with classical Q-learning when dealing with high-dimensional state spaces, such as those in video games or robotic control systems.

Algorithm 3 DQN Algorithm

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode $= 1, M$ **do**
    Initialize sequence s_1 and preprocessed sequence $\phi_1 = \phi(s_1)$
    for t $= 1, T$ **do**
        With probability ϵ select a random action a_t
        Otherwise select $a_t = \max_a Q(\phi(s_t), a; \theta)$
        Execute action a_t in the environment
        Observe reward r_t and new state s_{t+1}
        Set $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to θ
        Every C steps reset $\hat{Q} = Q$ with $\theta^- = \theta$
    end for
end for

As Algorithm 3 illustrates, a deep neural network predicts Q-values based on state input, reducing the dimensionality and dependency on the state-space size. DQN utilizes a replay memory to store transitions randomly sampled during the training process. This approach breaks the correlation between consecutive samples and smooths over changes in the data distribution, enhancing the stability of the neural network. To further stabilize training, DQN uses a separate, fixed network to estimate the target Q-values that are updated less frequently than the primary network used for value approximation. In DQN, two key innovations significantly enhance the stability and performance of the algorithm: the use of a replay buffer and a target network. These concepts address fundamental issues in training deep learning models with RL, specifically when dealing with correlated data and moving targets. Fig. 2.6 illustrates how the replay buffer and target network work in the training procedure of DQN.

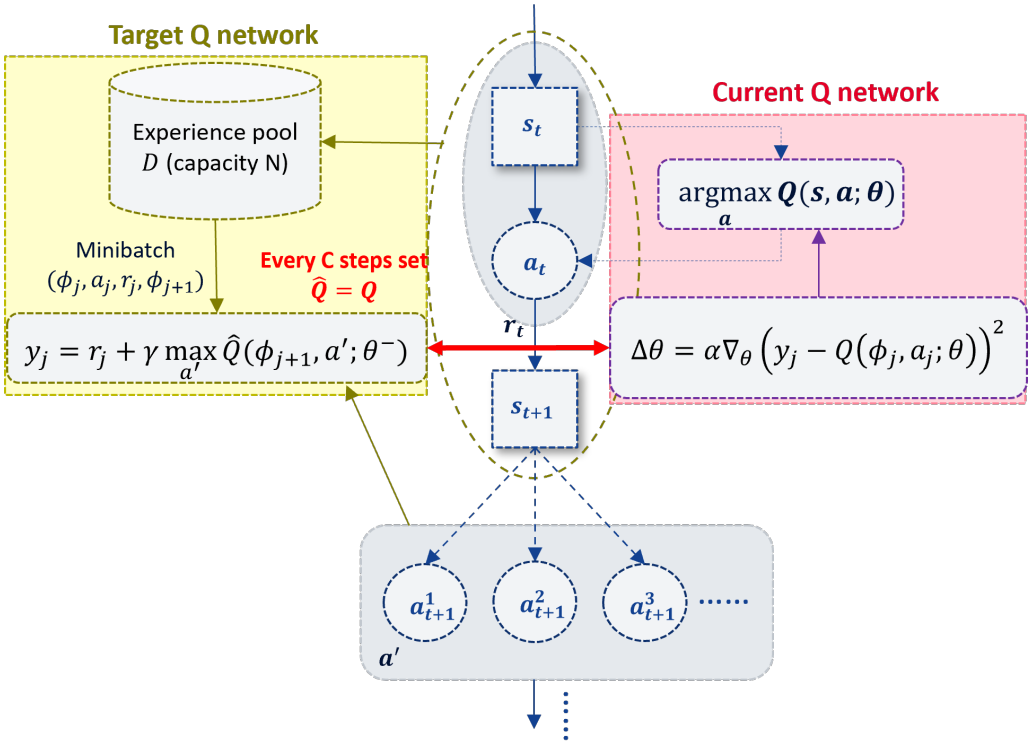

Figure 2.6: Update of Q networks in DQN

Replay buffer: A replay buffer is a structure that stores the experienced interaction sequences, consisting of tuples containing (s, a, r, s'). This concept is rooted in the idea of experience replay. The capacity of this buffer is often defined as finite, so older experiences are discarded once the capacity is reached. For each training step, a batch of experiences is randomly sampled from the buffer to compute the loss and update the model. This approach improves sample efficiency by reusing past experiences and helps smooth out learning and avoid catastrophic forgetting.

Target network: The target network is a copy of the leading network (the network being actively trained), but its weights are updated less frequently. The reason for introducing a target network is to stabilize the learning process by providing a fixed target for a period of time. The nature of RL targets a moving optimum, making the learning process chaotic and unstable. To address this, the weights in the target network are kept fixed for several iterations while the main network keeps updating. Typically, these weights are updated by copying over the weights from the main network every few thousand steps or gradually with a soft update rule, where the target network slowly tracks the main network. Fixing the target network provides a stable baseline from which the main network can learn, reducing the variability in the target during the learning updates.

DQN represents a typical extension from traditional RL to DRL by integrating deep learning, which facilitates handling environments with complex, high-dimensional data that traditional methods cannot process efficiently. In comparison with RL approaches, DRL has the following advantages:

- **Scalability:** DRL can manage larger, more complex environments thanks to the function approximation capabilities of deep neural networks.
- **Generality:** Neural networks in DRL generalize across states, allowing the agent to perform well in unseen states that resemble those it has encountered during training.

- **Efficiency:** While traditional RL might struggle with the curse of dimensionality in large state or action spaces, DRL leverages the efficiency of deep learning in dealing with high-dimensional data.

This progression from traditional RL to DRL broadens the scope of feasible applications and enhances our ability to tackle complex decision-making tasks in dynamic and uncertain environments. Due to its practical successes, DQN has been developed into several variations, including Double DQN [14], Dueling DQN [16], and Rainbow DQN [17].

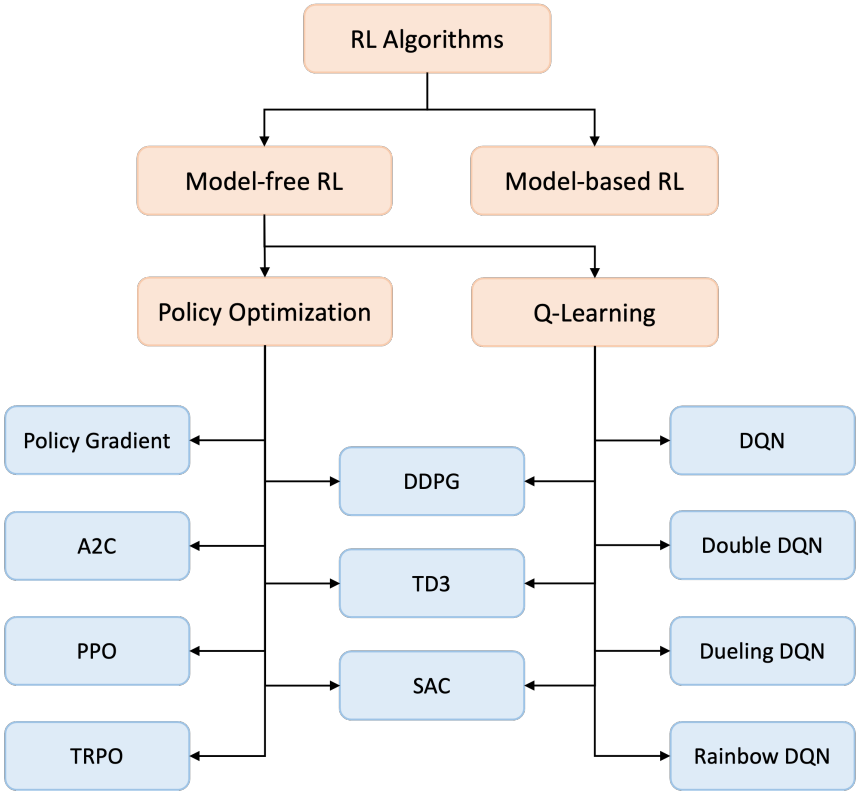

Figure 2.7: Categories of modern RL algorithms

2.4 Policy Gradient Algorithm

The critical steps in the previously discussed RL algorithms involve estimating the value function and updating the policy. A robust value function estimate is essential for deriving the optimal strategy. While the primary goal of RL is to determine the optimal policy, the reliance on value function estimation distinguishes these algorithms. In this section, we will explore more direct RL approaches that bypass the value function estimation and directly compute potential policy update directions. Specifically, we will focus on PG methods, starting with the basic policy gradient technique and progressing to the PG-based extension, actor-critic.

2.4.1 Policy Gradient

PG [18] methods represent a class of RL algorithms that directly optimize the policy as a mapping from states to actions without explicitly estimating value functions. Unlike value-based methods, which focus on learning the value of actions to indirectly determine the optimal policy, PG methods adjust the policy parameters directly based on the gradient of the expected return. This approach offers several advantages, particularly in continuous action spaces or environments where the policy needs to be highly expressive. While the

Bellman equation requires the optimal policy π^* to satisfy Eq. 2.8, PG methods take a different approach by maximizing the expectation of long-term RL return with:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau} \sim \pi(\tau)[r(\tau)], \tag{2.34}$$

where τ represents a trajectory of states and actions, and $r(\tau)$ is the total reward obtained from the trajectory. PG methods rely on optimizing the policy function parameterized by θ, denoted as $\pi(a|s;\theta)$, which represents the probability of selecting action a in state s given parameters θ. Then, the goal of solving the optimal policy π^* becomes deriving θ that maximizes the expected return. The objective of PG method is formulated as:

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[r(\tau)] = \int_{\tau\sim\pi_\theta} \pi_\theta(\tau) r(\tau) d\tau. \tag{2.35}$$

The parameters θ can be updated by finding the derivative of Eq. 2.35 and applying gradient ascent [18]. Using the Monte Carlo method, the gradient of the policy objective function $J(\theta)$ with respect to the parameters θ is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t \right] \tag{2.36}$$

where τ represents a trajectory of states and actions with a total length T. The term $\nabla_\theta \log \pi_\theta(a_t|s_t)$ allows the application of the likelihood ratio trick, simplifying the differentiation of probabilities. With derivations of J_θ at hand, the policy parameters θ are updated using gradient ascent given a learning rate α with:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta). \tag{2.37}$$

Algorithm 4 Policy Gradient Method

```
Initialize policy parameters θ randomly
for each episode do
    Generate an episode s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T under π_θ
    for t = 0 to T − 1 do
        G_t ← return from step t
        θ ← θ + αγ^t G_t ∇_θ log π_θ(a_t|s_t)
    end for
end for
```

The complete process of the PG method is illustrated in Algorithm 4. Compared to value-based RL algorithms, PG methods often exhibit more stable convergence properties, particularly when using function approximations. These methods suit environments with continuous or high-dimensional action spaces in DRL. Unlike methods that only learn deterministic policies, PG can learn stochastic policies, which are crucial in environments where randomness is inherent or exploration is beneficial. However, PG methods require many samples to estimate the gradient accurately, making them less sample-efficient than their value-based counterparts. Additionally, the gradient estimates can have high variance, leading to unstable training and the need for sophisticated variance reduction techniques.

2.4.2 Actor-critic

While PG methods provide a robust framework for directly learning policies, their practical implementation faces challenges like high variance in gradient estimates and poor sample efficiency. This leads us to the actor-critic architecture [19], which enhances basic PG

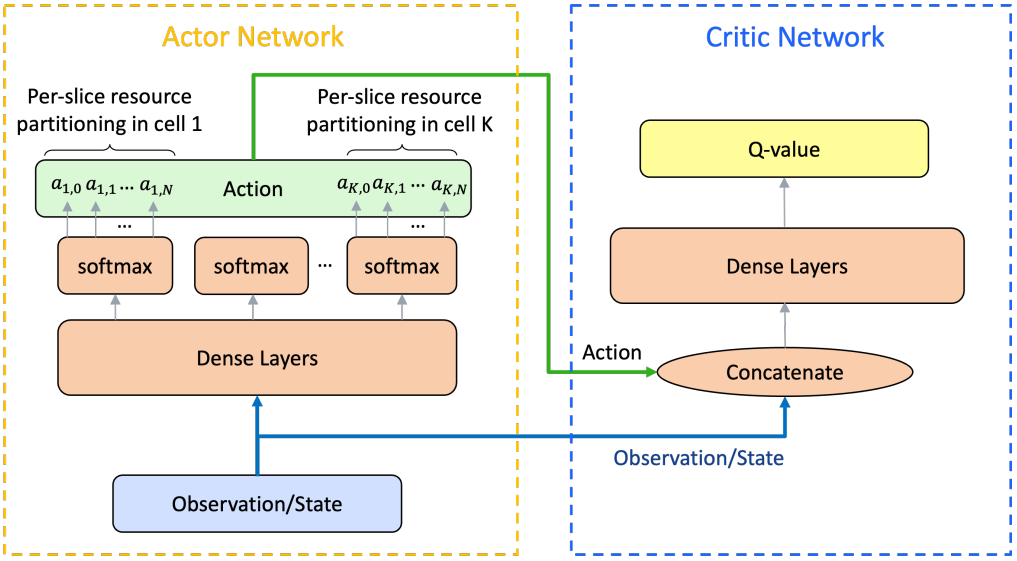

Figure 2.8: Actor-critic structure of RL-based slice resource partitioning

approaches by combining them with value function approximation. By combining the advantages of both PG and value-based approaches, actor-critic methods efficiently address the challenges of high variance and slow convergence rates associated with traditional policy gradient techniques.

As its name indicates, the actor-critic architecture consists of two main components:

1. **Actor:** The actor directly parameterizes the policy $\pi(a|s;\theta)$ with respect to parameter θ. The actor decides which action to take in a given state by stochastic sampling from the policy.

2. **Critic:** The critic evaluates the action taken by the actor by computing a value function, which could be either state value function $V(s)$ or action value function $Q(s,a)$. This component helps to estimate the performance of the policy.

The interplay between these two components allows the system to balance exploration, implemented by the actor, and exploitation, performed by the critic. This effectively learns optimal policies by reducing the variance of the updates and providing more stable and reliable learning signals. Fig. 2.8 demonstrates an example of the actor-critic structure for slice resource allocation.

The critic updates its value function parameters based on the difference between the computed values and the rewards obtained from the environment (the TD error). Meanwhile, the actor adjusts its policy parameters in the direction suggested by the critic to maximize expected rewards. Similarly, the actor-critic aims to find the optimal policy that maximizes the long-term return. The gradient of the objective concerning policy parameters θ is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) Q^\pi(s_t,a_t)\right] \tag{2.38}$$

This formulation directly incorporates the action-value function $Q^\pi(s,a)$. This derivation leads to the actor-critic update rules of two components:

1. **Critic update**: The critic updates its value function parameters ω to better estimate the return for each state-action pair. This is typically done using TD learning, particularly TD(0), with the update rule:

$$\omega \leftarrow \omega + \alpha_c \cdot \delta \cdot \nabla_\omega Q(s,a:\omega), \tag{2.39}$$

where $\delta = R + \gamma Q(s', a'\omega) - Q(s, a : \omega)$ is the TD error and α_c is the learning rate of critic.

2. **Actor update**: The actor updates its policy parameters θ in the direction suggested by the critic to maximize the expected return. The update rule using the policy gradient is:

$$\theta \leftarrow \theta + \alpha_a \ \gamma^t \cdot \delta \cdot \nabla_\theta \log \pi_\theta(a_t|s_t), \tag{2.40}$$

where α_a is the actor learning rate.

Algorithm 5 Actor-critic Method

Initialize policy parameters θ and value function parameters w randomly
Initialize environment and observe initial state s
for each episode **do**
  repeat
    Choose action a according to the current policy $\pi(a|s;\theta)$
    Execute action a in the environment
    Observe reward r and new state s'
    Calculate TD error: $\delta = r + \gamma Q(s', a'; w) - Q(s, a; w)$
    Update critic by minimizing loss $L = \delta^2$:

$$w \leftarrow w + \alpha_w \cdot \delta \cdot \nabla_w Q(s, a; w)$$

    Update actor using the policy gradient:

$$\theta \leftarrow \theta + \alpha_\theta \cdot \delta \cdot \nabla_\theta \log \pi(a|s;\theta)$$

    Set $s \leftarrow s'$
  until end of episode
end for

The actor-critic can achieve more stable and efficient learning by maintaining separate estimations for policy and value functions. With the basic actor-critic framework set, we can extend it to more sophisticated versions like DDPG and TD3, which adapt the actor-critic paradigm to continuous action spaces and improve stability and performance through architectural enhancements such as separate target networks, delayed policy updates, and noise for exploration.

2.5 Advances of RL Algorithms

While basic actor-critic methods provide a robust framework for balancing exploration and exploitation, they can struggle with complex environments that feature large state or action spaces, require handling delayed rewards, or exhibit significant observational noise. To address these challenges, advanced actor-critic algorithms such as DDPG [20] and TD3 [21] have been developed. In this dissertation, we have implemented a specialized version of the TD3 algorithm, tailored explicitly for network slicing, to serve as the RL mechanism for addressing our optimization problem in slicing resource allocation. Consequently, this section will provide a detailed examination of TD3 and its precursor, the DDPG algorithm.

2.5.1 DDPG

The DDPG [20] algorithm is an actor-critic approach based on the Deterministic Policy Gradient Algorithm (DPG) [22] algorithm. DDPG maintains a parameterized actor function and a critic function, both learned using the Bellman equation. The actor function specifies the policy by deterministically mapping states to specific actions, while the critic

function estimates the value of taking a specific action in a given state. The actor is updated by applying the chain rule to the expected return from the start distribution with respect to the actor parameters. The algorithm uses neural network function approximations to learn in large state and action spaces online, inspired by the success of DQN. The use of large, non-linear function approximation in DDPG allows for learning and generalization in large state spaces, although convergence is not guaranteed. DDPG has been shown to learn competitive policies for various physical control tasks, including tasks with high-dimensional pixel inputs, using a straightforward actor-critic architecture and learning algorithm.

DPG DPG uses a deterministic policy $\pi(s|\theta)$ that directly maps states to actions without the variance introduced by stochastic policies. Here, θ represents the parameters of the policy network. The critical innovation in DPG derives the policy gradient for deterministic policies. Unlike stochastic policy gradients that average over the distribution of actions, the deterministic policy gradient can be computed more directly, simplifying the computation and reducing the variance of gradient estimates. The deterministic policy gradient is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^\pi}[\nabla_\theta \pi(s|\theta) \nabla_a Q^\pi(s,a)|_{a=\pi(s|\theta)}], \tag{2.41}$$

where $Q^\pi(s,a)$ is the action-value function that evaluates the goodness of taking action a in state s under policy π, and ρ^π denotes the state distribution under policy π. By employing a deterministic policy, DPG reduces the computational burden associated with sampling actions, which is particularly beneficial in high-dimensional action spaces. The gradient estimates in DPG tend to have lower variance compared to those in stochastic policy gradients, leading to more stable learning.

Building on the principles of DPG, the DDPG algorithm extends the approach with function approximations using deep neural networks inspired by DQN. With the integration of deep learning, DDPG adapts the insights of DPG to more complex and realistic settings. DDPG also leverages experience replay to improve sample efficiency and break the correlation between consecutive updates and target networks for both actor and critic networks to enhance learning stability. Rather than abruptly copying the weights of the actor and critic networks to the target networks, DDPG uses soft updates, where the target networks are slowly adjusted towards the primary networks. This gradual transition ensures that the learning process remains stable and consistent.

The process of DDPG algorithm is illustrated in Algorithm 6, where the actor and critic networks are defined as $\pi(s|\theta^\pi)$ and $Q(s,a|\theta^Q)$ with network parameters θ^π and θ^Q, respectively. The objective of the actor network is to maximize the expected return from the start distribution:

$$J = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi}[Q(s,a|\theta^Q)] \tag{2.42}$$

Specifically, the critic network in DDPG is updated by minimizing the estimation loss of the value function:

$$L(\theta^Q) = \frac{1}{N}\sum_i (y_i - Q(s_i,a_i|\theta^Q))^2 \tag{2.43}$$

where $y_i = r_i + \gamma Q'(s_{i+1}, \pi'(s_{i+1}|\theta^{\pi'})|\pi^{Q'})$ is the target value for the current Q-value estimation, Q' and π' are the target networks for the critic and actor, respectively. Subsequently, the actor network is updated, referring to its gradient:

$$\nabla_{\theta^\pi} J \simeq \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\pi(s_i)} \nabla_{\theta^\pi}\pi(s|\theta^\pi)|_{s_i}. \tag{2.44}$$

In summary, DDPG refers to DPG by utilizing a deterministic policy function, which offers enhanced performance in environments with high-dimensional continuous action spaces.

Algorithm 6 DDPG

Initialize critic network $Q(s, a|\theta^Q)$ and actor $\pi(s|\theta^\pi)$ with weights θ^Q and θ^π
Initialize target network Q' and π' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\pi'} \leftarrow \theta^\pi$
Initialize replay buffer R
for episode = 1, M **do**
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state s_1
    for t = 1, T **do**
        Select action $a_t = \pi(s_t|\theta^\pi) + \mathcal{N}_t$ according to the current policy with noise
        Execute action a_t and observe reward r_t and new state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in R
        Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \pi'(s_{i+1}|\theta^{\pi'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\pi} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\pi(s_i)} \nabla_{\theta^\pi} \pi(s|\theta^\pi)|_{s_i}$$

        Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}; \quad \theta^{\pi'} \leftarrow \tau\theta^\pi + (1-\tau)\theta^{\pi'}$$

    end for
end for

Additionally, it incorporates advanced methodologies from DQN such as replay buffer and target networks, significantly improving the training efficiency of the models. Fig. 2.9 demonstrates the complete procedure of DDPG.

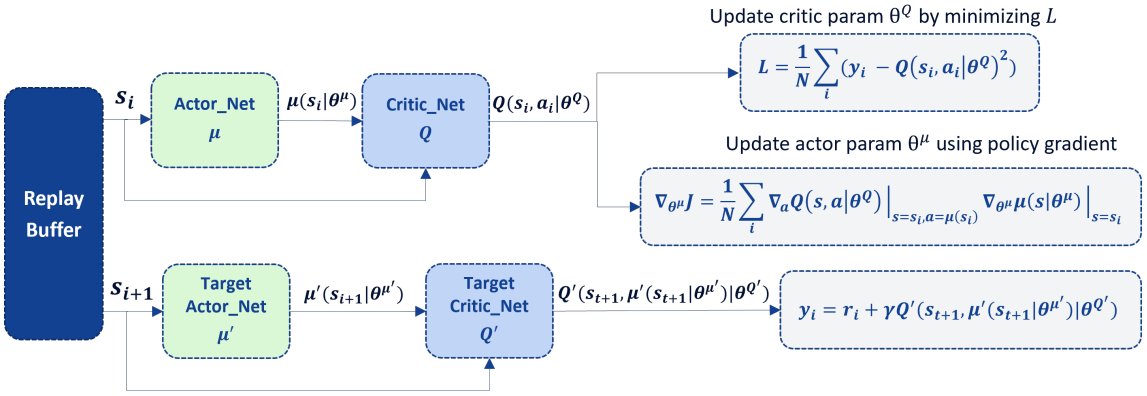

Figure 2.9: Update of actor-critic networks in DDPG algorithm

2.5.2 TD3

TD3 [21] is developed to address key challenges observed in DDPG, particularly the overestimation bias in Q-values due to the max operator in Bellman equations, a common problem also seen in Q-learning. This overestimation can lead to sub-optimal policy learning and instability during training. The TD3 algorithm builds on the foundation laid by DDPG by introducing key improvements, absorbing advantages from other RL approaches such as Double DQN. TD3 uses two critic networks and takes the minimum value of the two Q-functions to reduce overestimation bias. The key innovations of TD3 over the DDPG algorithm are:

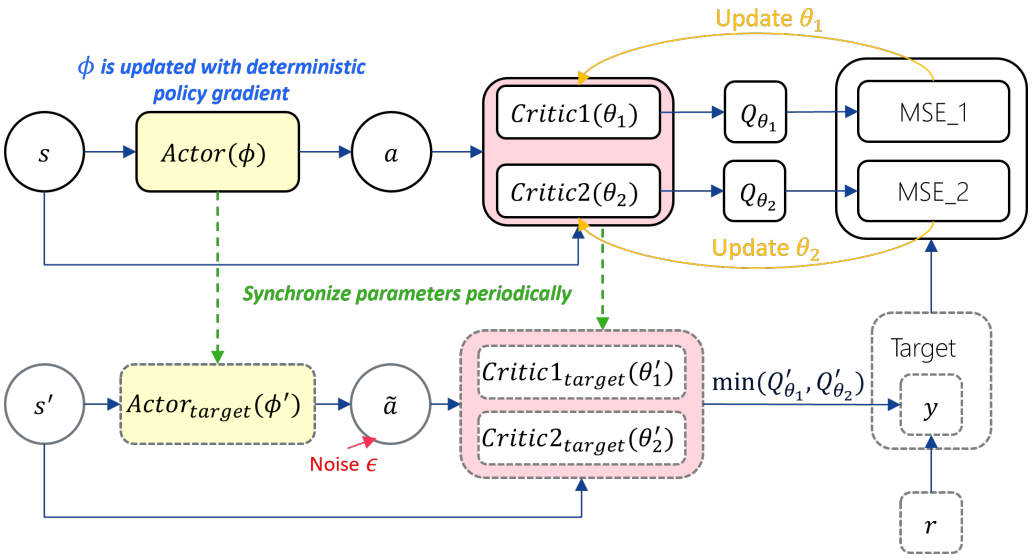

Figure 2.10: The process of TD3 algorithm

- **Double critic networks**: TD3 employs two separate critic networks (and their corresponding target networks) to estimate the action-value function. Besides the original critic network Q_{θ_1} and its target network $Q_{\theta_1'}$, TD3 defines an additional twin Q_{θ_2} and its target network $Q_{\theta_2'}$. By taking the minimum of the two estimated Q-values during the update steps, TD3 mitigates the positive bias in the policy improvement step, which is a common issue in DDPG.

- **Delayed policy updates**: Unlike DDPG, which updates the policy at every step, TD3 delays policy updates, reducing the frequency with which the policy is trained. This helps ensure that the value estimates are more stable and reliable before each policy update, reducing the risk of policy degradation due to poor value estimates.

- **Target policy smoothing**: TD3 adds noise to the target policy action, smoothing out the value estimates and preventing deterministic exploitation of Q-function inaccuracies.

With these modifications, targeting for the same objective as DDPG Eq. 2.42, the target for updating the Q-value in the critic network is given as:

$$y = r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a}), \tag{2.45}$$

with $\tilde{a} = \text{clip}(\pi(s') + \text{clip}(\epsilon, -c, c), a_{\text{Low}}, a_{\text{High}})$, where a_{Low} and a_{High} are the action bounds, and noise $\epsilon \sim \mathcal{N}(0, \sigma)$. The critic networks are updated by minimizing the Q-value estimation error:

$$L(\theta_i) = \mathbb{E}[(Q_i(s, a) - y)^2] \tag{2.46}$$

The actor policy is updated by the deterministic policy gradient:

$$\begin{aligned} \nabla_\phi J &\approx \mathbb{E}\left[\nabla_\phi Q_{\theta_1}(\mathbf{s}, \mathbf{a})\,|_{\mathbf{s}=\mathbf{s}_t, \mathbf{a}=\pi_\phi(\mathbf{s}_t)}\right] \\ &= \mathbb{E}\left[\nabla_{\mathbf{a}} Q_{\theta_1}(\mathbf{s}, \mathbf{a})\,|_{\mathbf{s}=\mathbf{s}_t, \mathbf{a}=\pi_\phi(\mathbf{s}_t)} \nabla_\phi \pi_\phi(\mathbf{s}_t)\right]. \end{aligned} \tag{2.47}$$

The TD3 algorithm is given in Algorithm 7, and Fig. 2.10 illustrates its procedure.

TD3 provides a mature way of handling continuous decision-making tasks in complex environments. It complements the deterministic policy gradient approach by addressing over-optimism that can undermine training in the actor-critic framework. This makes

Algorithm 7 Twin Delayed Deep Deterministic Policy Gradient (TD3)

Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$ and actor network π_ϕ
Initialize target networks $\theta_1' \leftarrow \theta_1, \theta_2' \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for each episode **do**
    Receive initial observation state s
    for t = 1, T **do**
        Select action a using policy $\mu_\phi(s)$ and exploration noise
        Execute action a, observe reward r, and new state s'
        Store transition (s, a, r, s') in $\mathcal{B}$
        Sample a mini-batch of N transitions from $\mathcal{B}$
        Set y using the clipped double-Q targets from target networks
        Update the critic networks by minimizing the loss $L(\theta_i)$
        if it is time to update the policy **then**
            Update the actor network by the deterministic policy gradient
            Update the target networks:

$$\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i', i = 1, 2; \quad \phi' \leftarrow \tau\phi + (1-\tau)\phi'$$

        end if
    end for
end for

TD3 stand out in comparison with DPG and DDPG. As we continue, the insights gained from TD3 pave the way for exploring more complex algorithms and creating more potential applications in multiple fields, including solving dynamic optimization problems in wireless communication.

2.6 Challenges in Applications

Having explored the concepts of RL from its fundamentals to advanced algorithms, we highlighted why RL has been successful in solving dynamic optimization problems. However, applying these successes to real-world scenarios presents several significant challenges, notably the exploration-exploitation dilemma, sample efficiency, and the transfer of learned policies across different tasks or environments.

Exploration-Exploitation Dilemma
One of the fundamental challenges in DRL is the exploration-exploitation dilemma, illustrated by Fig. 2.11, which involves deciding whether to explore the environment to discover new strategies (exploration) or exploit known strategies to maximize immediate rewards (exploitation). In most DRL applications, solutions are derived through real-time interactions, meaning the agent's policy is fine-tuned by enumerating collected explored actions. Simultaneously, the agent dedicates itself to providing recommended actions based on the current policy. This real-time exploration attempts to experience actions that deviate from the current policy, while in the exploitation phase, the agent sticks to the current policy to acquire more returns. However, under limited time or interactions, it is impossible to have an infinite exploration phase to go through all potential actions of all states.

This dilemma leads to the discussion of compelling exploration in the application of DRL. In complex environments, especially those with high-dimensional state and action spaces, sufficient exploration is necessary to ensure that the learning algorithm does not miss the areas that could contain optimal policies. Without adequate exploration, DRL algorithms can quickly converge to sub-optimal policies, particularly in non-convex optimization problems. Balancing exploration and exploitation is also crucial for stabilizing the learning updates in DRL. Too much exploitation can lead to premature convergence and poor policy

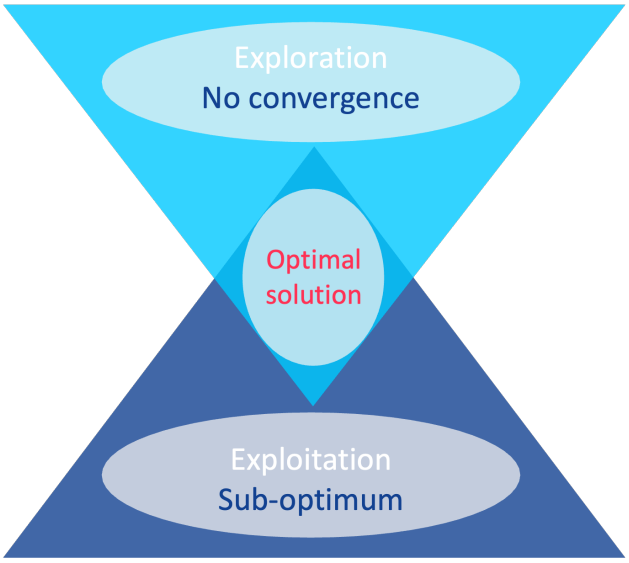

Figure 2.11: Exploration-Exploitation dilemma

performance, while excessive exploration can slow down the learning process or lead to high variance in policy updates.

The aforementioned ϵ-greedy strategy Eq. 2.32 is one of the methods to tackle this issue. It balances the choices of using the agent's recommendation or exploring other possibilities, which is widely applied in practical RL implementations. In the context of TD3, exploration is typically managed by adding noise to the policy actions (e.g., Gaussian noise). However, this method may not be sufficient for all scenarios, particularly those where the environment's dynamics are complex. Unfortunately, in wireless communication, network systems are often coupled with high complexity and high-dimensional data, significantly hindering the implementation of RL approaches despite their outstanding ability to solve dynamic problems.

Sample Efficiency
Another critical challenge in applying DRL to real-world scenarios is sample efficiency, which refers to the number of interactions or samples required to learn an effective policy. Many RL algorithms require substantial data to converge to an optimal or near-optimal policy. Collecting this data can be costly, time-consuming, or even infeasible in real-world applications. Improving sample efficiency is crucial for practical applications of RL, where each interaction with the environment might be expensive.

Given these challenges, TL presents a promising approach to address several issues associated with applying DRL algorithms like TD3 in practical scenarios. TL encourages reusing resolved knowledge, including pre-trained models and collected samples. DRL models can learn new tasks with fewer samples by transferring knowledge from similar tasks. This is particularly valuable in real-world applications where interaction with the environment can be costly or time-consuming. TL can also provide pre-learned models and policies covering a broad portion of the state-action space, reducing the need for exploration when adapting to new but related tasks. Moreover, learning policies that generalize across tasks can prevent overfitting to the specifics of a single environment, leading to more robust solutions in varied operational conditions.

In the next chapter, we will introduce TL as a promising approach to overcome several challenges in applying DRL, specifically TD3, to practical scenarios. By leveraging pre-trained models and transferring knowledge from similar tasks, TL can significantly enhance sample efficiency and reduce the need for extensive exploration, making RL more applicable and effective in real-world environments.

3. Transfer Learning

This dissertation explores the potential of applying TL to improve the sample efficiency and model reproducibility of pre-solved solutions in network slicing. In the previous chapters, we discussed solving dynamic optimization problems using RL approaches. In this chapter, we introduce TL, starting from its background and basic concepts, moving to specific methods, and finally presenting successful case studies in various application fields. Readers of this thesis are expected to have basic knowledge of ML and data science but may not be familiar with TL concepts and related works. This chapter is organized as follows: First, in Section 3.1, we introduce the background of TL and address its necessity in ML and practical problems. Then, in Section 3.2, we go through the key concepts and theories of TL with fundamental mathematical models. We further discuss the TL methods, such as the pre-train and fine-tuning framework and DA techniques in Section 3.3, and the current trends of TL solutions in practical use cases.

3.1 Introduction to TL

Before delving into the specifics of TL, consider the following two scenarios:

Generality of Image Classifiers: Consider a Convolutional Neural Net (CNN) trained to distinguish between images of cats and dogs. The next goal is to re-purpose this model for other visual recognition tasks, such as identifying images of tigers and wolves or classifying images of animated cats and dogs. The challenge lies in determining which aspects of the pre-trained model can be leveraged or modified to support these new classification tasks.

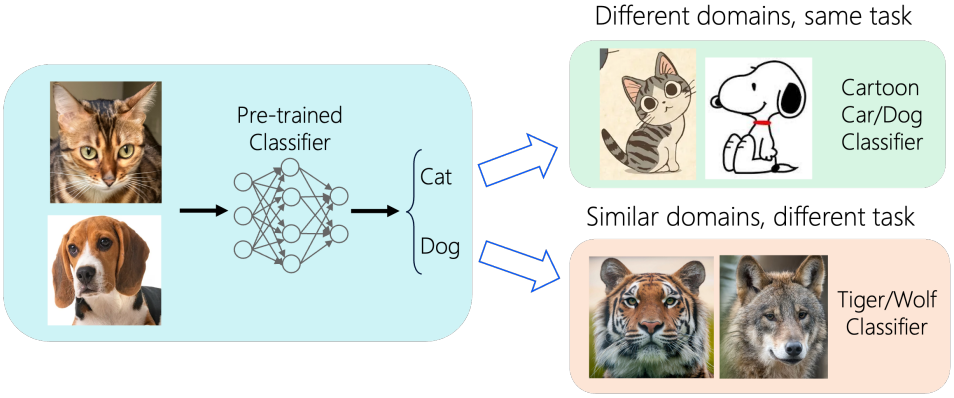

Figure 3.1: Generality of image classifiers

Commonality of Language Models: Imagine two pre-solved Natural Language Processing (NLP) models, one for translating German to English and another for translating German to Chinese, both developed using extensive language samples. Now, suppose we want to create a new model for translating German to Japanese, but we face a challenge due to the lack of sufficient language samples to ensure satisfactory performance. This situation prompts whether we can enhance the sparse sample set for German-Japanese translation by incorporating language samples from the more robust datasets of German-English and German-Chinese translations. Additionally, it is interesting to explore the effects of choosing which models to transfer knowledge regarding the similarity of translation.

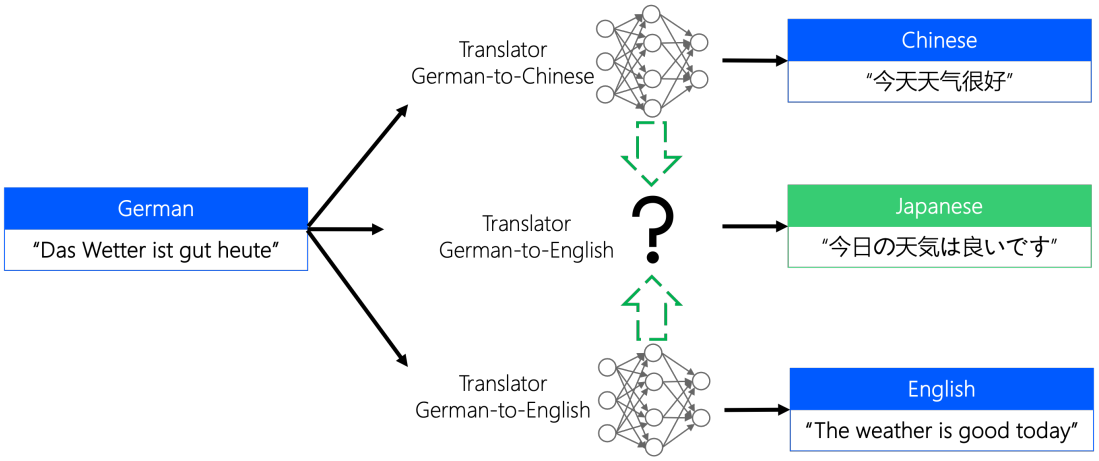

Figure 3.2: Commonality of language models

These scenarios highlight common challenges faced when deriving new models from scratch is time-consuming or when training samples are scarce. This leads to the concept of TL, which aims to leverage knowledge transfer from previously learned tasks to achieve faster deployment and improved outcomes in new applications. The concept of TL was originally discussed in the field of psychology under the term "transfer of learning" [23], emphasizing the effects of one learning process on another. In the field of AI, TL has become a crucial technique, characterized by its ability to adapt knowledge acquired from one problem and apply it to different but related problems.

3.1.1 Overview of TL

TL is a research domain within ML that focuses on reusing a pre-trained model for a new but related problem. Unlike traditional ML approaches, where a model is developed from scratch for every new task, TL allows for significant efficiencies and advancements by transferring the knowledge from one task to another. TL involves taking a model trained on a large dataset for a specific task and repurposing it for another related task. This methodology capitalizes on the commonalities between the source and target tasks. For instance, a model trained to recognize objects within photographs can be adapted to recognize objects in video clips without extensive retraining from the ground up. Hereby, we give a general definition to TL:

"Transfer learning can exploit the similarities between data, tasks, or models to apply models and knowledge learned in old domains to new domains."

TL is especially helpful in the modern context where deep learning models require enormous datasets, substantial computational power, and lengthy training times. TL offers a pragmatic solution by enabling the deployment of sophisticated models without the associated high costs and computational burdens. It ensures that even those with limited resources can participate in developing high-performing AI models, thus democratizing access to cutting-edge technology.

In [24], the authors describe three potential benefits that TL may bring to the model training process:

- **Jump Start:** The initial performance (before the model fine-tuning) on the target domain is higher than that of training a model from scratch.
- **Faster Convergence:** The rate of convergence of the transferred model during the training process is faster than that without knowledge transfer.
- **Higher Asymptote:** The process with TL achieves a higher converged performance after fine-tuning is completed.

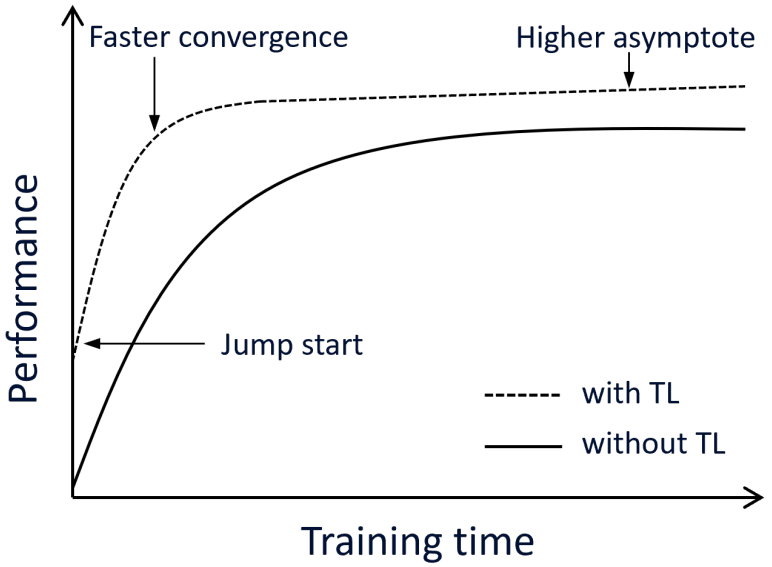

Figure 3.3: Three benefits of TL on training process

3.1.2 Importance and Applications

After a brief overview of TL, in this section, we will address why it is crucial to focus on TL research and continue applying it in practical use cases. We will also highlight specific application fields that particularly benefit with the help of TL.

3.1.2.1 Necessity of TL

The importance and necessity of applying TL can be illustrated in multiple aspects. It not only accelerates model development but also enhances performance, particularly in environments where data is scarce.

- **Efficiency:** TL dramatically reduces the computational resources needed. Traditional supervised ML models heavily depend on the availability of labeled data. However, manually labeling large datasets is time-consuming, laborious, and inefficient, posing significant challenges to training deep learning models. High-quality, large-scale data are often controlled by a limited number of companies or institutions, which poses a substantial barrier for smaller or startup companies. Due to insufficient data, developing a reliable model through general training is impossible for models targeting specific scenarios or applications. TL can identify similar labeled data within large datasets or find data from similar datasets that can address specific problems, thereby improving training outcomes.

- **Adaptability:** Moreover, TL can quickly adapt pre-existing solutions to new problems, showcasing remarkable flexibility. ML typically involves training a model with sufficient generalization capabilities on provided data to make accurate predictions in new, unknown scenarios and applications. However, developing a model that performs well in new environments based on limited training data is challenging. Under the scope of TL, DA and DG are methods designed to enhance the model generality. This dissertation will explore these aspects in depth and investigate the generalization methods of network solutions based on domain adaptation.

- **Performance Enhancement:** Models pre-trained on large datasets generally predict better and are more robust than those trained from scratch on smaller datasets. TL exploits this by fine-tuning pre-trained models to achieve higher accuracy and better generalization on new tasks. This widely observed benefit of TL enables the fast deployment of ML models.

- **Resource Limitation:** By reusing existing neural networks, organizations can optimize their computational resources, reduce energy consumption, and lower the environmental impact associated with training complex models from scratch. This is especially useful for startup companies or institutions with limited resources; because of the high demands for computational resources and large amounts of data, the development of some complex deep learning models is typically reserved for large-scale companies. These requirements become even more vital with the rise of LLM and visual-based CNN. TL offers a pre-train and fine-tuning mechanism that dramatically reduces the cost of training from scratch by transferring existing models from other use cases.

3.1.2.2 Applications

As an applicable technique in ML, the applications of TL extend across various fields. In this section, we list examples from multiple fields to illustrate the impact of applying TL on ML methods.

- **NLP**: TL has been particularly transformative in NLP. Traditional NLP faces numerous challenges related to data shortages or data biases. Since text information often exhibits domain specificity, a model trained on samples from one domain cannot be directly applied to another. As illustrated by the translator model example mentioned earlier in Fig. 3.2, if a new translator needs to be retrained for every pair of languages, problems such as insufficient data and difficulties in subsequent updates will frequently arise. Recent advanced models like Bidirectional Encoder Representations from Transformers (BERT) [25] and Generative Pre-Trained Transformer (GPT) [26] are pre-trained on vast text data banks and then fine-tuned for specific tasks such as text classification, summarization, and language understanding. This has led to unprecedented improvements in processing and understanding human languages.

- **CV**: The domain of CV has benefited significantly from TL. Visual images are more sensitive to model specificities than textual data. Factors such as shooting angle, lighting, background, and resolution can cause shifts in the statistical distribution of the visual data. Therefore, in the field of CV, TL is recommended to enhance model robustness. Pre-trained models on common CNN models, such as ImageNet [27], are commonly adapted for more specific tasks like detecting diseases from radiographic images or for autonomous driving systems where they help in object and pedestrian detection.

- **Healthcare**: In healthcare, TL is considered helpful in diagnostic procedures, particularly in the analysis of medical images where issues such as data scarcity and privacy are prevalent. The lack of labeled medical datasets due to confidentiality and ethical considerations often limits the development of robust models. Furthermore, healthcare data usually suffers from issues of non-renewability, as data collected under different systems or formats may not be easily combined, and patient data cannot be freely reproduced for research purposes. Despite these challenges, TL has proven a valuable tool. For instance, deep learning models trained on vast numbers of images have been fine-tuned to identify and diagnose specific pathologies in X-rays and MRI scans [28]. This adaptation process reduces the need for large-scale medical datasets and reduces the computational resources and time required to develop models from scratch.
- **Robotics**: Robotics employs TL to bridge the gap between simulation and real-world application. Robots trained in simulated environments can transfer their learned behaviors to physical environments, greatly reducing the risk and cost of direct training in unpredictable real-world settings.

3.2 Fundamentals of TL

As a subset of ML, the framework of TL is set up under the context of traditional ML frameworks but also introduces some new concepts. In this section, we will first introduce the basic concepts of TL by comparing them with ML definitions, then explore three fundamental questions in TL. Finally, we will discuss the basic solutions of TL.

3.2.1 Basic Concepts

TL introduces two new concepts, domain and task, which differentiate it from traditional ML approaches:

- **Domain:** A domain $\mathcal{D} := \{\mathcal{X}, \mathcal{Y}, P(\mathbf{x}, y)\}$ comprises an input feature space $\mathcal{X}$, a label space $\mathcal{Y}$, and the joint probability distribution of sample input $\mathbf{x}$ and its label y with $(\mathbf{x}, y) \sim P(\mathbf{x}, y)$, respectively. For any random sample $(\mathbf{x}_i, y_i)$, there is $\mathbf{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}$.
- **Task**: A task $\mathcal{T} := \{\mathcal{Y}, f(\cdot)\}$ includes the label space $\mathcal{Y}$ and the mapping function $f : \mathcal{X} \to \mathcal{Y}$, which defines the objective of the model within the corresponding domain.

In TL, the domain and task from which knowledge is transferred are usually referred to as the source domain and source task, respectively, while the domain and task to which the knowledge is transferred are termed the target domain and target task. Typically, the source domain possesses a large amount of general data or expertise, and the primary goal of TL is to transfer this knowledge from the source to the target domain. The definition of TL can vary across different works. In this dissertation, we formally define TL by extending and synthesizing the definitions from [29] and [30].

Definition 1 (Transfer Learning)**.** *Given a source domain* $\mathcal{D}_\mathsf{S} := \{\mathcal{X}_\mathsf{S}, \mathcal{Y}_\mathsf{S}, P_\mathsf{S}(\mathbf{x}, y)\}$ *with a sufficient number of samples* $\{(\mathbf{x}_k^\mathsf{S}, y_k^\mathsf{S}|_{k=1}^{n_\mathsf{S}})\}$ *for solving the source learning task* $\mathcal{T}_\mathsf{S} := \{\mathcal{Y}_\mathsf{S}, f(\cdot)\}$ *by minimizing the training loss* $l\big(f(\mathbf{x}), y\big)$ *over all samples referring to the model* f*:*

$$\begin{aligned} R_\mathsf{S}(f) :=& \mathbb{E}_{(\mathbf{x},y)\sim P_\mathsf{S}(\mathbf{x},y)}\Big[l\big(f(\mathbf{x}), y\big)\Big] \\ =& \int_y \int_x l\big(f(\mathbf{x}), y\big) \cdot P_\mathsf{S}\big(\mathbf{x}, y\big), \end{aligned} \tag{3.1}$$

and a target domain $\mathcal{D}_\mathsf{T} := \{\mathcal{X}, \mathcal{Y}, P_\mathsf{T}(\mathbf{x}, y)\}$ *which only has a limited number of samples* $\{(\mathbf{x}_k^\mathsf{T}, y_k^\mathsf{T}|_{k=1}^{n_\mathsf{T}})\}$ *with* $n_\mathsf{T} << n_\mathsf{S}$*, the objective of TL is to minimize the expectation of estimation loss in the target domain with* f*:*

$$\begin{aligned} R_\mathsf{T}(f) :=& \mathbb{E}_{(\mathbf{x},y)\sim P_\mathsf{T}(\mathbf{x},y)}\Big[l(f(\mathbf{x}), y)\Big] \\ =& \mathbb{E}_{(\mathbf{x},y)\sim P_\mathsf{S}(\mathbf{x},y)}\Big[\frac{P_\mathsf{T}\big(\mathbf{x}, y\big)}{P_\mathsf{S}\big(\mathbf{x}, y\big)} \cdot l\big(f(\mathbf{x}), y\big)\Big] \end{aligned} \tag{3.2}$$

As Eq. 3.2 illustrates, when $P_\mathsf{T}\big(\mathbf{x}, y\big) = P_\mathsf{S}\big(\mathbf{x}, y\big)$, TL in the target domain becomes a traditional ML problem that solves an optimal mapping function $f(\cdot)$ for samples from the same distribution. TL addresses the cases where $P_\mathsf{T}\big(\mathbf{x}, y\big) \neq P_\mathsf{S}\big(\mathbf{x}, y\big)$. In this dissertation, we won't cover the cases involving different input feature spaces $\mathcal{X}_\mathsf{S} \neq \mathcal{X}_\mathsf{T}$ or label spaces $\mathcal{Y}_\mathsf{S} \neq \mathcal{Y}_\mathsf{T}$, as our focus is more on the topic of DA.

In general, solving a TL problem can be considered as answering three fundamental questions: what to transfer, when to transfer, and how to transfer:

- **What to transfer:** This question addresses the type of knowledge or information that should be transferred from the source domain to the target domain. It is necessary to identify which parts of expertise, such as features, instances, or model parameters, are beneficial and relevant to the target task. Addressing this question upfront can ensure that the transfer is effective and does not negatively affect the learning process in the new context, known as negative transfer.
- **When to transfer**: Determining the appropriate time to transfer knowledge is essential. This question involves understanding under which scenarios the transfer of knowledge should occur. It requires evaluating the similarity and relevance of the source domain to the target domain and deciding if the transfer would improve the performance of the target model. The decision to transfer hinges on whether the source and target are sufficiently related to warrant the use of TL as opposed to training a new model from scratch.
- **How to transfer**: Considering how to implement knowledge transfer effectively is a critical aspect. This involves choosing the suitable algorithm or technique to facilitate TL. Methods can vary from transferring pre-trained models and fine-tuning them on the target task to more complex strategies like feature augmentation or transforming the feature space to align the source and target domains more closely. Each method has its merits and suitability depending on the specifics of the source and target tasks.

These three fundamental questions participate in the entire lifecycle of TL [31]. Addressing these questions before applying the transfer can lead to a smoother implementation of TL. Moreover, these questions are also helpful for evaluating the effectiveness of TL after the transfer is completed. By carefully considering these questions, we can also prevent negative transfer, which refers to failed TL procedures. Intuitively, we often aim to provide as much detailed information as possible from the source domain to achieve better results. However, transferring an abundance of knowledge does not always lead to positive improvements in the target domain [32].

This issue arises from the underlying mechanisms of how TL works. TL leverages the similarities between data across different domains to apply knowledge gained in the source domain to the unknown target domain. The key challenge is to accurately measure the similarity between these domains. Transferring more knowledge typically yields positive

results when the data similarity between the two domains is sufficiently high. Conversely, when domain similarity is limited, the effectiveness of TL can diminish significantly, sometimes even yielding worse results than training a new model from scratch in the target domain.

This detrimental effect, where knowledge transfer from the source to the target domain leads to negative effects, is known as negative transfer. Based on the statements in [31], negative transfer can be defined as follows:

Definition 2 (Negative Transfer)**.** *Denote the resulted error on target domain $\mathcal{D}_\mathsf{T}$ by transferring knowledge from source domain $\mathcal{D}_\mathsf{S}$ by implementing TL method Υ as $R\left(\Upsilon(\mathcal{D}_\mathsf{S}, \mathcal{D}_\mathsf{T})\right)$, negative transfer occurs when:*

$$R\left(\Upsilon(\mathcal{D}_\mathsf{S}, \mathcal{D}_\mathsf{T})\right) > R\left(\Upsilon^{'}(\Phi, \mathcal{D}_\mathsf{T})\right), \tag{3.3}$$

where Φ denote the empty set, and $\Upsilon^{'}$ is another algorithm. $R\left(\Upsilon^{'}(\Phi, \mathcal{D}_\mathsf{T})\right)$ denotes the derived target domain error without applying TL.

The main reason that causes negative transfer is the discrepancies between source and target domains while using inappropriate TL algorithms can also lead to negative transfer. Therefore, in practical applications, selecting domains with reasonable similarities and appropriate TL methods for effective knowledge transfer [33, 34] is crucial.

3.2.2 Rationale of TL

In this section, we will cover the basic rationales of TL solutions, starting from the nature of domain discrepancies and extending to discussions of some fundamental TL methods.

The basic idea of TL algorithms is to identify similarities between the source and target domains, using these similarities as a basis to rationally extend the common aspects to other parts of the domain. This necessitates the ability of TL to quantify and effectively utilize domain similarity. According to the definition of TL (see Definition 4), the difference between the source domain and the target domain is primarily reflected in the joint probability density, i.e., $P_\mathsf{T}(\mathbf{x}, y) \neq P_\mathsf{S}(\mathbf{x}, y)$. Consequently, the initial challenge that TL must address is measuring the difference between the two joint probability densities. Based on basic statistical knowledge, the relationship between joint probability distribution, marginal probability, and conditional probability is $P(\mathbf{x}, y) = P(\mathbf{x})P(y|\mathbf{x}) = P(y)P(\mathbf{x}|y)$.

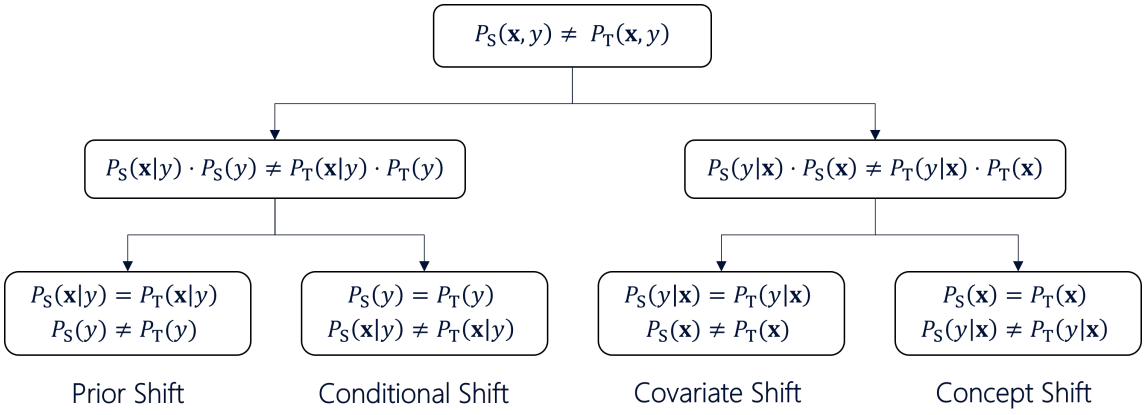

Figure 3.4: Categories of domain discrepancies according to joint distributions

Based on the assumptions of the TL problems, there must be a certain level of similarity between the source and target domains. This similarity is reflected in the probability distribution function, allowing us to categorize TL problems based on the nature of domain discrepancy [30], as illustrated in Fig. 3.4:

- **Prior Shift:** A prior shift occurs when the prior probabilities of the target classes differ from those in the source domain with $P_{\mathsf{S}}(y) \neq P_{\mathsf{T}}(y)$, while the conditional distributions are equivalent as $P_{\mathsf{S}}(\mathbf{x}|y) = P_{\mathsf{T}}(\mathbf{x}|y)$. This occurs rarely in practical use cases, where the data collection process or the population distribution changes over time or across different regions.

- **Conditional Shift:** In contrast to the prior shift, a conditional shift occurs when the conditional distributions of the input features given the class labels are different, while the prior probabilities of the labels remain the same, i.e., $P_{\mathsf{S}}(\mathbf{x}|y) \neq P_{\mathsf{T}}(\mathbf{x}|y)$, $P_{\mathsf{S}}(y) = P_{\mathsf{T}}(y)$. Conditional shift typically arises when the underlying conditions or contexts of data collection vary between the source and target domains. For instance, in CV tasks, changes in lighting, camera angles, or backgrounds can lead to a conditional shift.

- **Covariate Shift**: The covariate shift, also known as data drift, indicates scenarios where the marginal distributions of source and target domains are different $P_{\mathsf{S}}(\mathbf{x}) \neq P_{\mathsf{T}}(\mathbf{x})$, but their conditional distributions are identical as $P_{\mathsf{S}}(y|\mathbf{x}) = P_{\mathsf{T}}(y|\mathbf{x})$. This kind of domain discrepancy often occurs in real-world scenarios where specific running systems change or dynamic environment conditions evolve over time. The difference caused by network condition changes often leads to this domain drift.

- **Concept Shift**: Concept shift, also known as label drift, occurs when the input feature distribution of source and target domains remains the same as $P_{\mathsf{S}}(\mathbf{x}) = P_{\mathsf{T}}(\mathbf{x})$ while resulting in different label distributions $P_{\mathsf{S}}(y|\mathbf{x}) \neq P_{\mathsf{T}}(y|\mathbf{x})$. This shift means that even if the distribution of the input features remains stable, how those features relate to the output changes, potentially degrading the performance of a model trained on data sequences varying over time.

In response to the various types of domain discrepancy discussed above, TL has introduced several methods that leverage the correlations between domain samples to develop models that perform effectively in the target domain. The most widely used methods include sample re-weighting, the pre-train and fine-tuning approach, and representation (feature) learning:

- **Sample Re-weighting:** The idea of the sample re-weighting method is to enhance the participation of source domain samples that are more similar to those in the target domain during training. In practice, this method adjusts the frequency of samples in the training set based on their relevance to the target domain or modifies the training loss attributed to those samples. Relevant samples in the source domain for re-weighting can be collected based on prior knowledge if it is available. Some studies [35, 36] suggest using RL to manage this sample selection process by formulating a sample selector as an RL agent and using sample similarity as the reward metric.

- **Pre-train and Fine-tuning:** As one of the most frequently used TL methods, pre-train and fine-tuning initially train the models on the source domain with a large, generic dataset and fine-tune it on the target domain with a smaller, domain-specific dataset. This method leverages the broad generalizability learned during pre-training and adapts it to the specificity of the target domain during fine-tuning. This training mode has been widely used in ML applications. For example, BERT and GPT models both apply this method to enhance the generality of NLP models.

- **Representation Learning:** The representation learning method aims to minimize the difference between the source and target domains by solving a feature transition

that maps both into a common feature space that reduces domain variance while enhancing task-relevant information. This approach often involves extracting representative features from source domain samples and modifying the feature space to create robust and transferable representations across domains. Throughout this process, various distribution difference metrics, such as Euclidean distance, Kullback-Leibler Divergence (KL-divergence), and mutual information, are utilized to quantify domain discrepancies.

3.3 Selected TL Methods

In this section, we delve deeper into pre-train fine-tuning and representation learning methods, as they are more applicable in solving practical problems. Subsequently, we will discuss the ways of evaluating a TL approach and provide its open topics, including the TL methods for RL.

3.3.1 Pre-train and Fine-tuning

The method of pre-train and fine-tuning is generally split into two distinct phases: pre-training, where a model is trained on a large, often generic dataset, and fine-tuning, where the pre-trained model is adapted to a specific, often smaller, target task:

- **Pre-train:** During the pre-train phase, the model f characterized by parameters θ_P is trained on a large dataset regarded as source domain $\mathcal{D}_\mathsf{S}$ that is typically diverse and covers a broad range of topics or features. This dataset does not necessarily need to be closely related to the target tasks for which the model will eventually be used. The learning process of the pre-trained model f_{θ_P} respects to cost function $\mathcal{L}$ can be denoted as:
$$\theta_P^* = \arg\min_{\theta_P} \mathcal{L}(\mathcal{D}_\mathsf{S}; \theta_P). \tag{3.4}$$
Here, the model aims to learn a rich and robust representation of the input data features. For instance, models like BERT or GPT are pre-trained on extensive collections of text sourced from books, Wikipedia, or websites, allowing these models to comprehensively understand language dynamics, grammar, and context.

- **Fine-tuning:** The fine-tuning phase begins once the pre-trained model has converged. The target model f_{θ_F} is fine-tuned by using the parameters of the pre-trained model as the initial starting point, denoted as $\theta_F^0 = \theta_P^*$, and then trained on the target domain $\mathcal{D}_\mathsf{T}$ with a much smaller dataset. The training process on the target domain is then formulated as follows:
$$\theta_F^* = \arg\min_{\theta_F} \mathcal{L}(\mathcal{D}_\mathsf{T}; \theta_F | \theta_F^0 = \theta_P^*). \tag{3.5}$$
During fine-tuning, the model parameters are slightly adjusted to optimize performance for the specifics of the target task. This can involve minimal changes to the model's weights or require more significant adaptations, depending on the similarity between the pre-training data and fine-tuning data. In some cases, only the top layers of the model are adjusted, while in others, the entire network may continue to learn and adapt.

Sometimes, we can use the first few layers of the pre-trained model to initialize the target model and then fine-tune the subsequent layers. This is because the initial layers typically extract more general information, whereas the higher layers may contain domain-specific

features. This approach also addresses the fundamental question of what to transfer in TL. The pre-train and fine-tuning method is particularly popular in fields where obtaining large amounts of labeled data is difficult or expensive.

The pre-trained models on large datasets generally perform better on specific tasks, even with smaller training data sets, because they have already learned useful representations. And fine-tuning a pre-trained model is usually much faster and computationally cheaper than training a model from scratch on a smaller dataset. Besides, the same pre-trained model can be adapted to multiple tasks, making it a generalizable tool for various applications. However, this TL approach may risk overfitting during the fine-tuning phase, especially if the target dataset is very small or if the task is significantly different from the pre-training task. There's also the question of how much of the model to fine-tune and how to adjust the learning rates, which can significantly affect the final performance.

3.3.2 Representation Learning

Representation learning, also known as feature learning, focuses on automatically discovering the representations needed for feature detection or classification from raw data. In TL, especially for DA, it helps to leverage helpful information from one task (source domain) to improve performance on another (target domain), even when the direct difference between the two domains is enormous. In general, it involves developing a model f_R that can identify and extract a latent feature space $\mathcal{Z}$ from the source domain $\mathcal{X}$ with $f_R : \mathcal{X} \to \mathcal{Z}$, where $\mathcal{Z}$ may not be relevant to the source task but is also beneficial for the target domain task. A general process of representation learning can be illustrated in Fig. 3.5.

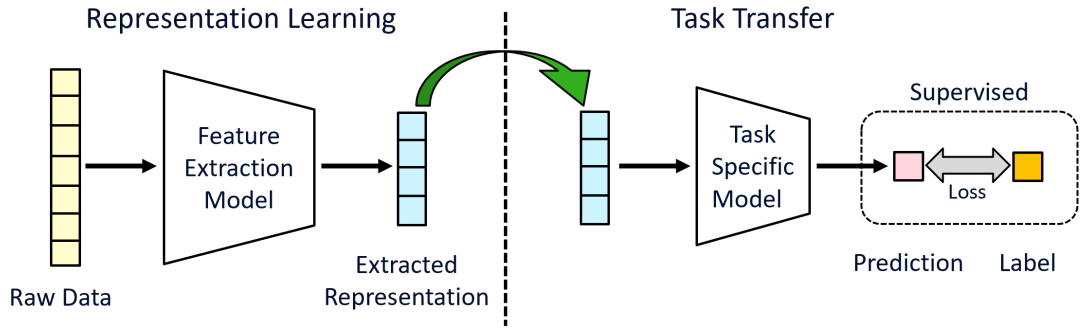

Figure 3.5: Overview of general representation learning process

Representation learning does not have a unified pipeline of solution derivation. Several techniques are employed to achieve effective transfer of knowledge:

- **Autoencoder:** Autoencoder [37] is a type of neural network used to learn efficient coding of unlabeled data, typically for dimension reduction or feature learning. Autoencoder has two main components: the encoder h and the decoder g. The encoder aims to compress the input data into a smaller, encoded representation with $z = h(\mathbf{x})$, where $z \in \mathcal{Z}$ is a compact version of the input data but retains most of its significant information. On the other hand, the decoder attempts to reconstruct the input data from this encoded representation by minimizing $\|\mathbf{x} - g\left(h(\mathbf{x})\right)\|_2^2$. In the context of representation learning, autoencoders can learn to encode the input data into a set of features that can be useful for classification or other predictive tasks.

- **Transformer:** The transformer model, introduced in [38], relies fundamentally on the self-attention mechanism. This allows each position in the input sequence to attend to every other position, thereby capturing intricate relationships in the data. A transformer model typically consists of an encoder and a decoder, but different from the autoencoder, each layer in the encoder and decoder is coupled with the

attention mechanism. Originally developed for NLP tasks, transformers learn contextual relationships between elements in a sequence. They are particularly effective for tasks that require understanding the context and relationships within data, which provides rich feature representations that can be adapted to different tasks beyond language processing.

In the field of domain adaptation, representation learning is frequently employed to identify common latent features in samples from two domains. However, if the source and target domains are substantially different, the learned representations may not transfer effectively. Additionally, the performance of TL is highly correlated with the representative of the extracted features. Complex models might overfit the differences in the source domain that are irrelevant to the target domain.

3.3.3 Other TL Methods

Feature Transformation Feature transformation [39] methods modify the feature space to align the source and target domains more closely. This approach often involves extracting representative features from source domain samples and adjusting the feature space to create robust and transferable representations across domains. Various distribution difference metrics, such as Euclidean distance, KL-divergence, and mutual information, are utilized to quantify domain discrepancies. Popular feature transformation techniques include:

- **Principal Component Analysis (PCA):** PCA reduces the dimension of the data while retaining most of the variability in the dataset, thereby aligning the feature space of the source and target domains.
- **Kernel PCA:** An extension of PCA that uses kernel methods to capture non-linear relationships in the data, making it more flexible for aligning complex feature spaces.
- **Variational Autoencoder (VAE):** VAE learns a probabilistic latent space from which they can generate new samples. This probabilistic approach helps create smooth and continuous feature representations, aiding in the alignment of the source and target domains.
- **VIB:** VIB is a method that maximizes the mutual information between the input and a compressed latent representation while minimizing the information about the input. This helps in learning robust feature representations that are invariant to domain shifts.

Generative Adversarial Net (GAN)
GAN [40] is a powerful tool for representation learning, particularly in generating new data samples that resemble the training data. A GAN network consists of a generator and a discriminator. The generator creates new data samples while the discriminator evaluates them against real data. This adversarial process helps the generator learn to produce realistic data, which can then be used to augment the target domain data or improve feature extraction.

Domain Adversarial Training
Similar to the concept of GAN, domain adversarial training methods use adversarial learning to make the features of the source and target domains indistinguishable. The main idea is to train a feature extractor that can produce domain-invariant features, which are then used to train the classifier. The process involves min-max fairness between two components: a feature extractor and a domain discriminator. Specifically, Domain Adversarial Neural Network (DANN) [41] integrate domain adversarial training into the neural

network architecture. The feature extractor learns to produce features that confuse the domain discriminator, while the domain discriminator learns to distinguish between source and target domain features.

Few-shot Learning
Few-shot learning [42] focuses on training models that can be generalized from a few training examples. Techniques such as prototypical networks and meta-learning enable models to quickly adapt to new tasks with limited data. Few-shot learning is particularly beneficial in TL, where the target domain often has scarce labeled data.

Deep Domain Confusion
Deep domain confusion methods aim to reduce domain discrepancy by incorporating domain confusion loss into the training process. This loss penalizes the model if it can distinguish between the source and target domains, encouraging it to learn domain-invariant features. Maximum Mean Discrepancy (MMD) measures the distance between two distributions. The MMD loss is added to the training objective in deep domain confusion methods to minimize the discrepancy between the source and target domain features.

The various TL methods, ranging from feature transformation to domain adversarial training, offer a robust toolkit for addressing domain discrepancies. These techniques enhance the transferability of models across different domains, thereby improving performance and generalization in target tasks. However, despite the advances in TL, several open topics and challenges remain, including:

- **Understanding Negative Transfer:** Investigating the causes of negative transfer and developing methods to mitigate its effects.
- **Transferability Across Domains:** Exploring how knowledge can be transferred across vastly different domains, especially in cases with limited labeled data.
- **Scalability:** Developing scalable TL methods that can handle large-scale datasets and complex models efficiently.

Additionally, applying TL to RL approaches is more complex compared to ML models. In the next section, we will specifically discuss TL methods for RL.

3.3.4 TL for RL

In this dissertation, we focus on applying TL to network optimization solutions, particularly RL-based solutions. Combining TL with RL can significantly enhance learning efficiency and effectiveness, especially in complex environments where learning from scratch can be prohibitively costly in terms of time and resources [43, 44].

TL enables RL agents to benefit from pre-existing knowledge and experiences, reducing the need for extensive interactions and enhancing sample efficiency. TL in RL can facilitate better initial performance, making it particularly valuable for domains with prohibitive sampling costs or safety concerns. By transferring knowledge between domains, RL frameworks can adapt and generalize more effectively, addressing the exploration-exploitation dilemma and improving overall performance. Here are some examples of successful applications of TL to RL:

- **Policy Transfer:** The methods of policy transfer directly apply a policy trained in one environment to another. More advanced versions, such as policy distillation, involve training a more compact and often faster policy that imitates a more complex

policy. It can be particularly useful when transferring between different types of environments or when simplifying policies for deployment. Following this method, [45] enabled policy transfer from complex teacher networks to simpler student networks, which can perform efficiently in similar tasks.

- **Representation Transfer in RL**: State representation learning focuses on developing a mapping from raw states into a feature space that encapsulates the essential dynamics of these states. This process facilitates faster learning in new but similar tasks. Universal value function approximation [46] is designed to generalize across both states and goals by learning a value function that is conditioned on both state and goal. This approach has demonstrated effectiveness in settings where agents need to adapt to varying goals within the same environment.

- **Reward Shaping**: Reward Shaping modifies the reward structure of the source task to more closely align with the target task, facilitating smoother policy transfer. Before applying a learned policy to a new task, the task inference method assesses how the new task relates to previously learned tasks and adapts the policy accordingly. These strategies work in tandem to enhance the effectiveness and efficiency of policy application across different tasks. [47] explores how knowledge from one task can be effectively transferred to improve learning in a different but related task. The adaptive utilization of shaping RL rewards aligns with the concept of TL, where insights gained from one domain can be leveraged to enhance performance in a different domain. The study demonstrates the potential for TL to optimize rewards in various environments by adapting shaping weights across different states and actions.

- **Multi-task RL**: This approach trains a single agent on multiple tasks simultaneously, encouraging the development of a generalized policy or value function. In multi-task RL, the process involves leveraging knowledge gained from previous tasks to enhance learning efficiency in new tasks. [48] use pre-trained language models to capture task context and project it to a lower-dimensional space, demonstrating the application of TL techniques in contextual representation learning. By incorporating task metadata to decide which information should be shared across tasks, the proposed method aims to alleviate negative interference, a common challenge in multi-task learning, showcasing the importance of TL concepts in improving multi-task RL performance.

Despite these successes, applying TL to RL is considered challenging in practical use cases. The difficulties include:

- **Domain Differences:** Variations in state space, action space, reward functions, transition dynamics, initial states, and trajectories between the source and target domains can hinder effective transfer.

- **Knowledge Transfer Quality:** Determining the amount and quality of necessary knowledge required for effective TL is crucial. Inadequate or excessive transfer can negatively impact the performance of the target agent.

- **Framework-specific Nature:** Many TL approaches are framework-specific, meaning that certain methods may only be applicable to specific RL algorithms designed for discrete or continuous action spaces.

In this dissertation, we aim to address these challenges by applying TL to RL-based resource allocation methods within the context of 5G slicing scenarios. This involves developing and evaluating techniques that can effectively transfer knowledge across varying network conditions, ensuring robust and efficient performance in dynamic and complex environments.

4. RAN Slice Resource Allocation with RL

In this chapter, we explore the application of RL methods to RAN slice resource management problems and conduct preliminary experiments for validation. First, we recap the motivation for applying RL methods to slice resource allocation to improve network performance, along with the associated challenges, in Section 4.1. In Section 4.2, we introduce the system model for multi-cell multi-slice scenarios, followed by the problem formulation for optimizing slice resource allocations in Section 4.3.

To address Question 1, in Section 4.4, we propose the MDP formulation for the slice resource allocation problem in multi-cell multi-slice scenarios in a centralized manner. We then discuss a distributed DQN-based resource allocation solution, including a brief introduction to the practical implementation process of distributed RL in the simulated network scenario. In Section 4.5, we present the numerical results from the preliminary experiments of the distributed DQN-based slice resource allocation solution. These results aim to validate the experimental environments and evaluate the effects of applying different definitions of RL rewards.

4.1 Motivation

As mentioned in Section 1.1, resource allocation for different slices in mobile communication networks is crucial for enhancing individual slice Key Performance Indicators (KPIs) and overall system performance. This is especially important for Self-Organizing Networks (SON), where dynamic systems need to adjust per-slice configurations to achieve better or more stable network KPIs. Traditionally, real-time network modifications require expert intervention, leading to high human resource and time costs. In advanced networks, such as SON systems, the network can adjust configurations based on its behavior, but these networks still face significant constraints in self-regulation.

In Chapter 2, we introduce the concept of RL, from basic to advanced algorithms. Given its effectiveness in solving dynamic optimization problems, RL is highly suitable for addressing issues in dynamic systems, as well as the SON problems as Fig. 4.1 indicated. RL algorithms are based on the concept of MDPs, enabling systems to learn optimal behavior based on the state of the environment. By modeling network systems as RL agents with appropriate formulations, networks can learn to reconfigure optimally in response to gradual changes in different environments. However, applying RL to RAN network scenarios is challenging due to the high complexity of these systems, as discussed in Section 2.6.

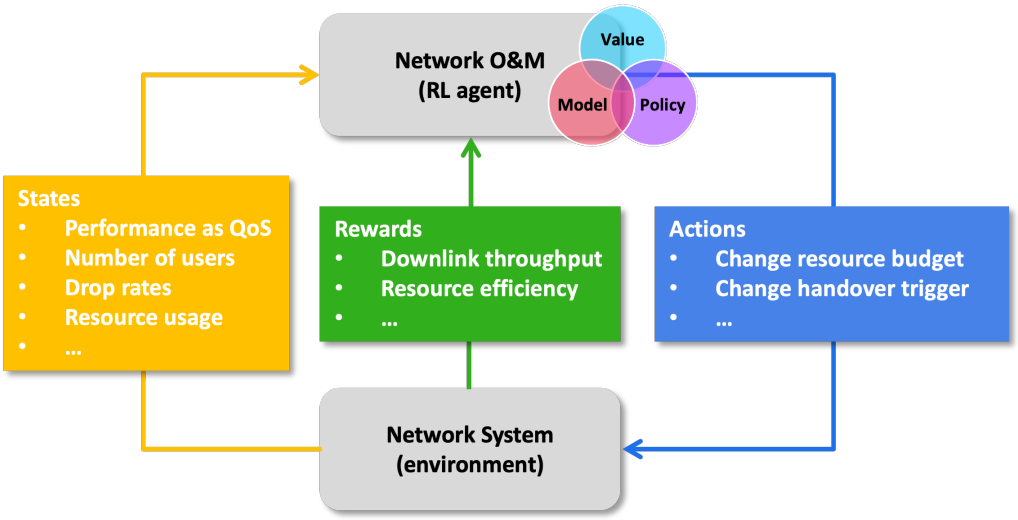

Figure 4.1: Framework of RL-based solution for SON problem

In this chapter, we aim to address Question 1, i.e., exploring the method of formulating the slice resource allocation problem such that RL can handle it. Additionally, we will focus on the parameter settings for per-slice resource budgets in a network slicing scenario. To analyze the influence of per-slice resource budget configurations on network slicing behaviors, we conducted a sanity check experiment examining the corresponding per-slice KPIs such as downlink throughput, number of active users, and actual slice load. The experiments were carried out using the SEASON II simulator [49]. Specifically, to test the feasibility of applying RL algorithms to the network slice resource allocation problem, we implemented a distributed DQN algorithm as a trial experiment for single cells to modify the configuration of per-slice resource budget parameters.

4.2 System Model

Assume a set of base stations (hereafter also referred to as cells, indicating the geographic area covered by the base station) $\mathcal{B} := \{1, 2, \ldots, B\}$ and a set of slices $\mathcal{N} := \{1, 2, \ldots, N\}$. At each time t, each cell $i \in \mathcal{B}$ observes its local state $\mathbf{s}_i(t)$, and let $\mathbf{s}(t) := [\mathbf{s}_1(t), \ldots, \mathbf{s}_B(t)] \in \mathcal{S}$ denote the state of the overall system, where $\mathcal{S}$ denotes the state space. The state of the system $\mathbf{s}(t)$ is controlled by the configured RAN slice resource budget $\mathbf{a} := [\mathbf{a}_1, \ldots, \mathbf{a}_B] \in \mathcal{A}$, where $\mathcal{A}$ denotes the space in which $\mathbf{a}$ lies. Each cell $i \in \mathcal{B}$ can choose its local configuration $\mathbf{a}_i := [a_{i,1}, \ldots, a_{i,N}]$. Note that the resource budget $a_{i,n}$ indicates the percentage of RAN resources as PRB allocated to the n-th slice in the i-th cell, and we have $\sum_{n \in \mathcal{N}} a_{i,n} \leq 1$ for each $i \in \mathcal{B}$.

The environment operates on discrete time steps $t = 0, 1, \ldots$, where at each step, the system selects its RAN slice resource budget $\mathbf{a}(t)$ based on a full observation (i.e., without information loss) of the state $\mathbf{s}(t)$. The dynamics of the environment are governed by the conditional probability mass function:

$$P\big(\mathbf{s}(t+1)|\mathbf{s}(t), \mathbf{a}(t)\big) = \Pr\big\{S_{t+1} = \mathbf{s}(t+1)|S_t = \mathbf{s}(t), \mathbf{a}(t)\big\}, \tag{4.1}$$

where $P(\cdot)$ defines the probability that the slice configuration $\mathbf{a}(t)$ in system state $\mathbf{s}(t)$ will lead to the next state $\mathbf{s}(t+1)$, and S_t denotes the random variable of the system state at time t.

Given a deterministic utility function $r(\cdot) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that indicates the system utility, the network system can be defined in terms of an MDP, determined by the tuple $\{\mathcal{S}, \mathcal{A}, P(\cdot), r(\cdot)\}$.

4.3 Optimization Problem Formulation

To achieve the optimal slice resource allocation solution, we aim to find an effective policy $\pi_{\mathbf{a}} : \mathcal{S} \rightarrow \mathcal{A}$ that helps determine the RAN slice resource budget $\mathbf{a}$ based on the network state $\mathbf{s}$. The objective is to maximize a cumulative function of the defined utility (or reward), typically the expected discounted sum over a potentially infinite horizon. The problem can be formulated as:

$$\max_{\pi_{\mathbf{a}}} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r\big(\mathbf{s}(t), \mathbf{a}(t)\big)\right] \quad \text{s.t.} \sum_{n \in \mathcal{N}} a_{i,n}(t) \leq 1, \forall i \in \mathcal{B}, \ldots \tag{4.2}$$

where γ is the discount factor with $0 \leq \gamma \leq 1$.

4.4 RL-based Slice Resource Allocation Optimization

To solve the problem formulated above, we aim to maximize the accumulated utility or reward, which indicates network performance, by adjusting the per-slice resource budget $\mathbf{a}$ in real time. We consider applying RL approaches to this optimization problem in dynamic systems.

4.4.1 RL Formulation

We define the state, action, and corresponding reward function as follows:

- **State:** The states are selected from the network KPIs, which can indicate or influence the utility function:
$$\mathbf{s}_i := [s_{i,1}, s_{i,2}, \ldots, s_{i,M}] \in \mathcal{S}_i \tag{4.3}$$
where M is the number of KPIs we consider as state indicators. For example, we may include KPIs such as cell load and throughput in the cell state. The joint cell states have a state space of $\mathcal{S} := \prod_{i \in \mathcal{B}} \mathcal{S}_i$.

- **Action:** The action is defined as the RAN slice resource budget $\mathbf{a}$. Each cell $i \in \mathcal{B}$ can choose its local configuration $\mathbf{a}_i := [a_{i,1}, \ldots, a_{i,N}]$. The per-cell per-slice action should satisfy $a_{i,n} \in \Omega$, where Ω can be defined as continuous space $\Omega := [0, 1]$ or a finite set of pre-defined discrete values within [0, 1]. The sum of resource budgets for each cell should be less than or equal to 1, so the action space for each cell is:
$$\mathcal{A}_i := \left\{ \mathbf{a}_i \in \Omega_i^N : \sum_{n \in \mathcal{N}} a_{i,n} \leq 1 \right\} \tag{4.4}$$
The joint cell actions have the corresponding action space of $\mathcal{A} := \prod_{i \in \mathcal{B}} \mathcal{A}_i$.

- **Reward:** We introduce two types of reward functions: resource efficiency and downlink average throughput (over users) of the mobile network systems.

  1. **Downlink Average Throughput:** Let the downlink average throughput over all users in cell i be denoted as $\phi_i(t)$ in Mbit/s. The reward based on the throughput is:
  $$r(t) := \sum_{i \in \mathcal{B}} \phi_i(t) \tag{4.5}$$

2. The resource efficiency for each cell is calculated from the per-slice downlink throughput $\phi_{i,n}(t)$ and the real slice load $l_{i,n}(t)$ in the network systems, where the load is defined as the percentage of frequency resource occupied in a cell. The resource efficiency (in Mbit/s/Hz) is denoted as:

$$\eta_{i,n}(t) := \frac{\phi_{i,n}(t)}{l_{i,n}(t) \times b} \quad \text{for } i \in \mathcal{B} \text{ and } n \in \mathcal{N} \tag{4.6}$$

where b is the bandwidth (in Hz) of the network systems. For each cell $i \in \mathcal{B}$, we take the weighted sum of resource efficiency for each slice:

$$\eta_i(t) := \sum_{n \in \mathcal{N}} \beta_n \eta_{i,n}(t) \tag{4.7}$$

where β_n is the weight factor for each slice, satisfying $\sum_{n \in \mathcal{N}} \beta_n = 1$. Thus, the reward function for resource efficiency is:

$$r(t) := \sum_{i \in \mathcal{B}} \eta_i(t) \tag{4.8}$$

Hereby, we give our answer to the Question 1: By defining the states, actions, and rewards in this manner, the slice resource allocation problem can be transformed into an RL problem, where the RL agent aims to learn the optimal policy $\pi_{\mathbf{a}}$ to maximize the accumulated reward. This approach leverages the ability of RL algorithms, such as DQN, to handle complex dynamic systems and optimize resource allocation effectively.

4.4.2 DQN-based Solution

As discussed in 2.3.5, DQN is a widely used RL algorithm that combines the principles of conventional Q-Learning with DNN. By replacing the Q-table with a Q-Network, DQN can effectively handle RL problems with larger state and action spaces. To train the DNN model with parameters θ, collecting a sufficient amount of samples in state-action pairs $(\mathbf{s}, \mathbf{a})$ is essential. The loss function $J(\theta)$ used to update the network parameters is defined as:

$$J(\theta) := \mathbb{E}\left[(\mathbf{y} - \hat{Q}(\mathbf{s}, \mathbf{a}; \theta))^2\right] \tag{4.9}$$

DQN implements the ϵ-greedy policy for sample collection by the RL agents. The network parameters are updated iteratively using the Stochastic Gradient Descent (SGD) method. The target Q-value $\mathbf{y}$ is calculated based on the RL discounted reward, incorporating the discount factor γ, similar to the Q-value update method in Q-Learning. For an RL agent at step j, the target Q-value is:

$$\mathbf{y}_j = r_j + \gamma \max_{\mathbf{a}_{j+1}} \hat{Q}(\mathbf{s}_{j+1}, \mathbf{a}_{j+1}; \theta) \tag{4.10}$$

The DQN model utilizes a double network structure to stabilize target Q-value updates and predictions, involving the primary network with parameters θ and the target network with parameters $\hat{\theta}$. The parameters of both networks are periodically synchronized by copying θ to target network $\hat{\theta}$. This double network architecture is illustrated in Fig. 4.2. This architecture helps reduce the overestimation bias and improve the stability and performance of learning process. By employing the DQN algorithm, we aim to effectively optimize the resource allocation for network slices in dynamic environments.

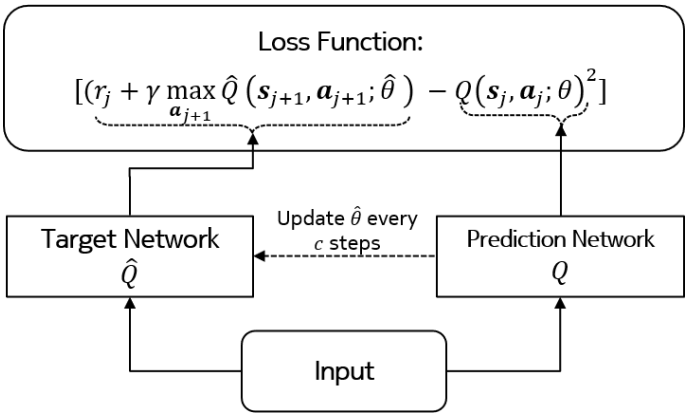

Figure 4.2: Double network structure of DQN model

4.4.3 Distributed DQN Approach

Solving the optimal per-slice resource budget for all cells in a network system is a challenging problem. If the entire system is formulated as a single RL agent (a centralized approach), the dimensions of both the state space $\mathcal{S}$ and the action space $\mathcal{A}$ would be extremely large. This high dimensionality would result in significantly longer convergence times for the RL algorithm, potentially preventing it from reaching an optimal solution.

At this stage, we only consider the local state within each cell as the input to the DQN. Later studies will explore coordinated multi-agent RL, incorporating both local observations and selected observations from other cells. For each cell $i \in \mathcal{B}$, the corresponding RL agent for the distributed DQN has the state $\mathbf{s}_i \in \mathcal{S}_i$, the action $\mathbf{a}_i \in \mathcal{A}_i$, and two options for the reward r_i: downlink average throughput or resource efficiency.

After formulating the RL agent, the new framework for the per-slice resource budget optimization problem is illustrated in Fig. 4.3:

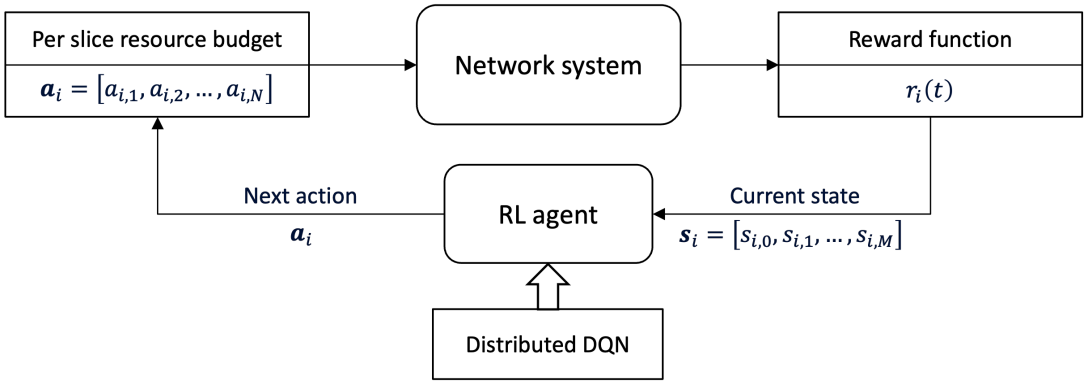

Figure 4.3: RL-based slice resource allocation optimization for single cell scenario

4.5 Experiments

Implementing the proposed RL algorithms directly on actual mobile networks is impractical due to the potential risks and resource constraints. The real network environments are highly complex, with numerous interconnected components and dynamic conditions that challenge RL algorithms, which require substantial data and computational resources to learn optimal policies. High dimensional state and action spaces make convergence to optimal solutions difficult and time-consuming. Additionally, ensuring network reliability is crucial, as disruptions can have widespread impacts on businesses, emergency services, and daily communications. RL algorithms, especially during exploration, can degrade service quality or even cause network outages, posing unacceptable risks. Furthermore, testing

RL algorithms in real networks is costly, requiring extensive monitoring and expert intervention to manage potential negative impacts, leading to significant operational expenses and prolonged instability.

4.5.1 Experiment Setup

As an alternative option, network simulators offer a controlled, reproducible environment where various network conditions can be tested without affecting actual users. This allows for rigorous experimentation and fine-tuning of RL algorithms. They eliminate the risk of real-world disruptions, enabling the safe exploration of different strategies. Simulators can scale to accommodate large networks and complex scenarios, providing flexibility in adjusting network parameters and conditions for comprehensive testing. They are cost-effective, reducing the need for extensive hardware, minimizing operational expenses, and accelerating development with immediate feedback and parallel experimentation. Additionally, simulators generate large volumes of synthetic data for training RL algorithms, including rare or extreme conditions, enhancing their robustness. Therefore, in this dissertation, we deployed a virtual environment capable of simulating real network system behaviors.

4.5.1.1 Season II Simulator

To imitate the complex behaviors of network systems, we use a system-level simulator, Season II [49], as the virtual environment. Season II was initially designed to address and study issues in SON. It is also capable of evaluating and visualizing network behaviors.

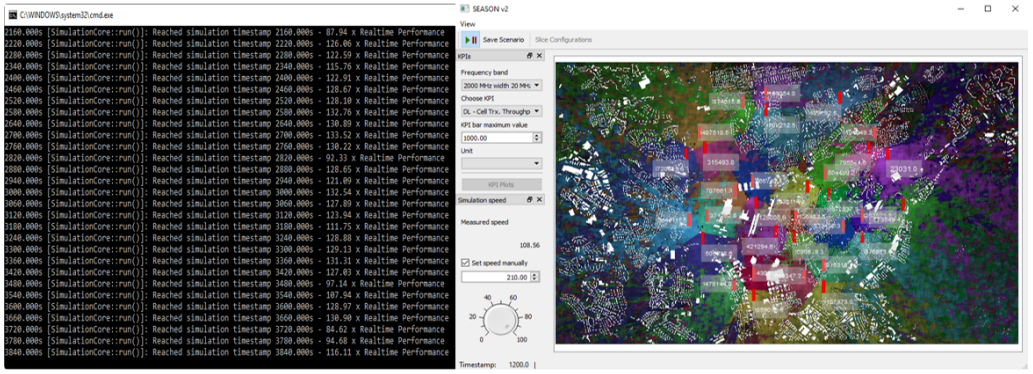

Figure 4.4: Graphic user interface and console of Season II

The main features of the Season II simulator include dynamic implementation, runtime reconfiguration, and two-way real-time communication. The real-time communication feature is particularly important for our use case, allowing us to receive network KPI reports and send new configurations periodically. Additionally, Season II integrates functions for multiple network simulation environments and scenarios, various service groups of users, and adjustable network parameters such as antenna settings, mobility (handover) parameters, network slicing parameters, and user group settings. The simulator processes simulations based on parameters in JSON format files and can dynamically react to configuration changes during the simulation. In summary, the ability of Season II to both report network KPIs and receive new configurations simultaneously makes it an ideal tool for our experiments.

4.5.1.2 Network Environment Setup

As for the environment setup of the experiments, we built a simple scenario of a mobile communication network system based on the Season II simulator, which contains 5 sites in total, and each site consists of 3 cells. The set of cells is noted as $\mathcal{B} = [1, 2, \ldots 15]$. The system is configured with two network slices, $\mathcal{N} = 1, 2$. Slice 1 has an expected bit rate of

5 Mbit/s, while slice 2 has an expected bit rate of 3 Mbit/s. The default resource budgets allocated to slice 1 and slice 2 are $a_{i,1} = 0.8$ and $a_{i,2} = 0.2$, respectively. We included 100 UEs in the scenario, with group A consisting of 60 UEs served by the slice 1 and group B consisting of 40 UEs served by the slice 2. These UEs move between different cells in the test environment according to specific traffic models. The slice settings are summarized in Table 4.1. This simple environment setup is intended to test the characteristics of the Season II simulator under dynamic system behavior and to implement preliminary RL algorithm experiments. Please note that in this trial experiment, we do not specify the types of slices; instead, we define them by their QoS requirements in terms of throughput and delay. This setup can be easily extended to a broader range of slice types.

Table 4.1: Slice Configurations

Slice name	Default Resource Budget	Expected Throughput	Number of UEs
Slice 1	0.8	5 Mbit/s	60
Slice 2	0.2	3 Mbit/s	40

4.5.1.3 Slice Resource Allocation Formulation

Based on the experimental scenario built in the Season II simulator, we update the formulation of the RL agent for further experiments. The utility functions we aim to optimize are set to reflect the behaviors and KPIs of the network systems. In the Season II simulator, network KPIs can be read from the returned reports, and the corresponding settings for the per-slice resource budgets can be modified by configuring the per-slice resource budget for each cell. We can formulate the RL agent with corresponding elements with these periodic reports and real-time configurations.

Based on our experience with mobile network systems, we select KPIs from the simulator reports related to the utility or reward functions of the RL agent. The selected KPIs for each slice $n \in \mathcal{N}$ in each cell $i \in \mathcal{B}$ as RL agent states are:

- **Per-slice Throughput:** As the throughput $\Phi_{i,n}$ from all UEs of slice $n \in \mathcal{N}$;
- **Slice Load:** The slice load $l_{i,n}$ indicates the actual percentage of resources that are occupied by slice $n \in \mathcal{N}$;
- **Per-slice Number of UEs:** The number of active UEs served by each slice, denote as $u_{i,n}^{(3)}$ for $n \in \mathcal{N}$.

For the action space of the RL agent for each cell, the DQN algorithm uses the benefits of DNNs to replace the Q-table in conventional Q-Learning, yet it still has limits on the dimension of the action space. Thus, we constrain the action space of distributed DQN to a discrete space. For convenience, we consider the case where the settings of the slice resource budgets are actions in $\mathbf{a}_i \in \Omega^2, i \in \mathcal{B}$, where $\Omega := 0.0, 0.1, \ldots, 1.0$ is a finite discrete space, and the action must strictly abide by the restriction of $\sum_{n \in \mathcal{N}} a_{i,n} = 1$.

The customized RL agent for each cell $i \in \mathcal{B}$ is defined as:

- **State**: $\mathbf{s}_i := [s_{1,1}, ..., s_{i,n}]$ with $s_{i,n} = [\Phi_{i,n}, l_{i,n}, u_{i,n}]$, $n \in 1, 2, k \in 1, 2, 3$.
- **Action**: $\mathbf{a}_i := [a_{i,1}, a_{i,2}]$ with an action space of

$$\mathcal{A}_i := \left\{ \mathbf{a}_i \in \Omega^2 \Big| \sum_{n=1,2} a_{i,n} = 1; \Omega = \{0.0, 0.1, \ldots, 1.0\} \right\}, \tag{4.11}$$

  where the cardinality of the action space is $|\mathcal{A}_i| = 11$.

- **Reward**:
  - Downlink Average Throughput of cell $i \in \mathcal{B}$: $r_i(t) := \phi_i(t)$.
  - Resource Efficiency of cell $i \in \mathcal{B}$: $r(t) := \eta_i(t)$.

The formulation above ensures that the RL agent for slice resource allocation aligns with the practical constraints and objectives of the simulated network environment, allowing us to effectively evaluate the proposed DQN algorithm.

4.5.2 Experiment I: Sanity Check

Before implementing the proposed solution with distributed DQN, we performed a sanity check with the experimental environment to analyze the basic behaviors of the network system and establish baselines for comparison. The experiments for the sanity check were organized similarly to those that would demonstrate a distributed DQN algorithm, except the RL agent's actions followed specific mechanisms for selection. To explore all potential configurations of per-slice resource budgets in test cell $\hat{i} \in \mathcal{B}$, we adjusted the real-time configuration using the following mechanisms:

- **Repeat Configuration:** The RL agent periodically updates the per-slice resource budgets of cell $I \in \mathcal{B}$ by iteratively choosing actions following a certain order throughout the entire action space $\mathcal{A}_I$.
- **Random Configuration:** This method randomly selects an action from the action space $\mathcal{A}_I$ based on a uniform distribution to update the per-slice resource budgets of cell $I \in \mathcal{B}$.

The basic setup for the sanity check experiment included the following steps:

1. Choose cell $I \in \mathcal{B}$ as the target cell for algorithm implementation.
2. Change the configuration of the RAN slice resource budget in cell $I \in \mathcal{B}$ only, keeping the settings of all other cells as default.
3. Set the reporting period of Season II to 1000 timestamps. After changing the configuration, the statistics within the subsequent 1000 timestamps are used to compute the next network state and reward.

For both reward functions, i.e., downlink average throughput and resource efficiency, we implemented the two methods, repeat configuration and random configuration, for the sanity checks. Each experiment collected 2000 samples of the changed configurations to ensure sufficient states and actions (updated configurations) for analysis. The results of all experiments are summarized in Fig. 4.5 and Fig. 4.6

The figures above demonstrate the change in per-slice throughput of $s^{(1)}_{I,n}$ for both slices at different configurations of per-slice resource budgets for target cell $I \in \mathcal{B}$. The solid dot markers indicate the mean value of "throughput per slice" at each configuration, while the vertical bars represent the corresponding standard deviations. It is evident that for all experiments with different setups, as the per-slice resource budget for each slice increases, the throughput also increases, as does the standard deviation. The throughput of slice 2 tends to saturate when the resource budget increases to a high value (e.g., higher than 0.7). This saturation occurs because users in this slice have fixed data rate requirements, and once these requirements are met, additional resources do not increase throughput.

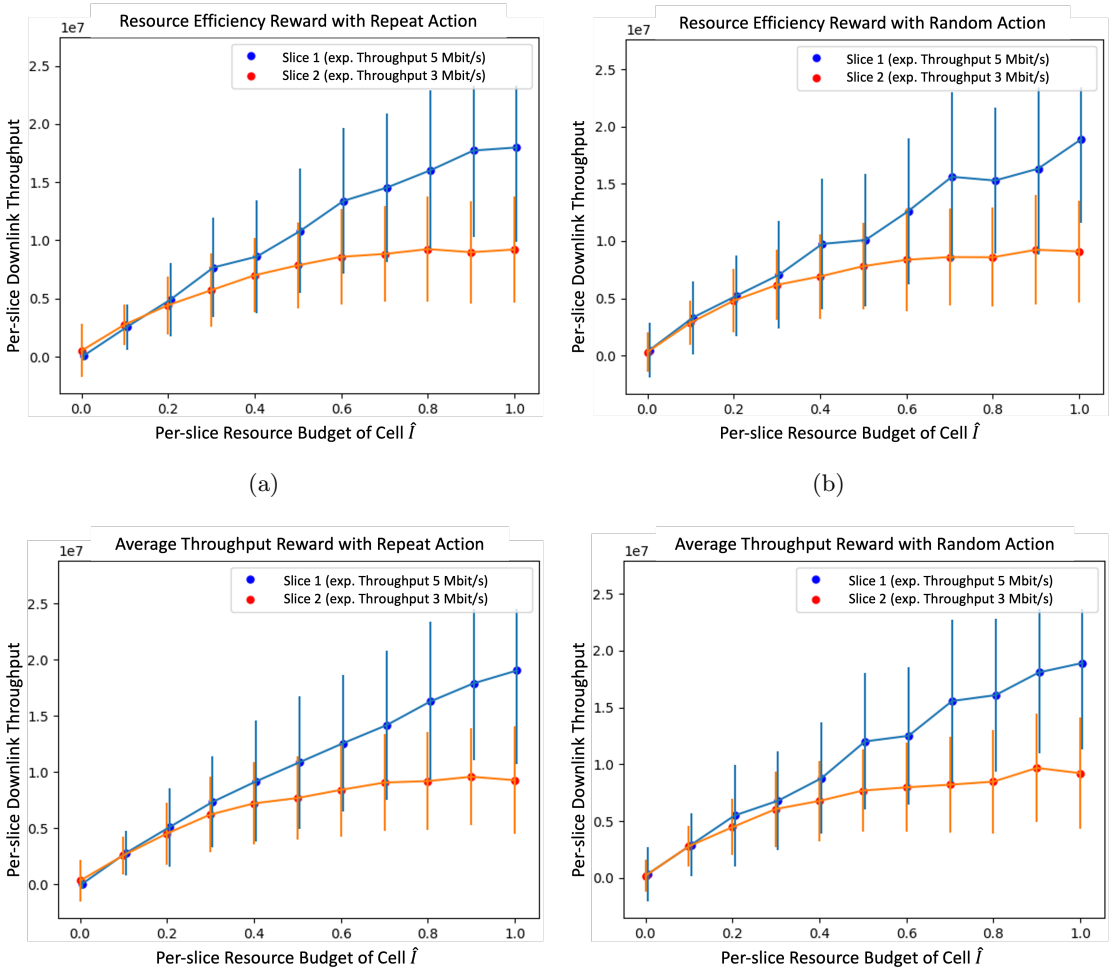

Figure 4.5: Sanity check for per-slice resource budget vs. per-slice throughput

Similar conclusions can be drawn from the comparison of results for per-slice resource budget vs. "slice load." Since "slice load" indicates the actual load on the slice, the per-slice resource budget in the configuration sets an upper bound on the actual slice load, which yields:

$$s_{i,n}^{(2)} <= a_{i,n}, \forall i \in \mathcal{B}, n \in \mathcal{N} \tag{4.12}$$

As shown in Fig. 4.6, the allocated loads on slice 1 are nearly fully occupied for per-slice resource budgets $a_{I,1} < 0.7$, implying that a resource budget less than 0.7 is insufficient to serve the existing users in slice 1.

In RL state, the number of UEs on each slice $u_{i,n}$ is rather dependent on the traffic configurations of user groups. Since the number of UEs per group was fixed in this experiment, it does not show significant changes. For future work, to simulate network behaviors under a more realistic environment, we should consider the traffic models of different groups of users as well. From the above sanity check experiments, we can conclude that under the toy environment we designed, Season II performs well in imitating the behaviors of the mobile network system. Season II returns system KPIs reports that correspond appropriately to configuration changes, aligning with theoretical properties. Additionally, we obtained multiple sets of baselines for comparison, which are critical for evaluating our proposed solution.

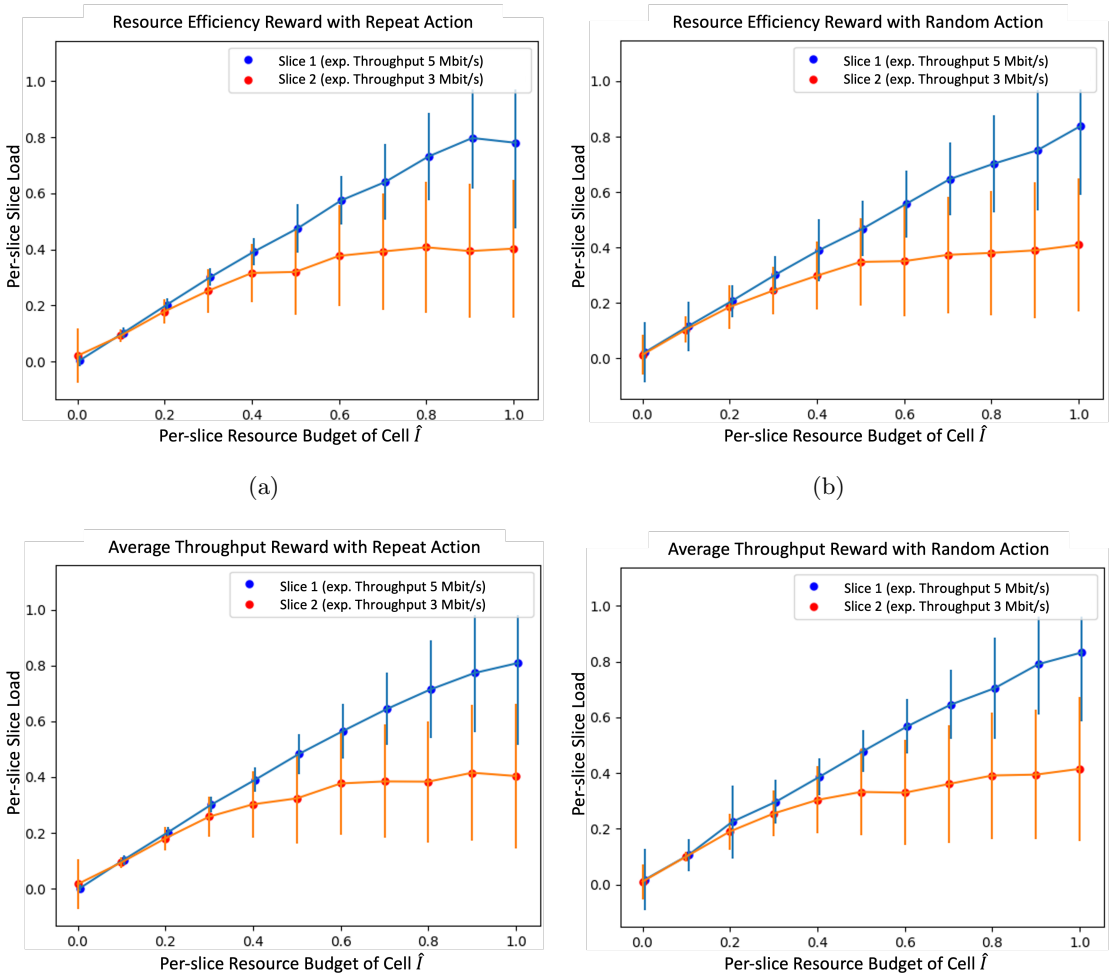

Figure 4.6: Sanity check for per-slice resource budget vs. slice load

4.5.3 Experiment II: DQN-based Approaches

In this experiment, we implement the distributed DQN algorithm using the same setup as in the sanity check. However, in this case, the RL agent updates the actions (i.e., the per-slice resource configurations) with the predicted optimal action from the DQN algorithm based on the current system states.

4.5.3.1 Resource Efficiency as RL Reward

First, we define the reward function of the RL agent as resource efficiency $\eta_I(t)$. For DQN implementation, we set the discount factor $\gamma = 0.09$ and build a replay buffer with capacity $D = 100$. The model is built with 3 hidden layers, with the number of neurons $(256, 128, 64)$. The learning rate of DQN model is 0.002 with Adam optimizer, and the batch size is 32. We run the RL agent for 2000 steps with the RL sample structure illustrated in Table 4.2:

Table 4.2: Elements of the RL agent with resource efficiency as reward

	Values
State $\in \mathbb{R}^6$	$\mathbf{s}_I := [s^1_{I,1}, s^1_{I,2}, s^1_{I,3}, s^2_{I,1}, s^2_{I,2}, s^2_{I,3}]$
Action $\in \mathbb{R}^{11\times 2}$	$\mathbf{a}_I \in \Omega^2$ with $\sum_{n\in\mathcal{N}} a_{I,n} = 1, a_{I,n} \in \{0.0, 0.1, \ldots, 1.0\}$
Reward $\in \mathbb{R}$	$\eta_I(t) = \sum_{n\in\mathcal{N}} \beta_n \eta_{I,n}(t)$ with $\beta_n = 0.5, \forall n \in \mathcal{N}$

Fig. 4.7 and Fig. 4.8 show that the configuration of the per-slice resource budget converges to a steady choice of action $\mathbf{a}_{\hat{i}} = [0.7, 0.3]$.

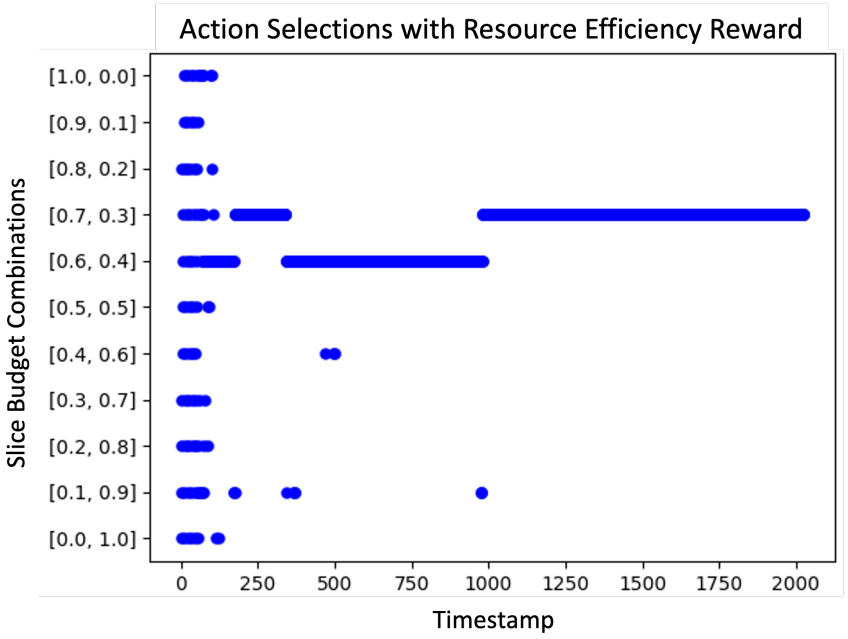

Figure 4.7: Change of slice budget combinations with resource efficiency as RL reward

In Fig. 4.8, the solid lines represent the configuration change along the time axis, while the dots represent the actual load on each slice. It is evident that the choice of configurations converges to $[0.7, 0.3]$ after about timestamp 1000. Most actual loads align with the configuration, but there are still discrepancies, especially in slice 2. It is common to observe that the actual load does not fully occupy the allocated slice load, which could also cause instability in the accumulated average resource efficiency curve.

We compare the results to the sanity check experiment as baselines to observe the benefits of implementing the distributed DQN. In Fig. 4.9, we plot the accumulated average resource efficiency curves with distributed DQN, repeat configuration, and random configuration separately.

In this case, the resource efficiency curve indicated by the blue, solid line is generally higher along the time axis compared to the other two curves, indicating changes without the distributed DQN. However, it does not always outperform the others. We can conclude that resource efficiency is not a direct criterion to reflect the benefits of configuration changes, so it is not an ideal reward function for RL agents. The reason is that resource efficiency mainly depends on the quality of the received signal. An increase in the allocated resource leads to an increased data rate, but the ratio between the data rate and resource remains the same if the signal quality does not change. Using Shannon channel capacity, the downlink throughput of UE k associated with base station i is usually modeled by:

$$r_k = a_{i,k} \log(1 + \text{SINR}_{i,k}) \tag{4.13}$$

where r_k denotes the data rate achieved by UE k in bit/s, $a_{i,k}$ denotes the amount of frequency resource in Hz, and $\text{SINR}_{i,k}$ is the downlink Signal-to-Interference-plus-Noise Ratio (SINR).

As shown in Eq. 4.13, the resource efficiency $r_k/a_{i,k}$ of UE solely depends on $\text{SINR}_{i,k}$. Thus, the per-cell averaged resource efficiency depends mainly on the performance of the scheduler, i.e., how to allocate the resource to each UE with different conditions of SINR. With the pre-defined scheduler, the configuration of the per-slice resource budget does not significantly influence the per-cell resource efficiency. However, we still observe a slight performance improvement compared to the baseline and the convergence of the configuration. The pre-defined proportional fairness scheduler, which tries to balance the service

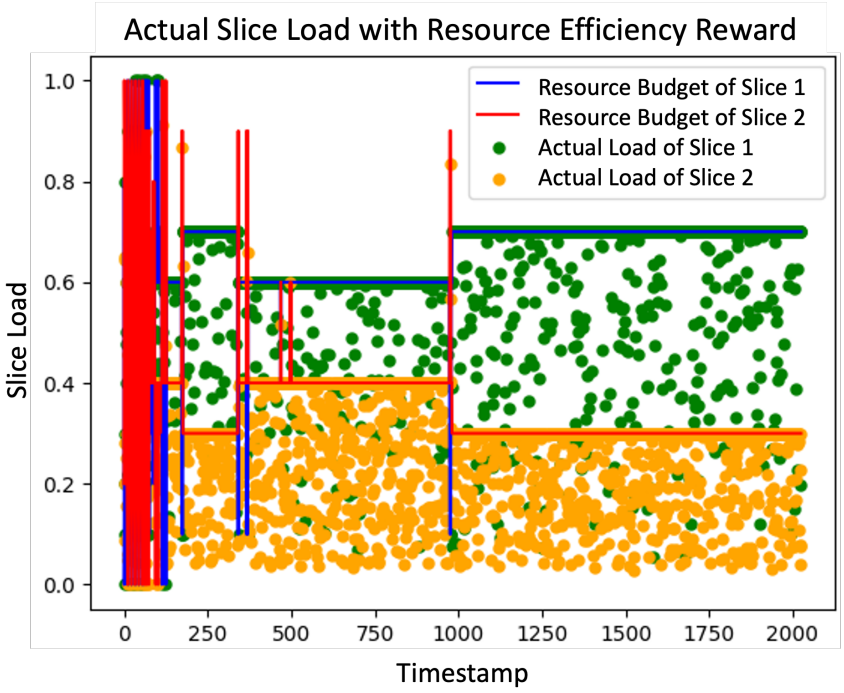

Figure 4.8: Comparison of actual load and allocated resource budgets with resource efficiency reward

quality for all UEs, including those with low signal quality, credits this improvement. When the total amount of allocated resources for a slice is insufficient, the scheduler still needs to allocate a good amount of resources to the UEs with poor signal quality, leading to decreased resource efficiency. Thus, the RL agent can still converge to a configuration that provides sufficient resources to each slice to improve resource efficiency.

4.5.3.2 Downlink Average Throughput as RL Reward

In the previous experiment, we used resource efficiency as the reward function, which provided some improvement, but the performance gains were not significant. In this experiment, we switch to using downlink average throughput as the reward function. Unlike resource efficiency, downlink average throughput is a more direct KPI that describes the behavior of mobile networks and can be read directly from the raw KPI reports generated by the simulator without additional calculations.

The setup for this experiment remains the same as the previous experiments, with the only change being the reward function, as shown in Table 4.3. The RL agent is run for 2000 steps with the same DQN algorithm structure as in Experiment II.

Table 4.3: Elements of the RL agent with downlink average throughput as reward

	Values
State $\in \mathbb{R}^6$	$\mathbf{s}_I := [s^1_{I,1}, s^1_{I,2}, s^1_{I,3}, s^2_{I,1}, s^2_{I,2}, s^2_{I,3}]$
Action $\in \mathbb{R}^{11\times 2}$	$\mathbf{a}_I \in \Omega^2$ with $\sum_{n\in\mathcal{N}} a_{I,n} = 1, a_{I,n} \in \{0.0, 0.1, \ldots, 1.0\}$
Reward $\in \mathbb{R}$	$\phi_I(t)$

Similar to the results in Experiment II, the predicted actions as new configurations still converge to a specific choice in the action space, as shown in Fig. 4.10 and Fig. 4.11.

In this experiment, the RL agent converges faster than in the previous experiment, reaching the configuration $[0.6, 0.4]$ after about 200 timestamps. We can observe similar results from Fig. 4.11. Most of the actual loads match the predicted configuration for slice 1, while slice 2 still has many outliers. This suggests that slice 1 may need a higher resource ratio, while slice 2 might require fewer resources. We compare the changes in downlink av-

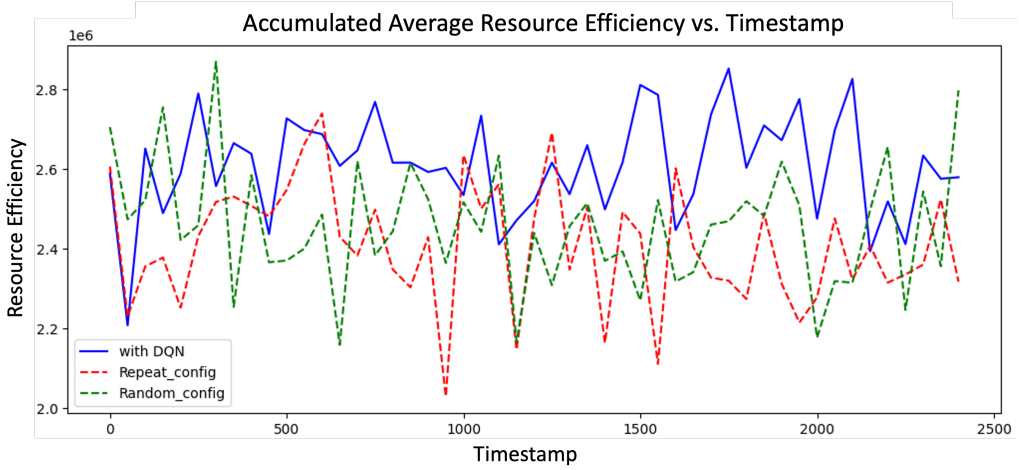

Figure 4.9: Compare accumulated average resource efficiency between different resource partition methods

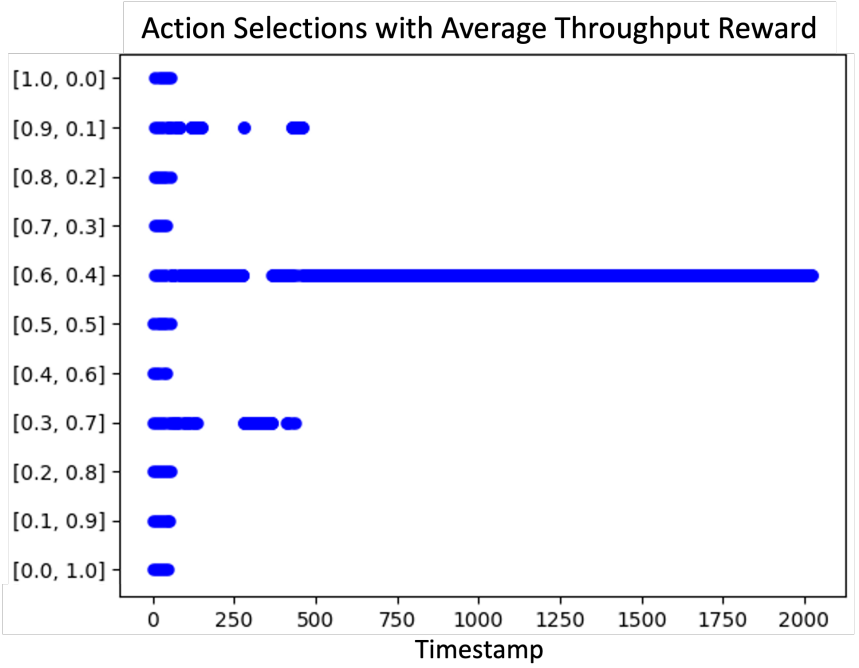

Figure 4.10: Change of slice budget combinations with average throughput as RL reward erage throughput across distributed DQN, repeat configuration, and random configuration methods. The curves are shown in Fig. 4.12.

It is evident that the distributed DQN consistently outperforms the other methods, especially after the timestamp of 750. This highlights the effectiveness of using downlink average throughput as the reward for the RL agent. This experiment demonstrates that direct KPIs, such as downlink average throughput, is a more suitable utility indicator for RL agents.

The mobile network system is a dynamic environment characterized by uncertainties and rapid changes contributing to its unpredictability. These challenges make it difficult to control and update configurations according to network behavior. However, the distributed DQN algorithm can maintain its KPIs at a stable level, proving its effectiveness in optimizing network performance.

4.6 Insights to the Thesis

Based on the experiments conducted above, we can draw the following insights:

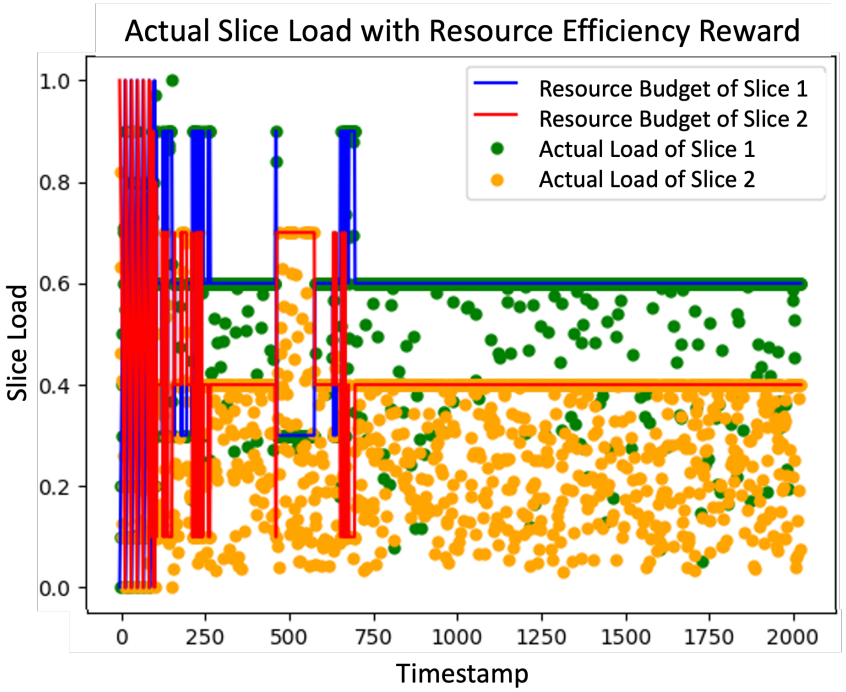

Figure 4.11: Comparison of actual load and allocated resource partitions with average throughput reward

1. **Season II Simulator as a Tool for RL-based Network Optimization:** The Season II simulator has proven to be a suitable tool for implementing RL-based algorithms for network system optimization problems. It responds appropriately to configuration changes in accordance with theoretical mechanisms.

2. **Effectiveness of the Distributed RL Approach:** The proposed distributed RL approach effectively addresses the optimization problem in network slicing environments.

3. **Evaluation of Different RL Agent Rewards:** Among the various types of RL agent rewards, downlink average throughput as a direct KPI demonstrates more observable effects compared to indirect rewards like resource efficiency.

These findings provide a promising starting point for further exploration of RL-based algorithms for optimizing configurations in mobile network systems. However, several areas require further investigation, as outlined below:

1. **Analysis of Different Reward Functions:** Future research should explore various reward functions, including those considered user-specific QoS requirements.

2. **Simulating Realistic Network Traffic Models:** To create a more realistic environment, future experiments should incorporate dynamic changes in the number of UEs based on realistic network traffic models rather than fixing the total number of UEs for each slice.

3. **Exploring RL Algorithms for Continuous Action Spaces:** The current DQN algorithm uses a discrete action space, limiting the precision of the optimal solution. Future research should investigate RL algorithms designed for continuous action spaces, such as DDPG or TD3. These algorithms could help achieve optimal solutions and handle multi-agent problems with more complex state and action spaces.

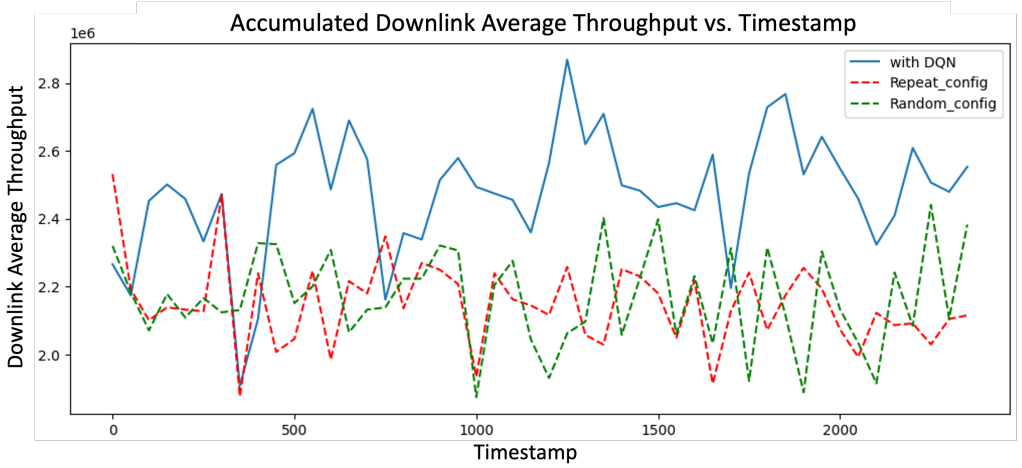

Figure 4.12: Compare accumulated average downlink throughput between different resource partition methods

4. **Addressing Multi-Agent Problems:** After thoroughly analyzing single-agent problems, future work should focus on multi-agent problems. The challenge lies in capturing the dependencies between multiple cells and achieving consensus on RAN slice resource budget decisions among neighboring cells.

5. **Applying TL:** A long-term goal is to apply the concept of TL to reuse the knowledge learned by an RL agent in one cell (or cell cluster) in another cell (or cell cluster). This could enhance the learning rate and sample efficiency, making the optimization process more effective.

These future research directions are expected to build on the findings of this study, further enhancing the application of RL algorithms in mobile network systems optimization.

5. Distributed DRL as Per-cell Scheme

In this chapter, we discuss applying a distributed DRL algorithm to multi-cell, multi-slice RAN slicing scenarios to address Question 2. Specifically, we aim to transform a centralized RL approach into a distributed manner while preserving inter-cell dependency information. We formulate the dynamic inter-cell slicing resource partitioning problem to enhance max-min slice performance while adhering to resource capacity constraints. For the distributed DRL implementation, we introduce a multi-agent DRL approach, exploring two coordination schemes: with and without inter-agent coordination. To ensure that the DRL agents respect resource constraints, we evaluate two methods: reward shaping and decoupled softmax layer embedding. As with the trial experiments in the previous chapter, the evaluation was conducted using a system-level simulator.

This chapter is based on the collaborative research work previously published in the paper:

> Inter-Cell Slicing Resource Partitioning via Coordinated Multi-Agent Deep Reinforcement Learning *Tianlun Hu, Qi Liao, Qiang Liu, Dan Wellington, Georg Carle* ICC 2022 - IEEE International Conference on Communications, 2022, pp. 1-6, doi: 10.1109/ICC45855.2022.9838518.

My contributions to this paper include conducting extensive literature research, conceptualizing and designing the methodology, implementing the proposed models, and rigorously evaluating their effectiveness. Additionally, I was responsible for drafting the manuscript. This chapter extends the discourse from the paper by delving deeper into the application of RL for network slice resource allocation, a topic initially introduced and explored in Chapter 4.

This chapter is organized as follows. In Section 5.1, we review the state-of-the-art slice resource allocation methods and discuss related works. Section 5.2 defines the system model, while Section 5.3 formulates the inter-cell, inter-slice resource partitioning problem. In Section 5.4, we propose distributed DRL solutions to this problem, exploring two schemes: with and without coordination. The numerical results are presented in Section 5.5. Finally, we conclude this chapter in Section 5.6.

5.1 Motivation

To meet the performance and coverage requirements of different slices, network operators aim to partition the radio PRBs resources across multiple base stations, also referred to as

cells. In this chapter, our objective is to satisfy the performance requirements of distinct slices with minimal inter-cell resource usage, thereby maximizing resource efficiency.

Existing model-based solutions approach the resource partitioning problem with mathematical models and solve it using various optimization techniques, such as linear programming [50, 51] and convex optimization [52, 53]. For instance, Addad et al. [50] formulated the network function deployment problem as a Mixed Integer Linear Programming (MILP) under resource constraints, latency, and bandwidth and proposed a heuristic algorithm to solve it. Cavalcante et al. [54] formulated a max-min fairness problem to control load-coupled interference in wireless networks, transforming the problem into a fixed point problem solvable by existing low-complexity iterative algorithms. However, these solutions fail to achieve optimal performance in real networks because the approximated models cannot fully represent complex network behaviors.

Recently, model-free solutions, particularly DRL, have shown significant potential in automatically managing radio access networks without the need for prior models. For example, Liu et al. [55] proposed an adaptive constrained RL algorithm based on interior-point policy optimization (IPO) for a single base station scenario. Liu et al. [56] designed the DeepSlicing algorithm to allocate resources to different slices, with each slice associated with a DRL agent and a coordinator to manage resource capacity in the base station. However, these works focus on single-cell scenarios.

In multi-cell scenarios, DRL solutions with discrete action spaces have been proposed, such as in [57] and [58]. Still, the achievable performance is limited due to the discrete nature of resource partitioning actions. Other works, such as [59] and [60], introduced resource management systems with continuous DRL for complex scenarios but did not address inter-cell interdependencies and inter-slice resource constraints.

As network deployments become denser, causing more severe inter-cell interference among a large number of cells, there is a need for a coordinated multi-agent DRL design capable of capturing complex inter-cell and inter-slice interactions with low model complexity. In this chapter, we investigate DRL-based resource partitioning problem in network slicing under a multi-cell scenario. We aim to improve the max-min slice performance while adhering to resource capacity constraints. To tackle inter-cell interference, we propose a multi-agent DRL approach with two coordination schemes: with and without inter-agent coordination. Additionally, we develop two methods to handle the constraints of instantaneous resource capacity in each agent.

5.2 System Model

We consider a network system consisting of a set of cells $\mathcal{K} := \{1, \ldots, K\}$ and a set of slices $\mathcal{N} := \{1, \ldots, N\}$. Each slice n has pre-defined throughput requirement ϕ_n^* and delay requirement d_n^*. The system runs on discrete time slots $t \in \mathbb{N}_0$. To adapt to the time-varying network traffic and satisfy the slice-aware service requirements in terms of both throughput and delay, the OAM adjusts the inter-slice resource partitioning for all cells periodically. The optimized slicing resource partitions are provided to the RAN scheduler in each cell, and used by the scheduler as the slicing resource budget for the further PRB allocation at a finer time-granularity (as shown in Fig. 1.6).

Considering the temporal and inter-cell interdependencies, we model the multi-cell system as a MDP defined by the tuple $(\mathcal{S}, \mathcal{A}, P(\cdot), r(\cdot), \gamma)$, where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ indicates the transition dynamics by a conditional distribution over the state space $\mathcal{S}$ and the action space $\mathcal{A}$, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function, and $\gamma \in [0, 1]$ is the discount factor.

The **state** at time slot t, denoted by $\mathbf{s}(t) := [\mathbf{s}_1(t), \ldots, \mathbf{s}_K(t)] \in \mathcal{S}$, is an observation of the entire system, where $\mathbf{s}_k(t) \in \mathcal{S}_k$ is the local state observed from cell k. The **action** at slot t,

denoted by $\mathbf{a}(t) := [\mathbf{a}_1(t), \ldots, \mathbf{a}_K(t)] \in \mathcal{A}$, includes the resource partitioning to each slice and each cell $a_{k,n}(t) \in [0,1]$, for $k \in \mathcal{K}, n \in \mathcal{N}$. We further introduce a "headroom" (or reserved bandwidth) to the allocated resource for two reasons: 1) improving the resource efficiency, and 2) converting the inequality action constraints to the equality ones. Let the headroom in cell k be denoted by $a_{k,0}(t) \in [0,1]$. The local action is then defined as $\mathbf{a}_k(t) := [a_{k,0}(t), \ldots, a_{k,N}(t)] \in \mathcal{A}_k$. Given the inter-slice resource constraints in each cell, the local action space $\mathcal{A}_k$ and the global action space $\mathcal{A}$ yield:

$$\mathcal{A}_k := \left\{ \mathbf{a}_k \middle| a_{k,n} \in [0,1], \forall n \in \mathcal{N} \cup \{0\}; \sum_{n=0}^{N} a_{k,n} = 1 \right\} \tag{5.1}$$

$$\mathcal{A} := \left\{ \mathbf{a} \middle| \mathbf{a}_k \in \mathcal{A}_k, \forall k \in \mathcal{K} \right\}. \tag{5.2}$$

Our objective is to satisfy the throughput and delay requirements(ϕ_n^*, d_n^*) for every slice $n \in \mathcal{N}$ and every cell $k \in \mathcal{K}$. Thus, given the observed average throughput $\phi_{k,n}(t)$ and average delay $d_{k,n}(t)$ at slot t for each slice n and cell k, we define the **reward** function as below:

$$r(t) := \min_{k \in \mathcal{K}, n \in \mathcal{N}} \min \left\{ \frac{\phi_{k,n}(t)}{\phi_n^*}, \frac{d_n^*}{d_{k,n}(t)}, 1 \right\}. \tag{5.3}$$

Eq. 5.3 means that if any per-slice throughput or delay in any cell does not meet the requirement, we have $r(t) < 1$. Otherwise, if all requirements are met, the reward is upper bound by 1. Note that the second term $d_n^*/d_{k,n}(t)$ is **inversely** proportional to the actual delay, namely, if the delay is longer than required, this term is smaller than 1.

5.3 Problem of Slice Resource Allocation

Our problem is to find the policy $\pi : \mathcal{S} \to \mathcal{A}$, which decides the inter-cell inter-slice resource partitioning $\mathbf{a} \in \mathcal{A}$ based on the observation of network state $\mathbf{s} \in \mathcal{S}$, to maximize the expectation of the cumulative discounted reward defined in Eq. 5.3 of a trajectory for a finite time horizon T. The problem is given by:

Problem 1.

$$\max_{\pi} \ \mathbb{E}_{\pi} \left[\sum_{t=0}^{T} \gamma^t r\big(\mathbf{s}(t), \mathbf{a}(t)\big) \right] \ s.t. \ \mathbf{a} \in \mathcal{A}, \tag{5.4}$$

where $\mathcal{A}$ is defined by Eq. 5.1 and Eq. 5.2.

The challenge of solving the above-defined problem is two-fold. Firstly, the reward function Eq. 5.3 depends on high-dimensional global state and action spaces and involves complex inter-agent dependencies. For example, increasing resource partition in one slice n and cell k improves its own service performance. However, it decreases the available resource allocated to other slices in the same cell and increases the interference received in the neighboring cells, which may further result in a general service degradation. The second challenge is caused by the intra-cell inter-slice resource constraints Eq. 5.1. Although various methods are proposed to solve the constrained MDP problems, e.g., by using the Lagrangian method [61] or Projection-based Safety layer [62], there still exists the problem of oscillations and overshooting caused by constraint-violating behavior during agent training.

5.4 Distributed DRL as Per-cell Scheme

In this section, we first present the distributed multi-agent DRL approach in terms of two different schemes to solve Problem 2: *distributed scheme without coordination*, and *distributed scheme with inter-agent coordination*. Then, we briefly introduce the actor-critic method to solve the DRL problem. Last but not least, we propose two methods to deal with the inter-slice resource constraints.

5.4.1 Multi-agent DRL with Coordination

The distributed approach allows each agent to learn a possibly different model and make its own decision on the local action, based on local or partial observation. In contrast to the conventional centralized approach, the distributed approach may not achieve the performance as well as the centralized one due to the limited observation. However, it may converge much faster and be more sample-efficient by using a less complex model based on local states and actions.

We first consider the distributed approach without coordination, i.e., each agent k only observes its **local state** $\mathbf{s}_k$. In particular, we include the following measurements and performance metrics into the state $\mathbf{s}_k$ for each cell $k \in \mathcal{K}$:

- Average per-slice user throughput $\{\phi_{k,n} : n \in \mathcal{N}\}$;
- Per-slice load $\{l_{k,n} : n \in \mathcal{N}\}$;
- Per-slice number of active users $\{u_{k,n} : n \in \mathcal{N}\}$.

Thus, with the above-defined three slice-specific features, the local state $\mathbf{s}_k$ has a dimension of $3N$.

Each agent k computes a **local reward** r_k, and makes decision on the **local action** $\mathbf{a}_k \in \mathcal{A}_k \subset [0,1]^{N+1}$. The local reward based on the local observations is computed by

$$r_k(t) := \min_{n \in \mathcal{N}} \min \left\{ \frac{\phi_{k,n}(t)}{\phi_n^*}, \frac{d_n^*}{d_{k,n}(t)}, 1 \right\}. \tag{5.5}$$

Each agent trains an independent model without communicating with others. Note that r_k depends not only on the local state-action pairs, but also on the states and actions of other agents, and we have $r(\mathbf{s}(t), \mathbf{a}(t)) = \min_{k \in \mathcal{K}} r_k(\mathbf{s}(t), \mathbf{a}(t))$. Thus, the distributed scheme approximates $r_k(\mathbf{s}, \mathbf{a})$ with $\tilde{r}_k(\mathbf{s}_k, \mathbf{a}_k)$, decomposes Problem 2 with K independent sub-problems, and finds the following local policies $\pi_k : \mathcal{S}_k \to \mathcal{A}_k$, $\forall k \in \mathcal{K}$:

$$\pi_k^* = \underset{\pi_k; \mathbf{a}_k \in \mathcal{A}_k}{\arg\max} \mathbb{E}_{\pi_k} \left[\sum_{t=0}^{T} \gamma^t \tilde{r}_k\big(\mathbf{s}_k(t), \mathbf{a}_k(t)\big) \right], \forall k \in \mathcal{K}. \tag{5.6}$$

The disadvantage of Eq. 5.6 is that, because $r_k(\mathbf{s}(t), \mathbf{a}(t))$ are strongly coupled to the joint actions and states of all neighboring agents, the approximation $\tilde{r}_k\big(\mathbf{s}_k(t), \mathbf{a}_k(t)\big)$ based on the local observations can be erroneous, which may result in poor learning performance.

To handle the curse of dimension, individual RL agent R_k for each cell $k \in \mathcal{K}$ is introduced for fast convergence in a distributed approach. However, the distributed RL agent would miss the information of inter-cell dependencies and, therefore, could conclude poor performance compared to the centralized one. We propose an inter-cell communication mechanism for a distributed approach to involve the neighbor cells' information in the state $\mathbf{s}_k$ of local agent $\mathbb{R}_k$ to mitigate the interference.

In recent years, a promising direction of *distributed learning with inter-agent coordination* has attracted much attention [63]. Allowing the agents to communicate to acquire a better estimate of the global state improves the performance of distributed method while maintaining low model complexity. To help the distributed agents better estimate $r_k(\mathbf{s}(t), \mathbf{a}(t))$ and capture the inter-agent dependencies, we propose to let the agents communicate and exchange additional information. Let each agent k send a message $\mathbf{m}_k$ to a set of its

neighboring agents, denoted by $\mathcal{K}_k$. Then, each agent k holds the following information: local state and action pair $(\mathbf{s}_k, \mathbf{a}_k)$ and received messages $\overline{\mathbf{m}}_k := [\mathbf{m}_i : i \in \mathcal{K}_k]$.

One option is to directly use all received messages $\overline{\mathbf{m}}_k$ along with $(\mathbf{s}_k, \mathbf{a}_k)$ to estimate $r_k(\mathbf{s}, \mathbf{a})$ with $\tilde{r}_k(\mathbf{s}_k, \overline{\mathbf{m}}_k, \mathbf{a}_k)$. However, if the dimension of the exchanged message is high, this increases the complexity of the local model.

An alternative is to extract from the received messages $\overline{\mathbf{m}}_k \in \mathbb{R}^{Z^{(m)}}$ useful information $\mathbf{c}_k \in \mathbb{R}^{Z^{(c)}}$ with $g : \mathbb{R}^{Z^{(m)}} \to \mathbb{R}^{Z^{(c)}} : \overline{\mathbf{m}}_k \mapsto \mathbf{c}_k$, such that $Z^{(c)} \ll Z^{(m)}$, where $Z^{(m)}$ and $Z^{(c)}$ stand for the corresponding dimension. We can then use $\tilde{r}_k(\mathbf{s}_k, \mathbf{c}_k, \mathbf{a}_k)$ to approximate r_k by capturing the hidden information in the global state while remaining low model complexity. Pioneer works such as [63] proposed to learn the extraction of the communication messages by jointly optimizing the communication action with the RL model. However, for practical systems, the joint training of multiple interacting models can easily result in unstable convergence problems. We want to leverage expert knowledge to extract the information to provide a robust and efficient practical solution. Knowing that the load-coupling inter-cell interference mainly causes the inter-agent dependencies, we propose to let each agent k communicate with its neighboring agent the slice-specific load information $l_{k,n}$, $\forall n \in \mathcal{N}$. Then, based on the exchanged load information, we compute the average per-slice neighboring load as the extracted information $\mathbf{c}_k(t)$. Namely, we define a deterministic function:

$$\begin{aligned} g_k :&\mathbb{R}^{N|\mathcal{K}_k|} \to \mathbb{R}^N : [l_{i,n} : n \in \mathcal{N}, i \in \mathcal{K}_k] \mapsto \mathbf{c}_k(t) \\ \text{with } \mathbf{c}_k(t) :=& \left[\frac{1}{|\mathcal{K}_k|} \sum_{i \in \mathcal{K}_k} l_{i,n}(t) : n \in \mathcal{N} \right]. \end{aligned} \tag{5.7}$$

Therefore, the proposed scheme is to find the following local policies $\pi_k : \mathcal{S}_k \times \mathbb{R}^N \to \mathcal{A}_k$ with distributed DRL agents $k \in \mathcal{K}$:

$$\pi_k^* = \underset{\pi_k; \mathbf{a}_k \in \mathcal{A}_k}{\arg\max} \mathbb{E}_{\pi_k} \left[\sum_{t=0}^{T} \gamma^t \tilde{r}_k\big(\mathbf{s}_k(t), \mathbf{c}_k(t), \mathbf{a}_k(t)\big) \right], \forall k \in \mathcal{K}. \tag{5.8}$$

5.4.2 Actor-Critic Method

We consider solving the DRL problems with actor-critic approaches [64], because of its effectiveness when dealing with high dimensional and continuous state and action spaces. Such approaches solve the optimization problem by using critic function $Q(\mathbf{s}_t, \mathbf{a}_t|\theta^Q)$ (in this section, we denote $\mathbf{s}(t)$ and $\mathbf{a}(t)$ by $\mathbf{s}_t$ and $\mathbf{a}_t$ respectively for brevity) to approximate the value function, i.e., $Q(\mathbf{s}_t, \mathbf{a}_t|\theta^Q) \approx Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$, and actor $\pi(\mathbf{s}_t|\theta^\pi)$ to update the policy π at every DRL step in the direction suggested by critic.

For DRL implementation, we deployed TD3 [65] as an off-policy DRL algorithm built on top of the actor-critic methods. As the extension of DDPG [66], TD3 overcomes the problem of DDPG that overestimates Q-values by introducing twin critic networks for both networks $Q_{\theta_1}, Q_{\theta_2}$ and target networks $Q_{\theta_1'}, Q_{\theta_2'}$. The actor is updated by policy gradient on the expected cumulative reward J with respect to actor parameter θ^π, as:

$$\begin{aligned} \nabla_{\theta^\pi} J &\approx \mathbb{E}\left[\nabla_{\theta^\pi} Q(\mathbf{s}, \mathbf{a}|\theta^Q)|_{\mathbf{s}=\mathbf{s}_t, \mathbf{a}=\pi(\mathbf{s}_t|\theta^\pi)} \right] \\ &= \mathbb{E}\left[\nabla_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}|\theta^Q)|_{\mathbf{s}=\mathbf{s}_t, \mathbf{a}=\pi(\mathbf{s}_t)} \nabla_{\theta^\pi} \pi(\mathbf{s}_t|\theta^\pi) \right] \end{aligned} \tag{5.9}$$

The critic parameter θ^Q is updated with temporal difference learning, given by:

$$\begin{aligned} L\left(\theta^Q\right) &= \mathbb{E}\left[\left(g_t - Q(\mathbf{s}_t, \mathbf{a}_t|\theta^Q) \right)^2 \right], \\ \text{where } g_t &= r_t + \gamma Q\left(\mathbf{s}_{t+1}, \pi\left(\mathbf{s}_{t+1}|\theta^\pi\right)|\theta^Q\right). \end{aligned} \tag{5.10}$$

5.4.3 Dealing with Resource Constraints

We compare two solutions to address the inter-slice resource constraints in Eq. 5.1. The first is to reshape the reward function with an additional term to penalize the violation of the resource constraints, and the second is to reconstruct the network architecture with an additional regularization layer.

5.4.3.1 Reward Reshaping

We add a penalty term to the original reward function Eq. 5.3 to penalize the actions violating the constraints $\sum_{n=0}^{N} a_{k,n} = 1, \forall k \in \mathcal{K}$. At time slot t, the penalty is defined by the allocated resource ratio exceeding the maximum quota. The modified reward function with penalty term is defined as:

$$\begin{aligned} r(t) &:= \min_{k\in\mathcal{K}, n\in\mathcal{N}} \min\left\{\frac{\phi_{k,n}(t)}{\phi_n^*}, \frac{d_n^*}{d_{k,n}(t)}, 1\right\} - \beta h_{k,n}(t) \\ \text{with } h_{k,n}(t) &:= \left|1 - \sum_{n=1}^{N} a_{k,n}(t)\right|, \end{aligned} \tag{5.11}$$

where β is the weight factor for leveraging the desired reward and the constraint-based penalty.

5.4.3.2 Embedding Decoupled Softmax Layer

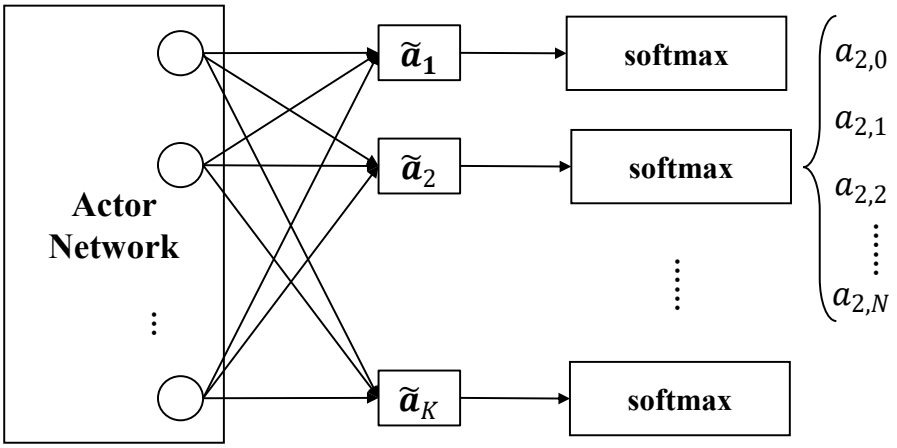

Figure 5.1: Actor output layer with decoupled softmax activation

In this method, we introduce a decoupled regularization layer into the output layer of the actor network, such that this layer becomes part of the end-to-end back propagation training of the neural network. Since the softmax function realizes for each $\mathbf{a}_k$ the following projection

$$\sigma : \mathbb{R}^{N+1} \to \left\{ \mathbf{a}_k \in \mathbb{R}^{N+1} \middle| a_{k,n} \geq 0, \sum_{n=0}^{N} a_{k,n} = 1 \right\},$$

the decoupled softmax layer well addresses the intra-cell inter-slice resource constraints $\sum_{n=0}^{N} a_{k,n} = 1$, $\forall k \in \mathcal{K}$ as shown in Fig. 5.1.

The benefit of applying the decoupled softmax layer versus the reshaping of the reward function is that, because the softmax regularization is part of the end-to-end backpropagation, the agent training is usually more stable and converges faster.

5.5 Experiments

In this section, we evaluate the performance of the proposed distributed schemes for inter-cell slicing resource partitioning introduced in Section 5.4.1 with the Season II simulator, which mimics real-life network scenarios with customized network slicing traffic, user mobility, and network topology. A small urban area of three sites is selected, as demonstrated in Fig. 5.2. At each three-sector site, three cells are deployed using LTE radio technology with 2.6 GHz. Thus, we have in total $K = 9$ cells. We use the realistic radio propagation model Winner+[67]. The system is built up with $N = 2$ network slices: slice 1 supporting

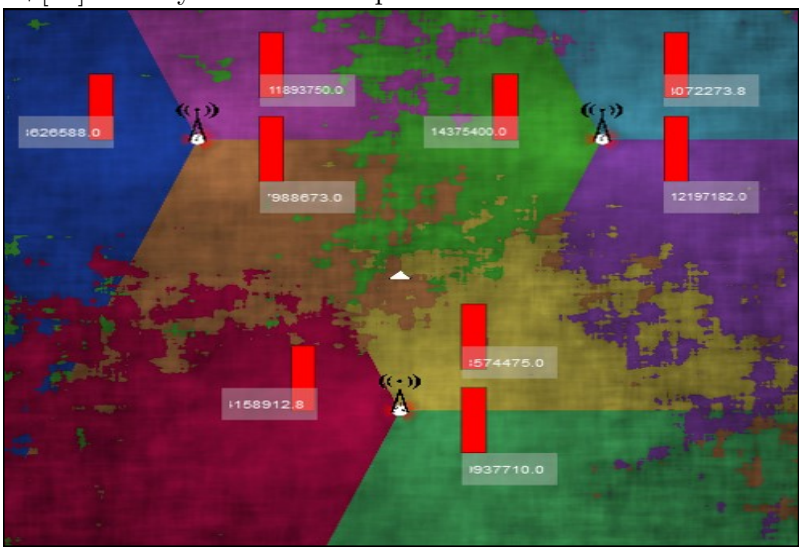

Figure 5.2: Environment setup for experiments

video traffic and slice 2 supporting HTTP traffic. We define slice-specific expected bit rates $\phi_1^* = 5$ Mbit/s and $\phi_2^* = 3$ Mbit/s respectively and the same network latency requirements $d_n = 1$ ms, $n = 1, 2$ (due to the current scheduler limitation of the simulator, we can only apply one latency requirement but different throughput requirements). All cells in the network have the same fixed bandwidth $B = 20$ MHz.

We define two groups of UEs associated with the defined two slices respectively, both with the maximum group size of 32, and both move uniformly randomly within the playground. To imitate the time-varying traffic pattern, we also apply a time-dependent traffic mask $\tau_n(t) \in [0, 1]$ for each slice $n = 1, 2$ to scale the total number of UEs in the scenario.

5.5.1 Schemes for Comparison

We compare the proposed distributed DRL schemes in Section 5.4 with the conventional centralized DRL approach and a traffic-aware baseline approach. The schemes to evaluate and compare are summarized as follows.

Table 5.1: Comparison of dimensions of DRL models used in simulation

	Centralized	**Dist. without Coord.**	**Dist. with Coord.**
State	Global state $\mathbf{s} \in \mathbb{R}^{54}$	Local state $\mathbf{s}_k \in \mathbb{R}^6$	Local state + coord. $[\mathbf{s}_k, \mathbf{c}_k] \in \mathbb{R}^8$
Action	Global action $\mathbf{a} \in [0, 1]^{27}$	Local action $\mathbf{a}_k \in [0, 1]^3$	Local action $\mathbf{a}_k \in [0, 1]^3$
Reward	Global reward r in Eq. 5.3	Local reward r_k in Eq. 5.5	Local reward r_k in Eq. 5.5

- **Cen-Pen**: centralized DRL approach with penalized reward as described in Section 5.4.3.1. We assume that a single agent has full observation of the global state $\mathbf{s} \in \mathcal{S}$, computes the global reward r based on Eq. 5.3, and makes the decision of the slicing

resource partitioning for all agents $\mathbf{a} \in \mathcal{A}$. The dimensions of the centralized and distributed DRL models used in the simulation are compared in Table 6.2.

- **Cen-Soft**: same centralized DRL approach as Cen-Pen but with embedded softmax layer as introduced in Section 5.4.3.2.
- **Dist**: distributed multi-agent DRL scheme as introduced in Section 5.4.1 with embedded softmax layer.
- **Dist-Comm**: coordinated distributed multi-agent DRL scheme with inter-cell communication introduced in Section 5.4.1 and embedded softmax layer.
- **Baseline**: a traffic-aware baseline that dynamically adapts to the current per-slice traffic amount. In each cell, the resources are split proportionally to the number of active UEs per slice.

5.5.2 RL Training Setup

As for DRL training, we use MLP architecture for actor-critic networks. In Cen-Soft and Cen-Pen schemes, the models of actor-critic networks are both built up with 3 hidden layers, with the number of neurons $(96, 64, 48)$ and $(120, 64, 32)$, respectively. While for distributed schemes, both actor-critic networks only have two hidden layers as $(48, 24)$ and $(64, 24)$. In all schemes, the learning rate of actor and critic are 0.0005 and 0.001 respectively with Adam optimizer and training batch size of 32. We choose a small DRL discount factor $\gamma = 0.1$, since the current action has a strong impact on the instantaneous reward while much less impact on the future. For training setups, we applied 2500 steps for exploration, 10000 steps for DRL learning, and final 2500 steps for evaluation.

5.5.3 Performance Evalutation

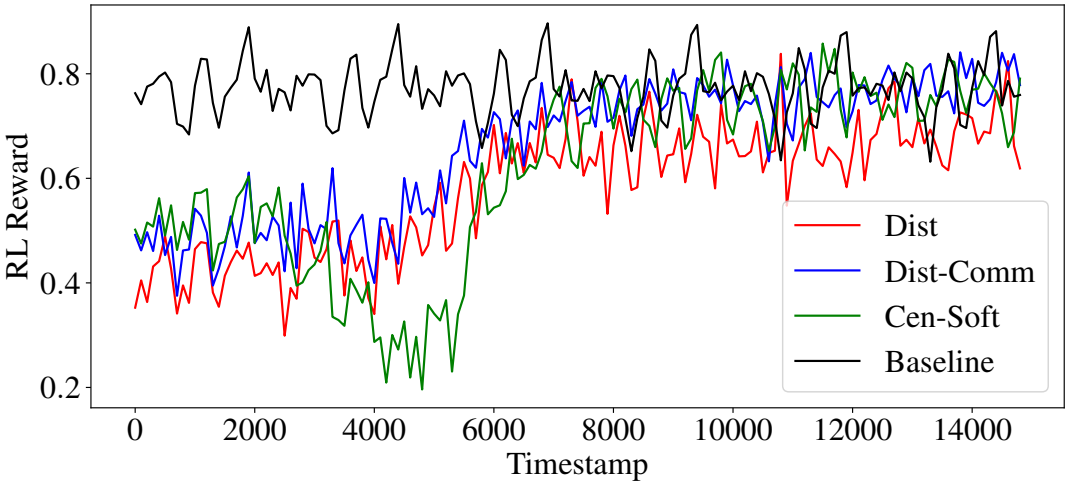

Figure 5.3: Comparison of reward among schemes

Fig. 5.3 demonstrates the comparison of reward defined in Eq. 5.5 during the training process among the schemes Cen-Soft, Dist, Dist-Comm and Baseline defined in Section 5.4.1, while Fig. 5.4 compares minimum resource efficiency among slices. The resource efficiency η_k for cell $k \in \mathcal{K}$ is given by $\eta_k = (1/N) \sum_{n \in \mathcal{N}} \phi_{k,n}(t)/(a_{k,n}B)$.

As shown in Fig. 5.3 and Fig. 5.4, all DRL approaches learn to achieve similar service performance to Baseline, while proving more than two-fold increase in resource efficiency by introducing the headroom in action choices. Note that Baseline dynamically captures time-varying traffic patterns and offers all resources to the UEs. It provides sufficiently good service performance while suffering from low resource efficiency.

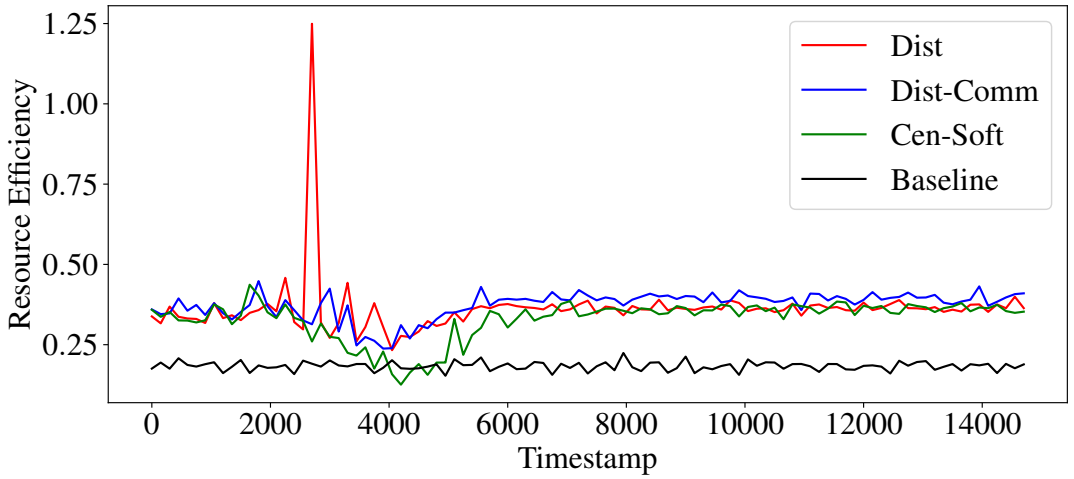

Figure 5.4: Comparison of resource efficiency among schemes

Another observation is that the proposed Dist-Comm scheme slightly outperforms Cen-Soft within the same training time period. The centralized approach converges slower and often experiences extremely poor performance during training because it has much higher action and state dimensions and requires longer training to converge to a good solution. In comparison between Dist and Dist-Comm schemes, it is obvious that inter-agent coordination helps Dist-Comm outperform Dist in terms of both service performance and resource efficiency. while scheme Dis without inter-cell communication concludes worse output than others. However, from the perspective of resource efficiency, all the DRL schemes perform better than the baseline, which means that the DRL resource scheduler could provide as good network performance as the baseline but with less resource occupation. The reason is that the baseline always takes full bandwidth in use, while the "headroom" in DRL schemes reserved partial resources during the training process to enable the DRL agent to explore optimal per-slice occupation.

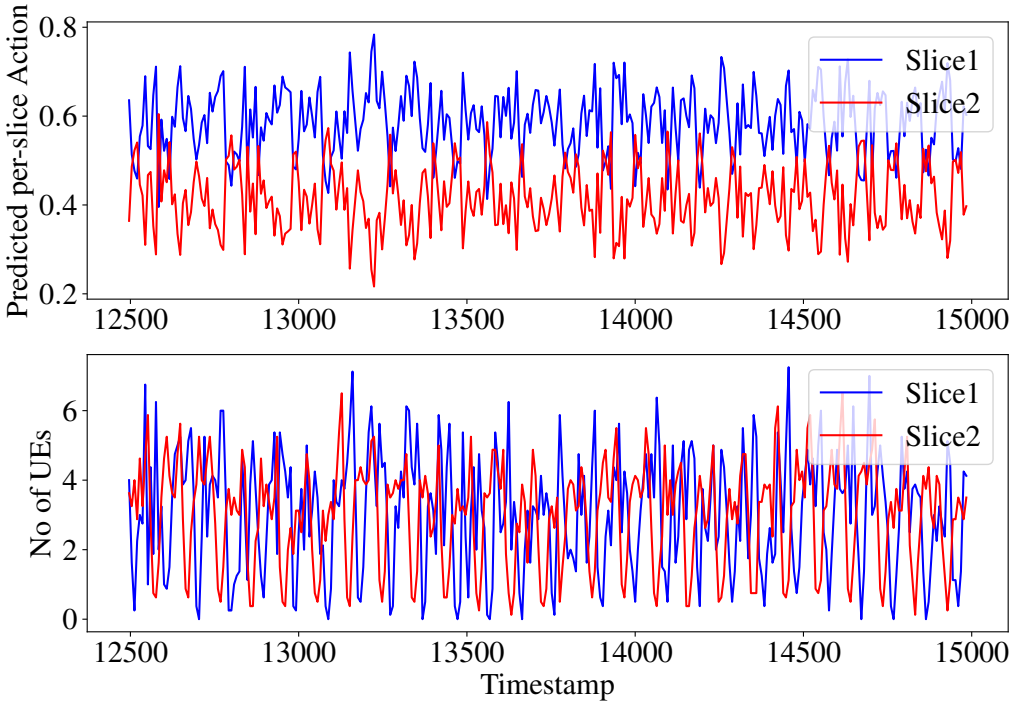

Figure 5.5: Adaptive action to traffic mask after training

Fig. 5.5 shows the predicted action, i.e., per-slice resource partitioning, and the pre-defined traffic mask of scheme Cen-Soft in cell $k = 9$. It verifies that the DRL approach predicts

actions that adapt well to network traffic dynamically concerning different slice-specific throughput requirements.

The above-illustrated results show the algorithms' performance in terms of our objectives, i.e., maximizing the minimum service quality among all slices and cells. In the following, let us take a deeper look into the general performance in terms of the service quality distributions. Fig. 5.6 illustrates the empirical complementary Cumulative Distribution Function (CDF) (or called survival function) that equals $1 - F_X(x)$ where $F_X(x)$ denotes the CDF. We observe that our proposed Dist-Comm achieves the best balance between the two slices, with both slices achieving $> 88\%$ of the satisfaction ratio with the expected throughput, while Baseline and Cen-Soft provide only 82% and 84% for slice 1 , respectively. Fig. 5.7 illustrates the CDF of the slice delay. And similar observation can be made, that the proposed Dist-Comm provides fairly balanced service quality to the two slices.

A summarized comparison of the average performance metrics among all approaches in the testing phase is listed in Table 5.2. We can see that Dist-Comm provides the best performance in terms of the desired reward, resource efficiency, and throughput and delay requirements. Moreover, it encourages a more balanced service quality between the two slices.

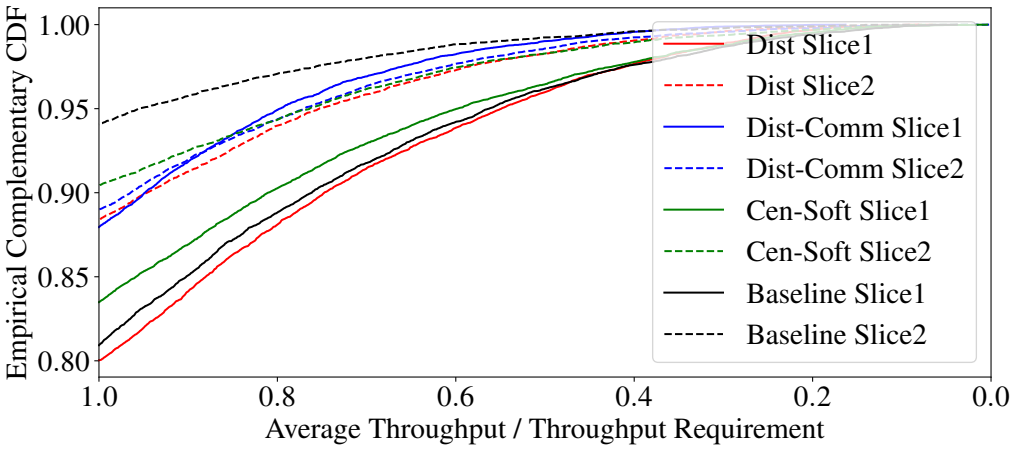

Figure 5.6: Comparing slice throughput from different approaches

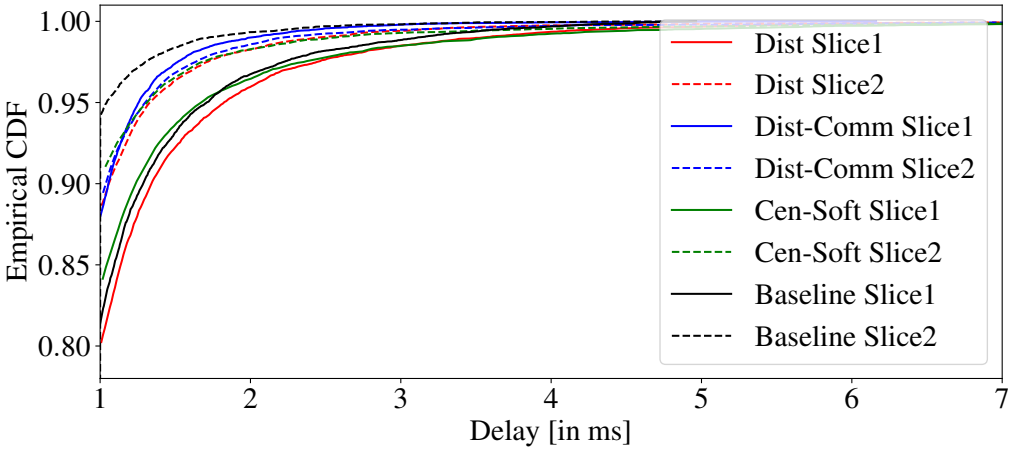

Figure 5.7: Comparing slice delay from different approaches

Last but not least, Fig. 5.8 illustrates the comparison between the solutions to resource constraints. The embedded softmax layer demonstrates a better performance than the reward shaping. It is also worth mentioning that the results shown in Fig. 5.8 were obtained with a different smaller environment consisting of 6 cells, with first 1000 timestamps for exploration, 6000 for training and final 1000 for testing, while with 9 cells we have diffi-

Table 5.2: Comparison of average performance metrics among different approaches

	Dist	Dist-Comm	Cen-Soft	Baseline
RL Reward	0.697	**0.775**	0.756	0.771
Resource Efficiency	0.367	**0.374**	0.362	0.183
Per-slice Throughput Satisfaction	(0.940, 0.969)	(**0.975**, 0.972)	(0.950, 0.972)	(0.942, **0.985**)
Per-slice Delay (ms)	(1.14, 1.06)	(**1.04**, 1.11)	(1.15, 1.07)	(1.15, **1.03**)

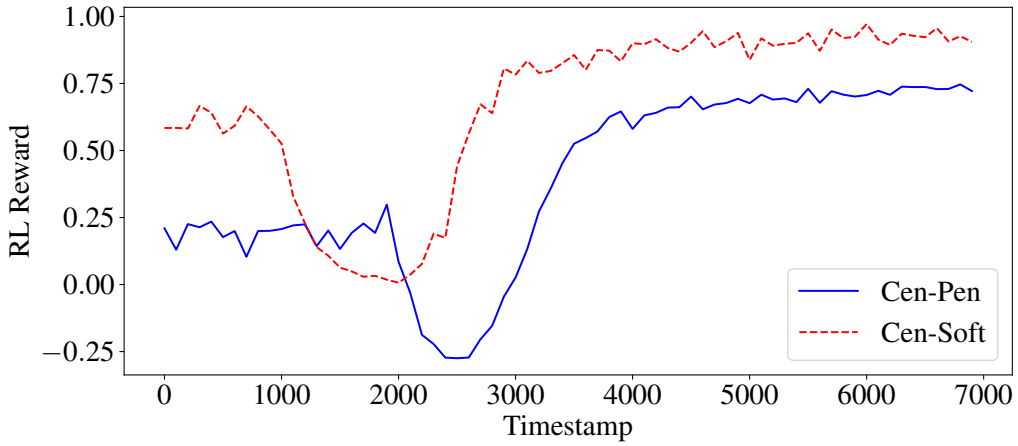

Figure 5.8: Comparing solutions to DRL with resource constraints

culties in obtaining converging results using reward shaping. Thus, a hypothesis is that the shaped reward function is more complex, and easily causes oscillating and unstable training experience.

5.6 Key Takeaways

In this section, we summarize the key insights gained from experiment analysis:

- **RL Learning Capability:** Both centralized and distributed DRL-based approaches effectively adapt to slice-aware traffic, providing high service quality. With the introduction of the headroom concept, these approaches offer more than a two-fold increase in resource efficiency compared to traffic-aware baselines.
- **Performance of Distributed Schemes:** The distributed DRL outperforms the centralized approach when both are trained within the same limited time period. Inter-agent coordination mechanism, which allows multiple agents to share load information, enhances the performance of the distributed scheme. This approach maintains lower model complexity, achieves faster convergence, and provides more balanced service quality among different slices.
- **Inter-Slice Resource Constraints:** When addressing inter-slice resource constraints, embedding a decoupled softmax layer proves superior to reward shaping. This method leads to faster convergence and avoids deep oscillations during training.

In this chapter, we tackled the dynamic inter-cell slicing resource partitioning problem to meet slice-aware service requirements and improve resource efficiency. Our approach involved jointly optimizing inter-cell inter-slice resource partitioning and resource headroom. Thereby, we answer the Question 2 by providing the following contributions:

1. **Distributed Multi-Agent DRL Solution:** We proposed a distributed multi-agent DRL solution to address the problem, exploring two schemes: with and without inter-agent coordination.
2. **Handling Resource Constraints:** We developed two methods, reward reshaping and decoupled softmax embedding, to enable DRL agents to be aware of inter-slice resource constraints.
3. **Extensive Evaluation:** We extensively evaluated our proposed solutions using a system-level simulator. The results demonstrated that the coordinated distributed scheme provides superior slice-aware service performance compared to the centralized approach, with the added benefit of more than doubling resource efficiency over traffic-aware baselines.

Despite the promising results of DRL in resource management, several challenges hinder its broader applicability and effectiveness. One major issue is the poor generality of DRL models, which often struggle to perform well outside the specific training scenarios due to their inability to generalize across different network conditions. Additionally, sample efficiency poses a significant challenge, as DRL algorithms typically require vast amounts of data to learn effective policies, making training time-consuming and computationally expensive. Lastly, model reproducibility is a critical concern, as the complex and stochastic nature of DRL agent can lead to inconsistent performance across different runs and environments.

Building on these findings, the next step is to integrate TL into the distributed DRL framework. TL can enhance the learning efficiency and performance of DRL agents by leveraging knowledge from previously learned tasks or domains. This approach is particularly useful in scenarios with complex state-action spaces or when dealing with multiple agents. In the upcoming chapters, we will explore how TL can be used to:

- Improve the learning rate and sample efficiency of DRL agents;
- Facilitate knowledge transfer between cells or cell clusters, allowing agents to adapt more quickly to new environments;
- Further enhance the coordination between multiple agents in a distributed DRL setting.

By incorporating TL, we aim to overcome the challenges associated with training DRL agents in large-scale, dynamic network environments, ultimately leading to more efficient and robust resource management solutions.

6. TL-aided DRL Approach I: Generalist-to-Specialist

In this chapter, we integrate the proposed distributed DRL approach with TL techniques to achieve higher model reproducibility and sample efficiency. Specifically, we investigate methods for transferring a general distributed DRL model trained on a large dataset to a specific domain.

We continue to address the dynamic inter-cell resource partitioning problem to meet the throughput and latency requirements for all slices under inter-slice resource constraints. In addition to the main objectives, we study two additional goals: maximizing the minimum service quality across all slices and cells and maximizing the average logarithmic utility over all slices. We refer to the multi-agent DRL algorithm with inter-agent coordination, as proposed in Section 5.4, as the DIRP algorithm. Furthermore, to address Question 3 and Question 5, we develop a novel TL-based DIRP algorithm to facilitate the transfer of DIRP solutions across different network environments. The effectiveness of TL-DIRP is analyzed through pre-trained model transfer, instance transfer, and combined model and instance transfer schemes.

This chapter incorporates findings from research published in the following papers:

> Knowledge Transfer in Deep Reinforcement Learning for Slice-Aware Mobility Robustness Optimization *Qi Liao, Tianlun Hu, Dan Wellington* ICC 2022 - IEEE International Conference on Communications, 2022, pp. 1-6, doi: 10.1109/ICC45855.2022.9838657.

For the paper mentioned above, my contributions were primarily in the design and implementation of applications combining TL with RL concepts. This work specifically addressed enhancing network mobility robustness by optimizing handover behaviors using RL methods, supplemented by TL techniques to boost performance. Although this specific network optimization task is not discussed in the dissertation, the methodologies developed for RL and TL are adapted and applied.

> Inter-Cell Network Slicing with Transfer Learning Empowered Multi-Agent Deep Reinforcement Learning *Tianlun Hu, Qi Liao, Qiang Liu, Georg Carle* IEEE Open Journal of the Communications Society, Volume. 4, 2023, pp. 1-15, doi: 10.1109/OJCOMS.2023.3273310

In the second publication, I was responsible for conducting the literature review, designing and implementing the proposed TL methods on top of RL-based solutions for slice resource allocation, and evaluating the experiments. Additionally, I took on the task of drafting the manuscript. In this dissertation, the work is extended by connecting to the discussions in previous chapters, particularly focusing on applying TL in Generalist-to-Specialist scenarios. This sets the stage for the initial discussions in Chapter 7, which covers other scenarios of applying TL to RL-based soluitions.

This chapter is organized as follows. In Section 6.2, we define the system model and formulate the inter-cell inter-slice resource partitioning problem. In Section 6.3, we propose the DIRP algorithm to solve the problem with inter-agent coordination. In Section 6.4, we enhance the DIRP algorithm with TL and investigate different types of transferable knowledge. The numerical results are demonstrated in Section 6.5. Finally, we conclude this paper in Section 6.6.

6.1 Introduction

With the increasing deployment of base stations in 5G and beyond, network slicing is becoming more complex. The lack of interference coordination in existing individualized approaches can degrade slice performance in multi-cell scenarios.

6.1.1 Motivation

Based on the findings from the last chapter, distributed schemes, such as those proposed in [57, 58, 68], independently train agents with individualized policies. These schemes show promising performance improvements in terms of convergence speed and communication overhead compared to centralized schemes. Zhao et al. [58] investigated dynamic resource allocation in network slicing with distributed DRL. However, their approach lacks inter-agent coordination, leading to uncoordinated interference in multi-cell slicing management. Efforts by Nie et al. [69] have addressed non-stationary environments by augmenting the state space of individual agents, but concerns about sample efficiency, lengthy exploration, and convergence speed remain, hindering practical implementation in large-scale networks.

Emerging TL techniques [70] have been increasingly studied to address challenges regarding algorithm scalability, model reproducibility, and sample efficiency in machine learning-based approaches [71, 72, 73]. The basic idea of TL is to utilize prior knowledge from pre-trained models to benefit the learning process in target models. While extensive TL research exists in the supervised learning domain [74, 75], these techniques cannot be directly applied to the RL domain [76, 77]. A few studies [78, 79] have explored TL in resource allocation for mobile networks, such as spectrum sharing in Vehicle-to-Everything (V2X) [80] and parameter optimization in network slicing. However, TL-assisted MADRL in inter-cell network slicing scenarios remains an open problem.

6.1.2 Related Works

The topics discussed in this chapter relate to network resource management, DRL, and TL methods in mobile networks.

Model-based Resource Management
Extensive works use model-based approaches to manage RAN slice resource allocation in

5G and beyond networks. Some studies [52, 53] investigated network slice resource allocation by assuming known and static resource demands, leveraging convex optimization methods to solve the problem with various utility functions. In machine-type communications, Beshley et al. [51] proposed a radio resource allocation method to dynamically select channel bandwidth based on QoS requirements and traffic aggregation in Machine-to-Machine (M2M) gateways. Addad et al. [50] analyzed virtual network function deployment in network slicing, formulating a mixed-integer linear programming model and proposing a heuristic algorithm under different resource constraints. Cavalcante et al. [81] formulated a max-min fairness problem to handle load-coupled interference, transforming it into a fixed-point problem solved with a low-complexity iterative algorithm. Recently, an inter-cell coordinated scheme for dense cellular network resource scheduling was proposed [82], addressing inter-cell interference and showing promising results. However, these approximated mathematical models cannot fully represent complex network characteristics. More importantly, applying these model-based solutions in OAM is challenging due to the lack of Channel State Information (CSI) measurements at fine time granularity.

DRL Approaches in Networking
Liu et al. [55] proposed a constrained DRL based on interior-point policy optimization (IPO) to solve the slicing resource allocation problem in a single base station scenario. Xu et al. [83] studied a DRL-based solution to extract the behavior of per-slice users with traffic-aware exploration and allocate sufficient RAN resources accordingly. Liu et al. [56] proposed a DRL-based algorithm named DeepSlicing, decomposing RAN slicing optimization into a master problem and several slave problems, addressed with a joint coordinator and associated DRL agent for each slice. However, these works focus on single-cell scenarios. Some studies [57, 58] explored multi-cell scenarios and proposed DRL solutions with discrete action spaces. Recent efforts [59, 60] extended the discrete action space into continuous action space, showing improved performance in handling complex scenarios. However, none addressed inter-cell dependencies and inter-slice resource constraints.

TL Methods in Networking
Xu et al. [84] proposed an aggregation TL method applied to MADRL for real-time strategy games, transferring knowledge from small-scale to large-scale multi-agent systems, improving the convergence speed of the algorithm. Zafar et al. [80] enhanced double Q-learning with TL for decentralized spectrum sharing in V2X communication networks, accelerating the learner model's convergence rate by transferring Q-values from the expert model. Mai et al. [79] proposed optimizing slice parameters, such as transmission power and spreading factor, with DDPG and TL, by pre-training a model on a centralized controller and using it as the initial model for local slice optimization tasks. Nagib et al. [78] studied TL to accelerate DRL algorithms for dynamic RAN slicing resource allocation in single-cell scenarios, transferring the model pre-trained from an expert base station to a learner base station. However, none of the above works studied TL in coordinated MADRL for inter-cell slicing resource partitioning.

6.1.3 Outline

In this chapter, we first develop a DIRP algorithm, which effectively solves the problem with an inter-agent coordination mechanism, allowing information sharing between cells. The optimization is based on two alternative objectives: 1) maximizing the minimum service quality over all slices and cells, and 2) maximizing the average logarithmic utility over all slices. The former ensures that all slice-specific requirements for throughput and delay are fulfilled, aligning with 3$^{\text{rd}}$ Generation Partnership Project (3GPP) specifications that the service provided by any network slice must comply with the SLA [85]. Max-min fairness, known to provide the best fairness guarantees, is a special case of the general class of well-known α-fair utility functions [86, 87]. The latter, as a classical concave

utility function and also part of the α-fair utility functions, compromises SLA fairness to improve resource efficiency [88].

Next, we design a TL-aided DIRP algorithm to further improve sample efficiency, model reproducibility, and algorithm scalability. We investigate the effectiveness of transferable knowledge through 3 schemes: pre-trained model transfer, instance transfer, and combined model and instance transfer. We derive several key insights from the simulation results when integrating TL in MADRL under these schemes.

6.2 System Model and Problem Formulation

In this section, we first describe the MDP-based system model in Section 6.2.1. Then, we formulate the optimization problem based on the MDP model in Section 6.2.2. Table 6.1 summarizes the notations used in this work.

Table 6.1: Table of notations

Symbol	Meaning
$\mathbf{s}$	Global state in $\mathcal{S}$
$\mathbf{a}$	Global action in $\mathcal{A}$
r	Global reward
$\mathbf{s}_k$	Local state in $\mathcal{S}_k$ of Agent k
$\mathbf{a}_k$	Local action in $\mathcal{A}_k$ of Agent k
r_k	Local reward of agent k
$\tilde{r}_k$	Approximated local reward of Agent k
$\mathbf{m}_k$	Message sent from Agent k to neighbors
$\overline{\mathbf{m}}_k$	Received messages from all neighbors of Agent k
$\mathbf{c}_k$	Extracted information from $\overline{\mathbf{m}}_k$ of Agent k
Q_θ	Current critic network with parameter θ
π_ϕ	Current actor network with parameter ϕ
$Q_{\theta'}$	Target critic network with parameter θ'
$\pi_{\phi'}$	Target actor network with parameter ϕ'
$\pi^{(\mathrm{G})}$	Generalist policy learned by central controller
π_k	Specialist policy learned by Agent k
$\mathcal{D}_S$	Source domain $\mathcal{D}_S := \mathcal{D}^{(\mathrm{G})}$, i.e., generalist domain
$\mathcal{T}_S$	Source task $\mathcal{T}_S := \mathcal{T}^{(\mathrm{G})}$, i.e., generalist task
$\mathcal{D}_T$	Target domain $\mathcal{D}_T := \mathcal{D}_k^{(\mathrm{S})}, k \in \mathcal{K}$, i.e., specialist domain
$\mathcal{T}_T$	Target task $\mathcal{T}_T := \mathcal{T}_k^{(\mathrm{S})}, k \in \mathcal{K}$, i.e., specialist task

6.2.1 System Model

Similar to the setup in the previous chapter, we consider a network system consisting of a set of cells $\mathcal{K} := \{1, 2, \ldots, K\}$ and a set of slices $\mathcal{N} := \{1, 2, \ldots, N\}$. Each slice $n \in \mathcal{N}$ has predefined throughput and delay requirements, denoted by ϕ_n^* and d_n^*, respectively. The network system runs on discrete time slots $t \in \mathbb{N}_0$. OAM adapts the inter-slice resource partitioning for all cells periodically to meet their performance requirements.

To capture the temporal and inter-cell dependencies, we model the multi-cell resource partition as an MDP defined by $\mathcal{M} := \{\mathcal{S}, \mathcal{A}, P(\cdot), r(\cdot), \gamma\}$, where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ denotes the transition probability distribution over state space $\mathcal{S}$ and action space $\mathcal{A}$. $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, which evaluates the per-slice QoS for all cells and $\gamma \in [0, 1]$ denotes the discount factor for cumulative reward calculation. Assuming that at each time step t, the network observes the global **state** $\mathbf{s}(t) := [\mathbf{s}_1(t), \ldots, \mathbf{s}_K(t)] \in \mathcal{S}$, where $\mathbf{s}_k(t)$ is the local state observed from cell k. The **action** at slot t denoted by $\mathbf{a}(t) := [\mathbf{a}_1(t), \ldots, \mathbf{a}_K(t)] \in \mathcal{A}$, includes the RAN slice resource budget, where the local action

$\mathbf{a}_k(t) \in \mathcal{A}_k$ indicates the partitioning ratio $a_{k,n}(t) \in [0,1]$ to each slice for $n \in \mathcal{N}$ aligning with inter-slice resource constraints. Thus, the local action space $\mathcal{A}_k$ and the global action space $\mathcal{A}$ yield

$$\mathcal{A}_k := \left\{ \mathbf{a}_k \middle| a_{k,n} \in [0,1], \forall n \in \mathcal{N}; \sum_{n=1}^{N} a_{k,n} = 1 \right\}. \tag{6.1}$$

$$\mathcal{A} := \left\{ \mathbf{a} \middle| \mathbf{a}_k \in \mathcal{A}_k, \forall k \in \mathcal{K} \right\}. \tag{6.2}$$

The goal is to maximize the satisfaction level of QoS in terms of throughput and delay requirements (ϕ_n^*, d_n^*) for every slice $n \in \mathcal{N}$ in each cell $k \in \mathcal{K}$. Thus, we design two alternative **reward** functions for the two alternative objective designs: **max-min fairness** and **maximizing the average logarithmic utilities**. The former provides the best fairness that guarantees overall slice requirements by giving the maximum protection to the most critical and resource-demanding slice. While the latter, although taking fairness into account, still tries to achieve a good fairness-efficiency trade-off.

The global **reward** function $r(t)$, based on the two alternative objectives, respectively, is defined as follows:

1. **Max-min fairness:** we define $r(t)$ as the minimum per-slice QoS satisfaction level based on the observed average throughput $\phi_{k,n}(t)$ and average delay $d_{k,n}(t)$ at time step t for each slice n in cell k, as

$$r(t) := \min_{k \in \mathcal{K}, n \in \mathcal{N}} \min \left\{ \frac{\phi_{k,n}(t)}{\phi_n^*}, \frac{d_n^*}{d_{k,n}(t)}, 1 \right\}. \tag{6.3}$$

   The reward formulation drops below 1 when the actual average throughput or delay of any slices fails to fulfill the requirements. Note that the reward is upper-bounded by 1 even if all slices achieve better performance than the requirements to achieve more efficient resource utilization. The second item in Eq. 6.3 is **inversely proportional** to the actual delay, namely, if the delay is longer than required, this term is lower than 1.

2. **Maximizing the average logarithmic utilities:** we define $r(t)$ as the average logarithmic utilities over the service satisfaction levels of all slices, given by

$$r(t) := \frac{1}{K \cdot N} \cdot \sum_{k \in \mathcal{K}, n \in \mathcal{N}} \log \left(\min \left\{ \frac{\phi_{k,n}(t)}{\phi_n^*}, \frac{d_n^*}{d_{k,n}(t)} \right\} + 1 \right). \tag{6.4}$$

   where the **service satisfaction level** per slice per cell $\min \left\{ \frac{\phi_{k,n}(t)}{\phi_n^*}, \frac{d_n^*}{d_{k,n}(t)} \right\} \geq 0$ is defined as the minimum between the throughput and delay satisfaction levels. Thus, if either throughput or delay does not meet the requirement, this term is below 1. The per-slice logarithmic utility function is always non-negative by adding an offset 1 within the log function with base 2. Note that unlike Eq. 6.3, the reward in Eq. 6.4 is not upper bounded by 1, because the service satisfaction level is not upper-bounded. However, if all slices' requirements are exactly met, we have $r(t) = 1$.

6.2.2 Problem Formulation

The problem is to find the optimal policy $\pi : \mathcal{S} \to \mathcal{A}$, which decides the inter-cell inter-slice resource partitioning $\mathbf{a} \in \mathcal{A}$ based on the observation of network state $\mathbf{s} \in \mathcal{S}$, to maximize the expectation of the cumulative discounted reward defined in Eq. 6.3 or Eq. 6.4 of a trajectory for a finite time horizon T. The problem is given by:

Problem 2.

$$\max_{\pi} \ \mathbb{E}_{\pi}\left[\sum_{t=0}^{T}\gamma^{t} r\big(\mathbf{s}(t),\mathbf{a}(t)\big)\right], s.t.\ \mathbf{a}\in\mathcal{A}, \tag{6.5}$$

where $\mathcal{A}$ is defined by Eq. 6.1 and Eq. 6.2, r is given by Eq. 6.3 or Eq. 6.4.

The challenges of solving the problem above are two-fold. First, the global reward functions depend on high-dimensional state and action spaces and involve complex inter-cell dependencies, which are difficult to accurately obtain in practical network systems. For example, increasing resource partition in one slice n and cell k improves service performance. However, it decreases the available resources allocated to other slices in the same cell and may aggravate the interference received in neighboring cells. Besides, because we aim at solving the inter-cell inter-slice resource partitioning problem in OAM, only a limited set of KPIs (e.g., averaged cell throughput and delay) at a medium time scale (e.g., every 15 minutes) is available. It is extremely difficult to derive closed-form expressions for the multi-cell network with the extracted data at the higher layers (above MAC layer) of the network system. Second, the dynamic of network systems, e.g., additional cell deployments, changes the properties of the problem, e.g., leading to expanded state and action space. This requires the solution of this problem to be efficient and scalable in terms of fast convergence speed, high sample efficiency, and low computational efforts.

6.3 Distributed DRL Per-cell Scheme - DIRP Algorithm

In this section, we summarize the distributed MADRL approach for inter-cell inter-slice resource partition introduced in Section 5.4 and nominate it as DIRP algorithm. Then, we briefly introduce the actor-critic method to solve the DRL problem. Next, we propose the method to tackle the inter-slice resource constraint with modified DRL network architecture.

6.3.1 DIRP Algorithm

The DIRP algorithm enables each agent to learn an individual policy and make its own decision on the local action based on local observations and neighboring information. Unlike centralized DRL, which collects global observations from all slices and cells in the network system, the DIRP algorithm may not achieve the same level of global performance due to its limited view of the entire network. However, it can converge much faster and be more sample-efficient by utilizing a less complex model with lower-dimensional state and action spaces. Additionally, the coordination mechanism can enhance the performance of distributed agents by providing supplementary side information about the environment.

Each agent k observes its local state $\mathbf{s}_k$ to capture local network observations. Specifically, we include the following measurements and performance metrics:

- Average per-slice user throughput $\{\phi_{k,n} : n \in \mathcal{N}\}$;
- Per-slice load $\{l_{k,n} : n \in \mathcal{N}\}$;
- Per-slice number of active users $\{u_{k,n} : n \in \mathcal{N}\}$;
- Per-slice throughput requirement $\{\phi^{*}_{k,n} : n \in \mathcal{N}\}$;
- Per-slice delay requirement $\{d^{*}_{k,n} : n \in \mathcal{N}\}$.

In the conventional distributed DRL approach, each agent k computes a local reward r_k and decides on the local action $\mathbf{a}_k \in \mathcal{A}_k \subset [0,1]^N$. The local reward, based on the local observations, is designed for either max-min fairness or maximizing average logarithmic utilities, as follows:

1. **Max-min Fairness:**

$$r_k(t) := \min_{n \in \mathcal{N}} \min \left\{ \frac{\phi_{k,n}(t)}{\phi_n^*}, \frac{d_n^*}{d_{k,n}(t)}, 1 \right\}, \tag{6.6}$$

2. **Maximizing Average Logarithmic Utilities:**

$$r_k(t) := \frac{1}{N} \cdot \sum_{n \in \mathcal{N}} \log \left(\min \left\{ \frac{\phi_{k,n}(t)}{\phi_n^*}, \frac{d_n^*}{d_{k,n}(t)} \right\} + 1 \right). \tag{6.7}$$

Note that r_k depends not only on the local state-action pair but also on the states and actions of other agents. The global reward can be formulated as $r(t) = \min_{k \in \mathcal{K}} r_k(\mathbf{a}(t), \mathbf{s}(t))$ with the local reward defined in Eq. 6.6 or $r(t) = \frac{1}{K} \sum_{k \in \mathcal{K}} r_k(\mathbf{a}(t), \mathbf{s}(t))$ with the local reward defined in Eq. 6.7. We approximate $r_k(\mathbf{s}, \mathbf{a})$ based on local observations $(\mathbf{s}_k, \mathbf{a}_k)$, denoted by $\tilde{r}_k(\mathbf{s}_k(t), \mathbf{a}_k(t))$. However, this estimation can be inaccurate because it neglects inter-cell dependencies and estimates local rewards independently.

Agents communicate and exchange additional information with neighboring cells to capture inter-agent dependencies in the DIRP algorithm. Each agent k sends a message $\mathbf{m}_k$ to a set of its neighboring agents, denoted by $\mathcal{K}_k$. Each agent k then holds the following information: local state and action pair $(\mathbf{s}_k, \mathbf{a}_k)$ and received messages $\overline{\mathbf{m}}_k := [\mathbf{m}_i : i \in \mathcal{K}_k]$. One option is to directly use all received messages $\overline{\mathbf{m}}_k$ along with $(\mathbf{s}_k, \mathbf{a}_k)$ to estimate $r_k(\mathbf{s}, \mathbf{a})$ as $\tilde{r}_k(\mathbf{s}_k, \overline{\mathbf{m}}_k, \mathbf{a}_k)$. However, if the dimension of the exchanged message is high, this increases the complexity of the local model.

An alternative is to extract useful information $\mathbf{c}_k \in \mathbb{R}^{Z^{(c)}}$ from the received messages $\overline{\mathbf{m}}_k \in \mathbb{R}^{Z^{(m)}}$ using a function $g : \mathbb{R}^{Z^{(m)}} \to \mathbb{R}^{Z^{(c)}} : \overline{\mathbf{m}}_k \mapsto \mathbf{c}_k$, such that $Z^{(c)} \ll Z^{(m)}$, where $Z^{(m)}$ and $Z^{(c)}$ represent the corresponding dimensions. We then use $\tilde{r}_k(\mathbf{s}_k, \mathbf{c}_k, \mathbf{a}_k)$ to approximate r_k, capturing hidden information in the global state while maintaining low model complexity.

Pioneering works such as [89] proposed learning the extraction of communication messages by jointly optimizing the communication action with the RL model. However, joint training of multiple interacting models often leads to extended convergence times and even diverged training. We leverage domain knowledge to extract information to provide a robust and efficient practical solution. Knowing that inter-agent dependencies are primarily caused by load-coupling inter-cell interference, we propose that each agent k communicates the slice-specific load information $l_{k,n}$, $\forall n \in \mathcal{N}$ with its neighboring agents. Based on the exchanged load information, we compute the average per-slice neighboring load as the extracted information $\mathbf{c}_k(t)$. We define a deterministic function:

$$\begin{aligned} g_k :& \mathbb{R}^{N|\mathcal{K}_k|} \to \mathbb{R}^N : [l_{i,n} : n \in \mathcal{N}, i \in \mathcal{K}_k] \mapsto \mathbf{c}_k(t) \\ \text{with } \mathbf{c}_k(t) :=& \left[\frac{1}{|\mathcal{K}_k|} \sum_{i \in \mathcal{K}_k} l_{i,n}(t) : n \in \mathcal{N} \right]. \end{aligned} \tag{6.8}$$

In this way, the DIRP algorithm solves Problem 2 with approximated local rewards while considering inter-cell dependencies by including neighboring information. The DIRP algorithm approximates $r_k(\mathbf{s}, \mathbf{a})$ with $\tilde{r}_k(\mathbf{s}_k, \mathbf{c}_k, \mathbf{a}_k)$, decomposes Problem 2 into K independent

subproblems, and finds the following local policies $\pi_k : \mathcal{S}_k \times \mathbb{R}^N \to \mathcal{A}_k$ for each DIRP agent $k \in \mathcal{K}$:

$$\pi_k^* = \underset{\pi_k; \mathbf{a}_k \in \mathcal{A}_k}{\arg\max} \mathbb{E}_{\pi_k} \left[\sum_{t=0}^{T} \gamma_k^t \tilde{r}_k \big(\mathbf{s}_k(t), \mathbf{c}_k(t), \mathbf{a}_k(t) \big) \right]. \tag{6.9}$$

6.3.2 Training Setup of DIRP

We follow the actor-critic method [64] to train the agents, which has proven effective when dealing with high dimensional and continuous state space. Such method solves the optimization problem by using critic function $Q(\mathbf{s}_t, \mathbf{a}_t|\theta)$ (in this subsection, we denote $\mathbf{s}(t)$ and $\mathbf{a}(t)$ by $\mathbf{s}_t$ and $\mathbf{a}_t$ respectively for brevity) to approximate the value function, i.e., $Q(\mathbf{s}_t, \mathbf{a}_t|\theta) \approx Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$, and actor $\pi(\mathbf{s}_t|\phi)$ to update the policy π at every DRL step in the direction suggested by critic. For brevity, we denote the network with parameters in the form Q_θ and π_ϕ for critic and actor respectively.

Same as the training process of distributed DRL, we use TD3 algorithm [65] as an off-policy DRL algorithm built on top of the actor-critic method. As an extension of DDPG [66], TD3 overcomes the DDPG's problem of overestimating Q-values by introducing a double critic structure for both current networks $Q_{\theta_1}, Q_{\theta_2}$ and target networks $Q_{\theta_1'}, Q_{\theta_2'}$. The minimum of the two Q-values is used to represent the approximated Q-value of the next state. Besides, the updates of the policy network are less frequent than the value network, which allows the value network to reduce errors before it is used to update the policy network. Moreover, TD3 uses target policy smoothing, i.e., adding noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along with changes in action.

The target actions are computed based on the next state collected in the sample, given by

$$\mathbf{a}'(\mathbf{s}_{t+1}) = clip\left(\pi'_{\phi'}\left(\mathbf{s}_{t+1}\right) + clip(\epsilon, -c, c), \mathbf{a}_\mathrm{L}, \mathbf{a}_\mathrm{H}\right) \tag{6.10}$$

where the added noise $\epsilon \sim \mathcal{N}(0, \sigma)$ is clipped to keep the target close to the original action, and $\mathbf{a}_\mathrm{L}, \mathbf{a}_\mathrm{H}$ are the lower and upper bounds of the action, respectively.

The target update in TD3 is given by:

$$y_t = r_t + \gamma \min_{i=1,2} Q_{\theta_i'}\left(\mathbf{s}_{t+1}, \mathbf{a}'(\mathbf{s}_{t+1})\right). \tag{6.11}$$

The critic parameters $\theta_i, i \in \{1, 2\}$ are updated with temporal difference (TD) learning, given by:

$$L\left(\theta_i\right) = \mathbb{E}\left[\left(y_t - Q_{\theta_i}\left(\mathbf{s}_t, \mathbf{a}_t\right)\right)^2\right]. \tag{6.12}$$

The actor is updated by policy gradient based on the expected cumulative reward J with respect to the actor parameter θ^π with:

$$\begin{aligned} \nabla_\phi J &\approx \mathbb{E}\left[\nabla_\phi Q_{\theta_1}\left(\mathbf{s}, \mathbf{a}\right)|_{\mathbf{s}=\mathbf{s}_t, \mathbf{a}=\pi_\phi(\mathbf{s}_t)}\right] \\ &= \mathbb{E}\left[\nabla_\mathbf{a} Q_{\theta_1}\left(\mathbf{s}, \mathbf{a}\right)|_{\mathbf{s}=\mathbf{s}_t, \mathbf{a}=\pi_\phi(\mathbf{s}_t)} \nabla_\phi \pi_\phi\left(\mathbf{s}_t\right)\right]. \end{aligned} \tag{6.13}$$

The parameters of the target networks are updated with the soft update to ensure that the TD-error remains small:

$$\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i', i = 1, 2; \phi' \leftarrow \tau\phi + (1-\tau)\phi'. \tag{6.14}$$

6.3.3 Dealing with Resource Constraints

To address the inter-slice resource constraints in Eq. 6.1, we propose a method by reconstructing the network architecture of DRL model with an additional regularization layer.

Based on the findings of comparing RL reward formulation in Section 5.4.3.2, we embed a decoupled regularization layer into the output layer of the actor network, such that this layer becomes part of the end-to-end back propagation training of the neural network. Since the softmax function realizes for each $\mathbf{a}_k$ the following projection

$$\sigma : \mathbb{R}^N \rightarrow \left\{ \mathbf{a}_k \in \mathbb{R}^N \middle| a_{k,n} \geq 0, \sum_{n=1}^{N} a_{k,n} = 1 \right\},$$

In summary, we provide the TD3-based DIRP algorithm with inter-agent coordination in Algo. 8.

Algorithm 8 The DIRP Algorithm

Initialize parameters for critics $Q_{\theta_1^k}$, $Q_{\theta_2^k}$, and actor π_{ϕ^k} with random parameters θ_1^k, θ_2^k, ϕ^k, $\forall k \in \mathcal{K}$
Initialize target networks $\theta_1'^k \leftarrow \theta_1^k$, $\theta_2'^k \leftarrow \theta_2^k$, $\phi'^k \leftarrow \phi^k$
Initialize empty replay buffer $\mathcal{B}_k$
Initialize $\epsilon \in [0,1]$ and decay $d \in [0,1]$ for ϵ-greedy exploration
Define time periods $\mathcal{H}^{(\text{Expl})}, \mathcal{H}^{(\text{Train})}, \mathcal{H}^{(\text{Eval})}$ for exploration, training, and evaluation phases, respectively
Repeat
for local agent $k \in \mathcal{K}$ **do**
    Observe local state $s_k(t)$ and information $\mathbf{c}_k(t)$
    Select and execute action:
    if $t \in \mathcal{H}^{(\text{Expl})}$ **then**
        $\mathbf{a}_k(t) \leftarrow$ **random choice**
    else if $t \in \mathcal{H}^{(\text{Train})}$ **then**
        $\mathbf{a}_k(t) \leftarrow \begin{cases} \pi_k(\mathbf{s}_k(t), \mathbf{c}_k(t)) + \varepsilon, & \text{if } U[0,1] > \epsilon \\ \textbf{random choice}, & \text{otherwise} \end{cases}$
        where $U[0,1]$ is the random value following uniform distribution in $[0,1]$.
        $\epsilon \leftarrow d\epsilon$
    else if $t \in \mathcal{H}^{(\text{Eval})}$ **then**
        $\mathbf{a}_k(t) = \pi_k(\mathbf{s}_k(t), \mathbf{c}_k(t))$
    end if
    Observe next state $\mathbf{s}_k(t+1)$, received information $\mathbf{c}_k(t+1)$, and compute $r_k(t)$
    Store instance in $\mathcal{B}_k$: $\Big(\big(\mathbf{s}_k(t), \mathbf{c}_k(t)\big), \mathbf{a}_k(t), \big(\mathbf{s}_k(t+1), \mathbf{c}_k(t+1)\big), r_k(t) \Big)$
    if time to update networks **then**
        Sample mini-batch of B instances from $\mathcal{B}_k$
        Compute target actions and targets using Eq. 6.12 and Eq. 6.13
        Update critic and actor based on Eq. 6.12 and Eq. 6.13
        if t mod policy_delay **then**
            Update target networks using Eq. 6.14
        end if
    end if
end for

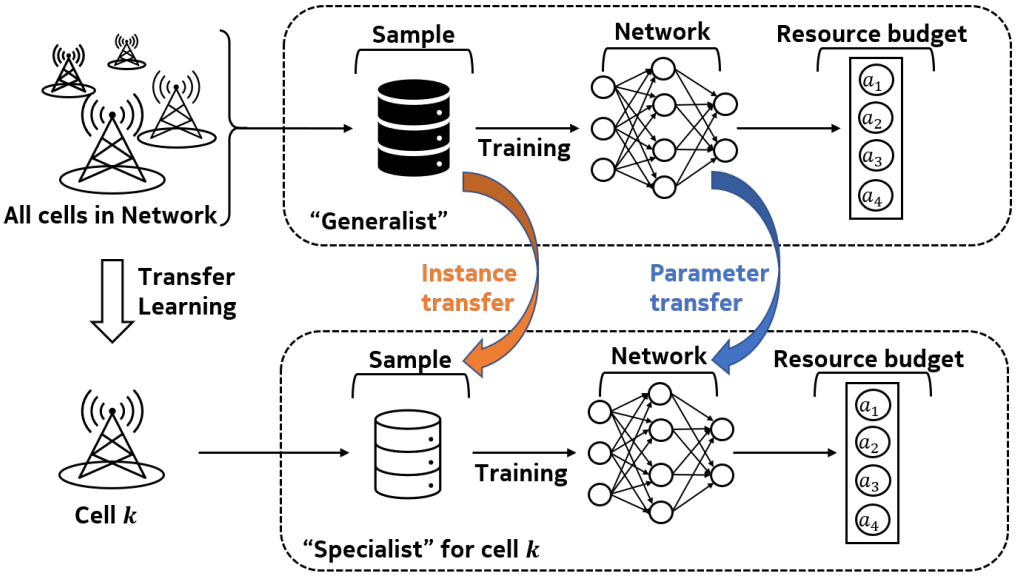

Figure 6.1: Generalist-to-Specialist TL scheme

6.4 TL-aided DIRP Algorithm

In this section, we introduce a TL method for multi-agent DRL approach with knowledge transfer in three ways, as sample knowledge transfer, model knowledge transfer, and full knowledge transfer, for faster training convergence, higher sample efficiency and model reproducibility.

As discussed in Section 6.3.1, the DIRP algorithm achieves a good trade-off between reducing model complexity and capturing the inter-cell dependencies. However, each agent needs to learn the local policy from scratch and still faces the well-known challenge of the exploration-exploitation dilemma. The environment dynamics and state transitions are usually unknown at the early stage of training, and the agent cannot exploit its knowledge until the state-action space is exhaustively explored. Moreover, because the local model is trained on a specific data domain, the learned model is sensitive to domain shift (a change in the data distribution between the training dataset and the dataset that it encounters when deployed). This means that even a slight change in the environment may result in deteriorated performance, and the agent may face a long period of retraining time.

To overcome the above-addressed challenges, we raise a hypothesis that some common hidden pattern may exist in the critic and actor networks across different agents and propose to enhance the developed coordinated MADRL algorithm with TL. We expect the TL to improve the model reproducibility and speed up the learning convergence by performing the following two major steps as demonstrated in Fig. 6.1:

1. **Centralized training of a "generalist":** A centralized controller collects the samples from all local agents $\Big(\big(\mathbf{s}_k(t), \mathbf{c}_k(t)\big), \mathbf{a}_k(t), \big(\mathbf{s}_k(t+1), \mathbf{c}_k(t+1)\big), r_k(t)\Big)$, $\forall k \in \mathcal{K}$ for a time period $t = 0, \ldots, T^{(\mathrm{G})}$ and trains a generalized model by interacting with the environment based on the same model in training for all agents.
2. **Distributed TL and fine-tuning to the "specialists":** After time slot $T^{(\mathrm{G})}$, we transfer the learned knowledge in the "generalist" to each local agent (i.e., the "specialists"), and fine-tune the customized model locally. The details of different types of transferable knowledge are provided later in Section 6.4.2.

6.4.1 TL Problem Formulation

Before introducing the TL problem in the context of MADRL, let us first introduce a general definition of TL.

A **domain** $\mathcal{D} := \{\mathcal{X}, P(X)\}$ consists of a feature space $\mathcal{X}$ and its probability distribution $P(X), X \in \mathcal{X}$. A **task** $\mathcal{T} := \{\mathcal{Y}, f(\cdot)\}$ consists of a label space $\mathcal{Y}$ and a predictive function $f(\cdot)$, where $f(\cdot)$ can be written as $P(Y|X), Y \in \mathcal{Y}$ and $X \in \mathcal{X}$. Formally, the general definition of the TL is given below.

Definition 3 (TL [70]). *Given a source domain $\mathcal{D}_S$ and a source learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and a target learning task $\mathcal{T}_T$, TL aims to improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$.*

In the context of DRL, a domain $\mathcal{D} := \{\mathcal{S}, P(\mathbf{s})\}$ consists of the state space $\mathcal{S}$ and its probability distribution $P(\mathbf{s}), \mathbf{s} \in \mathcal{S}$, while the task $\mathcal{T} := \{\mathcal{A}, \pi(\cdot)\}$ consists of the action space $\mathcal{A}$ and a policy function $\pi(\cdot)$. In general, the policy π is a mapping from states to a probability distribution over actions. With the actor-critic method introduced in Section 6.3.2, the policy directly maps the state space to optimized action, thus, we have $\pi : \mathcal{S} \to \mathcal{A}$.

In the scope of our proposed generalist-to-specialist TL-DIRP algorithm, we introduce the following definitions of the source domain, source task, target domain, and target task.

- **Source domain:** $\mathcal{D}_S := \mathcal{D}^{(\mathrm{G})}$ consists of the joint state and communicated message space $\mathcal{S}^{(\mathrm{G})} \times \mathbb{R}^N$ and its probability distribution $P\left(\mathbf{s}^{(\mathrm{G})}, \mathbf{c}^{(\mathrm{G})}\right)$, where $\mathbf{s}^{(\mathrm{G})} \in \mathcal{S}^{(\mathrm{G})} := \cup_{k\in\mathcal{K}}\mathcal{S}_k$ and $\mathbf{c}^{(\mathrm{G})} \in \mathbb{R}^N$. The state $\mathbf{s}^{(\mathrm{G})}$ and message $\mathbf{c}^{(\mathrm{G})}$ are collected by the centralized controller from all local agents.
- **Source task:** $\mathcal{T}_S := \mathcal{T}^{(\mathrm{G})}$ consists of the general action space $\mathcal{A}^{(\mathrm{G})}$ and the policy function $\pi^{(\mathrm{G})} : \mathcal{S}^{(\mathrm{G})} \times \mathbb{R}^N \to \mathcal{A}^{(\mathrm{G})}$. The general policy $\pi^{(\mathrm{G})}$ is trained on the instances collected by all agents.
- **Target domain:** $\mathcal{D}_T := \mathcal{D}_k^{(\mathrm{S})}, k \in \mathcal{K}$ consists of the joint local state and communication message space $\mathcal{S}_k \times \mathbb{R}^N$ and its probability distribution $P\left(\mathbf{s}_k, \mathbf{c}_k\right)$, where $\mathbf{s}_k \in \mathcal{S}_k$ and $\mathbf{c}_k \in \mathbb{R}^N$.
- **Target task:** $\mathcal{T}_T := \mathcal{T}_k^{(\mathrm{S})}, k \in \mathcal{K}$ consists of the local action space $\mathcal{A}_k$ and local policy $\pi_k : \mathcal{S}_k \times \mathbb{R}^N \to \mathcal{A}_k$.

The problem of TL from a source DRL agent as a "generalist"to a set of target DRL agents, i.e., the local "specialists", is formulated in Problem 3.

Problem 3. *Given source domain $\mathcal{D}^{(\mathrm{G})} := \left\{\mathcal{S}^{(\mathrm{G})} \times \mathbb{R}^N, P\left(\mathbf{s}^{(\mathrm{G})}, \mathbf{c}^{(\mathrm{G})}\right)\right\}$ and pre-trained source task $\mathcal{T}^{(\mathrm{G})} := \left\{\mathcal{A}^{(\mathrm{G})}, \pi^{(\mathrm{G})}(\cdot)\right\}$, TL aims to learn an optimal local policy for the target domain $\mathcal{D}_k^{(\mathrm{S})} := \left\{\mathcal{S}_k \times \mathbb{R}^N, P\left(\mathbf{s}_k, \mathbf{c}_k\right)\right\}, \forall k \in \mathcal{K}$ by leveraging the knowledge extracted from $\left(\mathcal{D}^{(\mathrm{G})}, \mathcal{T}^{(\mathrm{G})}\right)$, as well as the knowledge exploited in the target domain $\mathcal{D}_k^{(\mathrm{S})}$. The problem is given by*

$$\max_{\pi_k|\pi_k^{(0)}=\Lambda\left(\pi^{(\mathrm{G})}\right)} \mathbb{E}_{\pi_k}\left[\sum_{t=0}^{T}\gamma_k^t\tilde{r}_k\left(\mathbf{s}_k(t), \mathbf{c}_k(t), \mathbf{a}_k(t)\right)\right] \tag{6.15}$$
$$s.t.\ \left(\mathbf{s}_k, \mathbf{c}_k, \mathbf{a}_k\right) \in \Omega\left(\mathcal{D}^{(\mathrm{G})}, \mathcal{D}_k^{(\mathrm{S})}, \mathcal{A}^{(\mathrm{G})}, \mathcal{A}_k\right).$$

*where $\Lambda\left(\pi^{(\mathrm{G})}\right)$ is the **policy transfer strategy** which maps the pre-trained source policy $\pi^{(\mathrm{G})}$ to an initial local policy $\pi_k^{(0)}$, while $\Omega\left(\mathcal{D}^{(\mathrm{G})}, \mathcal{D}_k^{(\mathrm{S})}, \mathcal{A}^{(\mathrm{G})}, \mathcal{A}_k\right)$ is the **instance transfer strategy** which extracts the instances from the source domain and combines them with the experienced instances from the target domain.*

6.4.2 Generalist-to-Specialist TL

The problem defined in Eq. 6.15 offers various options for transferable knowledge:

- **pre-trained model transfer:** The policy transfer strategy $\Lambda(\cdot)$ simply maps the pre-trained source policy to itself, i.e., the local agent uses the pre-trained general policy $\pi^{(\mathrm{G})}$ as the initial policy $\pi_k^{(0)}$ and fine-tunes it by further interacting with the environment with locally made decisions.
- **Feature extraction:** $\Lambda(\cdot)$ keeps partial knowledge of $\pi^{(\mathrm{G})}$. In DRL, the policy $\pi^{(\mathrm{G})}\left(\mathbf{s}^{(\mathrm{G})}, \mathbf{c}^{(\mathrm{G})} | \boldsymbol{\phi}^{(\mathbf{G})}\right)$ is characterized by the pre-trained parameters (weights) of the neural networks. Feature extraction freezes partial of the layers (usually the lower layers) of the pre-trained neural networks while leaving the rest of them to be randomly initialized.
- **Instance transfer:** Except for the instances from the target domain, the agent also trains its policy using the extracted instances from the source domain. The instance transfer strategy $\Omega(\cdot)$ decides which instances are chosen from the source domain to be combined with the instances from the target domain in the local replay buffer.

The above-mentioned knowledge from the source domain and task can be transferred separately or in a combined manner. In this paper, we focus on studying the following three TL schemes:

- **pre-trained model transfer only:** Each local agent k uses the pre-trained general policy $\pi^{(\mathrm{G})}$ to initialize the local policy $\pi_k^{(0)}$. With the actor-critic method described in Section 6.3.2, we simply load the pre-trained parameters of the actor and critic networks from the generalist to the local agents. However, when the difference between the source and target domain is large, the local agent still needs extensive exploration to fine-tune the general policy to a customized local policy.
- **Instance transfer only:** Each local agent offloads a set of selected instances in the source domain from the centralized controller to the local replay buffer. Then, the local agent trains a policy from scratch with the replay buffer containing mixed offline instances from the source domain and the experienced online instances in the target domain. In this paper, we select the instances collected from the exact same local agent. Future work includes the similarity analysis between agents and instance selection from similar agents, which falls into the subject of **domain adaptation** [90].
- **Combined model and instance transfer:** To fully exploit the transferable knowledge, we combine the pre-trained model transfer and instance transfer. Firstly, each local agent retrieves $\pi^{(\mathrm{G})}$ from the centralized controller and uses it to initialize the local policy $\pi_k^{(0)}$. Then, we further investigate two options for local fine-tuning:
  - **Online fine-tuning with mixed replay buffer:**
    The local agent further online fine-tunes the policy with the replay buffer containing both the offloaded instances from the source domain and the locally experienced instances from the target domain.
  - **Offline fine-tuning with offloaded instances & online fine-tuning with experienced instances:**
    The local agent first offline fine-tunes $\pi^{(\mathrm{G})}$ with the offloaded instances. Then, the offline fine-tuned model is used to initialize $\pi_k^{(0)}$ and further fine-tuned online with the locally experienced instances in the target domain.

Note that our experiments focus on the pre-trained model transfer and instance transfer, while do not include the feature extraction. This is because feature exaction usually performs well when the target domain is highly similar to the source domain. However, in general, the similarity between the generalist domain and the specialist domain is not sufficiently high. Thus, the feature exaction method may better suit the scenario of inter-agent TL, while it may not be appropriate for generalist-to-specialist knowledge transfer.

We illustrate the TL-DIRP algorithm with a combined model and instance transfer in Algo. 9.

Algorithm 9 TL-Aided DIRP Algorithm

I. Generalist training in centralized controller
Initialize generalist critics $Q_{\theta_1^{(\mathrm{G})}}$, $Q_{\theta_2^{(\mathrm{G})}}$ and actor $\pi_{\phi^{(\mathrm{G})}}$ with random parameters $\theta_1^{(\mathrm{G})}$, $\theta_2^{(\mathrm{G})}$, $\phi^{(\mathrm{G})}$
Initialize target networks $\theta_1'^{(\mathrm{G})} \leftarrow \theta_1^{(\mathrm{G})}$, $\theta_2'^{(\mathrm{G})} \leftarrow \theta_2^{(\mathrm{G})}$, $\phi'^{(\mathrm{G})} \leftarrow \phi^{(\mathrm{G})}$
Initialize empty replay buffer $\mathcal{B}^{(\mathrm{G})}$
Define time periods $\mathcal{H}^{(\mathrm{G})}, \mathcal{H}^{(\mathrm{S})}$ for generalist training and specialist fine-tuning respectively
for $t \in \mathcal{H}^{(\mathrm{G})}$ **do**
    Collect observations of local states $\mathbf{s}_k(t)$ and received information $\mathbf{c}_k(t)$, $\forall k \in \mathcal{K}$
    Use general policy $\pi^{(\mathrm{G})}$ to select and execute action $\mathbf{a}_k(t), \forall k \in \mathcal{K}$
    Observe the next local states $\mathbf{s}_k(t+1)$ and information $\mathbf{c}_k(t+1)$, compute local rewards $r_k(t)$, $\forall k \in \mathcal{K}$
    Store K instances in replay buffer $\mathcal{B}^{(\mathrm{G})}$
    Train and update the general critics $Q_{\theta_i^{(\mathrm{G})}}, i = 1, 2$, actor $\pi_{\phi^{(\mathrm{G})}}$, and target critics $Q_{\theta_i'^{(\mathrm{G})}}, i = 1, 2$ and actor $\pi_{\phi'^{(\mathrm{G})}}$ using the TD3 algorithm in Section 6.3.2
end for
II. Specialist fine-tuning in local agents
Initialize parameters for critics $Q_{\theta_1^k}$, $Q_{\theta_2^k}$ and actor π_{ϕ^k} with $\theta_1^k \leftarrow \theta_1^{(\mathrm{G})}$, $\theta_2^k \leftarrow \theta_2^{(\mathrm{G})}$, $\phi^k \leftarrow \phi^{(\mathrm{G})}$, $\forall k \in \mathcal{K}$
Initialize target networks $\theta_1'^k \leftarrow \theta_1^k$, $\theta_2'^k \leftarrow \theta_2^k$, $\phi'^k \leftarrow \phi^k$
Offload selected instances from $\mathcal{B}^{(\mathrm{G})}$ to $\mathcal{B}_k$
for $t \in \mathcal{H}^{(\mathrm{S})}$ **do**
    for Local agent $k \in K$ **do**
        fine-tune local policy with Algo. 8 (except for the initialization steps)
    end for
end for

By introducing Algo. 9, we partially address Question 3 by proposing the transfer of general knowledge captured from multiple source domains and broadcasting it to individual agents. This approach assumes that we have sufficient samples to derive a generalist DRL agent. Later, in Chapter 7, we will explore another case where there is limited knowledge available for solving a general distributed DRL problem.

6.5 Experiments

In this section, we evaluate the performance of the proposed methods for inter-cell slicing resource partitioning introduced in Sections 6.3 and 6.4 with Season II simulator. To implement our proposed DRL solution, we build in the simulator a network with 4 sites (12 cells) covering an urban area of Helsinki city, as demonstrated in Fig. 6.2, consisting of 4 three-sector macro sites. All cells are deployed using LTE radio technology with 2.6 GHz. We use the realistic radio propagation model Winner+ [67].

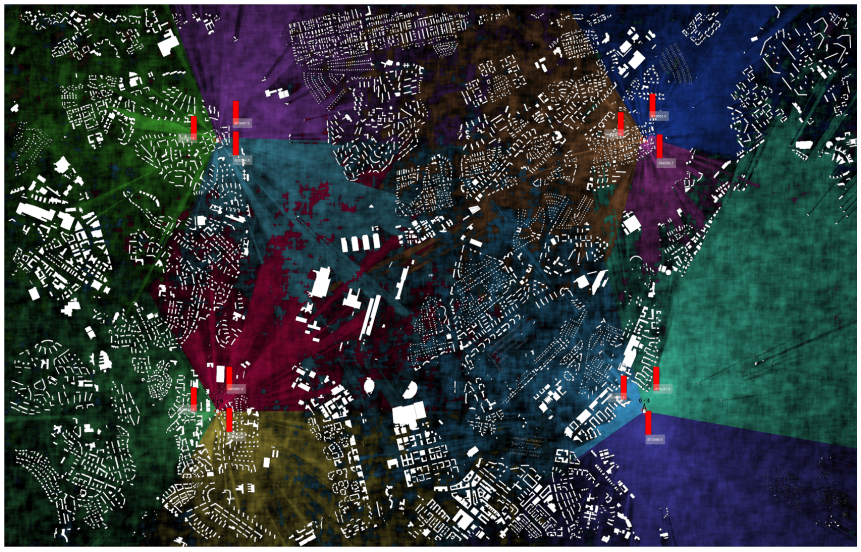

Figure 6.2: Network environment setup with 12 cells

The network is built up with $N = 4$ network slices, with per-slice throughput requirements of $\phi_1^* = 4$ Mbit/s, $\phi_2^* = 1$ Mbit/s, $\phi_3^* = 3$ Mbit/s, and $\phi_4^* = 0.5$ Mbit/s and per-slice delay requirements of $d_1^* = 1$ ms, $d_2^* = 1.5$ ms, $d_3^* = 2$ ms, and $d_4^* = 1$ ms respectively. All cells in the network have a fixed bandwidth of 20 MHz.

We define four groups of UEs associated with each defined slice respectively, i.e., 16 groups of UEs in total, all with the maximum group size of 10. UEs are moving uniformly randomly within the defined moving sphere of each group. The positions and moving radius of UEs groups are defined heterogeneously to ensure that each site can serve UE from all slices. To imitate the time-varying traffic pattern, we also apply a time-dependent traffic mask $\tau_n(t) \in [0, 1]$ for each slice $n \in \mathcal{N}$ to scale the total number of UEs in the scenario. In Fig. 6.3, we demonstrate the changes of the first 2 days from a three-week traffic mask. The UE traffic volume is updated every timestamp, which corresponds to 15 minutes in real time, also known as the typical KPI reporting time in OAM. In the experiments, the entire traffic mask is extended and periodically repeated after every 2016 timestamp corresponding to the three-week time period (96 timestamps per day).

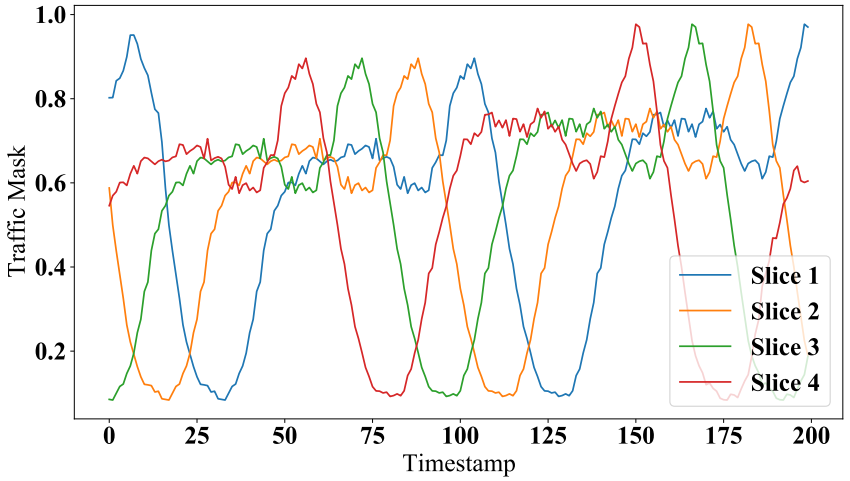

Figure 6.3: The first two days of a three-week traffic mask

6.5.1 Schemes for Comparison

For performance evaluation, we compare the proposed DIRP and TL-DIRP algorithms with the following three baselines:

Table 6.2: Comparison of dimensions of DRL models used in simulation

	BL-Cen	**BL-Dist**	**DIRP & TL-DIRP**
State	Global state $\mathbf{s} \in \mathbb{R}^{240}$	Local state $\mathbf{s}_k \in \mathbb{R}^{20}$	Local state with extracted message $[\mathbf{s}_k, \mathbf{c}_k] \in \mathbb{R}^{24}$
Action	Global action $\mathbf{a} \in [0,1]^{48}$	Local action $\mathbf{a}_k \in [0,1]^4$	Local action $\mathbf{a}_k \in [0,1]^4$
Reward	Global reward r_G^m in Eq. 6.3	Local reward r_k^m in Eq. 6.6	Local reward r_k^m in Eq. 6.6

- **BL-Cen:** The centralized DRL approach solving Eq. 6.5 referring to [91], assuming full observation of the global state $\mathbf{s} \in \mathcal{S}$. The agent computes the global reward and makes the decision of the slicing resource partitioning for all agents $\mathbf{a} \in \mathcal{A}$.
- **BL-Dist:** The distributed DRL approach without inter-agent coordination [58].
- **BL-Heur:** A traffic-aware heuristic approach that assumes perfect knowledge about per-slice traffic demand and dynamically adapts to the current per-slice traffic amount. It is implemented by dividing the resource in each cell $k \in \mathcal{K}$ to each slice proportionally to the amount of traffic demand per slice.

The DRL-based schemes to evaluate and compare are summarized in Table 6.2. Similarly, to evaluate the TL-DIRP algorithm and compare between different types of knowledge to transfer, we implement the proposed TL method in Section 6.4, i.e., centralized training of a generalist and then distributed fine-tuning to the specialist. We compare different transferable knowledge: instances, pre-trained model, and combined instances and pre-trained model. In addition, to ensure a safer exploration and better performance during online training, we perform the offline fine-tuning using the transferred instance before the online training in each local agent.

- **Gen:** centralized training of a general policy in the centralized controller based on the collected samples from all local agents, as described in Algo. 9.
- **Spec:** distributed fine-tuning of the specialists with full knowledge transfer. Each local agent initializes its critic and actor networks with the generalist model parameters. It also initializes the local replay buffer with the offloaded selected instances from the generalist buffer.
- **Spec-Instance:** distributed fine-tuning of the specialists with instance transfer only. The model parameters in each local agent are randomly initialized.
- **Spec-Model:** distributed fine-tuning of the specialists with model transfer only. Each local agent initializes its critic and actor networks by loading the generalist model parameters, while the local buffer is initialized as an empty queue.
- **TL-DIRP:** In addition to Spec (full knowledge transfer), we apply the offline fine-tuning based on the transferred instances before the online training.

Note that for "generalist-to-specialist"TL schemes with complete knowledge, we apply both max-min fairness and logarithmic utilities as local reward r_k for $k \in \mathcal{K}$ respectively:

- **TL-DIRP-Maxmin:** TL-DIRP approach with max-min fairness reward Eq. 6.6.
- **TL-DIRP-Log:** TL-DIRP approach with on logarithmic utility reward Eq. 6.7.

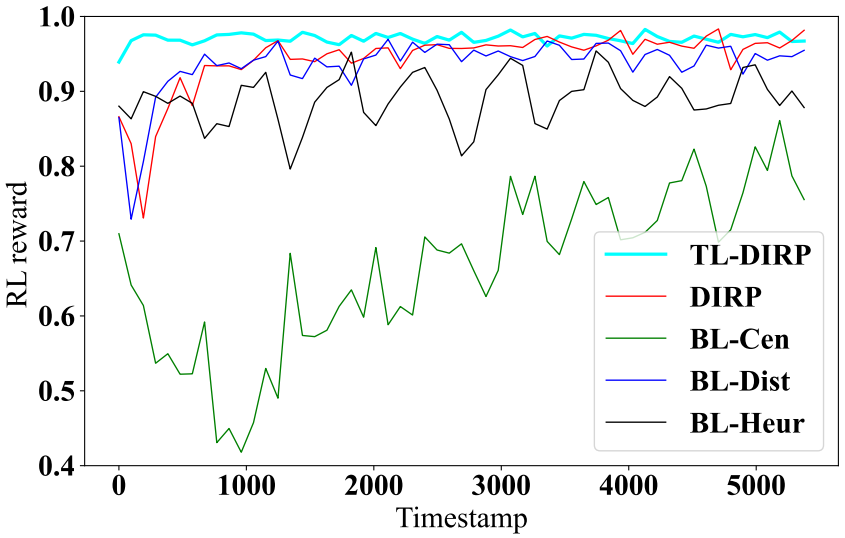

Figure 6.4: Comparison of reward among schemes

6.5.2 Hyperparameter Setup for Training

For DRL training, we use MLP architecture for actor-critic networks of TD3 algorithm. In the BL-Cen scheme, the actor and critic network models are built up with 3 hidden layers, with the number of neurons $(384, 192, 64)$ and $(324, 144, 64)$, respectively. While for BL-Dist and DIRP schemes, both actor-critic networks only have 2 hidden layers, with the number of neurons $(48, 24)$ and $(64, 24)$, respectively. In all schemes, the learning rates of actor and critic are 0.0005 and 0.001, respectively, with Adam optimizer and training batch size of 32. We choose a small DRL discount factor $\gamma = 0.1$ since the current action strongly impacts the instantaneous reward while having a weaker impact on the future reward. For the distributed DRL approaches, we only apply 100 steps for exploration, while for the centralized approaches, we apply 500 steps of exploration since the centralized agent has much higher dimensions of state and action. After the exploration phase, we apply 5000 steps for training and the final 500 steps for evaluation of all approaches.

In TL training, we apply the same DRL settings. For TL training setup, we set 100 steps for exploration, 5000 steps for learning, and 500 steps for evaluation in Gen and Spec-Model schemes, while in other TL execution schemes, we skip the exploration phase. The result of each process is derived from the average of 3 times of experiments.

In this work, we apply an orientated exploration strategy that chooses the new action under the recommendation of the traffic-aware heuristic policy, namely, the heuristic baseline BL-Heur. The reason is that we observe that BL-Heur provides sub-optimal performance without any training process. At the beginning of the exploration phase, the probability of using traffic-aware exploration is 0.5, and that of random exploration is also 0.5. Then, during the exploration, the probability of traffic-aware exploration gradually increases, and that of random exploration decreases.

6.5.3 Performance Evaluation

In this section, we provide the numerical results derived from experiments on the virtual network systems in Season II.

6.5.3.1 Comparison of the Distributed MADRL Schemes

In this comparison, we apply the reward design for max-min fairness to all approaches, i.e., global reward based on Eq. 6.3 for BL-Cen and local reward based on Eq. 6.4 for BL-Dist and DIRP. While for comparison between the different reward functions, we implement DIRP algorithm with both types of local reward r_k based on Eq. 6.6 and Eq. 6.7.

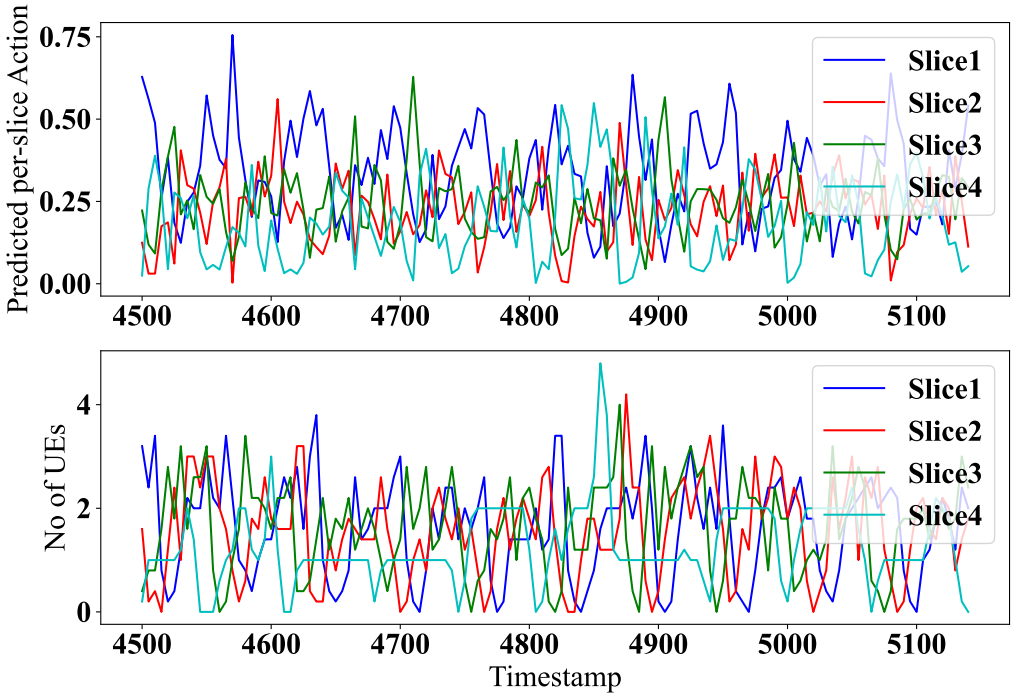

Figure 6.5: Adaptive action to traffic mask after training

Fig. 6.4 demonstrates the comparison of max-min fairness reward Eq. 6.3 during the training process among the baseline schemes BL-Cen, BL-Dist, BL-Heur, and the proposed DIRP and TL-DIRP algorithms.

As shown in Fig. 6.4, TL-DIRP provides the best performance among all approaches in terms of faster convergence, higher start point, and higher robustness after convergence.

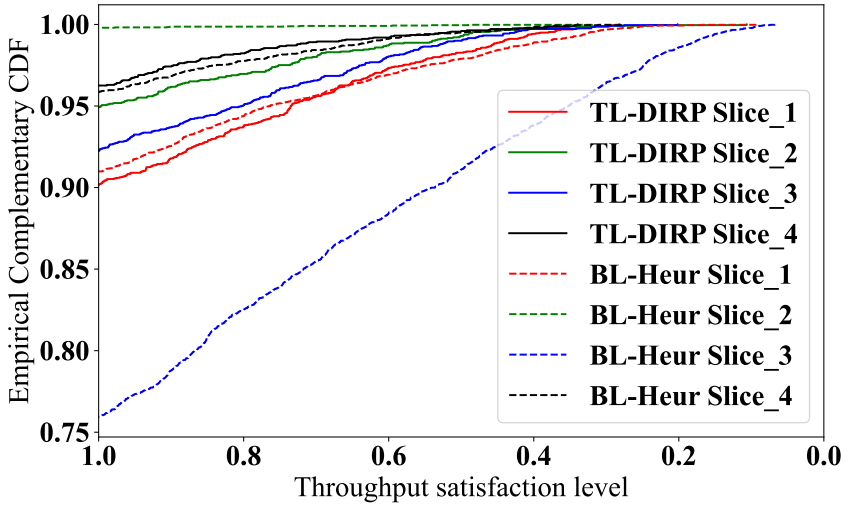

Figure 6.6: Comparing throughput QoS between TL-DIRP and BL-Heur

Compared to baselines, the DIRP algorithm achieves a significantly better global reward than BL-Heur after convergence. Note that BL-Heur is already a well-performed baseline because it assumes perfect traffic awareness and offers all resources to the UEs. On the other hand, BL-Cen fails to achieve performance as good as DIRP within the same training time. As Table 6.2 indicates, the dimensions of the state and action spaces of BL-Cen are much higher than the distributed approaches, making the training process more difficult for large-scale networks. Not only does BL-Cen converge slower, but it also often experiences poor performance at the early stage of training. The training curves are turbulent, corresponding to the time-varying traffic demand in Fig. 6.3, while DIRP is more robust compared to BL-Heur and BL-Cen.

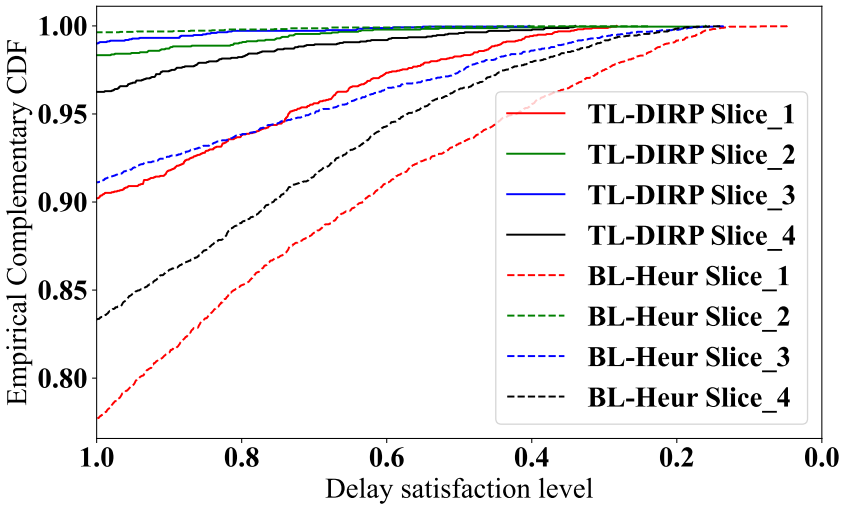

Figure 6.7: Comparing delay QoS between TL-DIRP and BL-Heur

In comparison between the two distributed schemes, according to Fig. 6.4, DIRP outperforms the BL-Dist scheme within the same training time period in terms of both converged global reward and convergence rate, which verifies the advantage of inter-agent coordination.

Fig. 6.5 shows the predicted action, i.e., per-slice resource partitioning as the ratio, and the actual traffic amount of DIRP in cell $k = 5$ after convergence. it verifies that the DRL approach well adapts its predicted actions to the dynamic network traffic demand with respect to different slice-specific QoS requirements.

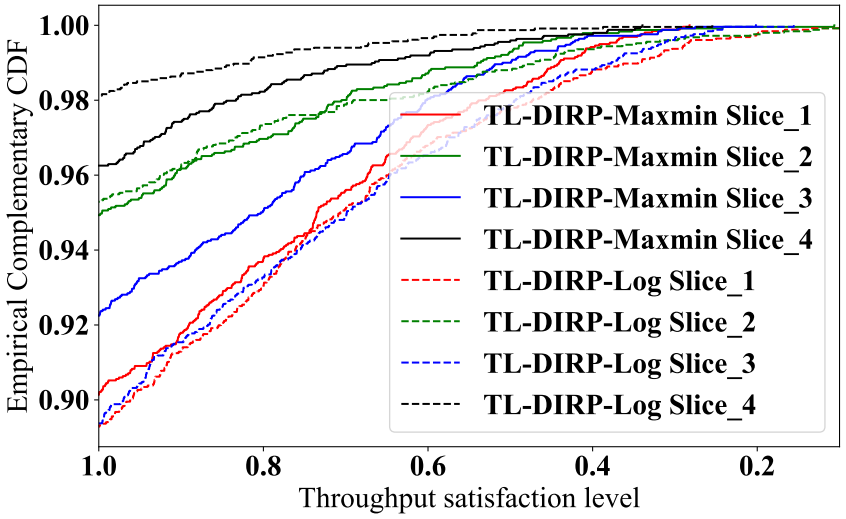

Figure 6.8: Comparing throughput QoS between utilities

Although DIRP shows better performance than baselines, it still faces two major challenges: slow convergence and oscillation. In Fig. 6.4, we show that TL-DIRP overcomes these challenges by transferring pre-learned knowledge. In particular, TL-DIRP achieves a much higher reward from the beginning of the learning process and quickly converges after a few hundred timestamps, while DIRP converges much slower because each local agent needs to learn from scratch. TL-DIRP outperforms DIRP in terms of both convergence rate and converged performance within the same time period.

Fig. 6.4 shows the evolving algorithms' performance during the training and testing process, while in the following, let us take a deeper look into the distributions of the converged service quality in terms of throughput and delay satisfaction level for each slice. Fig. 6.6 and Fig. 6.7 illustrate the empirical complementary CDF (or called survival function)

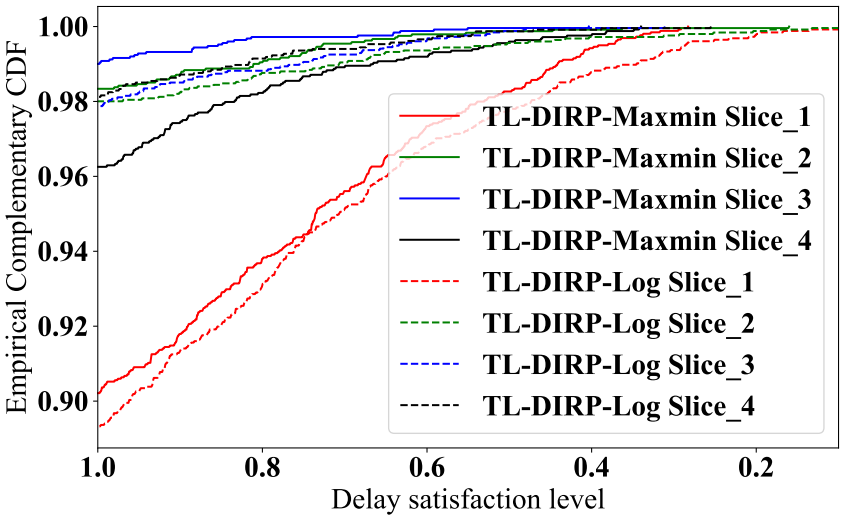

Figure 6.9: Comparing delay QoS between utilities

which equals $1 - F_X(x)$, where $F_X(x)$ denotes the CDF of per-slice throughput and delay satisfaction level between TL-DIRP and BL-Heur schemes, respectively.

Fig. 6.6 shows that TL-DIRP achieves 14% higher the worst-case throughput QoS among all slices than the traffic-aware baseline BL-Heur. It also guarantees that all the slices achieve a throughput satisfaction level above 90%, while BL-Heur serves Slice 3 with only 75% throughput satisfaction level.

Similar observation can be made for the delay satisfaction level in Fig. 6.7. TL-DIRP provides over 90% of the delay satisfaction level for all slices, while BL-Heur serves Slice 1 and 3 with only 77% and 83% respectively. In terms of the average delay satisfaction level over all slices, TL-DIRP achieves over 96% while BL-Heur only 88%. We observe that TL-DIRP attempts to fulfill more critical requirements by compromising resources from the less demanding slices while remaining sufficient satisfaction levels in others.

6.5.3.2 Comparison of RL Reward Formulation

Fig. 6.8 and Fig. 6.9 compare the two designs of the reward function, corresponding to max-min fairness Eq. 6.6 and maximizing average logarithmic utilities Eq. 6.7, respectively. They demonstrate the empirical complementary CDF of QoS in terms of throughput and delay satisfaction level for TL-DIRP with both reward functions. The results show that max-min fairness gives the maximum protection to the slice with the weakest performance, such that the minimum per-slice satisfaction level over all slices achieves 90% for both throughput and delay while maximizing average logarithmic utilities provides slightly lower satisfaction levels, about 89% for both throughput and delay, but higher maximum per-slice throughput satisfaction levels. This is because the logarithmic utility tends to distribute the resource more efficiently than max-min fairness, i.e., allocating more resources to the slice that can improve the averaged performance over all slices.

From an engineering perspective, max-min fairness is preferred for scenarios that require sufficiently good performance for all slices, especially those highly demanding ones. While the logarithmic utility is more suitable for cases that desire higher resource efficiency.

6.5.3.3 Comparison of TL Schemes

Fig. 6.10 illustrates the evolving rewards during the training and testing processes with different TL methods. Note that this comparison is based on max-min fairness in all TL methods. Here, we aligned the training process with the Spec scheme for comparison. The results are derived from the average of 3 independent instances of experiments. Spec

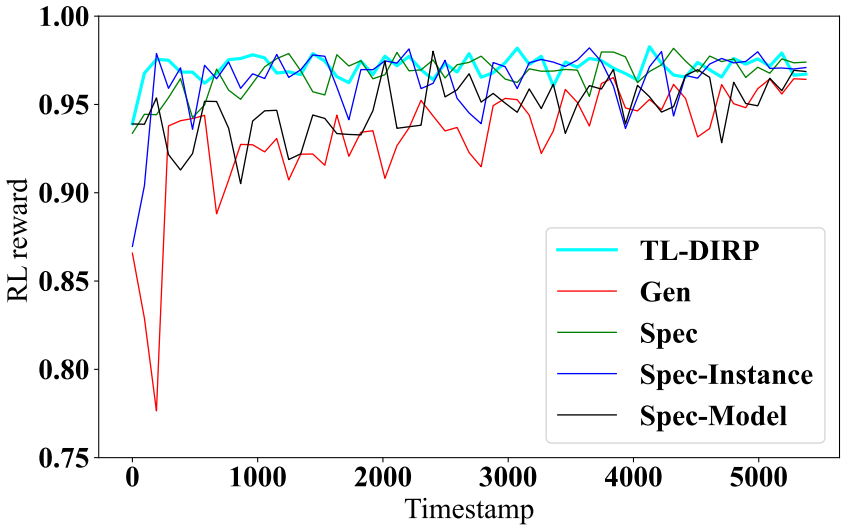

Figure 6.10: Comparing of reward among TL schemes

Table 6.3: Performance comparison among different schemes

	RL Reward	Min Slice Average Throughput Satisfy	Min Slice Average Delay Satisfy
BL-Cen	0.801	0.738	0.739
BL-Dist	0.948	0.952	0.962
BL-Heur	0.891	0.903	0.902
DIRP	0.968	0.967	0.967
Gen	0.961	0.960	0.961
Spec	0.972	0.968	0.968
Spec-Instance	0.971	0.970	**0.974**
Spec-Model	0.962	0.965	0.966
TL-DIRP	**0.973**	**0.971**	0.971

with complete knowledge transfer leads to higher reward and robustness compared to Spec-Instance and Spec-Model schemes with partial knowledge at the early stage of the training process, while in the latter two schemes, TL also helps in terms of convergence rate, compared with Gen. Furthermore, with offline fine-tuning TL-DIRP provides better performance with faster convergence and higher reward within the same training time. In most of the TL schemes, we observe that each specialist agent improves its performance with local fine-tuning from a higher starting point, which helps avoid risky action choices during exploration. With Spec-Instance, the agents behave the worst at the beginning of the training but converge fast later. On the other hand, Spec-Model also suffers from a weaker performance at the beginning and takes a longer time to learn. Eventually, Spec-Instance converges to a similar performance as Spec while Spec-Model achieves a slightly worse performance. Our guess is that there is still a substantial difference between the source and target domains. The initialized general policy cannot quickly adapt to the target task without transferring sufficient instances in the source domain (instances following a similar distribution to the target domain). Moreover, introducing offline fine-tuning with the transferred instances to the TL scheme further improves the performance by providing even faster convergence and more robust training.

In Fig. 6.11, we plot the change of local reward in each cell during the complete TL procedure from generalist training to specialist fine-tuning as described in Algo. 9. During the time in $\mathcal{H}^{(\mathrm{G})}$, the local rewards achieved by the generalist agent converge to a generally good reward over all cells. Later, in the local fine-tuning period $\mathcal{H}^{(\mathrm{S})}$, the Spec scheme further fine-tunes the general agent locally and concludes better performance in each cell.

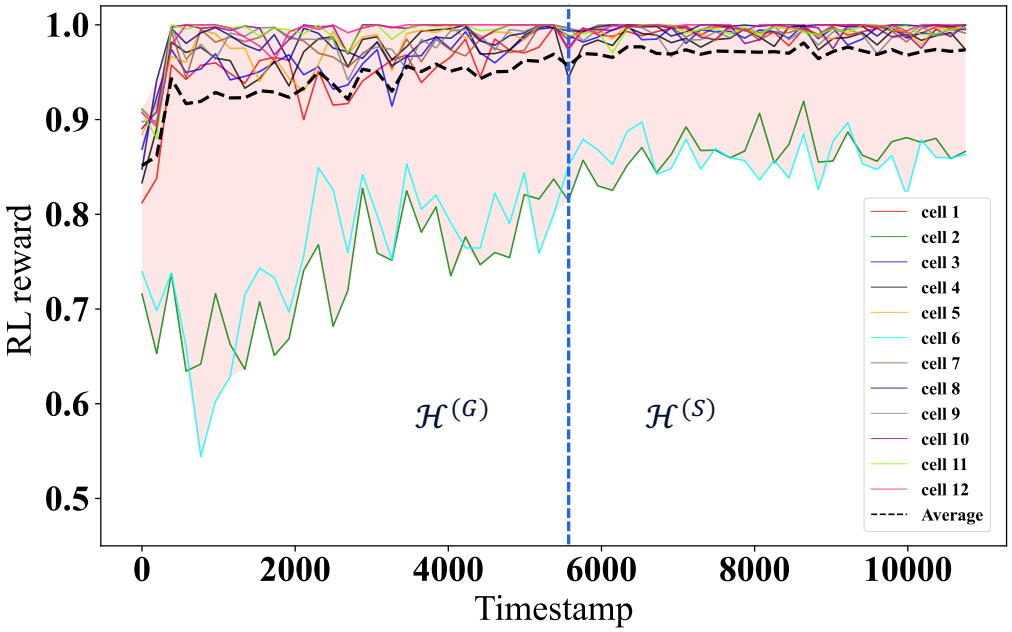

Figure 6.11: Change of local reward during TL scheme

The averaged local reward in $\mathcal{H}^{(S)}$ also indicates better robustness under time-varying traffic demand. We can also observe that the rewards from two cells are always lower compared to others during $\mathcal{H}^{(G)}$, and achieve relatively poor performance after knowledge transfer in $\mathcal{H}^{(S)}$. Fig. 6.12 shows the comparison of CQI distributions from all cells, it is clear to see that in cell 2 and cell 6 which derive poorer performance as shown in Fig. 6.11 correspondingly, the CQI histograms are significantly different compared to others. The difference in user distribution or radio propagation can make the "generalist"ambiguous on learning a general policy for all cells, and the derived policy is better for handling the samples from others. Thus, during $\mathcal{H}^{(G)}$, the rewards in these two cells are lower than others, while in $\mathcal{H}^{(S)}$, the performances in these two cells get better with local fine-tuning yet still worse than the others.

Summarized comparisons of the average performance metrics among all schemes in the testing phase are listed in Table 6.3. We can see that TL-DIRP as offline fine-tuned Spec provides the best performance in terms of the desired RL reward and minimum (worst-case) per-slice throughput satisfaction level among the schemes, while Spec-Instance provides a slightly better minimum per-slice delay satisfaction level. Moreover, TL-DIRP encourages a more balanced service quality between all slices in comparison to TL-DIRP-Log. It is also worth noting that the inference time for a pre-trained distributed DRL model to make a local decision is less than 4 milliseconds due to the small sizes of our defined neural networks.

The comparison of QoS performance metrics during the testing phase among all approaches are listed in Table 6.3. Same as indicated by the reward curves, the Spec-fine-tune scheme outperforms in all terms. Also, in comparison with the performance metrics of DIRP, Spec also provides better slice-specific performance.

6.6 Key Takeaways

Hereby, we summarize the key insights from the experiments above:

1. **Distributed vs. Centralized DRL Approaches:** Distributed DRL algorithms demonstrate superior learning capability for adapting to slice-aware traffic and maintaining good service quality in a network scenario with 12 cells. Due to high model complexity and large state and action spaces, the centralized scheme struggles to converge

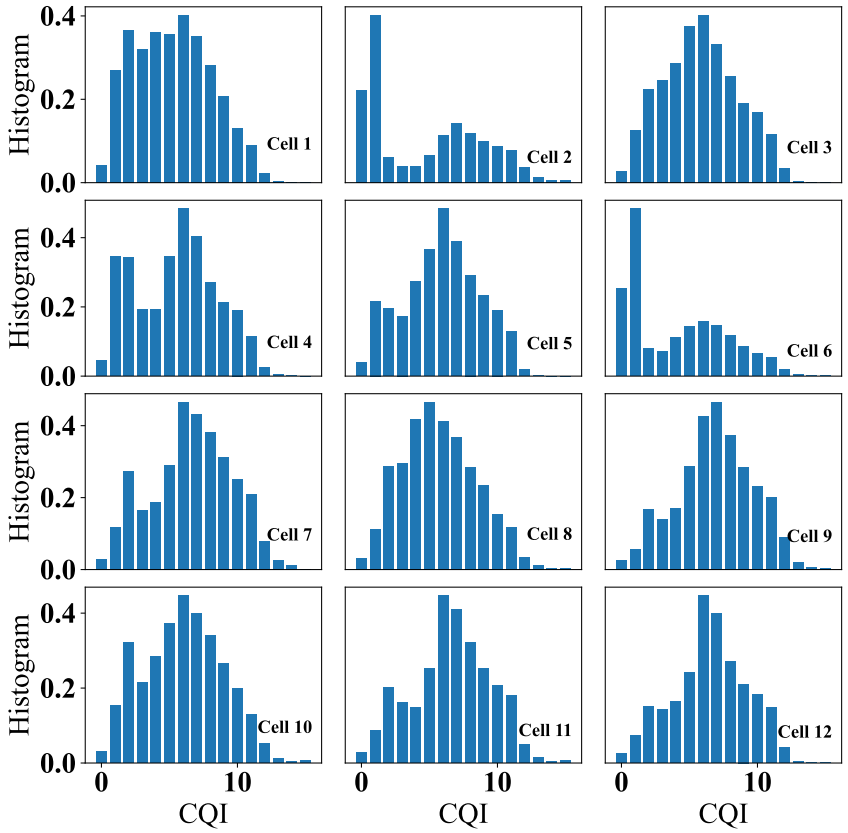

Figure 6.12: Comparison of Channel Quality Indicator (CQI) distribution between cells

Table 6.4: Comparison of average performance metrics among different TL approaches

	RL Reward	Min Per-slice Throughput/Requirement	Max Per-slice Delay (ms)
Gen	0.263	0.830	1.426
Spec	0.385	0.848	1.370
Spec-Data	0.363	0.841	1.388
Spec-Model	0.306	0.836	2.023
Spec-fine-tune	**0.400**	**0.853**	**1.339**

to an optimal reward within the same training time. As network scale increases, distributed schemes achieve even higher gains compared to centralized approaches.

2. **Inter-agent Coordination of MADRL:** The DIRP algorithm, which incorporates inter-agent coordination and allows multiple agents to share load information, outperforms distributed DRL without coordination. It achieves better performance in terms of converged reward and convergence rate while maintaining lower model complexity.

3. **Benefits of TL:** The proposed TL-DIRP algorithm further enhances the converged reward of DIRP by approximately 11.5%, achieves 87.5% faster convergence, and reduces exploration costs. It also provides about 15% higher QoS satisfaction for the most critical slice and an 8.8% higher average slice QoS satisfaction level than the traffic-aware baseline. Additionally, TL-DIRP demonstrates higher robustness in converged performance compared to DIRP without TL. With this finding, we can partially answer Question 5.

4. **Generalist-to-Specialist TL:** During the "generalist"training process of TL-DIRP, variations in CQI between cells can make learning a general policy ambiguous. Agents from different CQI conditions initially derive poorer performance. However,

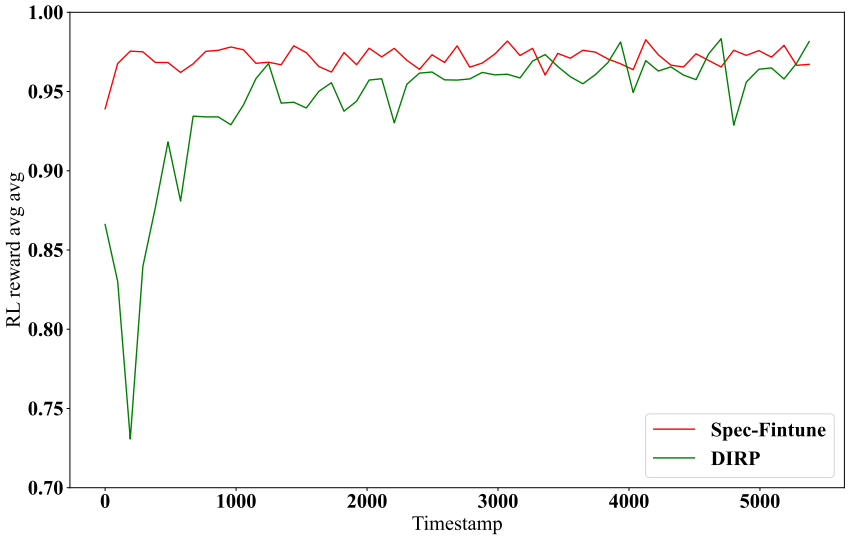

Figure 6.13: Comparing of reward between DIRP and Spec-fine-tune

agents achieve higher rewards and robustness with local fine-tuning in the "specialist" phase.

5. **Reward Functions Comparison:** The TL-DIRP approach, using both max-min fairness and logarithmic utility rewards, provides good performance across all slice QoS. Max-min fairness rewards better fulfill critical slice requirements by reallocating resources from less critical slices, whereas logarithmic utility rewards offer higher resource efficiency. Different reward definitions should be selected based on specific use cases.

6. **How to Transfer Knowledge:** The TL scheme that combines model and instance transfer, enhanced by offline fine-tuning, provides the best starting point and convergence rate performance. Transferring instances alone requires agents to train from scratch, resulting in low initial performance. Transferring a pre-trained model offers better initial performance but requires more time to converge. The disparity between source and target domains (e.g., CQI distribution) necessitates transferring sufficient instances to enable the general policy to adapt quickly to the target task. Offline fine-tuning with transferred instances further enhances performance by providing a higher start point and faster convergence.

In this chapter, we addressed the dynamic inter-cell resource partitioning problem to meet slice-aware service requirements by optimizing inter-cell inter-slice resource partitioning. We introduced the DIRP algorithm to solve this problem with inter-agent coordination. To enhance algorithm transferability, we developed the TL-DIRP algorithm using a generalist-to-specialist TL framework with various types of transferable knowledge. We evaluated the proposed solutions in a 12-cell network scenario in Season II. The results showed that the TL-DIRP algorithm offers superior slice-aware service performance compared to baseline approaches. Additionally, incorporating TL in MADRL improves training performance, including higher start points, faster convergence speeds, and higher asymptotes.

In addition to the generalist-to-specialist TL scheme, future work includes exploring TL among cells by transferring knowledge from a pre-trained DRL agent to another, such as transferring knowledge from a pre-trained cell to a newly deployed cell. However, as observed in numerical experiments, transferring knowledge between agents with different domains and tasks may initially degrade performance or cause negative transfer. Therefore, quantitative analysis is needed for efficient knowledge transfer to identify similar DRL agents.

7. TL-aided DRL Approach II: Specialist-to-Specialist

In this chapter, we further explore the TL-aided DRL approaches, focusing on transferring knowledge from one specific domain to another. Unlike generalized knowledge transfer, ensuring the effectiveness of TL in the target domain requires additional domain similarity analysis. By addressing these TL scenarios, we aim to tackle Question 4 and complete the answer to Question 5.

We propose a novel TL-aided MADRL approach with domain similarity analysis for inter-slice resource partitioning. First, we design a coordinated MADRL method for inter-cell resource partitioning in network slicing, where DRL agents share local information to mitigate inter-cell interference. The objective is to maximize the satisfaction of per-slice service requirements, focusing on average user throughput and delay in each cell. Additionally, we integrate a TL method to accelerate policy deployment among different agents. This new method consists of two parts: a feature-based inter-agent similarity analysis approach, which measures domain and task differences by extracting representative feature distributions in latent space, and a knowledge transfer approach combining model (policy) and instance transfer.

This chapter incorporates research that has been detailed in the following published papers:

> Network Slicing via Transfer Learning Aided Distributed Deep Reinforcement Learning *Tianlun Hu, Qi Liao, Qiang Liu, Georg Carle* GLOBECOM 2022 - 2022 IEEE Global Communications Conference, 2022, pp. 1-6, doi: 10.1109/GLOBECOM48099.2022.10000763.

My contributions to this paper involved the design, implementation, and evaluation of TL methods specifically tailored for the Specialist-to-Specialist use case, as well as the drafting of the paper. This dissertation expands upon this work in conjunction with the discussions presented in Chapter 6, and it completes the exploration of two typical TL use cases in RL-based slice resource allocation strategies.

> A Joint Industrial-Network Simulator for Leveraging Automation in 5G Private Networks *Alessandro Lieto, Stanislaw Strzyz, Patrick Agostini, Tianlun Hu* European Wireless 2022; 27th European Wireless Conference, 2022, pp. 1-5.

In the above paper, we introduced a sophisticated industrial network simulator. My role specifically involved implementing TL methods to assess an RL-based slice resource allocation solution within simulated environments. This contribution highlights the practical application of advanced ML techniques to optimize network operations in virtual settings, bridging theoretical concepts with tangible, industry-relevant applications.

This chapter is organized as follows. In Section 7.1, we illustrate the motivation of studying TL-aided DRL approaches by supplementing the literature research of TL application in networking. The system formulation and fundamental problem formulation are provided in Section 7.2 and Section 7.3, respectively. In Section 7.4, we introduce a domain similarity measurement method for addressing the question of "From whom to transfer" for Specialist-to-Specialist TL approaches. The numerical results and discussions are embedded in Section 7.5.

7.1 Motivation

As the network scale grows, the action and state space of the centralized problem increases exponentially, which challenges the convergence and sample efficiency of DRL. MADRL [58, 68] has been explored to address this issue by creating and training multiple cooperative DRL agents, where each agent focuses on an individual site or cell. However, training all individual DRL agents from scratch can still be costly and time-consuming due to expensive queries with real networks and unstable environments from the perspective of individual agents.

Recently, TL [70] methods have been increasingly studied to improve sample efficiency and model reproducibility in various machine learning fields [71, 72, 73]. The basic idea of TL is to utilize prior knowledge from pre-learned tasks to benefit the training process in new tasks. For example, the resource partitioning policy of one cell can be transferred to another cell with similar network settings, such as bandwidth, transmit power, and traffic patterns.

To effectively use TL methods, several key questions need to be addressed: *what to transfer*, *from whom to transfer*, and *how to transfer*. Most existing TL methods focus on supervised machine learning, such as computer vision and natural language processing [75], providing limited insights for DRL tasks [76, 92, 93, 79]. Therefore, it is imperative to study how TL can improve the performance of MADRL in terms of sample efficiency and fine-tuning costs for inter-cell resource partitioning problems.

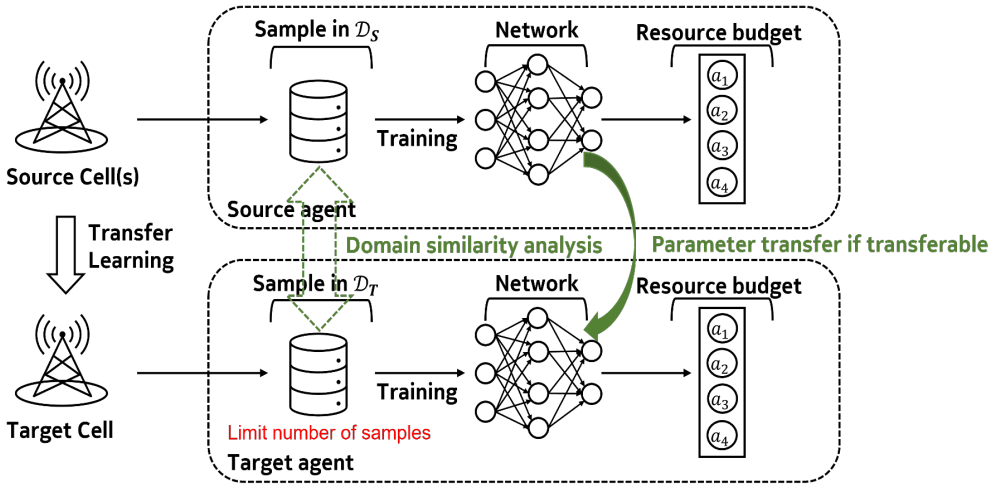

Figure 7.1: Specialist-to-Specialist TL scheme

In this chapter, we aim to address the challenge of transferring specific knowledge using domain similarity analysis, as illustrated in Fig. 7.1. This approach ensures that the TL

process is effective in the target domain, enhancing the overall efficiency and performance of MADRL-based slice resource allocation methods.

7.2 System Model

Aligning with the setup of Chapter 6, here we also consider a network consisting of a set of cells $\mathcal{K} := \{1, 2, \ldots, K\}$ and a set of slices $\mathcal{N} := \{1, 2, \ldots, N\}$. Each slice $n \in \mathcal{N}$ has pre-defined average user throughput and delay requirements, denoted as ϕ_n^* and d_n^*, respectively. The network system runs on discrete time slots $t \in \mathbb{N}_0$. The network operation and maintenance OAM adapts the inter-slice resource partitioning for all cells to periodically provide per-slice resource budgets to each cell. Then, within each cell, the RAN scheduler uses the provided resource budgets as constraints and performs resource scheduling and PRB allocation. In this chapter, we focus on the inter-cell inter-slice resource partitioning problem in network OAM.

Same as the formal RL formulation, we model the multi-cell resource partitioning system as a set of K distributed MDPs $\mathcal{M}_k := \{\mathcal{S}_k, \mathcal{A}_k, P_k(\cdot), r_k(\cdot), \gamma_k\}$ defined for each agent $k \in \mathcal{K}$ (with a slight abuse of notation, hereafter we use k for cell and agent interchangeably). $\mathcal{S}_k$ and $\mathcal{A}_k$ denote the state space and action space, respectively. $P_k(\cdot) : \mathcal{S}_k \times \mathcal{A}_k \times \mathcal{S}_k \to [0, 1]$ is the transition probability over $\mathcal{S}_k$ and $\mathcal{A}_k$ for cell k. $r_k : \mathcal{S}_k \times \mathcal{A}_k \to \mathbb{R}$ is defined as the reward function which evaluates the network service of all slices in cell k, and γ_k denotes the discount factor for cumulative reward calculation.

At each time step t, agent k collects state $\mathbf{s}_k(t) \in \mathcal{S}_k$ and decides an action $\mathbf{a}_k(t) \in \mathcal{A}_k$ according to policy $\pi_k : \mathcal{S}_k \to \mathcal{A}_k$, which indicates the per-slice resource partitioning ratio $a_{k,n} \in [0, 1]$ for $n \in \mathcal{N}$ while aligning with inter-slice resource constraints. Thus, the local action space $\mathcal{A}_k$ yields

$$\mathcal{A}_k := \left\{ \mathbf{a}_k \middle| a_{k,n} \in [0, 1], \forall n \in \mathcal{N}; \sum_{n=1}^{N} a_{k,n} = 1 \right\}. \tag{7.1}$$

For each cell $k \in \mathcal{K}$, our objective is to maximize the minimum service satisfaction level in terms of average user throughput and delay (ϕ_n^*, d_n^*) over all slices. Thus, for each agent k, we define the local reward function based on the observed per-slice average user throughput $\phi_{k,n}(t)$ and delay $d_{k,n}(t)$ at time t as

$$r_k(t) := \min_{n \in \mathcal{N}} \min \left\{ \frac{\phi_{k,n}(t)}{\phi_{k,n}^*}, \frac{d_{k,n}^*}{d_{k,n}(t)}, 1 \right\}. \tag{7.2}$$

The reward formulation drops below 1 when the actual average throughput or delay of any slices fails to fulfill the requirements. Note that the reward is upper-bounded by 1 even if all slices achieve better performance than the requirements to achieve more efficient resource utilization. The second item in Eq. 7.2 is **inversely proportional** to the actual delay, namely, if the delay is longer than required, this term is lower than 1.

7.3 Problem Formulation

RL Problem: The problem is to find a policy $\pi_k : \mathcal{S}_k \to \mathcal{A}_k$ for each $k \in \mathcal{K}$ that predicts optimal inter-slice resource partitioning $\mathbf{a}_k(t) \in \mathcal{A}_k$ base on the local state $\mathbf{s}_k(t) \in \mathcal{S}_k$ dynamically, to maximize the expectation of the cumulative discounted reward $r_k(t)$ defined in Eq. 7.2, in a finite time horizon T. The problem is given by:

$$\max_{\pi_k; \mathbf{a}_k(t) \in \mathcal{A}_k} \mathbb{E}_{\pi_k} \left[\sum_{t=0}^{T} \gamma_k^t r_k\big(\mathbf{s}_k(t), \mathbf{a}_k(t)\big) \right], \ \forall k \in \mathcal{K}, \tag{7.3}$$

where $\mathcal{A}_k$ is defined in Eq. 7.3.

In Chapter 5, we proposed a coordinated multi-agent DRL approach to transform an MADRL problem to the distributed DRL problem similar to Eq. 7.3, where the extracted information from neighboring cells is included into the state observation to better capture the inter-agent dependency. However, training all local agents in parallel from scratch can be costly and time-consuming. Moreover, the trained models are sensitive to environmental changes and the retraining cost can be high.

Thus, in this chapter, we raise the following new questions:
Can we reuse the knowledge in a pre-trained model? When is the knowledge transferable? And, most importantly, how to transfer the gained knowledge from one agent to another?

TL Problem: To tackle the TL problem, let us first introduce two definitions, *domain* and *task*, in the context of RL.

A *domain* $\mathcal{D} := \{\mathcal{S}, P(\mathbf{s})\}$ consists of a state feature space $\mathcal{S}$ and its probability distribution $P(\mathbf{s})$, for $\mathbf{s} \in \mathcal{S}$. A *task* $\mathcal{T} := \{\mathcal{A}, \pi(\cdot)\}$ consists of the action space $\mathcal{A}$ and a policy function $\pi : \mathcal{S} \rightarrow \mathcal{A}$.

Thus, our inter-agent TL problem is to find the optimal source agent among a set of pre-trained agents and transfer its knowledge (pre-trained model and collected instances) to the target agent, such that problem Eq. 7.3 can be solved in the target agent with fast convergence and a limited amount of samples. In particular, the problem is defined in Problem 4.

Problem 4. *Given a set of pre-trained source agents $\overline{\mathcal{K}} \subset \mathcal{K}$ with source domains $\boldsymbol{\mathcal{D}}^{(S)} := \left\{\mathcal{D}_i^{(S)} : i \in \overline{\mathcal{K}}\right\}$ and pre-trained tasks $\boldsymbol{\mathcal{T}}^{(S)} := \left\{\mathcal{T}_i^{(S)} : i \in \overline{\mathcal{K}}\right\}$, also given any target agent $k \notin \overline{\mathcal{K}}$ with target domain $\mathcal{D}_k^{(T)}$ and untrained task $\mathcal{T}_k^{(T)}$, find the optimal source agent $i_k^* \in \overline{\mathcal{K}}$ for target agent k to transfer knowledge such that*

$$
\begin{aligned}
i_k^* := \underset{\substack{\pi_k | \pi_k^{(0)} = \Lambda\left(\pi_i^{(S)}\right) \\ i \in \overline{\mathcal{K}}}}{\arg\max} \quad & \mathbb{E}_{\pi_k}\left[\sum_{t=0}^{T} \gamma_k^t r_k\big(\mathbf{s}_k(t), \mathbf{a}_k(t)\big)\right] \\
s.t. \ & (\mathbf{s}_k, \mathbf{a}_k) \in \Gamma\left(\mathcal{D}_i^{(S)}, \mathcal{D}_k^{(T)}, \mathcal{A}_i^{(S)}, \mathcal{A}_k^{(T)}\right),
\end{aligned}
\tag{7.4}
$$

where $\Lambda\left(\pi_i^{(S)}\right)$ is the policy transfer *strategy which maps a pre-trained source policy $\pi_i^{(S)}$ to the initial target policy $\pi_k^{(0)}$, while $\Gamma\left(\mathcal{D}_i^{(S)}, \mathcal{D}_k^{(T)}, \mathcal{A}_i^{(S)}, \mathcal{A}_k^{(T)}\right)$ is the* instance transfer *strategy which selects the instances from the source agent, combines them with the experienced instances from the target agent, and saves them in the replay buffer for model training or fine-tuning in the target agent. More details about the TL strategies will be given in Section 7.4.3.*

7.4 TL with Domain Similarity Analysis

In this section, we first recap the distributed MADRL approach to solve the slicing resource partitioning problem in Eq. 7.3. Then, to solve problem Eq. 7.4 to find the optimal source agent, we propose a novel approach to inter-agent similarity analysis based on the extracted features using VAE. Finally, for inter-agent TL, we introduce a TL strategy that combines the model (policy) transfer and instance transfer.

7.4.1 Distributed MADRL with Coordination

As stated in Eq. 7.3 , the distributed DRL approach allows each agent to learn a local policy and make its own decision on inter-slice resource partitioning based on local observation. Using the distributed DRL approach with inter-agent coordination proposed in Section 5.4, we briefly summarize the coordinated distributed DRL approach below because we would like to focus on the main contribution, namely, the inter-agent TL, in this chapter.

Each local agent k observes a local state $\mathbf{s}'_k$, which contains the following network measurements:

- Per-slice average user throughput $\{\phi_{k,n} : n \in \mathcal{N}\}$;
- Per-slice network load $\{l_{k,n} : n \in \mathcal{N}\}$;
- Per-slice number of users $\{u_{k,n} : n \in \mathcal{N}\}$.

Thus, with the above-defined three slice-specific features, the local state $\mathbf{s}'_k$ has the dimension of $3N$. Additionally, we introduce an inter-agent coordination mechanism through network information sharing among agents to better capture the inter-cell dependencies and estimate the global network performance. Let each agent k broadcast a message $\mathbf{m}_k$ to its neighboring group of agents, denoted by $\mathcal{K}_k$, which means, each agent k receives a collection of messages $\overline{\mathbf{m}}_k := [\mathbf{m}_i : i \in \mathcal{K}_k] \in \mathbb{R}^{Z^{(m)}}$. Instead of using all received messages in $\overline{\mathbf{m}}_k$, we propose to extract useful information $\mathbf{c}_k \in \mathbb{R}^{Z^{(c)}}$ to maintain the low model complexity. We aim to find an feature extractor $g : \mathbb{R}^{Z^{(m)}} \to \mathbb{R}^{Z^{(c)}} : \overline{\mathbf{m}}_k \to \mathbf{c}_k$, such that $Z^{(c)} \ll Z^{(m)}$. Then, we include the extracted features from the shared messages into the local state: $\mathbf{s}_k := [\mathbf{s}'_k, \mathbf{c}_k]$.

Knowing that the inter-agent dependencies are mainly caused by inter-cell interference based on cell load coupling [81], we propose to let each cell k share its per-slice load $l_{k,n}, \forall n \in \mathcal{N}$ to its neighboring cell. Then, we compute the extracted information $\mathbf{c}_k$ as the average per-slice neighboring load. Namely, we define a deterministic feature extractor, given by:

$$\begin{aligned} g_k :&\mathbb{R}^{N|\mathcal{K}_k|} \to \mathbb{R}^N : [l_{i,n} : n \in \mathcal{N}, i \in \mathcal{K}_k] \mapsto \mathbf{c}_k(t) \\ \text{with } \mathbf{c}_k(t) :=& \left[\frac{1}{|\mathcal{K}_k|} \sum_{i \in \mathcal{K}_k} l_{i,n}(t) : n \in \mathcal{N} \right]. \end{aligned} \tag{7.5}$$

With the extended local state including the inter-agent shared information, we can use classical DRL approaches, e.g., the actor-critic algorithms such as TD3 [65] to solve Eq. 7.3.

7.4.2 Domain Distance Measurement for Similarity Analysis

The distributed DRL approach introduced in Section 7.4.1 allows us to derive a set of pre-trained local agents. Still, given a target cell k, e.g., a newly deployed cell, or an existing cell but with a changed environment, more questions need to be answered: Can we transfer the pre-learned knowledge from at least one of the pre-trained agents? Which source cell provides the most transferable information? How to transfer the knowledge?

To solve the TL problem in Eq. 7.4, we develop a distance measure $\mathfrak{D}_{i,k}$ to quantify the inter-agent similarity between a source agent i and a target agent k. As Fig. 7.2 indicates, we aim to transfer the knowledge from the source agent with the highest similarity (reflected by the lowest distance measure).

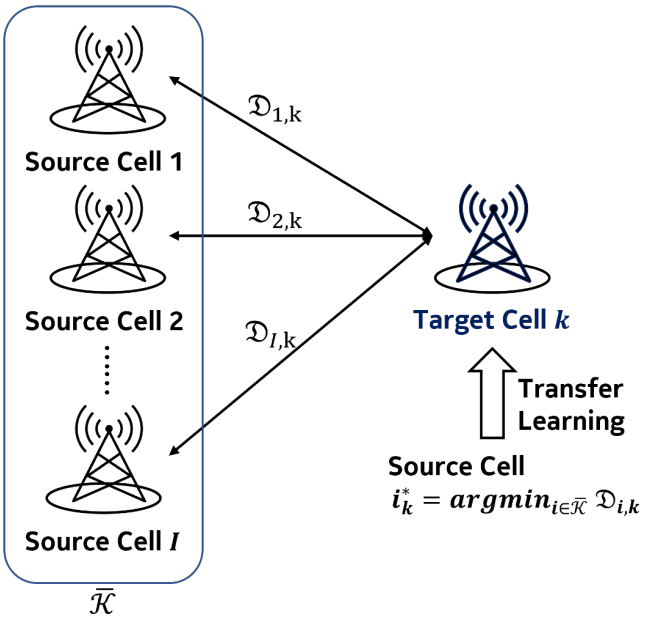

Figure 7.2: Domain similarity analysis for source domain selection

The ideal approach to analyze the domain and task similarity between two agents is to obtain their probability distributions of the state $P(\mathbf{s})$ and derive the conditional probability distribution $P(\mathbf{a}|\mathbf{s})$. However, the major challenge here lies in the limited samples in the target agent. Considering that the target agent is a newly deployed agent, there is no information available about its policy $P(\mathbf{a}|\mathbf{s})$, and $P(\mathbf{s})$ is very biased because all samples are collected under the default configurations (i.e., constant actions).

Thus, we need to design a distance measure constrained by very limited and biased samples in the target agent, without any information about its policy $P(\mathbf{a}|\mathbf{s})$. Our idea is to derive and compare the **joint state and reward distribution** under the same default action $\mathbf{a}'$, $P(\mathbf{s}, r|\mathbf{a} = \mathbf{a}')$, in both source and target agent. The rationale behind this is that, when applying the actor-critic-based DRL architecture, the critic function estimates the Q value $Q_\pi(\mathbf{a}, \mathbf{s})$ based on action and state. Hence, the conditional probability $P(r|\mathbf{s}, \mathbf{a})$ should provide useful information on the policy. With $\mathbf{a} = \mathbf{a}'$, we can consider to estimate $P(r|\mathbf{s}, \mathbf{a} = \mathbf{a}')$. To efficiently capture the information for both domain similarity (based on $P(\mathbf{s}|\mathbf{a} = \mathbf{a}')$) and task/policy similarity (based on $P(r|\mathbf{s}, \mathbf{a} = \mathbf{a}')$), we propose to estimate the joint probability $P(\mathbf{s}, r|\mathbf{a} = \mathbf{a}') = P(r|\mathbf{s}, \mathbf{a} = \mathbf{a}')P(\mathbf{s}|\mathbf{a} = \mathbf{a}')$.

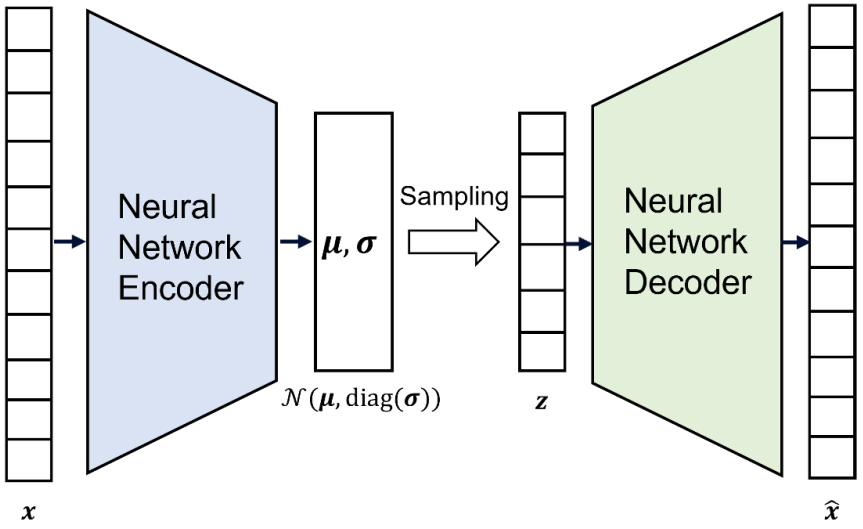

Figure 7.3: Variational autoencoder

Sample collection: To estimate the distance between $P(\mathbf{s}, r|\mathbf{a} = \mathbf{a}')$ of both the source and target agents, we use all available samples from the target agent k under the default action $\mathbf{a}'$, $\mathcal{X}_k = \{(\mathbf{s}_k(n), r_k(n))_{\mathbf{a}_k(n)=\mathbf{a}'} : n = 1, \ldots, N_k\}$, and select a subset of the samples from the source agent i with the same default action $\mathcal{X}_i = \{(\mathbf{s}_i(n), r_i(n))_{\mathbf{a}_i(n)=\mathbf{a}'} : n = 1, \ldots, N_i\}$. Note that in this section, we slightly abuse the notation by using n as the index of sample, and N_k as the number of samples with default action collected from agent k.

Feature extraction with VAE: To extract the representative features from the high-dimension vector $[\mathbf{s}, r]$, we propose to apply VAE [94] to map the samples into a low dimensional latent space. As Fig. 7.3 illustrates, for each sample $\mathbf{x} := [\mathbf{s}, r] \in \mathcal{X}$, the encoder of VAE estimates an approximated distribution $P(\mathbf{z})$ in latent space $\mathcal{Z}$ as a multi-variate Gaussian distribution with $\mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}))$, where diag denotes the diagonal matrix. The decoder samples a latent variable $\mathbf{z} \in \mathcal{Z}$ from the approximated distribution $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}))$ and outputs a reconstructed sample $\hat{\mathbf{x}}$ by training on the following loss function:

$$L := \|\mathbf{x} - \hat{\mathbf{x}}\|^2 + \alpha \cdot D_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma})) \,\|\, \mathcal{N}(\mathbf{0}, \mathrm{diag}(\mathbf{1}))\right), \tag{7.6}$$

where α is the weight factor and D_{KL} denotes the KL-divergence.

Inter-agent similarity analysis: Since VAE does not directly provide the probability distribution function $P(\mathbf{x})$, we propose to utilize the extracted features in the latent space to evaluate the inter-agent similarity. Considering the limited amount of samples (only those under default action), we propose to train a general VAE model based on the samples from all candidate source agents and the target agent, e.g., $\mathcal{X} = \bigcup_{j \in \overline{\mathcal{K}} \cup \{k\}} \mathcal{X}_j$. The idea is to extract the latent features from samples from all relevant agents with a general encoder and to distinguish the agents within a common latent space.

Thus, for each sample $\mathbf{x}_n \in \mathcal{X}$, we can derive its extracted features, i.e., the posterior distribution $P(\mathbf{z}_n|\mathbf{x}_n) = \mathcal{N}(\boldsymbol{\mu}_n, \mathrm{diag}(\boldsymbol{\sigma}_n))$. We denote the extracted latent space for agent k by $\mathcal{Z}_k$. Next, we can measure the inter-agent distance between an arbitrary source agent i and target agent k by calculating the KL-divergence based on the extracted latent variables from their collected samples:

$$\mathfrak{D}_{i,k} := \frac{1}{N_i N_k} \sum_{\substack{(\boldsymbol{\mu}_n, \boldsymbol{\sigma}_n) \in \mathcal{Z}_i \\ (\boldsymbol{\mu}_m, \boldsymbol{\sigma}_m) \in \mathcal{Z}_k}} D_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}_n, \mathrm{diag}(\boldsymbol{\sigma}_n)) \| \mathcal{N}(\boldsymbol{\mu}_m, \mathrm{diag}(\boldsymbol{\sigma}_m))\right). \tag{7.7}$$

This requires computing the KL-divergence of every pair of samples (n, m) for $n \in \mathcal{X}_i$ and $m \in \mathcal{X}_k$, which could be computing intensive.

Note that they are both Gaussian distributions. We can efficiently compute them with closed-form expression (as will be shown later in Eq. 7.8). Besides, from our experiment, we observed that $\boldsymbol{\sigma}_n \to \mathbf{0}$ for nearly all the collected samples $\mathbf{x}_n \in \mathcal{X}$, i.e., their variances are extremely small (to the level below $10e - 5$ from our observation). Thus, for our problem, we can use a trick to evaluate the distance measure more efficiently based on the following lemma.

Lemma 1. *Given two multi-variate Gaussian distributions $p = \mathcal{N}(\boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n)$, and $q = \mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$, where $\boldsymbol{\mu}_n, \boldsymbol{\mu}_m \in \mathbb{R}^L$, $\boldsymbol{\Sigma}_n = \boldsymbol{\Sigma}_m = \mathrm{diag}(\boldsymbol{\sigma}) \in \mathbb{R}^{L \times L}$ and every entry of $\boldsymbol{\sigma}$ is equal to a small positive constant $\sigma \ll 1$, the KL-divergence $D_{KL}(p||q)$ is proportional to $\sum_{l=1}^{L} (\mu_{n,l} - \mu_{m,l})^2$.*

Proof. It is easy to derive that

$$D_{KL}(p\|q) = \frac{1}{2}\Big[\log \frac{|\boldsymbol{\Sigma}_\mathbf{n}|}{|\boldsymbol{\Sigma}_\mathbf{m}|} - L + (\boldsymbol{\mu}_n - \boldsymbol{\mu}_m)^T \boldsymbol{\Sigma}_\mathbf{m}^{-1} (\boldsymbol{\mu}_n - \boldsymbol{\mu}_m) + \mathrm{Tr}\left\{\boldsymbol{\Sigma}_\mathbf{m}^{-1} \boldsymbol{\Sigma}_\mathbf{n}\right\}\Big]. \tag{7.8}$$

Because $\mathbf{\Sigma}_n = \mathbf{\Sigma}_m = \text{diag}([\sigma^2, ..., \sigma^2])$, we have the first term in Eq. 7.8 equals to 0, and the last term equals to L. Thus, we obtain

$$D_{KL}(p\|q) = \frac{1}{2\sigma^2}\sum_{l=1}^{L}(\mu_{n,l} - \mu_{m,l})^2. \tag{7.9}$$

□

With Lemma 1, we can measure the distance between two agents more efficiently, based on the extracted $\boldsymbol{\mu}_n$ and $\boldsymbol{\mu}_m$ in the source and target latent spaces. Thus, to solve Problem (4), we propose to choose the source agent:

$$i_k^* := \arg\min_{i \in \overline{K}} \mathfrak{D}_{i,k}, \tag{7.10}$$

where $\mathfrak{D}_{i,k}$ is computed based on Eq. 7.7 and Eq. 7.9.

7.4.3 Specialist-to-Specialist TL

In general, the pre-learned knowledge can be transferred from a source agent i to the target agent k with various policy transfer strategies $\Lambda(\cdot)$ and instance transfer strategy $\Gamma(\cdot)$:

- **Model transfer:** The policy transfer strategy $\Lambda(\cdot)$ simply initializes the target agent policy $\pi_k^{(0)}$ by loading the parameters (e.g., weights of the pre-trained neural networks) of the pre-trained policy $\pi_i^{(S)}$ from the source agent i.
- **Feature transfer:** The policy transfer strategy $\Lambda(\cdot)$ keeps partial information extracted from the source agent pre-trained policy $\pi_i^{(S)}$. In particular, the target agent loads partial of the layers (usually the lower layers) of the pre-trained neural networks of $\pi_i^{(S)}$, while leaving the rest of them to be randomly initialized. Then, during training, the loaded layers are frozen and only the randomly initialized layers are fine-tuned with the instances newly collected by the target agent.
- **Instance transfer:** The instance transfer strategy $\Gamma(\cdot)$ transfers the collected instances from the source agent i to the target agent k and saves them in the replay buffer of target agent. Then, the target agent trains a policy from scratch with randomly initialized parameters and mixed instances collected from both source and target agents.

The above-mentioned knowledge from the source domain and source task can be transferred separately or in a combined manner. In this chapter, we propose the **integrated transfer method** with both model and instance transfer. Specifically, the target agent k initializes its local policy $\pi_k^{(0)}$ by loading the pre-trained policy of the source agent $\pi_i^{(S)}$ and fine-tunes the policy by sampling from the replay buffer containing both types of instances: the instances transferred from the source agent and those locally experienced. Here, we skip the feature transfer because it practically performs well only when the similarity between the source domain/task and target domain/task is very high. Although this assumption may hold for some regression and classification tasks, we empirically find that it fails in this context of MADRL.

7.5 Experiments

In this section, we evaluate the performance of the proposed solution within Season II. In addition, we introduce a traffic-aware baseline that allocates resources proportionally to the data traffic demand per slice. Note that the baseline assumes perfect information about per-cell per-slice traffic demands, which has already provided very good results.

7.5.1 Network Environment Setup

We build a radio access network with 4 three-sector sites (i.e., $K = 12$ cells). All cells are deployed using LTE radio technology with 2.6 GHz under a realistic radio propagation model Winner+ [67]. Each cell has $N = 4$ slices with diverse per-slice requirements in terms of average user throughput and delay. In the cells with label $1, 2, 3, 7, 8, 9$, we define per-slice average throughput requirements of $\phi_1^* = 4$ Mbit/s, $\phi_2^* = 3$ Mbit/s, $\phi_3^* = 2$ Mbit/s, and $\phi_4^* = 1$ Mbit/s respectively, and per-slice delay requirements of $d_1^* = 3$ ms, $d_2^* = 2$ ms, $d_3^* = d_4^* = 1$ ms. In the cells with label $4, 5, 6, 10, 11, 12$, we define per-slice throughput requirements as $\phi_1^* = 2.5$ Mbit/s, $\phi_2^* = 2$ Mbit/s, $\phi_3^* = 1.5$ Mbit/s, and $\phi_4^* = 1$ Mbit/s, and delay requirements of $d_n^* = 1$ ms, $\forall n \in \mathcal{N}$. All cells have the same radio bandwidth of 20 MHz.

We define four groups of UE associated with four slices in each cell, respectively. Each UE group has the maximum size of 32 and moves randomly among the defined network scenario. To mimic the dynamic behavior of real user traffic, same to the experiment in 6, we apply a varying traffic mask $\tau_n(t) \in [0, 1]$ to each slice to scale the total number of UEs in each cell as Fig. 6.3.

7.5.2 DRL Training Configuration

For MADRL training, we implemented TD3 algorithm at each local agent using MLP architecture for actor-critic networks. In each TD3 model, both actor and critic neural works consist of two layers with the number of neurons as $(48, 24)$ and $(64, 24)$, respectively. The learning rates of actor and critic are 0.0005 and 0.001 accordingly with Adam optimizer and training batch size of 32. We set the discount factor as $\gamma = 0.1$ since the current action has a stronger impact on instant network performance than future observation. As for the training, for distributed DRL agents we applied 3000 steps for exploration, 5500 steps for training, and final 250 steps for evaluation. For the TL training process, we apply the same model setups as DRL approaches, while only setting 4000 steps for training and 250 for evaluation since knowledge transfer saves time for exploration.

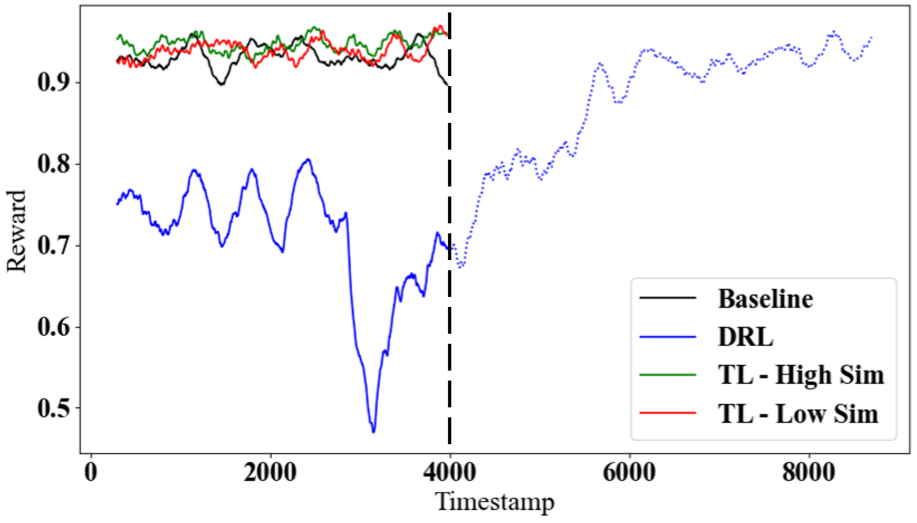

Figure 7.4: Comparing reward during the training process

7.5.3 Evalutation of TL-aided DRL

In Fig. 7.4 we compare the evolution of reward during the training processes among the baseline, DRL approach (proposed in Section 7.4.1), and TL approaches when transferred from source agent with low and high similarity (proposed in Section 7.4.2 and 7.4.3), respectively. For DRL, we present the first 4000 step, i.e., the same training time as TL approaches with a solid line and the rest training curve with a dashed line.

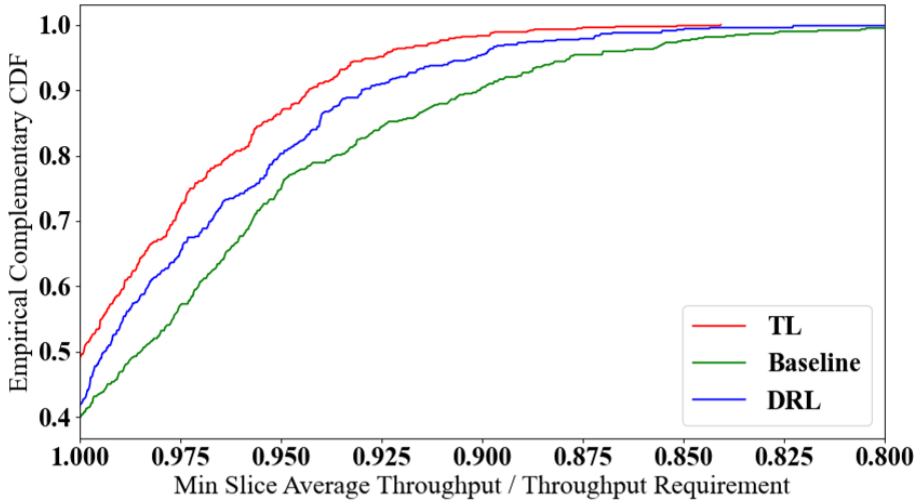

Figure 7.5: Comparing CDF of minimum slice throughput satisfaction

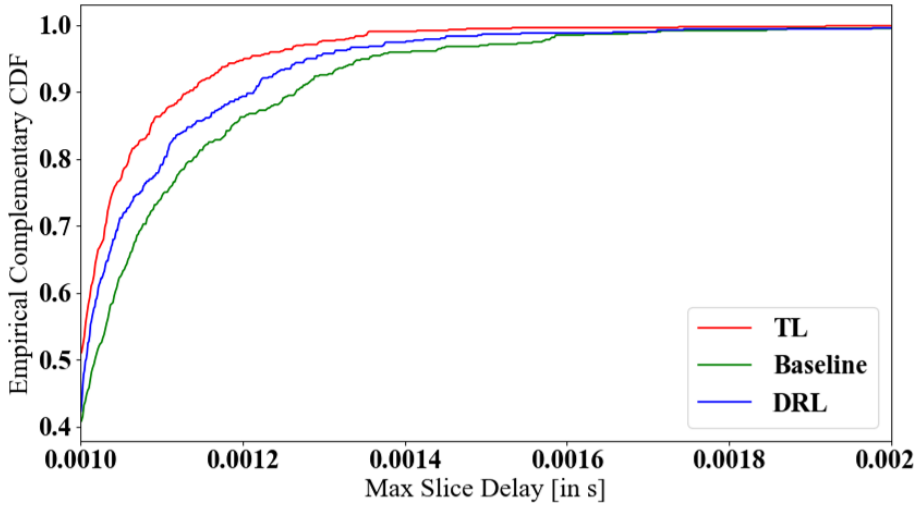

Figure 7.6: Comparing CDF of maximum slice delay

As shown in Fig. 7.4, the distributed DRL approach learns to achieve a similar reward as the baseline after a lengthy exploration phase, while both TL approaches start with much higher start compared to DRL. After a short fine-tuning period, the TL approaches outperform the baseline with higher robustness, especially during the period with higher traffic demands and strong inter-cell interference where the baseline has sharp performance degradation. Besides, in comparison between the TL from agents with different similarity measures, we observe that with higher similarity, TL provides higher starts at the early stage of training, while both of them converge to similar performance after the training converges.

For performance evaluation, we compare the statistical results on minimum per slice throughput satisfaction level and maximum per slice delay, respectively, among all cells among the methods baseline, distributed DRL and the proposed TL approach after convergence. Fig. 7.5 illustrated the empirical complementary CDF which equals $1 - F_X(x)$ where $F_X(x)$ is the CDF of minimum per slice throughput satisfaction level. We observe that the TL approach provides the best performance compared to others by achieving only about 12% failure to satisfy 0.95 of the requirement, while converged DRL and baseline conclude 19% and 25% failure rate respectively. By average satisfaction level, the TL approach concludes 0.92 while DRL and baseline only provide 0.90 and 0.87. A similar observation can be made from Fig. 7.6, which illustrates the CDF of maximum slice delay

in ms. The TL approach provides 1.5 ms maximum average per-slice delay, while DRL achieves 1.7 ms and baseline achieves 1.8 ms.

7.5.4 Domain Similarity Analysis

We implemented the similarity analysis method introduced in Section 7.4.2 with a VAE model in MLP architecture, both networks of encoder and decoder consist of 3 layers with the number of neurons as $(64, 24, 4)$ and $(4, 24, 64)$ respectively. To achieve a good trade-off between low dimensional latency space and accurate reconstruction with VAE, we map the original sample $x \in \mathbb{R}^{17}$ to the latent variable $\mathbf{z} \in \mathbb{R}^4$.

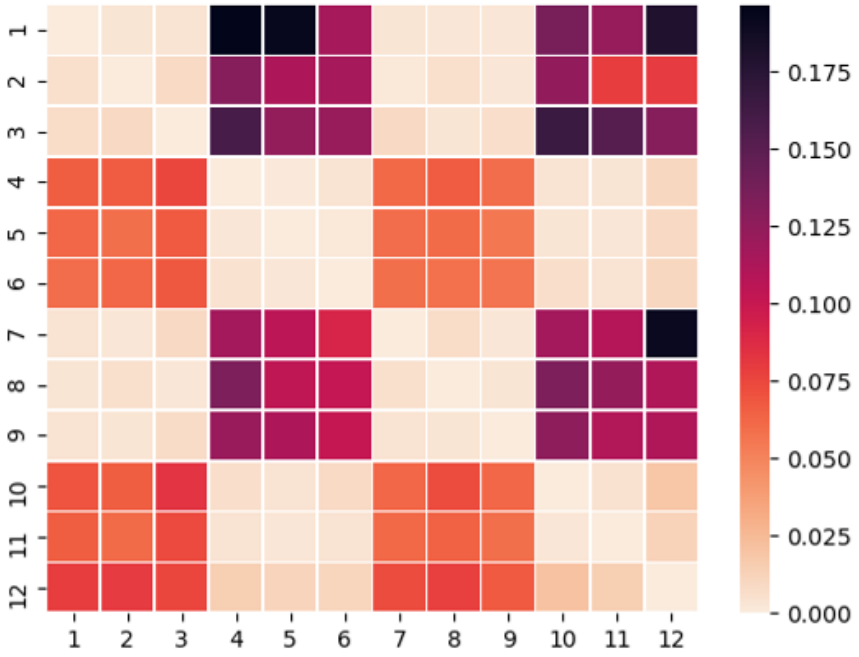

Figure 7.7: Inter-agent distance measure

Fig. 7.7 illustrates the results of inter-agent similarity analysis as a metric of distance measure proposed in Eq. 7.7. It shows that our proposed method can distinguish cells with different per-slice service quality requirements and gather the cells with similar joint state-reward distribution.

7.5.5 TL Evaluation based on Domain Distance Measurement

In Fig. 7.8 we compare the benefits of TL in the training process by transferring knowledge from source agents with different average inter-agent distance measures. The TL gains are derived by comparing the reward to the DRL approach at the same training steps. The results show that before 200 steps of TL training, the TL approaches with the lowest distance measure provide about 3% higher gain than the one with the largest distance. As the training process continues, the gains in all TL approaches increase with local fine-tuning and the difference between transferring from highly similar and less similar agents is getting smaller. However, TL from the most similar agent provides higher gains for all training steps.

7.6 Key Takeaways

The numerical results from the experiments and discussions above provide several significant insights:

- **Distributed DRL Performance:** All distributed DRL-based approaches consistently achieve superior per-slice network service compared to the traffic-aware baseline

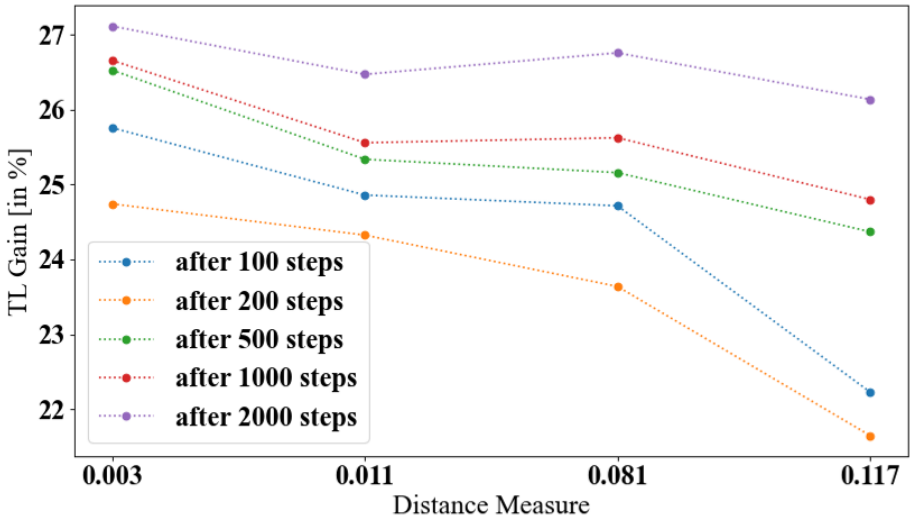

Figure 7.8: TL performance gain depending on distance measure

post-convergence. However, TL schemes notably outperform conventional DRL approaches in terms of convergence rate, as well as initial and converged performance.

- **VAE-based Similarity Measure:** Our proposed VAE-based similarity measure effectively quantifies the distance between agents. This measure can be utilized to map the defined distance to the TL performance gain, facilitating more effective knowledge transfer.

- **Transfer from Similar Agents:** The performance gains achieved by transferring knowledge from highly similar agents are significantly greater compared to less similar agents, particularly when the number of training steps is limited (i.e., with fewer online training samples). Although the advantage diminishes as the number of online training steps increases, a slight performance gain is consistently achieved by transferring knowledge from the most similar source agent.

In this chapter, we developed a novel integrated TL method to transfer learned DRL policies among different local agents under the scenarios that differ from Chapter 6, thereby accelerating policy deployment. This method incorporates a new inter-agent similarity measurement approach and an innovative knowledge transfer approach. We evaluated our proposed solution through extensive simulations using a system-level simulator. The results demonstrate that our approach significantly outperforms conventional DRL solutions, showcasing its effectiveness in enhancing network service quality and efficiency. Building on the success of our Generalist-to-Specialist and Specialist-to-Specialist TL-aided DRL approaches, future research will focus on investigating per-slice distributed solutions. This involves further refining the granularity of resource allocation and optimizing individual slice performance within the distributed network framework. By continuing to explore these advanced methods, we aim to develop more robust and efficient solutions for dynamic network environments.

8. IDLA: Per-Slice Scheme for Resource Allocation

In this chapter, we downgrade the granularity of the slicing resource allocation solution from the cell-wise scheme to the slice-wise scheme to explore the capability of TL in RAN slicing scenarios. We introduce a novel framework that integrates deep learning models with constrained optimization methods, allowing for generalization to arbitrary slice combinations. This solutions are proposed to address Question 6 and Question 7. First, we derive a general DNN model to approximate the slice network utility, capable of handling slices with different requirements. Leveraging the efficient computation of the partial derivatives of the slice utility function approximated by the DNN model, we design a Lagrangian method for optimal per-slice resource allocation while adhering to inter-slice resource constraints. We evaluate the proposed algorithm in a system-level network simulator, demonstrating that our algorithm achieves near-optimal QoS satisfaction and promising generalization performance compared to state-of-the-art solutions, including widely used DRL approaches.

This chapter partially draws from research that has been published in the following paper:

> Fast and Scalable Network Slicing by Integrating Deep Learning with Lagrangian Methods *Tianlun Hu, Qi Liao, Qiang Liu, Antonio Massaro, Georg Carle* GLOBECOM 2023 - 2023 IEEE Global Communications Conference, 2023, pp. 1-6, doi: 10.1109/GLOBECOM54140.2023.10436849.

In contributing to this paper, my roles were multifaceted, involving conceptualizing and designing innovative slice-wise resource allocation solutions. I was also responsible for the implementation and rigorous evaluation of these solutions, as well as the task of drafting the manuscript. In this dissertation, we extend the scope of the original paper. The extension includes the development and integration of a TL -based approach tailored explicitly for optimizing slice-wise resource allocation. The subsequent chapter will present a detailed discussion of this TL-based methodology, providing a deeper insight into its advantages and implementation challenges in real-world network environments.

This chapter is organized as follows. We define the system model in Section 8.2 and formulate the slice-aware resource allocation problem in Section 8.3. In Section 8.4, we propose the solutions with DNN-based slice utility estimator and constrained nonlinear

optimization. The numerical results are shown in Section 8.5. We conclude this chapter in Section 8.6.

8.1 Motivation

As discussed in previous chapters, RL methods have been increasingly explored to tackle complex allocation problems in dynamic mobile networks. However, RL-based solutions struggle to scale to large state and action spaces. In Chapter 5, we investigated coordinated multi-agent DRL to handle the high-dimensional continuous action space and complex resource optimization in network slicing, embedding inter-slice resource constraints within the neural network architecture. Although effective, the proposed solution was explicitly trained for a fixed network scenario, making it difficult to generalize to different slice setups in terms of slice type and number. Additionally, recent works lack the discussion of generalizing the solution to different multi-cell network scenarios with flexible slice setups.

This limitation prompts us to revisit conventional approaches, where the problem of resource allocation in network slicing is often optimized under the assumption that the resource demand of slices is known in advance. Existing works derive their solutions by formulating analytical closed-form models and solving the network slicing problem using constrained nonlinear optimization methods. For instance, in [95], the authors initially formulated the slice-wise resource allocation problem and streamlined it by finding the upper and lower bounds of network utility using the Lagrangian method, subsequently obtaining a sub-optimal solution with a greedy algorithm. Although effective, this model is still tailored to specific slice configurations. In [96], a flexible slice deployment solution with dynamic slice configurations was proposed, formulating a slice model with adjustable parameters and solving resource partitioning through an optimization process. However, recent findings [97, 98] show that these approximated models cannot accurately represent the diverse demand and performance of slices.

In this chapter, we aim to address the slice resource allocation problem with a per-slice scheme by combining conventional optimization approaches and DNN models. We present a novel algorithm called IDLA, that optimizes slicing resource allocation and can be generalized to adapt to arbitrary slice combinations under time-varying dynamics. This integrated approach leverages the strengths of both conventional optimization and advanced deep learning techniques to overcome the limitations of current methods and provide a robust solution for dynamic network environments.

8.2 System Model

We consider a discrete-time network system that comprises a set of cells denoted by $\mathcal{C} := \{1, 2, ..., C\}$. The set of slices in each cell $c \in \mathcal{C}$ can be time-varying, denoted by $\mathcal{S}_c(t) := \{1, 2, ..., S_c(t)\}$, where $S_c(t)$ is the number of slices served by cell c at time slot $t \in \mathbb{N}_0$. Each slice $s \in \mathcal{S}_c(t)$ in cell c needs to meet the pre-defined QoS requirements, e.g., throughput and delay requirements denoted by ϕ_s^* and d_s^* respectively. Note that although here the slices are defined by throughput and delay requirements, the problem formulation and the proposed approach in the following sections can be generalized to a broader set of requirements.

OAM dynamically partitions the inter-slice resource to provide per-slice resource budgets to each cell periodically. Within each cell, the RAN scheduler allocates PRBs to individual services, using the provided resource budgets as upper-bound constraints. Same as the previous chapters, the focus of this chapter is to solve the inter-slice resource partitioning

problem in network OAM. At each time slot t, OAM optimizes slicing resource partitioning $\mathbf{x}_c(t)$ for each cell c, i.e., the ratio of the radio resource to allocate to each slice, given by

$$\mathbf{x}_c(t) := \left[x_{c,1}(t), \ldots, x_{c,S_c(t)}(t)\right] \in \mathcal{X}_c(t),\ \forall c \in \mathcal{C}, \tag{8.1}$$

$$\text{where } \mathcal{X}_c(t) := \left\{ [0,1]^{S_c(t)} \middle| \sum_{s \in \mathcal{S}_c(t)} x_{c,s}(t) \leq 1 \right\}. \tag{8.2}$$

Let $\mathbf{x}(t) := [\mathbf{x}_1(t), \ldots, \mathbf{x}_C(t)]$ be a collection of the per-cell slicing partitioning, the performance of each slice $s \in \mathcal{S}_c(t)$ in cell $c \in \mathcal{C}$ at time t is measured by the QoS satisfaction level $r_{c,s}(\mathbf{x}(t))$, defined as

$$r_{c,s}(\mathbf{x}(t)) := \min \left\{ \frac{\phi_{c,s}(\mathbf{x}(t))}{\phi_s^*}, \frac{d_s^*}{d_{c,s}(\mathbf{x}(t))}, 1 \right\}, \tag{8.3}$$

where $\phi_{c,s}(\mathbf{x}(t))$ and $d_{c,s}(\mathbf{x}(t))$ are the throughput and delay associated with slice s at cell c at time slot t, respectively. This performance metric takes the minimum between throughput and delay and is upper bounded by 1, such that both requirements need to be met to achieve the satisfaction level of 1.

Remark 1. *Theoretically, due to the inter-cell and possibly inter-slice interference, the achievable throughput and delay not only depends on the locally allocated resource to its own slice and cell but also the resource occupation of other slices in the neighboring cells. Thus, in Eq. 8.3, the QoS metric $r_{c,s}$ is written as a function of the global slicing partitioning $\mathbf{x}(t)$.*

8.3 Problem Formulation

Our objective is to find an efficient and scalable solution to optimize the utility of QoS satisfaction over all slices and cells by optimizing slicing resource partitioning at each time slot. The per-slot optimization problem is formulated in Problem 5.

Problem 5 (Global Problem)**.**

$$\begin{aligned} &\max_{\mathbf{x}(t)}\ U(\mathbf{r}(\mathbf{x}(t))) \\ &\text{subject to } \mathbf{r}(\mathbf{x}(t)) := \left[r_{c,s}(\mathbf{x}(t)) : c \in \mathcal{C}, s \in \mathcal{S}_c(t)\right], \\ &\qquad (8.1), (8.2), (8.3),\ \forall t. \end{aligned} \tag{8.4}$$

Note that the utility function can be defined based on various system designs. For example, a common utility function to consider fairness is the sum of the logarithmic function of the local performance metric:

$$U(\mathbf{r}(\mathbf{x}(t))) := \sum_{s \in \mathcal{S}_c(t), c \in \mathcal{C}} \log\left(r_{c,s}(\mathbf{x}(t)) + 1\right). \tag{8.5}$$

In this chapter, we use Eq. 8.5 as an example of the utility function. However, by leveraging the superior approximation capability of deep learning, our proposed approach can be applied to a wide range of utility functions. The challenge of solving Problem 5 is multifaceted. Firstly, the utility function complexity poses a challenge to function approximation, particularly due to limited measurements in OAM. In contrast to RAN, where user and channel feedback can be collected with fine time granularity (e.g., in milliseconds), OAM only collects averaged cell- and slice-level KPIs with a coarse granularity (e.g., in

minutes). Consequently, deriving closed-form expressions becomes extremely challenging. Secondly, the flexible slice configurations and inter-slice constraints further complicate the problem, resulting in slow convergence and poor adaptability of deep learning-based approaches. Finally, the high scalability of OAM demand, e.g., up to over $100k$ cells, makes it challenging to use either large global deep learning models or collaborative multi-agent local models that require extensive exploration to learn from scratch.

8.4 Per-slice Scheme: IDLA Algorithm

In this section, we introduce IDLA algorithm to address the aforementioned challenges. The structure of IDLA model is illustrated in Fig. 8.1. First, we design and train a DNN to approximate the per-slice utility function. Then, with the derived slice-based utility model, we decompose Problem 5 into distributed cell-based resource allocation problems with inter-slice resource constraints. This decomposition allows the IDLA algorithm to adapt to a flexible number of slices per cell. Next, we use the Lagrangian method to solve the constrained decomposed problem, where the partial derivatives can be efficiently computed based on the DNN-based utility model. Finally, by leveraging the automatic differentiation engine of deep learning libraries, we improve the efficiency of the Lagrangian method.

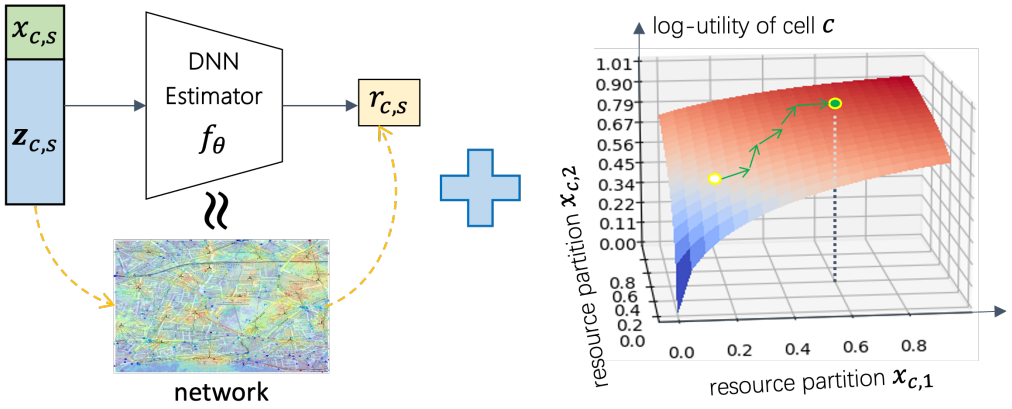

Figure 8.1: IDLA framework - DNN-based estimator + non-linear optimization

8.4.1 DNN-based Slice-wise Network QoS Estimator

The complexity of global utility in Eq. 8.5 is caused by the dependency of each local utility $U_{c,s}(r_{c,s}(\mathbf{x}(t)))$ on the global slice resource partition $\mathbf{x}(t)$. Our idea is to investigate whether each local per-slice QoS satisfaction level $r_{c,s}(t)$ can be approximated by a single general DNN $f_{\boldsymbol{\theta}}(x_{c,s}(t), \mathbf{z}_{c,s}(t)), \forall c, s$ based on the local observations only, including the allocated slice resource $x_{c,s}(t)$ and a set of slice-based KPIs $\mathbf{z}_{c,s}(t)$, i.e., to find

$$f_{\boldsymbol{\theta}}(x_{c,s}(t), \mathbf{z}_{c,s}(t)) \approx r_{c,s}(\mathbf{x}(t)), \forall s \in \mathcal{S}_c(t), c \in \mathcal{C}, \tag{8.6}$$

where the DNN is parameterized by $\boldsymbol{\theta}$.

Data Collection: The DNN model serves as a general slice-based QoS estimator, trained on slice-wise data collected from network KPIs of different cells and slice configurations. Such data are collected in network OAM periodically, e.g., every 15 minutes, as a standard practical setting. Based on the experts' prior knowledge, to predict the slice utility $r_{c,s}(t)$, computed with the achievable throughput $\phi_{c,s}(t)$ and latency $d_{c,s}(t)$, $\forall c, s$, the following network KPIs are highly correlated:

- Per-slice required throughput ϕ_s^* and required delay d_s^*;

- Per-slice PRB utilization ratio $p_{c,s}(t)$, defined as the ratio of the PRBs occupied by the slice, which can be seen as the input of the allocated resource $x_{c,s}(t)$. This is because, if resource $x_{c,s}(t) := p_{c,s}(t)$ was allocated to the slice, the corresponding achieved throughput and delay would be the same as $\phi_{c,s}(t)$ and $d_{c,s}(t)$, respectively;
- The previous H states of per-slice average number of active users $\mathbf{v}_{c,s}^{(H)}(t) := [v_{c,s}(t-H), \ldots, v_{c,s}(t-1)]$;
- The previous H states of per-slice average CQI $\mathbf{q}_{c,s}^{(H)}(t) := [q_{c,s}(t-H), \ldots, q_{c,s}(t-1)]$.

Note that we collect multiple historical states of the average number of active users and CQI, in the hope that the historical slice states not only capture temporal correlation but also reflect some hidden information extracted from the missing global states, e.g., experienced inter-cell and inter-slice interference. Also, we follow the realistic assumption that for model inference, the real-time $v_{c,s}(t)$ and $q_{c,s}(t)$ are unknown while only the previous states within $[t-H, t-1]$ are available.

Thus, a set of the local observations as the input samples during a time period $[1, T]$ is then denoted by:

$$\mathcal{X}_{t=1}^{T} := \{(x_{c,s}(t), \mathbf{z}_{c,s}(t)) : \text{ for } t = 1, \ldots, T, \forall c, s\}, \tag{8.7}$$

$$\text{where } \mathbf{z}_{c,s}(t) := [\mathbf{v}_{c,s}(t), \mathbf{q}_{c,s}(t), \phi_s^*, d_s^*] \in \mathbb{R}^{2H+2}, \tag{8.8}$$

while the set of output samples is denoted by:

$$\mathcal{Y}_{t=1}^{T} := \{r_{c,s}(t) : \text{ for } t = 1, \ldots, T, \forall c, s\}, \tag{8.9}$$

where the QoS satisfaction level $r_{c,s}(t)$ is computed by Eq. 8.3 based on the observed throughput $\phi_{c,s}(t)$ and delay $d_{c,s}(t)$.

Local Utility Approximation: We learn a general slice QoS estimator $f_{\boldsymbol{\theta}} : \mathbb{R}^{2H+3} \to \mathbb{R} : (x_{c,s}, \mathbf{z}_{c,s}) \mapsto r_{c,s}$, as defined in Eq. 8.6, such that the local utility in Eq. 8.5 can be approximated by:

$$U_{c,s}(r_{c,s}(\mathbf{x}(t))) \approx \log(f_{\boldsymbol{\theta}}(x_{c,s}(t), \mathbf{z}_{c,s}(t)) + 1). \tag{8.10}$$

With the collected data Eq. 8.7 and Eq. 8.9, we can train a MLP with $[x_{c,s}(t), \mathbf{z}_{c,s}(t)] \in \mathcal{X}_{t=1}^{T}$ as inputs and $r_{c,s}(t) \in \mathcal{Y}_{t=1}^{T}$ as output. Because $f_{\boldsymbol{\theta}}(\cdot)$ is a general distributed model that can apply to any slice, at each time t the samples from any cell and slice can contribute to model training, leading to a much higher sample efficiency and faster learning speed than training a large number of local models for distinct cells and slices. Moreover, a general model, that includes the throughput and delay requirements into the input features, can handle flexible slice configurations, even with unseen requirements.

Remark 2. *More details of data augmentation for unseen samples of different slice configurations will be given in Section 8.5. Moreover, in Section 8.5 we validate the viability of learning Eq. 8.6 not only on the simulated data but also on a collected real dataset from a commercial LTE network.*

8.4.2 Lagrangian Method for Slicing Resource Partitioning

With DNN-based utility approximation Eq. 8.10 at hand, we can decompose Problem 5 into independent per-cell optimization problems with intra-cell and inter-slice resource constraints. For each cell $c \in \mathcal{C}$ at time t, optimization problem is written as:

Problem 6 (Decomposed Local Problem)**.**

$$\max_{\mathbf{x}_c} \; F(\mathbf{x}_c) := \sum_{s \in \mathcal{S}_c(t)} \log\Big(f_{\boldsymbol{\theta}}\big(x_{c,s}(t), \hat{\mathbf{z}}_{c,s}(t)\big) + 1\Big) \tag{8.11}$$

$$\text{subject to } (8.1), (8.2), \; \forall t, \forall c \in \mathcal{C}, \tag{8.12}$$

where $\hat{\mathbf{z}}_{c,s}(t)$ *is the local observations defined in Eq. 8.8.*

Problem 6 is a classical constrained non-linear optimization problem. Note that the objective in Eq. 8.12 is a monotonic non-decreasing function over $\mathbf{x} \in \mathbb{R}_+$, i.e., we have $F(\mathbf{x}'_c) \geq F(\mathbf{x}_c)$ if $\mathbf{x}'_c \geq \mathbf{x}_c$ (entrywise greater). Therefore, the optimal solution to the problem with the equality constraint is also an optimal solution to the original problem, and we can solve it by using the Lagrange multiplier method. Since the problem is independently formulated for each time slot t and cell $c \in \mathcal{C}$, hereafter in this subsection we omit the index of t for brevity. For each cell $c \in \mathcal{C}$, the Lagrangian is given by:

$$\mathcal{L}(\mathbf{x}_c, \lambda_c) := \sum_{s \in \mathcal{S}_c} \log f_{\boldsymbol{\theta}}(x_{c,s}) + \lambda_c\big(1 - \sum_{s \in \mathcal{S}_c} x_{c,s}\big), \tag{8.13}$$

where $f_{\boldsymbol{\theta}}(x_{c,s}) := f_{\boldsymbol{\theta}}\big(x_{c,s}(t), \hat{\mathbf{z}}_{c,s}(t)\big)$ is the learned DNN in Eq. 8.12, and $\lambda_c \in \mathbb{R}^+$ is the real non-negative Lagrangian multiplier. Then, we can solve the primal and dual problems:

$$\mathbf{x}^*_c(\lambda_c) = \arg\max_{\mathbf{x}_c \in \mathbb{R}^+} \mathcal{L}(\mathbf{x}_c, \lambda_c), \tag{8.14}$$

$$\lambda^*_c = \arg\min_{\lambda_c \geq 0} \mathcal{L}(\mathbf{x}^*_c(\lambda_c), \lambda_c), \tag{8.15}$$

by computing the partial derivatives with respect to each variable and performing Gradient Descent (GD) iteratively:

$$\begin{cases} x^{(i+1)}_{c,s} := \left[x^{(i)}_{c,s} + \delta^{(i)}_x \cdot \dfrac{\partial \mathcal{L}_c\left(\mathbf{x}^{(i)}_c, \lambda^{(i)}_c\right)}{\partial x^{(i)}_{c,s}} \right]_+, \forall s \in \mathcal{S}_c \\ \lambda^{(i+1)}_c := \left[\lambda^{(i)}_c - \delta^{(i)}_\lambda \cdot \left(1 - \sum_{s \in \mathcal{S}_c} x^{(i+1)}_{c,s}\right) \right]_+, \end{cases} \tag{8.16}$$

where i is the index of iteration, δ_x and δ_λ are the positive updating rates of $x_{c,s}, \forall s \in \mathcal{S}_c$ and λ_c, respectively, and $[x]_+$ is equivalent to $\max\{x, 0\}$. The partial derivative of $\mathcal{L}_c$ with respect to $x_{c,s}$, $\forall s \in \mathcal{S}_c$ is given by:

$$\frac{\partial \mathcal{L}_c\left(\mathbf{x}^{(i)}_c, \lambda^{(i)}_c\right)}{\partial x^{(i)}_{c,s}} = \frac{1}{f_{\boldsymbol{\theta}}\left(x^{(i)}_{c,s}\right) + 1} \cdot \frac{\partial f_\theta\left(x^{(i)}_{c,s}\right)}{\partial x^{(i)}_{c,s}} - \lambda^{(i)}_c. \tag{8.17}$$

8.4.3 Efficient Implementation of Lagrangian Method

One major limitation of the Lagrangian methods is that, if the function is non-linear and non-convex, multiple solutions or folds might exist on the functional surface, and searching on one path may easily get stuck in a local optima. To overcome this, we exploit the automatic differentiation engine of deep learning libraries and design a robust search strategy. By using the automatic differentiation module *torch.autograd* of PyTorch [99], we can efficiently compute the partial derivative of the trained function with respect to any input variables on tensors, e.g., the partial derivative $\partial f_\theta\left(x^{(i)}_{c,s}\right) / \partial x^{(i)}_{c,s}$ in Eq. 8.17. This allows fast parallel computing of multiple searching paths. Thus, we propose the following search strategy:

(1) Based on the assumption that the network states between two successive time steps change smoothly, we propose to initialize the starting points for the optimization of each time slot t with the optimized solution of the previous time slot $t-1$, i.e., $\mathbf{x}_c^{(0)}(t) := \mathbf{x}_c^*(t-1)$;

(2) To find a better (possibly local) optima, we take P neighboring points near $\mathbf{x}_c^{(0)}(t)$ and run the GD optimizations from all P initial points in parallel. After GD optimizations have finished, we select the best solution among them.

The proposed IDLA algorithm is summarized in Algo. 10, where $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $i^{(\max)}$, and η denote the normal distribution for taking neighboring points with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, the maximum iteration steps, and criterion for stopping iteration, respectively.

Algorithm 10 IDLA Algorithm

for $t \in \mathcal{T}$ and $c \in \mathcal{C}$ **do**
  $i \leftarrow 0$
  $\mathbf{x}_c^{(i)}(t) \leftarrow \begin{cases} \textit{default action}, & \text{if } t = 0 \\ \mathbf{x}_c^*(t-1), & \text{Otherwise} \end{cases}$
  Take P neighboring points as:
  $\mathbf{x}_{c_p}^{(i)}(t) := \mathbf{x}_c^{(i)}(t) + \boldsymbol{\epsilon}$, $p \in [1, ..., P]$ with $\boldsymbol{\epsilon} \in \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
  Parallelly compute for all $p \in [1, ..., P]$:
  Initialize Lagrangian multiplier $\lambda_{c_p}^{(i)}$
  Initialize update rate $\delta_{x_p}^{(i)} > 0, \delta_{\lambda_p}^{(i)} > 0$
  while $i \leq i^{(\max)}$ and $\|\mathbf{x}_{c_p}^{(i)}(t) - \mathbf{x}_{c_p}^{(i-1)}(t)\| \geq \eta$ **do**
    Compute partial derivative with (8.17) $\forall s \in \mathcal{S}_c(t)$
    Update optimization variables and Lagrangian multipliers with (8.16)
    Decrease update rate $\delta_{x_p}^i, \delta_{\lambda_p}^i$
    $i \mathrel{+}= 1$
  end while
  $\mathbf{x}_{c_p}^*(t) \leftarrow \mathbf{x}_{c_p}^{(i)}(t)$
  Choose the best solution among all P points that provides the highest utility:
  $\mathbf{x}_c^*(t) := \arg\max_{\mathbf{x}_{c_p}^*(t)} \sum_{s \in \mathcal{S}_c(t)} \log\left(f_{\boldsymbol{\theta}}\left(x_{c_p,s}^*(t)\right)\right)$.
end for

8.5 Experiments

We evaluate the performance of the proposed algorithm by implementing it in Season II and compare the real-time processing performance of the IDLA scheme with two state-of-art schemes including a cell-wise DRL scheme and a traffic-aware baseline that allocates resources proportionally to data traffic demand per slice. We also compare it with an oracle scheme obtained by brute force optimization with the theoretically optimal performance. In addition, we explore the flexibility of the IDLA algorithm when facing slice configuration changes and its transferability from sample-collecting network configurations to a new network configuration.

8.5.1 Network Environment Setup

We built a network system consisting of 4 three-sector base stations with the operating frequency band of 2.6 GHz, i.e., $C = 12$ cells. We defined 4 types of services, where the slice combination $\mathcal{S}_c(t)$ can be configurable and time-varying. Each service has different requirement as average user throughput ϕ_s^* for $s = 1, 2, 3, 4$ defined as $\{2, 1, 1.5, 0.5\}$ Mbit/s, respectively. All cells are provided with the same bandwidth of 20 MHz. In addition, to

imitate the real user traffic, we apply a varying traffic mask $\tau_s(t) \in [0, 1]$, which is collected from a real network system, for each slice $s \in \mathcal{S}_c(t)$ to reflect the daily periodic pattern of per-slice user traffic.

8.5.2 Per-Slice Resource Allocation with IDLA

Before the training process of network QoS estimator $f_{\boldsymbol{\theta}}(\cdot)$, we collected the training samples from the built network scenario in the simulator following the pipeline introduced in Section 8.4.1.

8.5.2.1 Network Sample Collection

We implemented a data augmentation strategy to cover a wider range of (unseen) sample space. The data augmentation strategy is summarized as follows:

(1) For the per-slice samples that achieve lower network QoS than the requirements, i.e., for $r_{c,s}(t) < 1$, we generated augmented samples by replacing the QoS requirements (ϕ_s^*, d_s^*) in the input training sample with the achieved QoS $(\phi_{c,s}(t), d_{c,s}(t))$ and replacing the QoS satisfaction $r_{c,s}(t)$ (sample output) with 1. If the achieved $(\phi_{c,s}(t), d_{c,s}(t))$ were given as requirements, then the requirement would be met.

(2) Conversely, for the per-slice samples that achieve no less network QoS than the requirement, i.e., for $r_{c,s}(t) = 1$, we generated augmented samples by replacing the slice resource partition $x_{c,s}(t)$ in the input training samples with a random value $x'_{c,s}(t) \in [x_{c,s}(t), 1]$. Because the QoS is upper bounded by 1 based on Eq. 8.3, if more resource was given to the slice, due to the monotonicity of $r_{c,s}$ over $x_{c,s}$, the achieved $r_{c,s}$ would be 1 as well.

8.5.2.2 Training of Slice-wise QoS Estimator

Then, for the training of QoS estimator $f_{\boldsymbol{\theta}}(\cdot)$, we built the DNN model with MLP architecture consisting of 4 hidden layers with the number of neurons $(36, 24, 16, 16)$. As proposed in 8.4.1, to better capture the temporal correlation, we used $h = 5$ steps of the historical network reports for estimator training, i.e., the training input $[x_{c,s}, \mathbf{z}_{c,s}] \in \mathbb{R}^{13}$. The training data was collected from the network environment defined in 8.5.1. The model was trained for 200 epochs on 75% training samples and 25% testing samples with Adam optimizer with respect to Mean Absolute Error (MAE) loss.

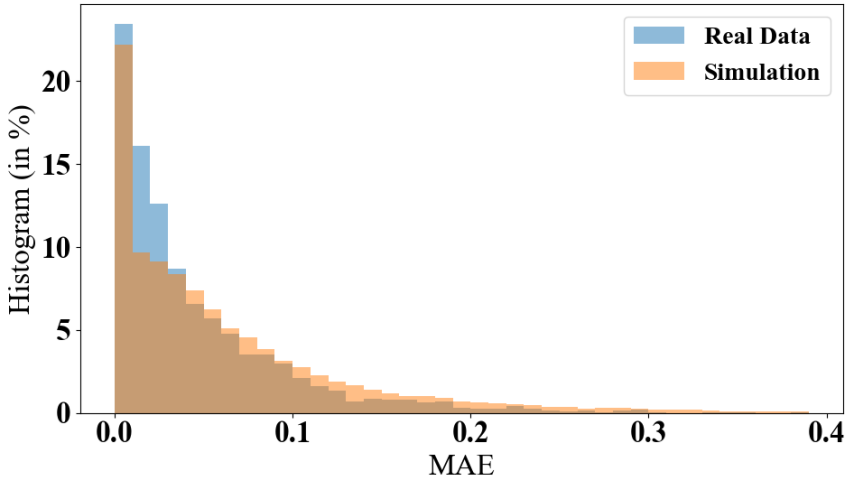

Figure 8.2: Network QoS estimator MAE histogram

Fig. 8.2 shows the histogram of the MAE of network QoS estimator after training was completed. To validate the viability of training a utility estimator, We investigated the

DNN model not only on the simulated data but also on a dataset collected from a real commercial LTE network. By incorporating historical network reports, the estimator can provide accurate network QoS predictions based on the given slice resource partition. The average MAE was 0.0639 and 0.0573 for the simulation and real dataset respectively. The close performances indicate that our method is valid for handling real network systems.

8.5.3 Performance Evaluation

With the derived slice QoS estimator $f_{\boldsymbol{\theta}}(\cdot)$ in hand, we further implement the optimization proposed in 8.4.3 to obtain optimal resource partitions. We compare the performance of the following schemes:

- **IDLA** scheme: our proposed algorithm in Algo. 10 with $P = 5$ neighboring start points, where the offset ϵ for each point follows a normal distribution $\mathcal{N}(0, 0.05)$.
- **DRL** scheme: a distributed TD3 algorithm-based DRL approach similar to our previous work [100], which solves cell-wise optimal slicing resource partitions regarding the reward defined by minimum of network QoS Eq. 8.10 among all slices.
- **Traffic** scheme: a traffic-aware baseline that dynamically adapts slicing resource partitions in each cell proportionally to the current per-slice traffic amount, assuming perfect knowledge of traffic amount.
- **Oracle** scheme: an oracle scheme that provides the near-optimum for the constrained optimization problem. It is derived by using brute-force search for the optimal utility based on the pre-trained $f_{\boldsymbol{\theta}}(\cdot)$ over all potential combinations of $\mathbf{x}_c \in \mathcal{X}_c$ with an interval of 0.05.

To evaluate the performance of IDLA against the other schemes, we implemented an online experiment in the network simulator with dynamic slice configuration, i.e., during the processing of the schemes, we changed the combination of network slices. For a fair comparison, we divide the whole online process into 3 time periods, denoted by $\mathcal{H}_0$, $\mathcal{H}_1$, and $\mathcal{H}_2$ respectively:

- $\mathcal{H}_0$ ($t \in [0, 1000)$): First, we set the network system with 3 slices with combination $\mathcal{S}_c(t) := [1, 2, 4], t \in \mathcal{H}_0, c \in \mathcal{C}$. Both **IDLA** and **Oracle** schemes are under the stage of sample collection, while **DRL** is under the exploration phase for buffer collection without agent training. The **Traffic** scheme provides slice resource partitioning proportional to instantaneous slice traffic demands.
- $\mathcal{H}_1$ ($t \in [1000, 3000)$): The network keeps the same slice configuration as $\mathcal{H}_0$. The **IDLA** and **Oracle** schemes optimize resource allocation based on the pre-trained $f_{\boldsymbol{\theta}}(\cdot)$ over the samples collected within $\mathcal{H}_0$, and **DRL** enter the phase of online training, with the samples collected within $\mathcal{H}_0$ also stored in the replay buffer.
- $\mathcal{H}_2$ ($t \in [3000, 5000]$): At $t = 3000$, the network slice configuration changes to slice combination $\mathcal{S}_c(t) := [1, 2, 3, 4], t \in \mathcal{H}_2, c \in \mathcal{C}$, i.e., we introduce a new slice with the corresponding user group into the network system with the same resource constraints.

In Fig. 8.3, we compare the averaged per-slice user throughput $\phi_{c,s}(t)$ referring to its requirements ϕ_s^* over all cells for $s \in \mathcal{S}_c(t)$ of all schemes during the entire process $\{\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2\}$. Note that in $\mathcal{H}_0$ and $\mathcal{H}_1$, there are only 3 slices with index $[1, 2, 4]$, while later in period $\mathcal{H}_2$, we add a new slice with index 3.

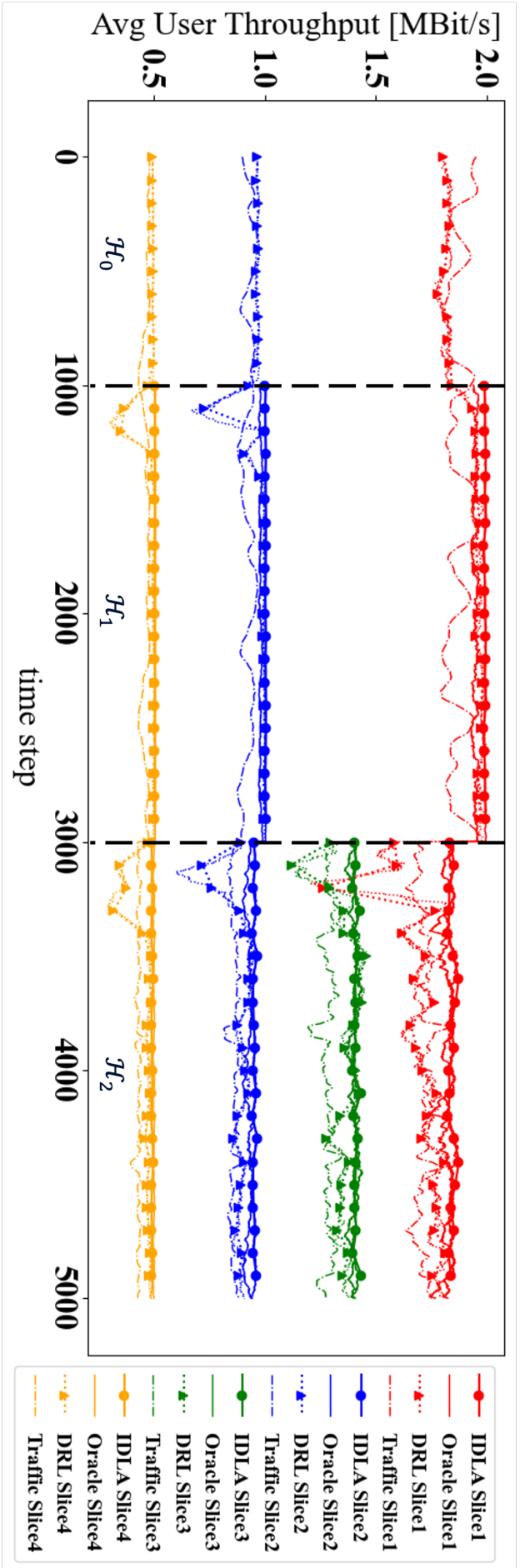

Figure 8.3: Comparison of average user throughput among schemes

After the offline training of QoS estimator $f_{\boldsymbol{\theta}}(\cdot)$, the **IDLA** provides the best performance among all schemes and faster convergence than **DRL** scheme in both online processing phases $\mathcal{H}_1$ and $\mathcal{H}_2$. Moreover, **IDLA** quickly adapts to the new slice configuration (with an added slice) in $\mathcal{H}_2$ and provides robust performance. On the contrary, due to the poor scalability of the cell-wise agent, the **DRL** scheme needs to retrain the model when the slice configuration changes. It is worth noting that since the total network resource remains the same after adding a new slice, the user throughputs in other slices decrease correspondingly to serve the users in the new slice.

To compare the converged performance of all schemes, in Fig. 8.4 we compare the empirical CDF of the converged network QoS satisfaction level of all schemes under both slice configurations. The **IDLA** scheme provides the highest probability of QoS satisfaction under both slice configurations with 0.973 and 0.629 respectively, while **Oracle** and **DRL** schemes achieved similar converged QoS satisfaction. Note that, theoretically, with brute-force search **Oracle** scheme should find a near-optimal solution if the utility estimator can be learned with 100% accuracy. However, in this experiment, the performance of **Oracle** is not as good as **IDLA** due to the estimation error of the utility estimator and the discretization of the searching grid space. In general, **IDLA** provides the best performance in terms of convergence rate, converged performance, and scalability.

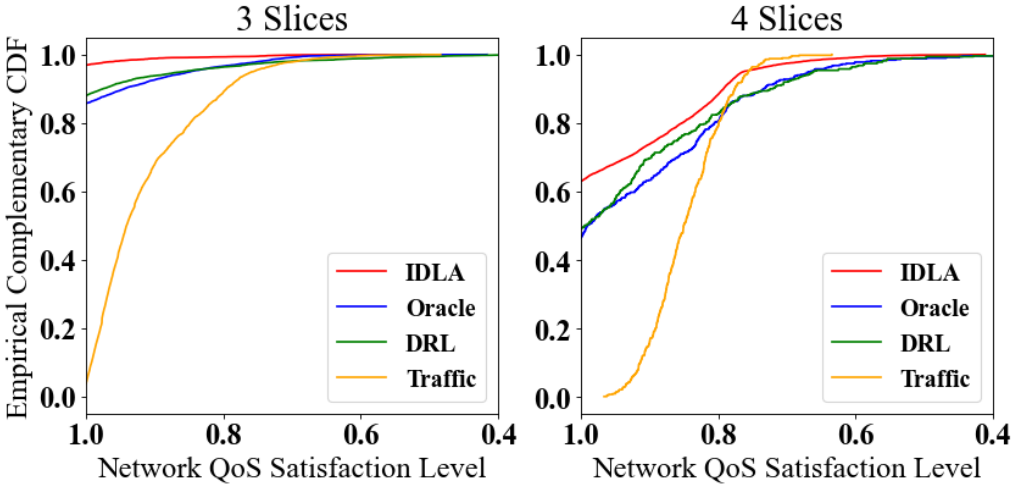

Figure 8.4: Comparison of network utility

8.6 Key Takeaways

In this chapter, we introduced a novel algorithm IDLA that integrates the strong generalization capabilities of the Lagrangian method with the superior approximation capabilities of deep learning. which addresses the resource partitioning problem in network slicing, ensuring compliance with inter-slice resource constraints. Experimental results demonstrate that the IDLA algorithm can achieve near-optimal performance with fast convergence and high generality compared to state-of-the-art solutions. Furthermore, we evaluated the scalability of our approach by deploying the derived model in different network scenarios with varying slicing configurations. The slice-wise resource scheduler in our proposed algorithm provides high scalability and generality, enabling fast and efficient deployment in real network systems.

In the next chapter, we aim to improve IDLA solution by incorporating TL techniques, particularly focusing on DA methods to improve the generality of the DNN model. TL can significantly reduce the training time and data requirements for new scenarios by leveraging knowledge from previously trained models. By applying domain adaptation techniques, we can further refine the DNN model to adapt to diverse and evolving network

conditions, ensuring that the IDLA algorithm maintains high performance across various deployments. Specifically, we will explore the following key aspects:

- **Pre-trained Model Adaptation:** Investigating the use of pre-trained models from similar domains and fine-tuning them for specific network scenarios.
- **Instance-based Transfer:** Utilizing data samples from different but related domains enhances the training dataset and improves the robustness of the model.
- **Feature-based DA:** Extracting and aligning feature distributions between the source and target domains to ensure the DNN model can effectively generalize across different environments.

9. TL-aided IDLA

Based on the per-slice resource partitioning algorithm IDLA, in this chapter, we investigate the TL methods for improved model reproducibility and sample efficiency. Building on the IDLA framework, we address domain gaps across network environments by improving the generality of the slice-wise estimator through TL techniques and thereby tackle the Question 8. Specifically, we implement a VIB-based regression model, consisting of an encoder for invariant feature extraction and a network performance estimator, replacing the traditional MLP-based regression. This VIB model adapts to a broader range of network scenarios, enhancing the derivation of slicing resource partitions following the IDLA approach.

We assess the performance of VIB-based model in mitigating domain discrepancies by comparing its estimation accuracy in source and target domains across various sample combinations and network setups. The evaluation includes comparisons with two baseline methods: one using domain sample re-weighting and the other employing a conventional MLP regression model. Numerical results indicate that the VIB approach significantly enhances target domain accuracy with fewer samples, while maintaining robust performance in source domains. In domain adaptation scenarios, the VIB method outperforms both MLP and loss re-weighting baseline methods. We also implement the IDLA method with the proposed VIB-based estimators in a system-level network simulator to evaluate their practical efficiency in diverse slicing scenarios. Specifically, we demonstrate a dynamic network slicing scenario with varying slicing configurations in real-time, showcasing the high flexibility and scalability of the TL-aided IDLA algorithm compared to a DRL slicing approach and IDLA without TL.

The rest of the chapter is organized as follows. In Section 9.2, we first define the network slicing model, then formulate the slicing resource allocation and DA problems. In Section 9.3, we introduce VIB-aided IDLA as DA process for higher solution generality. The numerical results are demonstrated in Section 9.4, and we conclude this chapter in Section 9.5.

9.1 Motivation

To overcome the challenges of poor model reproducibility and limited sample efficiency in dynamic mobile networks, recent advancements have turned to TL techniques [70]. TL enables the rapid adaptation of pre-solved solutions to new tasks, thereby reducing processing time and data demand. Leveraging prior knowledge from pre-learned tasks, TL

enhances the training process for new tasks that share common features. This approach has demonstrated significant advantages across various machine learning applications, inspiring the introduction of knowledge transfer methods in the field of wireless communication [71, 72, 73]. For network optimization problems, several works [101, 102, 103] have highlighted TL's potential in reducing resource consumption and improving efficiency.

However, effective TL implementation must address challenges such as data imbalance, domain similarity analysis, and negative knowledge transfer. Domain adaptation (DA) [104], which addresses disparities between source and target domains to enhance knowledge transferability, has gained significant attention [105]. In the wireless communication field, DA is applied to boost the robustness and generalizability of network solutions [106, 107, 108]. For instance, the information bottleneck (IB) method [109] identifies maximum informative representations of samples through rate-distortion theory [110], characterizing the trade-off between quantization ratio and expected distortion. This approach has practical applications in classification and generative modeling tasks [111], paving the way for exploring Information Bottleneck (IB)-based methods in wireless communication to achieve flexible and generalizable solutions across different network environments.

The method of TL has been well explored in various fields, while its application in wireless communication remains relatively scarce but is gradually gaining attention. Yang *et al.* [112] applied TL to adapt a pre-trained beamforming model in a massive MIMO system, mitigating hardware limitations and requiring fewer channel data for derivation. TL was also recommended in [113] for radio map estimation, where the authors fine-tuned a source model trained on a specific network for other environments, highlighting TL's efficacy with limited training data. Tailored for DRL, Janiar *et al.* proposed a TL approach [114] to accelerate DRL training in wireless networks with an integrated feature extractor that quantifies the disparity between source and target domains. Experimental results demonstrated significant reductions in training time, outperforming conventional DRL approaches.

Among TL techniques, DA has attracted significant attention in practical applications [115, 116, 117]. Guan *et al.* [118] introduced an uncertainty-aware domain adaptation network (UaDAN) for object detection, applying conditional adversarial learning to align samples with varying degrees of similarity and leveraging uncertainty metrics for adaptive learning. Chen *et al.* [119] introduced CrossTrainer, a DA system that leverages loss re-weighting to improve model reproducibility across different sample sources, though the sensitivity of the loss re-weighting hyperparameter requires expensive tuning and retraining procedures. In end-to-end communication, Raghu *et al.* [120] advocated for DA via autoencoders to reduce the need for frequent retraining amidst changing channel conditions, proposing a method for adapting a Gaussian mixture density network with minimal target distribution samples and validating its effectiveness through simulation. Chen *et al.* [121] introduced an IB-based GAN for medical imaging, using IB theory to enhance cross-domain image translation by preserving relevant features while discarding extraneous information, maintaining object consistency in domain adaptation.

In wireless communication, Zhou *et al.* [108] presented SemiAMR, a semi-supervised network utilizing adversarial learning for cross-domain modulation recognition without the need for pre-training on labeled target domain data, improving classification accuracy. Shi *et al.* [122] addressed the simulation-to-reality gap in network configuration prediction through a teacher-student DNN approach for DA, merging simulation and real-world data to enhance training outcomes.

Fig. 9.1 illustrates the process of slicing resource partitioning in OAM, dynamically allocating per-slice resource budgets as ratios of network resource partitions in each cell at

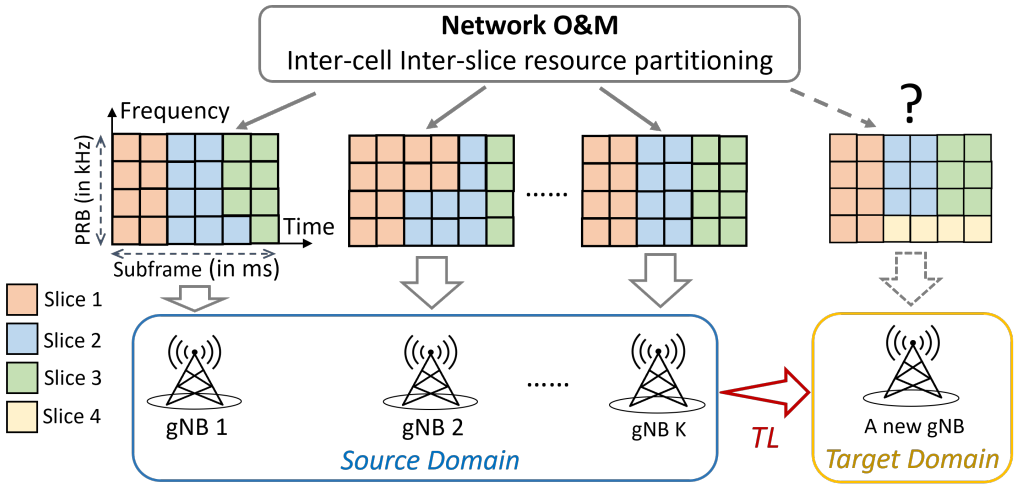

Figure 9.1: TL of network slicing resource allocation

medium time scales, i.e., minutes or quarters. The RAN resource scheduler then periodically allocates PRBs to each service based on these partitions. However, deploying new base stations with dynamic slice configurations and sparse samples poses challenges for slicing resource schedulers using existing strategies. To address this, we propose a TL-aided deep learning approach with DA to optimize slicing resource allocation for OAM, ensuring adaptability to diverse slice configurations and network conditions with high flexibility and generality. Our objective is to optimize slice-wise resource allocation for both existing (source domain) and newly deployed (target domain) network slices by deriving common solutions based on limited domain samples.

9.2 System Model and Problem Formulation

In this section, first, we define the dynamic network slicing system model in Section 9.2.1. Then, in Section 9.2.2, we formulate the scalable slice-wise resource allocation as a nonlinear optimization problem. The DA problem formulation under the context of TL is given later in Section 9.3.1.

9.2.1 System Model

In this work, we consider multi-cell slicing scenarios within a network system comprising a set of cells $\mathcal{C} := \{1, 2, \ldots, C\}$ with dynamically configured slices from a collection denoted as set $\mathcal{S} := \{1, 2, \ldots, S\}$. For each cell $c \in \mathcal{C}$, the set of slicing can be time-varying, represented as $\mathcal{S}_c(t) \subset \mathcal{S}$ with cardinality $|\mathcal{S}_c(t)| = S_c(t)$, where $S_c(t)$ is the number of slices in cell c at time slot $t \in \mathbb{N}_0$. At each time slot t, the instantaneous slice performance is measured by the network QoS in terms of per-slice user throughput $\phi_{c,s}(t)$ and delay $d_{c,s}(t)$, referring to pre-defined requirements ϕ_s^* and d_s^*, respectively. It is worth noting that the slice types in this work are defined with network QoS referencing throughput and delay requirements. The problem formulation and proposed solutions in the following sections can be easily generalized to a broader set of requirements.

For each cell $c \in \mathcal{C}$ at time slot t, OAM optimizes inter-slice resource partitions $\mathbf{x}_c(t)$, composed of ratios of resource allocation to all slices, given by

$$\mathbf{x}_c(t) := \left[x_{c,1}(t), \ldots, x_{c,S_c(t)}(t)\right] \in \mathcal{X}_c(t), \ \forall c \in \mathcal{C}, \tag{9.1}$$

$$\text{where } \mathcal{X}_c(t) := \left\{ [0,1]^{S_c(t)} \middle| \sum_{s \in \mathcal{S}_c(t)} x_{c,s}(t) \leq 1 \right\}. \tag{9.2}$$

Then, the overall collection of network slicing resource partitions is denoted as $\mathbf{x}(t) := [\mathbf{x}_1(t), \ldots, \mathbf{x}_C(t)]$. The slice-wise performance of slice $s \in \mathcal{S}_c(t)$ in cell $c \in \mathcal{C}$ is measured

by the QoS satisfaction level $y_{c,s}(\mathbf{x}(t))$, which refers to the metric for assessing partitioned slice resource resulted by proposed methods, defined as

$$y_{c,s}(\mathbf{x}(t)) := \min\left\{\frac{\phi_{c,s}(\mathbf{x}(t))}{\phi_s^*}, \frac{d_s^*}{d_{c,s}(\mathbf{x}(t))}, 1\right\}, \tag{9.3}$$

where $\phi_{c,s}(\mathbf{x}(t))$ and $d_{c,s}(\mathbf{x}(t))$ are the instantaneous slice throughput and delay at time slot t, respectively. As Eq. 9.3 illustrates, slice performance is determined as the minimum satisfaction level among slice throughput and delay, which is upper-bounded by 1. In other words, the slice performance $y_{c,s}$ is measured as 1 only if both throughput $\phi_{c,s}$ and delay $d_{c,s}$ meet the requirements; otherwise, it is measured as the minimum between $\frac{\phi_{c,s}}{\phi_s^*}$ and $\frac{d_s^*}{d_{c,s}}$.

9.2.2 Problem Formulation

In this work, our primary objective is to provide efficient and scalable solutions for slicing resource partitioning to optimize the utility of network QoS across all slices and cells at each time slot t. In Problem 7, we formulate the general problem of optimizing global network resource partitions concerning the QoS performance at each time slot.

Problem 7 (Global Slicing Problem)**.**

$$\begin{aligned} &\max_{\mathbf{x}(t)} \; U(\mathbf{y}(\mathbf{x}(t))) \\ &\text{subject to } \mathbf{y}(\mathbf{x}(t)) := \left[y_{c,s}(\mathbf{x}(t)) : c \in \mathcal{C}, s \in \mathcal{S}_c(t)\right], \\ &\qquad (9.1), (9.2), (9.3), \; \forall t. \end{aligned} \tag{9.4}$$

We define the network utility function as the sum of the logarithms of the local network performance metrics:

$$U(\mathbf{y}(\mathbf{x}(t))) := \sum_{c\in\mathcal{C}, s\in\mathcal{S}_c(t)} \log\left(y_{c,s}(\mathbf{x}(t)) + 1\right). \tag{9.5}$$

Note that the utility function can be defined based on various system designs and requirements. For example, here we choose Eq. 9.5 as a utility function because we aim to leverage the resource partitions for overall slicing performance without considering specific slice priorities. However, with the superior approximation capability of deep learning, our proposed approach can be applied to a wide range of utility functions.

Solving Problem 7 presents multifaceted difficulties. Firstly, the complexity of the utility function poses challenges for function approximation, mainly due to limited measurements in OAM. In contrast to RAN, where user and channel feedback can be collected with fine time granularity (e.g., in milliseconds), OAM only collects averaged cell and slice-level KPIs with a coarse granularity (e.g., in minutes). Consequently, deriving closed-form expressions becomes exceptionally challenging. Secondly, the flexible slice configurations and inter-slice constraints further complicate the problem, resulting in slow convergence and poor adaptability of deep learning-based approaches. Finally, OAM high scalability demand makes it challenging to use large global models or collaborative multi-agent local models that require extensive exploration to learn from scratch.

Moreover, the challenges of deriving the solution for Problem 7 also raise another concern: since deriving slicing resource partitions for one specific network scenario can be time-consuming and complicated, deploying the same approach when the network scenario changes or under a new environment can incur significant costs in both time and resources. Hence, we pose the question: Can we enhance the adaptability of this solution to accommodate various network environments or setups more effectively?

9.3 TL-aided IDLA Algorithm

In Chapter 8, we introduced the slice-wise resource partitioning solution IDLA for scalable slice configurations within the same network scenario. However, the transferability of IDLA from one specific network scenario to another raises concerns about its performance. This is because even a minor configuration change in the network system can introduce a drift in the slice sample space, which the derived slice-based model may not have encountered before, leading to a possible degradation in the accuracy of the QoS estimator. Consequently, the Lagrangian method loses the gradient guidance necessary to find optimal solutions. To address these challenges, in this section, we propose to enhance the generality of the estimator with a DA model structure. We first provide the domain definitions under the context of the proposed IDLA framework and formulate the DA problem of slice-wise QoS estimation model in Section 9.3.1. Then, in Section 9.3.2, we introduce a VIB-based estimator. The derivation of this model involves finding a common latent sample space, which extracts representative features across different slice configurations. The underlying hypothesis is that certain common hidden patterns exist in samples across diverse cells and slices. By leveraging TL, we aim to identify these representative sample space features, leading to the derivation of a general slice QoS estimator model. Lastly, we evolve VIB-based estimator into IDLA approach.

9.3.1 DA Problem Formulation

Before we formulate the DA problem of the IDLA solution, let us introduce two general definitions related to TL:

- **Domain**: A domain $\mathcal{D} := \{\mathcal{X}, \mathcal{Y}, P(X,Y)\}$ comprises an input feature space $\mathcal{X}$, a label space $\mathcal{Y}$, and the joint probability distribution of random variables X and Y with the sample space $\mathcal{X}$ and $\mathcal{Y}$, respectively.
- **Task**: A task $\mathcal{T} := \{\mathcal{Y}, f(\cdot)\}$ includes the label space $\mathcal{Y}$ and the mapping function $f : \mathcal{X} \to \mathcal{Y}$.

In the following, we use upper case letters for random variables, e.g., X, Y, and lower case letters for the particular realizations (measured samples) of the random variables, e.g., $\mathbf{x}, y$. Formally, we extend the general definition of the TL problem in [104] as follows:

Definition 4 (Transfer Learning)**.** *Given a source domain $\mathcal{D}_\mathsf{S} := \{\mathcal{X}_\mathsf{S}, \mathcal{Y}_\mathsf{S}, P_\mathsf{S}(X,Y)\}$ with a sufficient number of samples $\Omega^{(\mathrm{S})} := \left\{\left(\mathbf{x}_k^{(\mathrm{S})}, y_k^{(\mathrm{S})}\right) : k = 1, \ldots, N_\mathsf{S}\right\}$ for solving the source learning task $\mathcal{T}_\mathsf{S} := \{\mathcal{Y}_\mathsf{S}, f(\cdot)\}$ by minimizing the training loss $l\big(f(\mathbf{x}), y\big)$ over all samples referring to the model $f(\cdot)$:*

$$\begin{aligned} R_\mathsf{S}(f) :=& \mathbb{E}_{(\mathbf{x},y)\sim P_\mathsf{S}(\mathbf{x},y)}\Big[l\big(f(\mathbf{x}), y\big)\Big] \\ =& \int l\big(f(\mathbf{x}), y\big) \cdot P_\mathsf{S}\big(\mathbf{x}, y\big) d\mathbf{x} dy, \end{aligned} \tag{9.6}$$

and a target domain $\mathcal{D}_\mathsf{T} := \{\mathcal{X}_\mathsf{T}, \mathcal{Y}_\mathsf{T}, P_\mathsf{T}(X,Y)\}$ which only has a limited number of samples $\Omega^{(\mathrm{T})} := \left\{\left(\mathbf{x}_k^{(\mathrm{T})}, y_k^{(\mathrm{T})}\right) : k = 1, \ldots, N_\mathsf{T}\right\}$ with $N_\mathsf{T} << N_\mathsf{S}$, the objective of TL is to minimize the expectation of estimation loss in the target domain with $f(\cdot)$:

$$\begin{aligned} R_\mathsf{T}(f) :=& \mathbb{E}_{(\mathbf{x},y)\sim P_\mathsf{T}(\mathbf{x},y)}\Big[l(f(\mathbf{x}), y)\Big] \\ =& \int l\big(f(\mathbf{x}), y\big) \cdot \frac{P_\mathsf{T}\big(\mathbf{x}, y\big)}{P_\mathsf{S}\big(\mathbf{x}, y\big)} P_\mathsf{S}\big(\mathbf{x}, y\big) d\mathbf{x} dy \\ =& \mathbb{E}_{(\mathbf{x},y)\sim P_\mathsf{S}(\mathbf{x},y)}\left[\frac{P_\mathsf{T}\big(\mathbf{x}, y\big)}{P_\mathsf{S}\big(\mathbf{x}, y\big)} \cdot l\big(f(\mathbf{x}), y\big)\right]. \end{aligned} \tag{9.7}$$

Note that in this work, the source and target tasks are identical with the same label space $\mathcal{Y}_\mathsf{T} = \mathcal{Y}_\mathsf{S}$, the objective of TL in this work is to leverage the source data and limited target data to learn a slice QoS estimator model $f(\cdot)$ capable of performing well in the target domain. Under the context of the slice-based QoS estimator in the IDLA approach, each slice $s \in \mathcal{S}_c$ of cell $c \in \mathcal{C}$ can be regarded as a domain $\mathcal{D}_{c,s}$. While all $\mathcal{D}_{c,s}$ and $\mathcal{T}_{c,s}$, $\forall s \in \mathcal{S}_c, c \in \mathcal{C}$ share the same sample spaces, such as $x_{c,s} \in [0,1]$ in Eq. 9.2, $\mathbf{o}_{c,s} \in \Re^{2H+2}$ in Eq. 8.8, and $y_{c,s} \in [0,1]$ in Eq. 8.9, respectively. For brevity, in this section we denote $\bar{\mathbf{x}}_{c,s}(t) := (x_{c,s}(t), \mathbf{o}_{c,s}(t))$ with $\bar{\mathbf{x}}_{c,s}(t) \in \bar{\mathcal{X}}_{c,s}$, where $\bar{\mathcal{X}}$ stands as an extension of $\mathcal{X}$ because of adding $\mathbf{o}(t)$. For each domain, the set of input samples for the QoS estimator is given by:

$$\bar{\mathcal{X}}_{c,s} := \{\bar{\mathbf{x}}_{c,s}(t) : \text{ for } t = 1, \ldots, T\} \subset \bar{\mathcal{X}}, \tag{9.8}$$

and the set of output samples is denoted by:

$$\mathcal{Y}_{c,s} := \{y_{c,s}(t) : \text{ for } t = 1, \ldots, T\} \subset \mathcal{Y}, \tag{9.9}$$

where $\bar{\mathcal{X}} := \bigcup_{c\in\mathcal{C}, s\in\mathcal{S}_c(t)} \bar{\mathcal{X}}_{c,s}$ and $\mathcal{Y} := \bigcup_{c\in\mathcal{C}, s\in\mathcal{S}_c(t)} \mathcal{Y}_{c,s}$ are the collections of input samples and output samples, respectively. Then, the local domain can be denoted as $\mathcal{D}_{c,s} := \{\bar{\mathcal{X}}, \mathcal{Y}, P_{c,s}(\bar{X}, Y)\}$, and the local task of solving local utility estimation is $\mathcal{T}_{c,s} := \{\mathcal{Y}, f_{c,s}(\cdot)\}$, where $f_{c,s} : \bar{\mathcal{X}} \to \mathcal{Y}$.

Based on TL formulation, the problem of DA for training a slice-based QoS estimator can be formulated as Problem 8. For brevity, we omit the time index t starting from this section.

Problem 8 (DA Problem for Slice QoS Estimator)**.** *Given a set of source domains $\mathcal{D}_\mathsf{S} := \{\mathcal{D}_{c,s} : c \in \bar{\mathcal{C}}, s \in \bar{\mathcal{S}}_c\}$, where each domain comprises collected samples from source cells $\bar{\mathcal{C}} \subset \mathcal{C}$ and slices $\bar{\mathcal{S}}_c \subset \mathcal{S}_c$, and a target domain $\mathcal{D}_\mathsf{T} := \{\bar{\mathcal{X}}, \mathcal{Y}, P_\mathsf{T}(\bar{X}, Y)\}$ representing the slice $s' \in \mathcal{S}_{c'}$ of cell $c' \notin \bar{\mathcal{C}}$, we aims to derive a slice-based QoS estimator $f^{(\mathrm{TL})}(\cdot)$ based on the samples from $\mathcal{D}_\mathsf{S}$, such that the expectation of QoS estimation loss over samples in $\mathcal{D}_\mathsf{T}$ is minimized. The problem is given by*

$$\begin{aligned}\min_{f^{(\mathrm{TL})}} \; R_\mathsf{T}\left(f^{(\mathrm{TL})}\right) :=& \mathbb{E}_{(\bar{\mathbf{x}},y)\sim P_\mathsf{T}(\bar{\mathbf{x}},y)}\left[l\left(f^{(\mathrm{TL})}(\bar{\mathbf{x}}), y\right)\right] \\ =& \mathbb{E}_{(\bar{\mathbf{x}},y)\sim P_\mathsf{S}(\bar{\mathbf{x}},y)}\left[\frac{P_\mathsf{T}(\bar{\mathbf{x}},y)}{P_\mathsf{S}(\bar{\mathbf{x}},y)} l\left(f^{(\mathrm{TL})}(\bar{\mathbf{x}}), y\right)\right], \\ &\text{subject to } (9.8), (9.9),\end{aligned} \tag{9.10}$$

$P_\mathsf{S}(\bar{\mathbf{x}}, y) := \sum_{s\in\bar{\mathcal{S}}_c, c\in\bar{\mathcal{C}}} \omega_{c,s} P_{c,s}(\bar{\mathbf{x}}, y)$ is the mixture distribution of samples from all source domains used to derive the model $f^{(\mathrm{TL})} : \bar{\mathcal{X}} \to \mathcal{Y}$, where $\omega_{c,s}$ are the weights assigned to each local distribution with $\sum_{s\in\bar{\mathcal{S}}_c, c\in\bar{\mathcal{C}}} \omega_{c,s} = 1$. In this work, we assume the domain weights are equally distributed as $\omega_{c,s} := \frac{1}{|\bar{\mathcal{C}}|\cdot|\bar{\mathcal{S}}_c|}, \forall s \in \bar{\mathcal{S}}_c, c \in \bar{\mathcal{C}}$. $P_\mathsf{T}(\bar{\mathbf{x}}, y)$ denotes the distribution of samples collected in slice s' of cell c'.

The challenges in developing an adaptive QoS estimator $f^{(\mathrm{TL})}(\cdot)$, that performs well on both source and target domains, arise primarily from the differences between $P_\mathsf{S}(\bar{\mathbf{x}}, y)$ and $P_\mathsf{T}(\bar{\mathbf{x}}, y)$. In our problem, the source and target task functions are the same, i.e., $P_{c_i,s_p}(y|\bar{\mathbf{x}}) = P_{c_j,s_q}(y|\bar{\mathbf{x}})$ with respect to a common slice-based estimator $f^{(\mathrm{TL})}(\bar{\mathbf{x}}) = y$ in Eq. 8.6. Hence, we can express the relationship as:

$$\frac{P_\mathsf{T}(\bar{\mathbf{x}}, y)}{P_\mathsf{S}(\bar{\mathbf{x}}, y)} = \frac{P_\mathsf{T}(\bar{\mathbf{x}}) P_\mathsf{T}(y|\bar{\mathbf{x}})}{P_\mathsf{S}(\bar{\mathbf{x}}) P_\mathsf{S}(y|\bar{\mathbf{x}})} = \frac{P_\mathsf{T}(\bar{\mathbf{x}})}{P_\mathsf{S}(\bar{\mathbf{x}})}. \tag{9.11}$$

The difference between domains depends solely on the distributions of input samples. This allows us to rewrite Eq. 9.10 as:

$$\min_{f^{(\mathrm{TL})}} \; \mathbb{E}_{(\bar{\mathbf{x}},y)\sim P_\mathsf{S}(\bar{\mathbf{x}},y)}\left[\frac{P_\mathsf{T}(\bar{\mathbf{x}})}{P_\mathsf{S}(\bar{\mathbf{x}})} \cdot l\left(f^{(\mathrm{TL})}(\bar{\mathbf{x}}), y\right)\right]. \tag{9.12}$$

9.3.2 VIB-based Slice QoS Estimation

From Eq. 9.12, it is evident that the TL problem becomes an authentic machine learning problem if $P_{\mathsf{T}}(\bar{\mathbf{x}}) = P_{\mathsf{S}}(\bar{\mathbf{x}})$, i.e., training and validating model $f^{(\mathrm{TL})}$ on the same sample distribution. We aim to map samples from different domains into a common representative latent feature space. This common space should extract features informative enough to distinguish and represent the patterns of various domains, allowing the derivation of a general QoS estimator capable of handling samples from different distributions.

Information Bottleneck: In this work, we propose to derive an adaptive slice-based network QoS estimator $f^{(\mathrm{TL})}$ based on source domain samples from $P_{\mathsf{S}}(\bar{\mathbf{x}}, y)$ that performs well on target samples from $P_{\mathsf{T}}(\bar{\mathbf{x}}, y)$ using a domain adaptation approach inspired by the VIB method [123]. Specifically, we aim to find intermediate representations of $\bar{X}$ as latent variables Z by solving an encoder $g_{\boldsymbol{\epsilon}}$ parameterized by $\boldsymbol{\epsilon}$. We aim for $Z \in \mathcal{Z}$ to be maximally informative about the training output $Y \in \mathcal{Y}$. The informative level is measured by the mutual information:

$$I(Z, Y; \boldsymbol{\epsilon}) := \int P(\mathbf{z}, y|\boldsymbol{\epsilon}) \log \frac{P(\mathbf{z}, y|\boldsymbol{\epsilon})}{P(\mathbf{z}|\boldsymbol{\epsilon})P(y|\boldsymbol{\epsilon})} d\mathbf{z}dy. \tag{9.13}$$

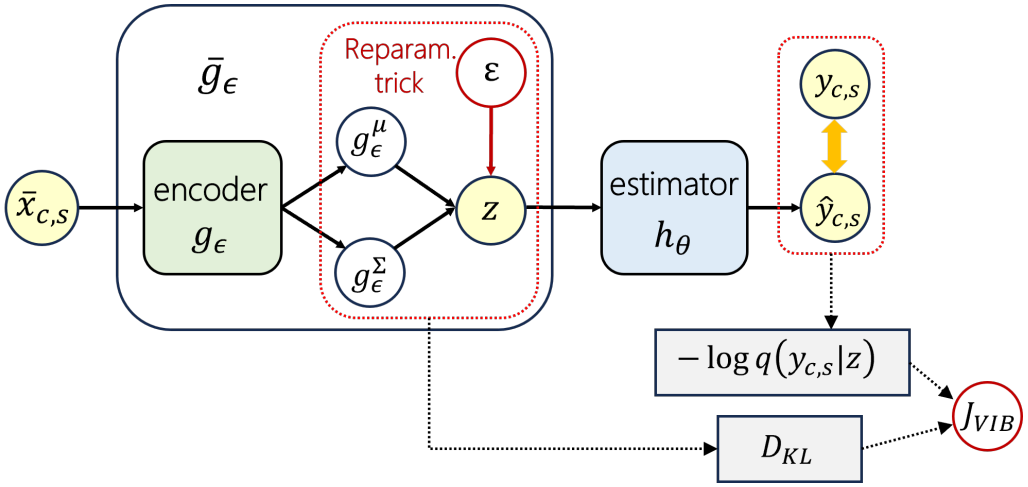

Figure 9.2: VIB-based slice QoS estimator

We aim for a representative space $\mathcal{Z}$ that is informative enough for capturing $\mathcal{Y}$ while maintaining a low dependency on $\bar{\mathcal{X}}$. To restrict the complexity of the representation space, we introduce a constraint on the mutual information between $\bar{X}$ and Z as $I(Z, \bar{X}; \boldsymbol{\epsilon}) \leq I_c$. Here, I_c represents the information bottleneck [109], which sets the limit of information about $\bar{X}$ that Z can obtain. The problem of finding the encoder $g_{\boldsymbol{\epsilon}}$ is given by:

$$\max_{\boldsymbol{\epsilon}} . I(Z, Y; \boldsymbol{\epsilon}), \text{ subject to } I(Z, \bar{X}; \boldsymbol{\epsilon}) \leq I_c. \tag{9.14}$$

And by introducing a constant $\beta \geq 0$, the objective function for solving Eq. 9.14 is:

$$\mathcal{L}_{IB}(\boldsymbol{\epsilon}) := I(Z, Y; \boldsymbol{\epsilon}) - \beta I(Z, \bar{X}; \boldsymbol{\epsilon}). \tag{9.15}$$

By maximizing the first term of Eq. 9.15, the function encourages the representation Z to be predictive of output Y, while minimizing the second term encourages Z to "forget" $\bar{X}$. This means Z is forced to be a minimal sufficient statistic of $\bar{X}$ for predicting Y. This objective function aligns with Shannon rate-distortion theory [110], where $I(Z, \bar{X})$ represents the "rate" of information about $\bar{X}$ encoded in Z, and $I(Z, Y)$ inversely relates to "distortion", reflecting how well the representation Z predicts Y. The weight factor β adjusts the trade-off between the information compression rate and prediction distortion.

VIB-based model: To derive the solution for Eq. 9.15, we propose using the VIB approach, approximating the problem with variational inference. We assume $P(\mathbf{z}|\bar{\mathbf{x}}, y) = P(\mathbf{z}|\bar{\mathbf{x}})$ corresponding to the Markov chain $\mathcal{Y} \to \mathcal{X} \to \mathcal{Z} \to \hat{\mathcal{Y}}$, i.e., the representation variables Z are not directly dependent on output labels Y, while $\hat{Y}$ denoted the prediction of Y generated from $\mathcal{Z}$ without direct dependence on Y. Under this assumption, the joint distribution of $P(\bar{\mathbf{x}}, \mathbf{z}, y)$ can be factorized as:

$$\begin{aligned} P(\bar{\mathbf{x}}, \mathbf{z}, y) =& P(\mathbf{z}|\bar{\mathbf{x}}, y)P(y|\bar{\mathbf{x}})P(\bar{\mathbf{x}}) \\ =& P(\mathbf{z}|\bar{\mathbf{x}})P(y|\bar{\mathbf{x}})P(\bar{\mathbf{x}}). \end{aligned} \tag{9.16}$$

Recalling the objective function Eq. 9.14, besides $P(\bar{\mathbf{x}}, y)$ which is determined by the sample distribution, the only content we need now is the encoding function $P(\mathbf{z}|\mathbf{x})$, while other distributions can be derived based on the Markov chain.

Following the derivation process of VIB, we can simplify two terms in Eq. 9.15 respectively:

$$I(Z, Y) \geq \int P(\bar{\mathbf{x}})P(\mathbf{z}|\bar{\mathbf{x}})P(y|\bar{\mathbf{x}}) \log Q(y|\mathbf{z}) d\bar{\mathbf{x}} d\mathbf{z} dy, \tag{9.17}$$

$$I(Z, \bar{X}) \leq \int P(\bar{\mathbf{x}})P(\mathbf{z}|\bar{\mathbf{x}}) \log \frac{P(\mathbf{z}|\bar{\mathbf{x}})}{R(\mathbf{z})} d\bar{\mathbf{x}} d\mathbf{z}, \tag{9.18}$$

where $Q(y|\mathbf{z})$ and $R(\mathbf{z})$ are the variational approximations of $P(y|\mathbf{z})$ and $P(\mathbf{z})$ respectively. It is necessary to address that our proposed VIB-based estimator aims to extract latent variables Z as sufficient representatives of the original slice samples from different domains and thereby capture common latent features in between to conduct robust slice QoS estimations.

In this way, solving the Lagrangian Eq. 9.15 boils down to maximizing $\mathcal{L}_{IB}$ to find the optimal $g_{\boldsymbol{\epsilon}}$ for encoding $P(\mathbf{z}|\bar{\mathbf{x}}; \boldsymbol{\epsilon})$. In the context of slice-based QoS estimation presented in this work, the joint sample distribution $P(\bar{\mathbf{x}}, y)$ can be approximated based on collected samples from domains, as indicated by Eq. 9.8 and Eq. 9.9, using the empirical distribution. Therefore, in practice, we can compute $\mathcal{L}_{IB}$ by:

$$\mathcal{L}_{IB} \approx \frac{1}{N_{\mathsf{S}}} \sum_{c \in \bar{\mathcal{C}}, s \in \bar{\mathcal{S}}_c} \Big[\sum_{\mathbf{z}} P(\mathbf{z}|\bar{\mathbf{x}}_{c,s}) \log Q(y_{c,s}|\mathbf{z}) - \beta \; P(\mathbf{z}|\bar{\mathbf{x}}_{c,s}) \log \frac{P(\mathbf{z}|\bar{\mathbf{x}}_{c,s})}{R(\mathbf{z})} \Big], \tag{9.19}$$

where N_{S} is the number of collected samples from source domains. Assuming $P(\mathbf{z}|\bar{\mathbf{x}}_{c,s})$ follows a Gaussian distribution, we define the encoding $P(\mathbf{z}|\bar{\mathbf{x}}_{c,s}) := \mathcal{N}(\mathbf{z}|g^{\mu}_{\boldsymbol{\epsilon}}(\bar{\mathbf{x}}_{c,s}), g^{\Sigma}_{\boldsymbol{\epsilon}}(\bar{\mathbf{x}}_{c,s}))$ with respect to the encoder $g_{\boldsymbol{\epsilon}}$. To enable differenitability for backpropagation, we sample $\mathbf{z}$ from $P(\mathbf{z}|\bar{\mathbf{x}}_{c,s})$ using the reparameterization trick [124] with respect to a Gaussian random variable $\varepsilon \sim \mathbb{N}(0, \mathbf{I})$. For convenience, we denote $\bar{g}_{\boldsymbol{\epsilon}}(\bar{\mathbf{x}}_{c,s}, \varepsilon)$ as deterministic function by compositing of $g_{\boldsymbol{\epsilon}}$ and reparameterization trick.

In summary, we can derive the VIB model for slice-based QoS estimation with domain adaptation by minimizing the following objective function:

$$J_{VIB} = \frac{1}{N_{\mathsf{S}}} \sum_{c \in \bar{\mathcal{C}}, s \in \bar{\mathcal{S}}_c} \mathbb{E}\Big[\log Q(y_{c,s}|\bar{g}_{\boldsymbol{\epsilon}}(\bar{\mathbf{x}}_{c,s}, \varepsilon)) + \beta \; D_{KL}\big[P(\mathbf{z}|\bar{\mathbf{x}}_{c,s}) \| R(\mathbf{z})\big] \Big], \tag{9.20}$$

the variational estimation $Q(y_{c,s}|\bar{g}_{\boldsymbol{\epsilon}}(\bar{\mathbf{x}}_{c,s}, \varepsilon))$ can be defined as a DNN model $h_{\boldsymbol{\theta}} : \mathcal{Z} \to \mathcal{Y}$ for QoS estimation, considering the derived representations $\mathbf{z} = \bar{g}_{\boldsymbol{\epsilon}}(\bar{\mathbf{x}}_{c,s}, \varepsilon)$. Therefore, the proposed VIB model for QoS estimation comprises two sub-models, namely, $g_{\boldsymbol{\epsilon}}$ as the encoder and $h_{\boldsymbol{\theta}}$ as the QoS estimator. The model structure is illustrated in Fig. 9.2.

9.3.3 IDLA with DA

In Chapter 8, we propose to collect slice-wise network observations as in Eq. 8.8 to capture common features from different slices for learning a general slice QoS estimator $f_{\boldsymbol{\theta}}$. For DA problems, the model is required to be generalizable to different source domains but also "predictable" for adapting to unseen domains. This implies that DA model should be capable of distinguishing domains and learning hidden patterns through the collected source samples. Following the approach proposed in Section 9.3.2, we can derive a VIB-based model as an adaptive QoS estimator.

Based on the VIB model, in the IDLA approach, the Lagrangian function for solving local Problem 6 of each cell $c \in \mathcal{C}$ is reformulated as:

$$\mathcal{L}_{VIB}(\mathbf{x}_c, \lambda_c; \varepsilon) := \sum_{s \in \mathcal{S}_c} \log\left(h_{\boldsymbol{\theta}} \circ \bar{g}_{\boldsymbol{\epsilon}}(\bar{\mathbf{x}}_{c,s}, \varepsilon) + 1\right) + \lambda_c\Big(1 - \sum_{s \in \mathcal{S}_c} x_{c,s}\Big). \tag{9.21}$$

Similarly, we can solve the primal and dual problems and solve each local optimization with GD iteratively:

$$\begin{cases} x_{c,s}^{(i+1)} := \left[x_{c,s}^{(i)} + \delta_x^{(i)} \cdot \dfrac{\partial \mathcal{L}_{VIB}\left(\mathbf{x}_c^{(i)}, \lambda_c^{(i)}\right)}{\partial x_{c,s}^{(i)}}\right]_+, \forall s \in \mathcal{S}_c \\ \lambda_c^{(i+1)} := \left[\lambda_c^{(i)} - \delta_\lambda^{(i)} \cdot \left(1 - \sum_{s \in \mathcal{S}_c} x_{c,s}^{(i+1)}\right)\right]_+. \end{cases} \tag{9.22}$$

The VIB model is composed of models $g_{\boldsymbol{\epsilon}}$ and $h_{\boldsymbol{\theta}}$, and their differentiability is guaranteed by implementing the reparameterization trick for sampling latent representation $\mathbf{z}$, the derivatives of $\mathcal{L}_{VIB}$ are derivable with respect to $x_{c,s}$ as:

$$\frac{\partial \mathcal{L}_{VIB}}{\partial x_{c,s}^{(i)}} = \frac{1}{h_{\boldsymbol{\theta}} \circ \bar{g}_{\boldsymbol{\epsilon}} + 1} \cdot \frac{\partial h_{\boldsymbol{\theta}} \circ \bar{g}_{\boldsymbol{\epsilon}}}{\partial \bar{g}_{\boldsymbol{\epsilon}}} \cdot \frac{\partial \bar{g}_{\boldsymbol{\epsilon}}}{\partial x_{c,s}^{(i)}} - \lambda_c^{(i)}. \tag{9.23}$$

With these changes, the VIB can further proceed with the slicing resource partitioning following the pipeline of the IDLA algorithm. Similarly, to ensure a robust and efficient optimization process, for VIB-aided IDLA algorithm, we also implement the searching strategies presented in Section 8.5.2. Specifically, for parallel exploration in Π different paths, we select the optimal among them after convergence with:

$$\mathbf{x}_c^* := \arg\max_{\mathbf{x}_{c\pi}^*} \sum_{s \in S_c} \log\left(h_{\boldsymbol{\theta}} \circ \bar{g}_{\boldsymbol{\epsilon}}\left(x_{c_\pi,s}^*, \mathbf{o}_{c,s}, \varepsilon\right) + 1\right). \tag{9.24}$$

In Algo. 11, we demonstrate the complete process of IDLA algorithm with VIB-based estimator, where $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $i^{(\max)}$, and η denote the normal distribution for taking neighboring points with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, the maximum iteration steps, and criterion for stopping iteration, respectively.

9.4 Experiments

In this section, we evaluate the performance of the proposed VIB-based model and IDLA algorithm under Season II, where we can define various types of slices regarding slice service requirements and build up network environments with arbitrary scales. First, we introduce the network environments for assessing our proposed methods, focusing on the analysis of sample-based domain discrepancies. Utilizing samples gathered from corresponding domains, we derive slice-wise network QoS estimators. Subsequently, to evaluate the slicing

Algorithm 11 VIB-aided IDLA Algorithm

for $t \in T$ and $c \in C$ **do**
  $i \leftarrow 0$
  $\mathbf{x}_c^{(i)}(t) \leftarrow \begin{cases} \textit{default action}, & \text{if } t = 0 \\ \mathbf{x}_c^*(t-1), & \text{otherwise} \end{cases}$
  Take Π neighboring points as:
  $\mathbf{x}_{c_\pi}^{(i)}(t) := \mathbf{x}_c^{(i)}(t) + \boldsymbol{\epsilon}$, $\pi \in [1, ..., \Pi]$, with $\boldsymbol{\epsilon} \in \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
  Parallelly compute for all $\pi \in [1, ..., \Pi]$:
  Initialize Lagrangian multiplier $\lambda_{c_\pi}^{(i)}$
  Initialize update rates $\delta_x^{(i)} > 0, \delta_\lambda^{(i)} > 0$
  while $i \leq i^{(\max)}$ and $\|\mathbf{x}_{c_\pi}^{(i)}(t) - \mathbf{x}_{c_\pi}^{(i-1)}(t)\| \geq \eta$ **do**
    Compute partial derivatives with respect to (9.23)
    Update optimization variables multipliers with (9.22)
    Decrease update rates $\delta_x^{(i)}, \delta_\lambda^{(i)}$
    $i \mathrel{+}= 1$
  end while
  $\mathbf{x}_{c_\pi}^*(t) \leftarrow \mathbf{x}_{c_\pi}^{(i)}(t)$
  Choose the optimal among Π points with (9.24)
end for

resource allocation performance of the VIB-aided IDLA algorithm in real-time, we deployed in an evolving network slicing scenario within Season II, benchmarking its performance against other state-of-the-art solutions, including a IDLA approach with DNN-based estimator, a DRL approach and a traffic-aware resource allocation method. Finally, we assess the adaptability and scalability of the VIB-based models by comparing their DA efficacy against traditional DNN models and a sample re-weighting baseline to address their DA performance under data imbalance scenarios.

9.4.1 Network Environment Setup

To implement the proposed methods and evaluate their transferability and scalability, we aimed for building a network system that can imitate a realistic scenario where slicing settings, such as slice requirements or served user behaviors, may vary within a consistent network architecture. Therefore, to induce domain disparities, we constructed a virtual network system consisting of 4 three-sector base stations operating at 2.6 GHz located in a small area of Helsinki city, represented by a network environment with $C = 12$ cells under the buildings map of Helsinki in Season II. All cells are provided with the same bandwidth of 20 MHz.

We defined a set of network slices with distinct service requirements with varying user throughput ϕ^* varying from 0.5 to 4 Mbit/s. The slice combination of any cell c can be configurable and time-varying, while each slice $s \in \mathcal{S}_c(t)$ is correspondingly assigned to a specific service. Besides, to manually create diverse user behaviors across cells, we categorized users into groups with varying moving radius $U^{(\mathrm{R})}$ from 200 to 100 meters, and the maximum user number per service $U^{(\mathrm{N})}$ changing between $12 - 36$.

These deliberate discrepancies among slices and user groups aim to simulate diverse domain sample characteristics under an identical network architecture. During experiments, we have the flexibility to construct slicing scenarios by selecting various combinations of slices and user groups under the structure of the pre-built network system. Specifically, to address the performance of the proposed DA method, we are able to acquire network data from a particular set of cells, denoted as source domains, to train slice-aware network performance estimation models. Subsequently, we evaluate these models' efficacy on the

cell not included in the source domain, i.e., denoted as the target domain, to assess their transferability. For convenience, we omit the units in the rest of the section.

9.4.2 Deriving of VIB-based Slice QoS Estimator

In this subsection, we first outline the process for collecting training samples from the network system we established above for training slice-wise QoS estimators. This is followed by an introduction of training setups for these estimators, including the specification of training hyperparameters. We explored various configurations of slices and user groups to generate diverse per-slice samples for DA assessment. The per-slice samples that exhibited common characteristics were aggregated as source domains for training slice-wise QoS estimators. Subsequently, we deployed these models on target domain samples after training convergence to evaluate the DA features of the proposed estimator model. These target domains, while distinct, retain relevance to their corresponding source domains. In Section 9.4.4, we introduce the selection of source and target domain pairs by identifying specific combinations of network slice setups as indicated in Table 9.1 to facilitate evaluation.

9.4.2.1 Domain Sample Collection

Before implementing the training of proposed slice QoS estimators, we first demonstrate the data collection process within the Season II network system. Following the data collection pipeline presented in Section 8.5.2.1, we collect per-slice samples across all cells, incorporating $H = 5$ steps of historical data to enhance the capture of temporal correlations. Consenquently, this forms the training input $[x_{c,s}, \mathbf{o}_{c,s}] \in \Re^{17}$ refer to Eq. 8.8.

It is worth mentioning that consistent with DA assumption, we have limited data volumes in target domains, which results in a much smaller set of samples compared to those from source domains. Before initiating the training of QoS estimators with these per-slice samples, we conducted an analysis of the discrepancy between source and target domains. This preliminary step was essential for assessing the feasibility and potential challenges of implementing DA techniques across domains. Accordingly, we derived the domain discrepancy measurements utilizing the approach from Section 7.5.2. This discrepancy metric is determined by averaging the Kullback-Leibler (KL) divergence across latent variables of the source and target samples with Eq. 7.7.

where i, j are the domain labels and $(\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k)k \in i, j$ are the parameters of latent variable distributions extracted with $P^{\text{VAE}}(\mathbf{z}_k|\mathbf{x}_k) = \mathcal{N}(\boldsymbol{\mu}_k, \text{diag}(\boldsymbol{\sigma}_k))$ from $\mathbf{x}_n \in \mathcal{X}_i$ and $\mathbf{x}_m \in \mathcal{X}_j$. Please note that the encoder used here differs from the one in the VIB-based estimation model.

9.4.2.2 Model Training

In this section, we evaluate the DA performance of proposed models by comparing slice QoS estimation accuracy across source and target domains in each defined comparison pair. Regarding the DA approach, in addition to our proposed VIB model, we implement a Label Distribution Smoothing (LDS) [125] technique to address the data distribution imbalance issue across different slices. LDS decicates to smooth the domain label distribution by convolving the empirical distribution with a symmetric kernel. In this work, we obtained a smoothed version of the domain label distribution. This smoothed distribution was used to re-weight the loss function during training, inversely proportional to the effective density of each domain label, thereby giving more emphasis to rarer domain samples and improving the model performance on under-represented data. The weighted estimation loss in the form of the MAE for LDS training can be expressed as:

$$L_{LDS} = \frac{1}{N} \sum_{i=1}^{N} w_i \cdot |\hat{y}_i - y_i|. \tag{9.25}$$

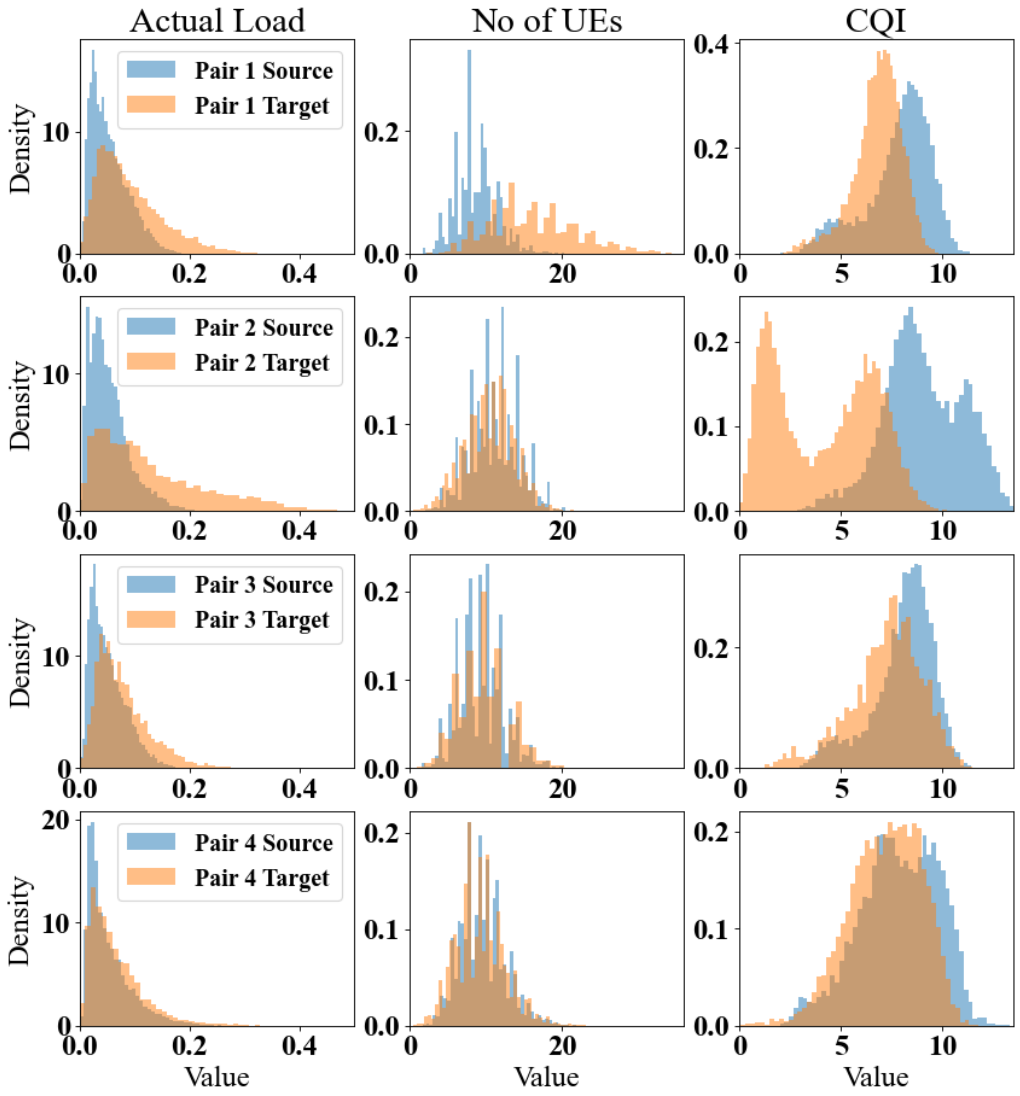

Figure 9.3: Compare distributions of input between domains

The weights for each sample are given as $w_i = \frac{1}{\tilde{p}(y_i)}$, where $\tilde{p}(y_i)$ represents the effective presenting frequency of y_i determined by its distribution, emphasizing domains with fewer samples to potentially mitigate the impact of label imbalance. Specifically, we compare the following estimation models with corresponding model configurations:

- **VIB**: The VIB-based model incorporated both an encoder and a predictor, both structured as MLPs with hidden layers $(36, 24, 16)$ for the encoder and $(24, 16, 16)$ for the predictor, respectively. To achieve a good trade-off between representative richness and estimation precision, we mapped the original input samples $[x_{c,s}, \mathbf{o}_{c,s}] \in \Re^{17}$ into latent variables $z_{c,s} \in \Re^{12}$. It is trained by targeting Eq. 9.20, where β was set to 0.002 to balance the model ability to generalize across domains while maintaining meaningful information in the latent representation. The distribution $r(z)$ of the latent variables is assumed as a standard Gaussian $\mathbb{N}(0, I)$.

- **DNN**: The conventional DNN-based estimator, constructing with a MLP architecture comprising 5 hidden layers with neurons $(64, 36, 24, 16, 16)$, targeting MAE as estimatino loss metric $L_{DNN} = \frac{1}{N}\sum_{i=1}^{N} |\hat{y}_i - y_i|$;

- **LDS**: While not as a model itself, the LDS technique is applied to DNN estimator model to address label imbalance through effective re-weighting, enhancing the DA

performance of DNN estimator, serves as a baseline to assess the performance of VIB models. The LDS model is built upon the DNN setups, while training respect to weighted loss Eq. 9.25;

For each domain pair, we trained all model types over 200 epochs, partitioning the data into 75% for training and 25% for testing, utilizing the Adam optimizer.

9.4.3 Evaluation of Resource Allocation Performance

In this subsection, we assess the online performance of slice QoS estimators by implementing the proposed IDLA procedure presented in 9.3. Our objectives were twofold: firstly, we intend to evaluate the online performance of the estimators within the framework of the proposed IDLA algorithm, and secondly, we aim to explore the adaptability of the IDLA approach in practical network dynamics. Specifically, we investigated how the IDLA would respond to domain shifts when facing network slicing changes. In this subsection, we implement the IDLA approaches based on derived models in a scenario with dynamically changing networks. For comparison, we also implemented a DRL-based approach and a traffic-aware slicing resource partitioning mechanism as baselines. In detail, the following schemes were implemented for online comparisons:

- **IDLA-VIB**: the proposed IDLA algorithm based on the VIB-based slice QoS estimator. The IDLA algorithm is implemented by setting the number of initial neighboring points $P = 5$ with offset $\varepsilon \sim \mathbb{N}(0, 0.05)$;
- **IDLA** scheme: the proposed IDLA algorithm with a DNN-based QoS estimator. The initial neighboring point setting is identical to the **IDLA-VIB**;
- **DRL**: a distributed cell-wise DRL approach using a TD3 algorithm, as discussed in Section 5.4, which solves optimal slicing resource partitions in a cell-wise manner regarding defined DRL reward as the minimum QoS satisfaction level among all slices;
- **Traffic**: a traffic-aware approach that dynamically adapts slicing resource partitions in each cell proportionally to the current per-slice user traffic demands, which assumes perfect knowledge of the traffic amount;

We applied the IDLA approaches with the help of derived slice performance estimators, i.e., VIB-based and DNN-based models, that were derived with source domain data from the corresponding network environments. To evaluate the adaptation performance of the IDLA-VIB scheme against the others, we implemented an online assessment in the network simulator under dynamic network slicing configurations and various user groups, i.e., during the network processing, we changed network slicing combinations or user behaviors. For fair comparisons, we divided the whole online process into 3 time periods, denoted by H_1, H_2, H_3, respectively:

- $\mathcal{H}_1$ ($t \in [0, 1000)$): First, in time phase $\mathcal{H}_1$ we set 3 slices with combination $\mathcal{S}_c(t) := [1, 2, 4], t \in \mathcal{H}_1, c \in \mathcal{C}$ in network, with slice throughput requirements as $\phi^*_{\mathcal{H}_1} \in \{2.0, 1.0, 0.5\}$ Mbit/s respectively.
- $\mathcal{H}_2$ ($t \in [1000, 2000)$): At time step 2000, the network scenario enters phase $\mathcal{H}_2$, and we changed the network slicing configuration by introducing a new slice with index 3, i.e., the new slice combination becomes $\mathcal{S}_c(t) := [1, 2, 3, 4]$. The new slice throughput requirements become $\phi^*_{\mathcal{H}_2} \in \{2.0, 1.0, 1.5, 0.5\}$ Mbit/s.

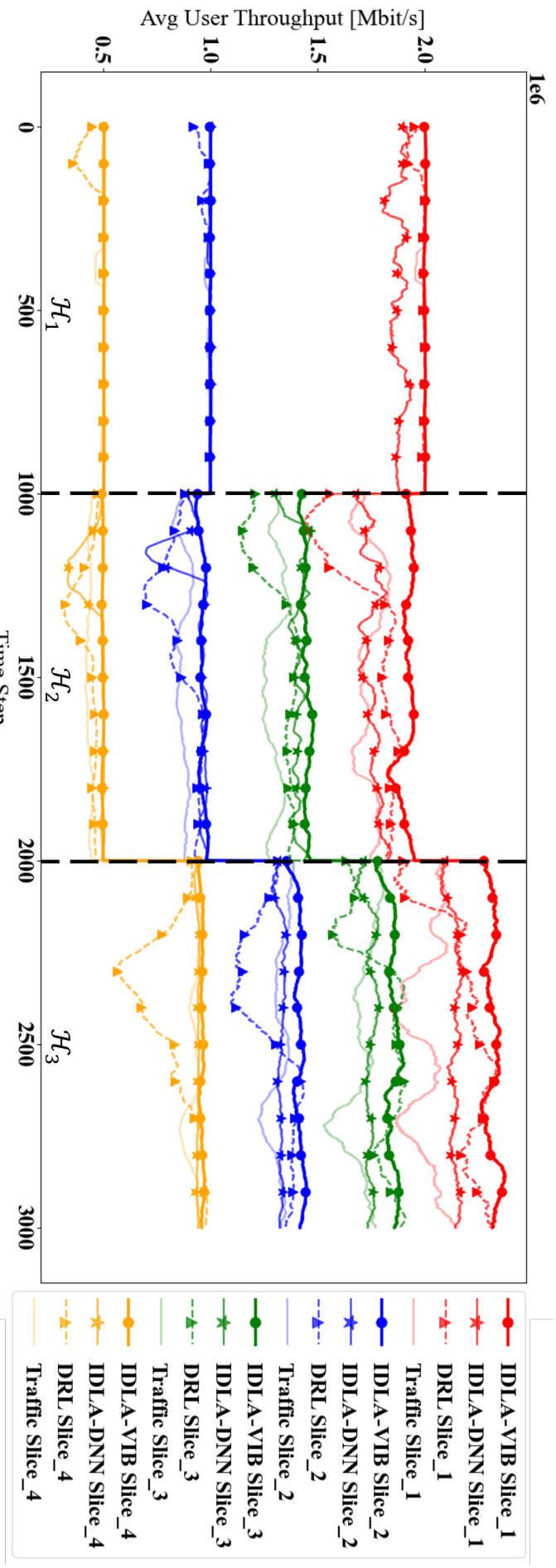

Figure 9.4: Comparison of online slice throughput in dynamic slicing

- $\mathcal{H}_3$ ($t \in [2000, 3000]$): We further changed the slicing configuration at the switching point between phase $\mathcal{H}_2$ and $\mathcal{H}_3$ by increasing the throughput requirement of each slice by 0.5 Mbit/s, i.e., the new slice requirements are $\phi^*_{\mathcal{H}_3} \in \{2.5, 1.5, 2.0, 1.0\}$ Mbit/s.

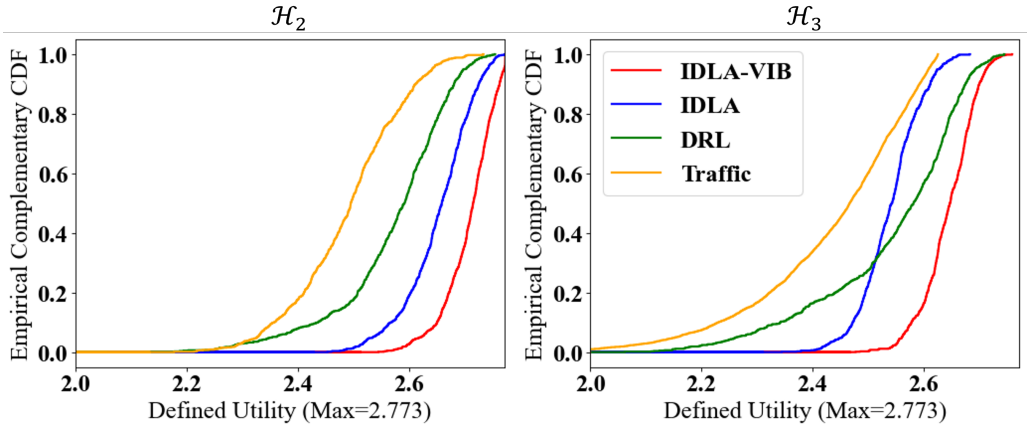

Figure 9.5: Comparing CDF of network utility

Table 9.1: Table of Source-Target Domain Pairs

	Source Domain	**Target Domain**	**Distance Metric**
Pair 1	$\phi^*_{\mathsf{S}} \in \{2.0, 1.5, 1.0, 0.5\}$, $U^{(\mathrm{R})}_{\mathsf{S}} = 500$, $U^{(\mathrm{N})}_{\mathsf{S}} \in \{16, 24\}$	$\phi^*_{\mathsf{T}} \in \{2.0, 1.5, 1.0, 0.5\}$, $U^{(\mathrm{R})}_{\mathsf{T}} = 500$, $U^{(\mathrm{N})}_{\mathsf{T}} = 36$	0.311
Pair 2	$\phi^*_{\mathsf{S}} \in \{2.0, 1.5, 1.0, 0.5\}$, $U^{(\mathrm{R})}_{\mathsf{S}} \in \{250, 360\}$, $U^{(\mathrm{N})}_{\mathsf{S}} = 24$	$\phi^*_{\mathsf{T}} \in \{2.0, 1.5, 1.0, 0.5\}$, $U^{(\mathrm{R})}_{\mathsf{T}} \in \{750, 900\}$, $U^{(\mathrm{N})}_{\mathsf{T}} = 24$	1.312
Pair 3	$\phi^*_{\mathsf{S}} \in \{2.0, 1.5, 1.0, 0.5\}$, $U^{(\mathrm{R})}_{\mathsf{S}} = 500$, $U^{(\mathrm{N})}_{\mathsf{S}} = 24$	$\phi^*_{\mathsf{T}} \in \{2.5, 2.0, 1.5, 1.0\}$, $U^{(\mathrm{R})}_{\mathsf{T}} = 500$, $U^{(\mathrm{N})}_{\mathsf{T}} = 24$	0.383
Pair 4	$\phi^*_{\mathsf{S}} \in \{2.0, 1.0, 0.5\}$, $U^{(\mathrm{R})}_{\mathsf{S}} = 500$, $U^{(\mathrm{N})}_{\mathsf{S}} = 24$	$\phi^*_{\mathsf{T}} \in \{2.0, 1.5, 1.0, 0.5\}$, $U^{(\mathrm{R})}_{\mathsf{T}} = 500$, $U^{(\mathrm{N})}_{\mathsf{T}} = 24$	0.958

To assess transferability, during this process, we kept collecting new samples from the network simulators and fine-tuned the estimators periodically every 200 steps for 100 epochs for **IDLA-VIB** and **IDLA** schemes, while the **DRL** fine-tunes itself through online interaction when entering a new phase with new slicing setups. For fairness comparison, the initial exploration phase of the **DRL** scheme was excluded at the beginning of $\mathcal{H}_1$. To simulate realistic user traffic patterns, we applied varying traffic masks $\tau_s(t) \in [0, 1]$ collected from a real network system to each slice $s \in \mathcal{S}_c(t)$, reflecting the daily periodic pattern of the per-slice user traffic, where each step represents 15 minutes in real time. Similarity, a traffic mask is applied to avoid overlapping of traffic peaks. On each slice, the traffic mask has 16 steps shift to the next.

Fig. 9.4 illustrates the trends in average throughput across all cells for each slice during phase $\mathcal{H}_1$, $\mathcal{H}_2$, and $\mathcal{H}_3$ by implementing different slicing resource allocation schemes. The **IDLA-VIB** and **IDLA** schemes demonstrate consistent throughput levels across all slices before encountering network condition changes, with **IDLA-VIB** offering slightly better and more robust performance. In contrast, the **DRL** and **Traffic** schemes exhibit more fluctuating performance, particularly after each configuration changes. At each network change point, there is a visible impact on throughput for all schemes. However, the IDLA-based schemes show better adaptability and quickly stabilize within 300 steps, whereas **IDLA-VIB** exhibits faster convergence with online fine-tuning. The **DRL** scheme, which is tailored to specific cases, requires a period of exploration and training after each con-

figuration change, shows fluctuating performance after the change, and tends to converge after 600 steps. Although the **Traffic**-aware baseline initially knows traffic demands well, its adaptability varies. Across the slices, the **IDLA-VIB** scheme consistently outperforms the others in maintaining higher throughput levels. This underscores the VIB-based estimator robustness and efficiency in optimizing slicing resource allocation under dynamic conditions. Overall, the **IDLA-VIB** scheme outperforms the other schemes, demonstrating its capability to effectively manage and optimize resource allocation for varying slice requirements.

In Fig. 9.5, we compare the convergence performance of each scheme referring to defined network utility Eq. 9.5 during phase $\mathcal{H}_2$ and $\mathcal{H}_3$, which are regarded as target domains in this comparison. The network utility Eq. 9.5 is taken by taking the logistic of minimum among slice throughput and delay satisfaction level, with a maximum defined utility of 2.773. In general, the **IDLA-VIB** scheme exhibits the highest utility across the phases, closely followed by the **IDLA** scheme. In phase $\mathcal{H}_2$, the **IDLA-VIB** scheme achieves a 53.56% ratio of 95% utility, compared to **IDLA** and **DRL**, which only reach 38.22% and 33.38%, respectively. Similarly, in phase $\mathcal{H}_3$, **IDLA-VIB** continues to lead, achieving the highest ratio at 41.28%. This is in contrast to 30.68% with **DRL** and 10.06% with **IDLA**. The superior performance of the **IDLA-VIB** scheme, in particular, underscores its robustness and adaptability in optimizing network performance under slicing configuration changes. This comparison shows that the IDLA algorithm can provide scalable and fast deployable resource slicing resource allocation solutions, and the high transferability and adaptivity of VIB models can further expand its efficiency.

9.4.4 Evaluation of VIB-based QoS Estimator

To assess the models' adaptability to samples across domains, we set up 4 domain pairs with variant network living setups under the experimental network scenario. Specifically, our defined categories of domain setups are summarized as follows:

- **Pair 1: Different maximum number of users.** In the source domain, the maximum number of users ranges between $U_{\mathsf{S}}^{(\mathrm{N})} \in \{16, 24\}$, whereas in the target domain, this expands to a larger group with $U_{\mathsf{T}}^{(\mathrm{N})} = 36$;
- **Pair 2: Different user moving radius.** In the source domain, users move within a radius of $U_{\mathsf{S}}^{(\mathrm{R})} \in \{250, 360\}$ meters, compared to the target domain, where the moving radius increases to $U_{\mathsf{T}}^{(\mathrm{R})} \in \{750, 900\}$ meters, indicating a significantly wider range of movement;
- **Pair 3: Different slice combinations**. The source domain features slice types with throughput requirements of $\phi_{\mathsf{S}}^{*} \in \{2.0, 1.5, 1.0, 0.5\}$ Mbit/s. In contrast, the target domain requires higher throughputs for each slice, with values at $\phi_{\mathsf{T}}^{*} \in \{2.5, 2.0, 1.5, 1.0\}$ Mbit/s, making an increase of 0.5 Mbit/s per slice.
- **Pair 4: Different number of slices**. The source domain includes 4 slice types with throughput requirements of $\phi_{\mathsf{S}}^{*} \in \{2.0, 1.0, 0.5\}$ Mbit/s. The target domain expands this set by introducing an additional slice, making the slice combination $\phi_{\mathsf{T}}^{*} \in \{2.0, 1.5, 1.0, 0.5\}$ Mbit/s.

Table 9.1 specifies these domain configurations pairwisely. In the final column of Table 9.1, we provide the domain discrepancy measures derived with Eq. 7.7, higher metric values indicate greater domain differences, notably in pairs 3 and 4, attributed to varying slice types despite similar user counts and CQI distributions.

To demonstrate the domain difference, Fig. 9.3 illustrates a comparison of sample distributions of four key parameters - actual load, number of users, and CQI - between the source and target domains for each evaluated domain pair. The distribution comparisons highlight distinct differences, such as the number of users in pair 1 and CQI in pair 2, which also lead to variations in per-slice resource occupations as depicted by the actual load. However, for some domain pairs, these differences are not as apparent from the sample distribution.

9.4.4.1 Comparison of DG Ability

To investigate the DG ability of models, we first trained these estimators exclusively on source domain samples following the procedures outlined in Section 9.4.2.1 and subsequently evaluated their accuracy on both source and target domains.

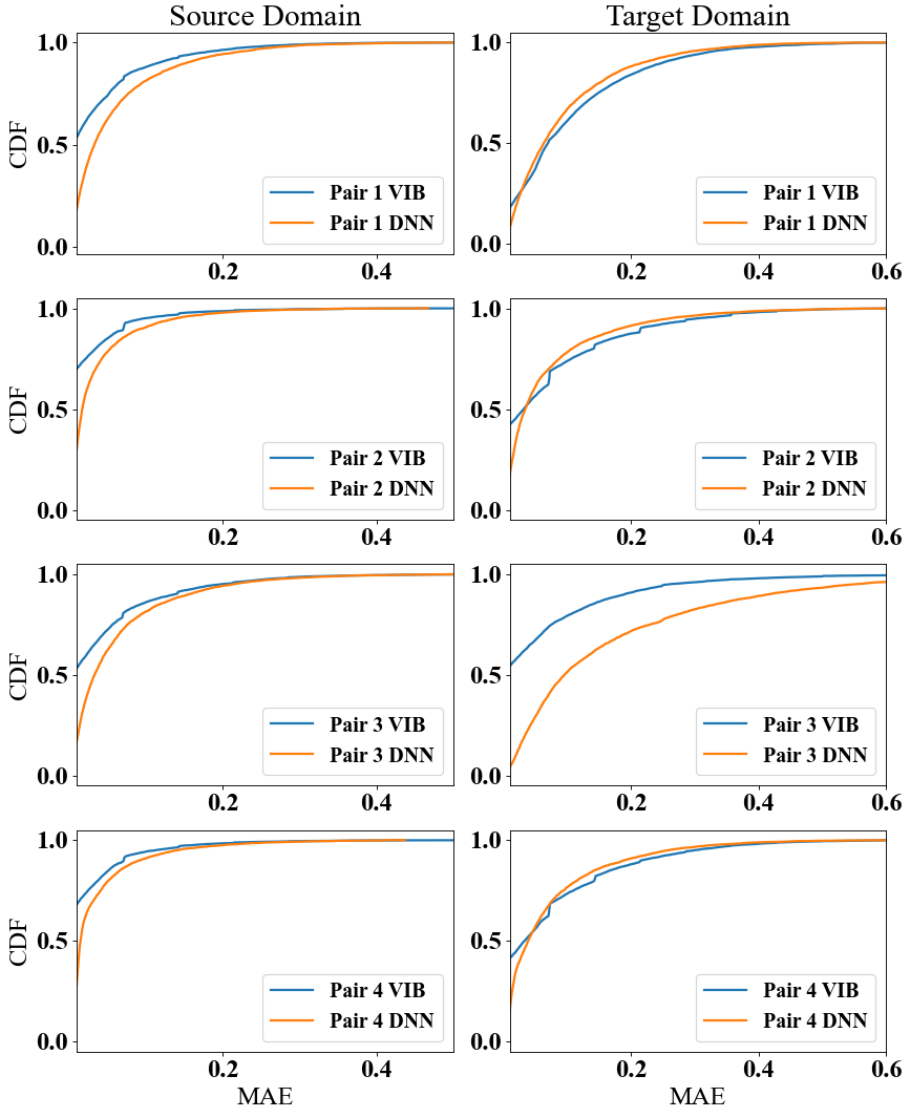

Figure 9.6: Compare MAE CDF on source and target domains

Fig. 9.6 presents the distributions of MAE for both VIB and DNN models across all domain pairs, comparing the accuracy on source and target domains. Each subfigure corresponds to a specific domain pair and model type. As CDFs illustrate, the VIB models consistently achieve lower MAE values in source domains compared to DNN models, while in target domains, the performance of both models is comparable except domain pair 3. Additionally, both models exhibit varying degrees of performance degradation in target domains in comparison with their source domains. However, the VIB model demonstrates a better ability to balance MAE across domains, indicating superior estimation accuracy and

enhanced DG and robustness capabilities. Specifically, in the comparisons of domain pair 1 and 2, the MAE distributions for both source and target domains are distributed towards low errors. In contrast, for pairs 3 and 4, the target domains show much worse and broader MAE distributions, reflecting greater variability in estimation accuracy compared to source domains. The metrics in Table corroborate this 9.1, which indicate that the differences between source and target domains in pair 3 and 4 are more significant, consistent with the observed decline in QoS estimator performance demonstrated in Fig. 9.6.

9.4.4.2 Comparison of DA Ability

In response to the diminished performance in target domains with substantial domain gaps identified in DG scenarios, we further explored the models' DA capabilities by incorporating varying proportions of limited target domain samples in the training data. Specifically, we implemented the training process of estimators under the DA scenarios, utilizing target domain sample inclusion ratio of $[0\%, 20\%, 50\%, 70\%, 90\%]$. It is important to note that even with a 90% inclusion rate of target samples, the aggregate proportion of target domain data within the training set remains significantly lower than the portion of the source domain data, which reflects the basic assumption of DA scenarios. Table 9.2 demonstrates the estimation error in MAE of models in comparison under different usage ratios of target domain samples under all domain pairs.

As Table 9.2 illustrates, VIB models outperform DNN estimators in all domain pairs, demonstrating notable reductions in estimation errors as the target sample ratio increases. Particularly in pairs 2 and 4, where the VIB model presents remarkable adaptability, significantly narrowing the performance loss from source to target domains, while the DNN model exhibited lesser flexibility and adaptability across varying domain conditions. When integrated with the LDS method, the DNN model showed improved performance, particularly in managing data imbalance, indicating the effectiveness of LDS in adaptability. However, even with the enhancement provided by LDS, conventional models still exhibit lower estimation accuracy compared to VIB. Specifically, in pair 1, which has the smallest domain discrepancy, the VIB model offers improvements ranging from 18.02% to 35.84% under varying inclusion rates of target samples. This is compared to the 14.33% to 18.17% improvement range provided by the LDS model. This improvement becomes even more evident as domain discrepancy increases. The last column of each sub-table showcases the baseline estimation accuracy achieved by applying models exclusively on target domain samples. The estimation errors of both VIB and DNN models suggest that including target domain samples may slightly impact their performance on source domains. On the contrary, with the LDS method, the models' accuracy in the source domain is not adversely affected by including fewer samples. It even achieves higher accuracy in source domains with the re-weighted loss. In Fig. 9.7, we visualize the models' estimation errors in target domains across various target sample ratios, where shadowed solid lines depict the baselines for each model. The comparison focuses on the impact of incorporating varying ratios of target domain samples into the training process under the DA scenarios.

In domain pairs 1 and 3 with narrower domain discrepancies, the VIB based estimator enhances target domain accuracy as the ratio of target sample inclusion increases, stabilizing around 70% inclusion rate in pair 1 and also showing marked improvements in pair 3. While in pairs 2 and 4, which have wider domain gaps, the VIB model exhibits even more significant reductions in target domain MAE, suggesting robust adaptability in response to increased proportions of target domain samples. The DNN model, across all domain pairs, shows improvements as well, but these are more restrained compared to those of the VIB model, and it does not achieve as low an MAE as the VIB model does. Additionally, the LDS-enhanced DNN model improves upon the basic DNN model performance, but it does

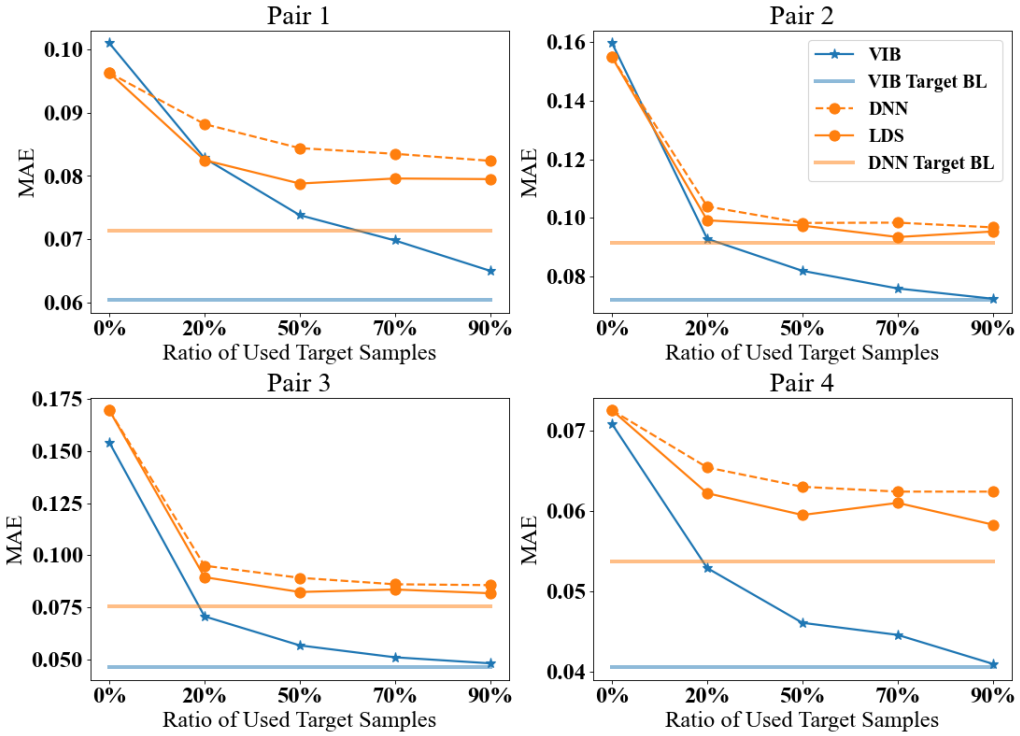

Figure 9.7: Compare MAE with involvement of target samples

not consistently reach the enhanced performance levels of the VIB model, particularly in domain pairs with wider gaps.

Overall, incorporating target domain samples into the training significantly improved the VIB model accuracy in the target domain, with a little compromising of source domain performance in the cases of small domain discrepancies, while the DNN model does not exhibit clear enhancements from the inclusion of target domain samples. The VIB model consistently demonstrates superior generalization to the target domain across all pairs under both DG and DA contexts, particularly with higher ratios of target samples. The numerical results of estimation performance underscore VIB model superior adaptability and robustness relative to the DNN model.

9.5 Key Takeaways

In this chapter, we present a comprehensive framework that merges the robust generalization capabilities of the Lagrangian method with the approximation potential of deep learning. Our proposed IDLA algorithm addresses the resource partitioning challenges inherent in network slicing while considering inter-slice resource constraints. The empirical results from a system-level network simulator underscore the efficacy of our approach, demonstrating superior transferability, faster convergence, and better scalability compared to state-of-the-art solutions.

Implementing the IDLA algorithm in real-time network slicing scenarios revealed its high scalability in resource partitioning. When compared to other approaches, such as DRL methods and traffic-aware allocation mechanisms, the IDLA algorithm, especially when combined with VIB-based estimators, provided significant improvements in managing slicing resources dynamically and efficiently. It achieved twice the convergence speed, showcasing its potential for rapid and efficient deployment in diverse network environments. Furthermore, the evaluation within dynamically changing network environments highlighted the adaptability of the IDLA-VIB scheme. Despite the challenges posed by varying network conditions and slicing configurations, the VIB-based approach consistently outperformed

other models, achieving higher throughput levels across different slices with the highest ratios of utility satisfaction. This demonstrated its robustness in optimizing slicing resource allocation under dynamic conditions.

The transferability of the VIB-based model was further emphasized when deployed in dynamically changing network settings. The model exhibited higher adaptability compared to the original IDLA approach without TL, underscoring the flexibility of VIB in optimizing resource allocation under varying slice configurations. Additionally, the VIB-aided slice-wise network QoS estimators demonstrated enhanced DA capabilities compared to traditional DNN-based models. This was evident under various TL tasks, where the VIB-based models showed lower MAE values across both source and target domains, indicating improved estimation accuracy and robust DA performance.

In conclusion, our proposed IDLA algorithm, enhanced with TL and DA methods, provides a highly scalable and transferable solution for network slicing. By ensuring near-optimal performance, fast convergence, and high generality, the IDLA algorithm with the VIB-based DA model sets a new standard for slicing resource partitioning. This approach paves the way for its scalable and generalized application in real-world network systems, highlighting its potential for effective deployment in dynamic and diverse network environments.

Table 9.2: Comparison of Estimator Performance as Slice QoS Estimation Error

(a): Estimation Error in Domain Pair 1 (in 0.01)

Model	Domain	Target Sample Ratio (in %)					Target Only
		0	20	50	70	90	
VIB	$\mathcal{D}_S$	3.73	3.76	3.96	4.10	4.20	—
	$\mathcal{D}_T$	10.10	**8.28**	**7.38**	**6.98**	**6.48**	**6.04**
DNN	$\mathcal{D}_S$	6.10	6.36	6.50	6.49	6.50	—
	$\mathcal{D}_T$	**9.63**	8.82	8.44	8.35	8.24	7.13
LDS	$\mathcal{D}_S$	—	6.67	6.33	6.28	6.29	—
	$\mathcal{D}_T$	—	8.25	7.88	7.96	7.95	—

(b): Estimation Error in Domain Pair 2 (in 0.01)

Model	Domain	Target Sample Ratio (in %)					Target Only
		0	20	50	70	90	
VIB	$\mathcal{D}_S$	1.44	1.63	1.58	1.68	1.75	—
	$\mathcal{D}_T$	15.95	**9.29**	**8.20**	**7.60**	**7.25**	**7.21**
DNN	$\mathcal{D}_S$	3.06	3.13	3.11	3.17	3.19	—
	$\mathcal{D}_T$	**15.74**	10.39	9.83	9.84	9.68	9.16
LDS	$\mathcal{D}_S$	—	3.30	3.16	3.22	3.22	—
	$\mathcal{D}_T$	—	9.92	9.74	9.35	9.54	—

(c): Estimation Error in Domain Pair 3 (in 0.01)

Model	Domain	Target Sample Ratio (in %)					Target Only
		0	20	50	70	90	
VIB	$\mathcal{D}_S$	3.79	4.05	4.20	4.42	4.53	—
	$\mathcal{D}_T$	**15.41**	**7.07**	**5.68**	**5.11**	**4.82**	**4.64**
DNN	$\mathcal{D}_S$	6.05	6.18	6.23	6.31	6.41	—
	$\mathcal{D}_T$	16.95	9.50	8.92	8.61	8.57	7.57
LDS	$\mathcal{D}_S$	—	6.58	6.41	6.29	6.30	—
	$\mathcal{D}_T$	—	8.95	8.24	8.36	8.18	—

(d): Estimation Error in Domain Pair 4 (in 0.01)

Model	Domain	Target Sample Ratio (in %)					Target Only
		0	20	50	70	90	
VIB	$\mathcal{D}_S$	1.88	2.08	2.19	2.23	2.33	—
	$\mathcal{D}_T$	**7.08**	**5.29**	**4.61**	**4.46**	**4.10**	**4.06**
DNN	$\mathcal{D}_S$	3.78	3.86	3.97	3.93	3.96	—
	$\mathcal{D}_T$	7.25	6.54	6.30	6.24	6.24	5.37
LDS	$\mathcal{D}_S$	—	4.03	3.92	3.99	3.82	—
	$\mathcal{D}_T$	—	6.22	5.95	6.10	5.83	—

10. Conclusion and Outlook

Mobile networks are essential for facilitating connectivity in all human interactions, whether between individuals or devices, underscoring their vital role within wireless communication. As we move towards future communication networks, sophisticated algorithms will be required to automate complex cognitive processes currently managed by human reasoning. This thesis has focused on developing and applying advanced DRL and TL techniques to address the challenges of network resource allocation in dynamic and heterogeneous environments. Through comprehensive research, experimentation, and evaluation, several key insights and contributions have been made that will shape the future of network automation.

Specifically, we have explored DRL and TL methods within the context of network slicing, addressing the optimization problems associated with slice resource allocation. The discussions were divided into 3 main topics: ***RL application in network slicing***, ***integration of TL***, and ***comparison of solution granularity***. For the third topic, we investigated 3 levels of solution granularity: a centralized scheme that handles resource partitioning across the entire network, a per-cell distributed scheme that decomposes the central solver into cell-level models with lower complexity, and finally, a per-slice scheme that offers the highest scalability and flexibility, theoretically serving as a "one-size-fits-all" solution. Within each distributed scheme, we examined RL-based solutions and the integration of TL methods.

Each chapter concluded with a summary based on numerical assessments from experiments, demonstrating the effectiveness of the proposed solutions within their respective contexts. The proposed algorithms were rigorously tested using system-level network simulators, validating their practical efficiency and scalability. The results highlighted the superiority of TL-aided RL and IDLA algorithms in dynamically and efficiently managing slicing resources. These findings pave the way for real-world deployment, showcasing the potential for rapid and effective network automation.

In this final chapter, we will assess and evaluate the applications of RL and TL, highlighting their efficacy in solving resource allocation tasks within network slicing scenarios. We will also discuss their limitations in certain use cases and suggest future research directions based on the findings of this dissertation.

10.1 RL Application in Network Slicing

This dissertation has focused on the application of RL techniques to enhance network slicing in dynamic communication networks. In this initial Chapter 2, we provided a comprehensive introduction to RL, covering fundamental concepts and advanced algorithms such as DQN, DDPG, and TD3. These algorithms were evaluated for their potential to optimize network performance autonomously, especially in the context of 5G and future networks. The research demonstrated that RL can significantly improve network performance by dynamically adjusting configurations in response to changing conditions.

To address the optimization of slice resource allocation, we explored both centralized and distributed RL approaches. Initially, we formulated the slice resource allocation problem using MDP models in multi-cell, multi-slice scenarios. The centralized approach served as a benchmark for comparison. To validate the effectiveness of RL in network slicing, we proposed a distributed DQN-based resource allocation solution. Preliminary experiments showed promising results, demonstrating the practical implementation and benefits of RL in a simulated network environment. The distributed approach aimed to address the complexity and scalability issues inherent in centralized solutions. Further extending the distributed approach, we explored multi-agent DRL algorithms to enhance inter-cell and inter-slice resource partitioning. This approach aimed to maximize slice performance while adhering to resource constraints.

Our key findings and contributions to exploring multi-agent DRL application in network slicing include:

1. **Distributed Multi-Agent DRL Solution:** We proposed a distributed multi-agent DRL framework, exploring schemes with and without inter-agent coordination. This approach allowed agents to share load information, enhancing overall performance.

2. **Handling Resource Constraints:** To respect resource constraints, we developed two methods: reward reshaping and decoupled softmax layer embedding. These methods ensured that the agents operated within the defined resource limits.

3. **Performance Evaluation:** Using a system-level simulator, we extensively evaluated the proposed solutions. The coordinated distributed scheme outperformed the centralized approach, providing superior slice-aware service performance and more than doubling resource efficiency compared to traffic-aware baselines.

4. **Effectiveness of RL in Network Slicing:** RL, particularly in distributed and multi-agent settings, effectively addresses the dynamic and complex nature of network slicing, offering significant improvements in resource management.

5. **Practical Implementation:** The proposed algorithms were rigorously tested in simulated environments, validating their practical applicability and scalability in real-world network scenarios.

6. **Enhanced Performance:** The integration of TL techniques further boosted the performance of DRL algorithms, demonstrating faster convergence rates, improved initial and final performance, and enhanced robustness.

In conclusion, the application of RL in network slicing presents a promising avenue for achieving efficient and autonomous network management. The contributions of this dissertation lay a strong foundation for future advancements in network automation, paving the way for more adaptive, scalable, and robust communication networks.

10.2 Integration of TL

This dissertation has explored the integration of TL with RL to enhance the sample efficiency and model reproducibility of network slicing solutions. By leveraging TL, we aimed to overcome the limitations of RL models, such as sample inefficiency and lack of generality, to provide more robust and scalable network management solutions. We began with a detailed introduction to TL, covering its background, basic concepts, and specific methods, including the pre-train and fine-tuning framework and DA techniques. The necessity of TL in ML and practical applications was emphasized, highlighting how TL can significantly improve the learning rate and initial performance of RL agents by leveraging pre-trained models and knowledge from similar tasks.

In our research, we integrated TL into the distributed DRL approach, specifically focusing on the methods of transferring the per-cell DRL-based slice resource allocation method, denoted as the DIRP algorithm, under two TL scenarios: Generalist-to-Specialist and Specialist-to-Specialist. This integration aimed to enhance the performance of RL in dynamic network environments by addressing inter-slice resource constraints and optimizing inter-cell resource partitioning. Building on the IDLA framework, we further investigated TL methods to enhance model reproducibility and sample efficiency. We introduced a VIB-based regression model to improve the generality of the slice-wise estimator across different network environments. Our contributions and findings are summarized as follows:

Generalist-to-Specialist with TL-aided DIRP

1. **Improved Sample Efficiency and Performance:** The TL-aided DIRP algorithm demonstrated higher rewards, faster convergence, and better performance in diverse network scenarios compared to baseline approaches.

2. **Enhanced Algorithm Transferability:** By using pre-trained model transfer, instance transfer, and combined model and instance transfer schemes, we significantly improved the transferability of the DIRP algorithm across different network environments.

3. **Experiment Evaluation:** The proposed solutions were evaluated using a system-level simulator, showing superior slice-aware service performance and increased resource efficiency over traffic-aware baselines.

Specialist-to-Specialist TL with Domain Similarity Analysis

1. **Domain Similarity Analysis:** We further explored TL-aided DRL approaches that focus on transferring knowledge from one specific domain to another. We developed a novel TL-aided multi-agent DRL approach with domain similarity analysis for inter-slice resource partitioning. Using a feature-based inter-agent similarity analysis approach, this method measures domain and task differences.

2. **Specialist-to-Specialist Transfer:** We achieved greater performance gains by focusing on transferring knowledge between highly similar agents, particularly in scenarios with limited training steps.

TL for Per-slice Resource Allocation

1. **Enhanced Adaptability:** The VIB-based model adapted to a broader range of network scenarios, significantly improving the accuracy of slicing resource partitions.

2. **Mitigating Domain Discrepancies:** The VIB approach effectively mitigated domain discrepancies, outperforming conventional MLP regression models and domain sample re-weighting methods.

3. **Practical Efficiency:** Implemented in a system-level network simulator, the TL-aided IDLA algorithm demonstrated high flexibility and scalability in dynamic network slicing scenarios, outperforming traditional DRL and IDLA approaches without TL.

In conclusion, the integration of TL into RL-based network slicing solutions and the IDLA algorithm has demonstrated significant improvements in performance, efficiency, and adaptability. Key findings include:

- **Enhanced Performance and Convergence:** TL techniques have significantly improved the learning rate and overall performance of RL agents, resulting in faster convergence and higher rewards.

- **Scalability and Flexibility:** The proposed TL-aided approaches, particularly the Variational Information Bottleneck (VIB)-based IDLA algorithm, have showcased superior scalability and flexibility in managing dynamic slicing resources.

- **Robustness in Diverse Scenarios:** TL methods have enhanced the robustness of RL models across various network environments, ensuring consistent performance even with domain discrepancies.

These advancements highlight the significant promise of integrating TL into RL-based slice resource allocation methods. The improvements in efficiency, scalability, and robustness lay a strong foundation for future research and practical implementations in dynamic and heterogeneous network environments.

10.3 Comparison of Solution Granularity

10.3.1 Centralized vs. Distributed DRL Approaches

One of the core aspects of this research was the comparison between centralized and distributed DRL approaches for network slicing. While effective in controlled environments, centralized methods faced significant scalability and convergence challenges when applied to large-scale networks. In contrast, distributed DRL approaches, particularly those incorporating inter-agent coordination mechanisms, demonstrated superior performance in terms of convergence rates, resource efficiency, and overall service quality.

We proposed a distributed multi-agent DRL solution to enhance resource allocation in network slicing. By exploring two coordination schemes—one with inter-agent coordination and one without—we enabled agents to share local information, improving convergence rates and service quality. We developed two methods to ensure that DRL agents respect resource constraints: reward reshaping and decoupled softmax embedding. These methods allowed for more effective management of inter-slice resource constraints. The proposed solutions were rigorously tested using a system-level simulator, showing that the coordinated distributed scheme more than doubled resource efficiency over traffic-aware baselines while providing superior slice-aware service performance compared to the centralized approach.

10.3.2 Per-slice Resource Allocation with IDLA

We developed the IDLA algorithm to further enhance resource allocation. This approach combines deep learning models with constrained optimization techniques, enabling robust and scalable resource allocation for various slice combinations. The IDLA algorithm addresses the limitations of traditional RL methods, offering a practical solution for dynamic network environments.

A general DNN model was derived to approximate slice network utility, capable of handling slices with different requirements. This model facilitated efficient computation of the partial derivatives of the slice utility function. Leveraging the DNN model, a Lagrangian method was designed for optimal per-slice resource allocation while adhering to inter-slice resource constraints. This approach ensures near-optimal QoS satisfaction and generalizes well across various network scenarios. The slice-wise resource scheduler in the IDLA algorithm demonstrated high scalability and flexibility, enabling fast and efficient deployment in real network systems. Experimental results showed that the IDLA algorithm achieved near-optimal performance with fast convergence compared to state-of-the-art solutions.

Summary of Findings
The research provided several significant insights into network slicing solutions:

1. **Scalability and Efficiency:** Distributed DRL approaches, particularly with inter-agent coordination, are more scalable and efficient than centralized methods, making them suitable for large-scale network environments.

2. **Advanced Resource Allocation:** The IDLA algorithm combination of deep learning and constrained optimization techniques offers a robust and flexible solution for dynamic network conditions, outperforming traditional RL methods.

3. **Enhanced Performance through TL:** Integrating TL techniques into DRL models significantly improved sample efficiency, learning rates, and overall performance, demonstrating the potential for rapid and effective network automation.

This dissertation contributes to a thorough discussion of effectively managing network resources in dynamic and heterogeneous environments by investigating these various granularities of slice resource allocation. Combining with the advanced RL and TL techniques, the findings lay a strong foundation for future network slicing and automation advancements for more adaptive, scalable, and robust communication networks. The framework of the potential application of this dissertation is provided in Fig. 10.1. It showcases cells C_1 to C_N, which collectively form the source domain $\mathcal{D}_S$, each populated with various network slices. These slices are transferred to a target domain represented by cell C_T, highlighting a process where learned strategies and network management solutions are adapted from source to target, facilitating quick deployment and resource optimization. The figure also indicates a centralized knowledge data repository that gathers data from current network systems, which is essential for TL by providing a rich dataset for training and refining RL models. TL schemes like per-cell (DIRP) and per-slice (IDLA) indicate strategic approaches to applying TL, either focusing on entire cells or individual slices, suggesting flexibility in optimization strategies. The implementation in new network systems shown at the bottom part of the diagram implies that these transferred strategies enhance performance and efficiency, particularly useful in dynamic network environments where rapid adaptation to changing conditions is crucial. This approach not only accelerates the deployment of network resources but also significantly improves the adaptability and efficiency of network automation solutions.

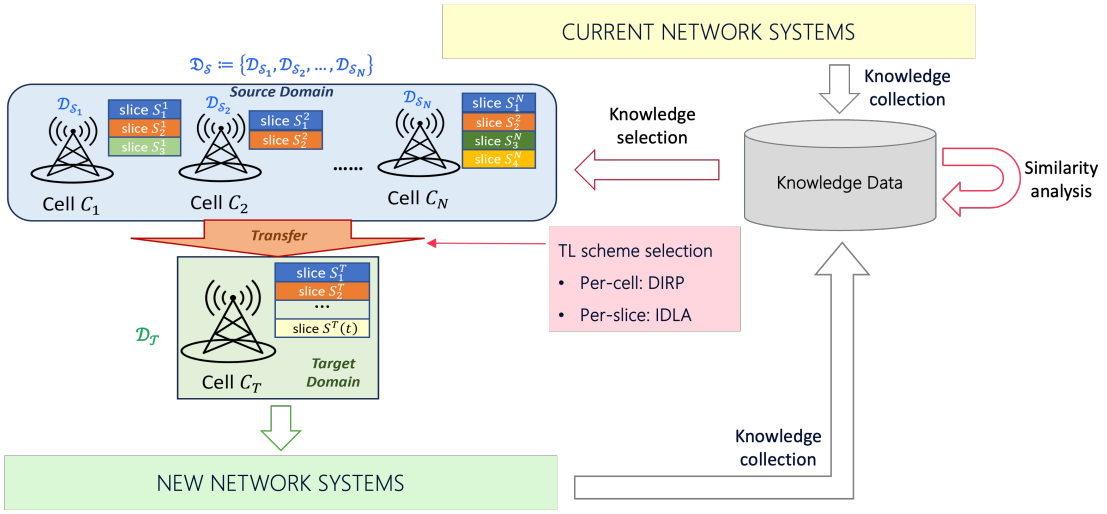

Figure 10.1: Knowledge and data services for network automation

10.4 Answers to Research Questions

Answer to Question 1: How to convert slice resource allocation into the form that RL approaches can handle?

To convert slice resource allocation into a form suitable for RL, we modeled the problem as a MDP. This involves defining states, actions, and rewards that accurately represent the dynamic environment of network slicing. The state space includes variables such as current resource allocation, network traffic, and slice performance metrics. Actions correspond to possible adjustments in resource distribution among slices, and the reward function is designed to reflect KPIs like throughput, latency, and resource efficiency. This MDP formulation enables the application of RL algorithms to optimize slice resource allocation dynamically.

Answer to Question 2: How to distribute the centralized resource allocation problem into the per-cell scheme while maintaining the correlation of inter-cell dependency?

To distribute the centralized resource allocation problem into a per-cell scheme while preserving inter-cell dependencies, we introduced a distributed multi-agent DRL approach. Each cell operates as an independent agent that collaborates with neighboring cells. Inter-agent coordination is achieved through mechanisms such as information sharing and coordination protocols, which help maintain inter-cell dependency. Techniques like reward shaping and decoupled softmax embedding were developed to ensure resource constraints are respected across the network. This distributed approach allows for scalable and efficient resource allocation in large-scale networks.

Answer to Question 3: How to proceed TL methods on the top of RL solutions?

We employed a two-phase approach: pre-training and fine-tuning to integrate TL methods with RL solutions. Initially, RL models are trained on a source domain with ample data to develop a robust base policy. This pre-trained model is then fine-tuned on the target domain with limited data, allowing it to adapt to specific network conditions. We explored different TL techniques such as model transfer, instance transfer, and combined transfer methods. When applied to new or varying network scenarios, this process enhances the learning rate, sample efficiency, and initial performance of RL agents.

Answer to Question 4: How to determine the source of knowledge transfer given a target slicing scenario?

Determining the source of knowledge transfer involves analyzing the similarity between the source and target domains. We developed a domain similarity analysis using feature-based inter-agent similarity measures. This method evaluates potential source domains' characteristics and performance metrics to identify the most relevant and similar one for transfer. By extracting representative feature distributions in latent space, we can effectively match target slicing scenarios with the best-suited pre-trained models, ensuring efficient and effective knowledge transfer.

Answer to Question 5: Is TL helpful in enhancing the generality of RL solutions?

TL is significantly helpful in enhancing the generality of RL solutions. Our research demonstrated that TL techniques improve the adaptability of RL models to diverse network environments. By leveraging pre-trained models, TL reduces the need for extensive training data and computational resources in the target domain. This approach results in faster convergence rates, higher initial performance, and greater robustness against domain discrepancies, thereby extending the applicability and effectiveness of RL solutions across different network scenarios.

Answer to Question 6: How to address the resource constraints under different granularity of solutions?

We developed specific methods tailored to each granularity level to address resource constraints under different granularities. We used comprehensive resource allocation algorithms for centralized schemes that consider the entire network resource demands and constraints. For per-cell distributed schemes, we implemented reward shaping and decoupled softmax embedding to manage resources at a local level while maintaining network-wide constraints. We combined deep learning models with constrained optimization techniques in per-slice schemes to ensure efficient and compliant resource distribution. These methods collectively ensure optimal resource management across varying granularity levels.

Answer to Question 7: How to facilitate slicing solutions under dynamic slice configurations?

Facilitating slicing solutions under dynamic configurations involves developing adaptable algorithms capable of real-time adjustments. We proposed the IDLA algorithm, which combines deep learning with constrained optimization to manage varying slice configurations. The IDLA algorithm's flexibility allows it to handle different numbers of slices and configuration changes efficiently. This approach ensures consistent and optimal resource allocation in dynamic environments by continuously learning and adapting to new conditions.

Answer to Question 8: How to improve model generality under domain shift caused by slice configuration changing?

Improving model generality under domain shifts caused by slice configuration changes was achieved by incorporating advanced TL techniques, specifically domain adaptation methods. We introduced a VIB-based regression model to enhance the generality of slice-wise estimators. This model focuses on invariant feature extraction and robust performance estimation across different domains. By mitigating domain discrepancies through VIB, we ensured that models remain accurate and reliable despite changes in slice configurations, thereby maintaining high performance across diverse network scenarios.

10.5 Outlook to Future Research

Building on the numerical results from the integration of RL and TL methods for network slicing, there are several potential experiments for future research to further enhance and refine these approaches. This chapter outlines potential research directions based on the insights and findings of this dissertation.

Advanced Reward Function Analysis One significant area for future research is the analysis of different reward functions. The current study highlighted the effectiveness of using downlink average throughput as a KPI. However, future research should explore a variety of reward functions, including user-specific QoS requirements, to optimize RL performance in network slicing. This could involve developing more sophisticated reward structures that capture the nuances of user experiences and network performance metrics.

Incorporating Realistic Network Traffic Models To create more accurate and adaptable RL models, future experiments should incorporate dynamic changes in the number of UE based on realistic network traffic models. This approach would provide a more authentic simulation environment, allowing RL algorithms to adapt to varying traffic conditions and improve their generalizability and robustness.

Enhancing Multi-Agent Coordination Future work should focus on improving multi-agent coordination mechanisms to better capture dependencies between multiple cells and achieve consensus on RAN slice resource budget decisions among neighboring cells. This involves developing advanced coordination strategies that enhance the efficiency and effectiveness of distributed RL approaches in complex network environments.

Expanding TL Applications The long-term goal is to expand the use of TL to enhance sample efficiency and learning rates by transferring knowledge across different network environments. Future research should explore TL applications in other network management tasks, such as load balancing and interference management, to significantly improve network automation capabilities and make the optimization process more effective and adaptable. Additionally, further exploration of advanced DA methods is crucial to enhancing the generality and adaptability of RL models across diverse network scenarios. Implementing sophisticated DA techniques will help mitigate domain discrepancies, ensuring consistent performance of RL models in various network environments.

Real-World Validation and Implementation Finally, implementing and validating the proposed TL-aided RL approaches in real-world network environments is essential to ensure their practical applicability and effectiveness. This involves rigorous testing and adaptation of the algorithms in live network settings, providing valuable insights into their operational performance and scalability.

10.6 Conclusion

This dissertation has advanced the state-of-the-art network slicing and resource allocation through innovative applications of RL and TL. The proposed solutions offer practical, scalable, and efficient approaches for managing the complexities of next-generation networks. The findings and methodologies developed here provide a strong foundation for future research and development.

The methodologies developed in this thesis offer scalable solutions for network slicing, which is essential for managing future networks' increasing complexity. By reducing operational costs and complexity, these approaches facilitate more autonomous and self-optimizing networks, critical for supporting diverse applications and services. The proposed algorithms ensure efficient resource allocation and adherence to QoS requirements,

supporting a wide range of applications with varying performance needs. This capability enables advanced services such as autonomous driving, remote surgery, and immersive AR/VR experiences.

The integration of TL techniques ensures that developed models can quickly adapt to new network conditions and configurations, maintaining high performance despite rapidly changing traffic patterns and emerging technologies. The insights and methodologies from this research lay a strong foundation for future communication systems, such as 6G. The principles of RL, TL, and dynamic resource allocation explored in this thesis will be critical for managing next-generation networks' increased complexity and demands.

Furthermore, the successful application of TL and RL highlights their potential for broader use in various aspects of network management, including fault detection, traffic prediction, and energy-efficient operations. Future research can extend these techniques to other domains within the communication network ecosystem. Developing policy transfer techniques within TL-aided RL frameworks opens new avenues for creating more generalized and adaptable network management solutions. These techniques can facilitate the transfer of learned policies across different network segments, enhancing the overall efficiency and performance of network management.

The proposed algorithms, particularly those enhanced with TL and domain adaptation methods, have demonstrated their potential for real-time network optimization. This capability is crucial for ensuring that next-generation networks can meet the stringent demands of emerging applications and services. By continuing to refine and expand these advanced methods, we aim to develop more robust and efficient solutions for dynamic network environments, paving the way for more adaptive, scalable, and efficient communication networks.

In conclusion, the integration of TL into RL-based network slicing solutions and the IDLA algorithm has shown significant improvements in performance, efficiency, and adaptability. These advancements lay a strong foundation for future research and practical implementations in dynamic and heterogeneous network environments. As we move towards more intelligent and adaptive communication systems, integrating RL, TL, and advanced optimization techniques will be crucial for realizing the full potential of next-generation networks.

Bibliography

[1] Rachid El Hattachi and Javan Erfanian. 5G white paper. Technical report, 02 2015.

[2] Mansoor Shafi, Andreas F. Molisch, Peter J. Smith, Thomas Haustein, Peiying Zhu, Prasan De Silva, Fredrik Tufvesson, Anass Benjebbour, and Gerhard Wunder. 5g: A tutorial overview of standards, trials, challenges, deployment, and practice. *IEEE Journal on Selected Areas in Communications*, 35(6):1201–1221, 2017. doi:10.1109/JSAC.2017.2692307.

[3] Nicolas Huin, Paolo Medagliani, Sébastien Martin, Jérémie Leguay, Lei Shi, Shengming Cai, Jinchun Xu, and Hao Shi. Hard-isolation for network slicing. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 955–956, 2019. doi:10.1109/INFCOMW.2019.8845282.

[4] Xin Li, Mohammed Samaka, H. Anthony Chan, Deval Bhamare, Lav Gupta, Chengcheng Guo, and Raj Jain. Network slicing for 5g: Challenges and opportunities. *IEEE Internet Computing*, 21(5):20–27, 2017. doi:10.1109/MIC.2017.3481355.

[5] Abdelbaset S. Hamza, Shady S. Khalifa, Haitham S. Hamza, and Khaled Elsayed. A survey on inter-cell interference coordination techniques in ofdma-based cellular networks. *IEEE Communications Surveys Tutorials*, 15(4):1642–1670, 2013. doi:10.1109/SURV.2013.013013.00028.

[6] Mahmudur Rahman and Halim Yanikomeroglu. Enhancing cell-edge performance: a downlink dynamic interference avoidance scheme with inter-cell coordination. *IEEE Transactions on Wireless Communications*, 9(4):1414–1425, 2010. doi:10.1109/TWC.2010.04.090256.

[7] Meiyu Wang, Yun Lin, Qiao Tian, and Guangzhen Si. Transfer learning promotes 6g wireless communications: Recent advances and future challenges. *IEEE Transactions on Reliability*, 70(2):790–807, 2021. doi:10.1109/TR.2021.3062045.

[8] Cong T. Nguyen, Nguyen Van Huynh, Nam H. Chu, Yuris Mulya Saputra, Dinh Thai Hoang, Diep N. Nguyen, Quoc-Viet Pham, Dusit Niyato, Eryk Dutkiewicz, and Won-Joo Hwang. Transfer learning for wireless networks: A comprehensive survey. *Proceedings of the IEEE*, 110(8):1073–1115, 2022. doi:10.1109/JPROC.2022.3175942.

[9] Manuel S. Santos and John Rust. Convergence properties of policy iteration. *SIAM J. Control. Optim.*, 42:2094–2115, 2003.

[10] Anna Winnicki and R Srikant. On the convergence of policy iteration-based reinforcement learning with monte carlo policy evaluation. In *International Conference on Artificial Intelligence and Statistics*, pages 9852–9878. PMLR, 2023.

[11] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[12] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[13] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[14] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL: http://arxiv.org/abs/1509.06461, `arXiv:1509.06461`.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[16] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[17] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[18] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[19] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[20] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[21] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[22] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

[23] C. W. Bray. Transfer of learning. *Journal of Experimental Psychology*, 11(6):443–467, 1928.

[24] Lisa A. Torrey and Jude W. Shavlik. Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 249–256. IGI Global, 2010.

[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[26] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[27] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi:10.1109/CVPR.2009.5206848.

[28] Edoardo Giacomello, Daniele Loiacono, and Luca Mainardi. Brain mri tumor segmentation with adversarial networks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[29] Qiang Yang, Yu Zhang, Wenyuan Dai, and Sinno Jialin Pan. *Transfer Learning*. Cambridge University Press, 2020.

[30] Wouter M. Kouw. An introduction to domain adaptation and transfer learning. *CoRR*, abs/1812.11806, 2018. URL: http://arxiv.org/abs/1812.11806.

[31] Jindong Wang and Yiqiang Chen. *Introduction to Transfer Learning: Algorithms and Practice*. Springer Nature, 2023. URL: jd92.wang/tlbook.

[32] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

[33] Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime Carbonell. Characterizing and avoiding negative transfer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11293–11302, 2019.

[34] Wen Zhang, Lingfei Deng, Lei Zhang, and Dongrui Wu. A survey on negative transfer. *IEEE/CAA Journal of Automatica Sinica*, 10(2):305–329, 2022.

[35] Miaofeng Liu, Yan Song, Hongbin Zou, and Tong Zhang. Reinforced training data selection for domain adaptation. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 1957–1968, 2019.

[36] Yash Patel, Kashyap Chitta, and Bhavan Jasani. Learning sampling policies for domain adaptation. *arXiv preprint arXiv:1805.07641*, 2018.

[37] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pages 353–374, 2023.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[39] A. Kusiak. Feature transformation methods in data mining. *IEEE Transactions on Electronics Packaging Manufacturing*, 24(3):214–221, 2001. doi:10.1109/6104.956807.

[40] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. arXiv:1406.2661.

[41] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *J. Mach. Learn. Res.*, 17(1):2096–2030, jan 2016.

[42] Yaqing Wang, Quanming Yao, James Kwok, and Lionel M. Ni. Generalizing from a few examples: A survey on few-shot learning, 2020. arXiv:1904.05046.

[43] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[44] Zhuangdi Zhu, Kaixiang Lin, Anil K. Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(11):13344–13362, 2023. doi:10.1109/TPAMI.2023.3292075.

[45] Simon Schmitt, Jonathan J. Hudson, Augustin Zídek, Simon Osindero, Carl Doersch, Wojciech M. Czarnecki, Joel Z. Leibo, Heinrich Küttler, Andrew Zisserman, Karen Simonyan, and S. M. Ali Eslami. Kickstarting deep reinforcement learning. *ArXiv*, abs/1803.03835, 2018.

[46] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR.

[47] Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, 33:15931–15941, 2020.

[48] Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning*, pages 9767–9779. PMLR, 2021.

[49] Nokia Siemens Networks. White paper: Self-organizing network (son): Introducing the nokia siemens networks son suite-an efficient, future-proof platform for son. 2009.

[50] R. A. Addad, Miloud Bagaa, T. Taleb, D. Dutra, and H. Flinck. Optimization model for cross-domain network slices in 5g networks. *IEEE Transactions on Mobile Computing*, 19:1156–1169, 2020.

[51] Halyna Beshley, Mykola Beshley, Mykhailo Medvetskyi, and Julia Pyrih. Qos-aware optimal radio resource allocation method for machine-type communications in 5g lte and beyond cellular networks. *Wirel. Commun. Mob. Comput.*, 2021:9966366:1–9966366:18, 2021.

[52] Francesca Fossati, Stefano Moretti, P. Perny, and S. Secci. Multi-resource allocation for network slicing. *IEEE/ACM Transactions on Networking*, 28:1311–1324, 2020.

[53] Tengteng Ma, Yong Zhang, Fanggang Wang, Dong Wang, and Da Guo. Slicing resource allocation for embb and urllc in 5g ran. *Wirel. Commun. Mob. Comput.*, 2020:6290375:1–6290375:11, 2020.

[54] Renato L. G. Cavalcante, Qi Liao, and Slawomir Stańczak. Connections between spectral properties of asymptotic mappings and solutions to wireless network problems. *IEEE Transactions on Signal Processing*, 67(10):2747–2760, 2019. doi:10.1109/TSP.2019.2908147.

[55] Yongshuai Liu, Jiaxin Ding, and Xin Liu. A constrained reinforcement learning based approach for network slicing. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–6, 2020. doi:10.1109/ICNP49622.2020.9259378.

[56] Qiang Liu, Tao Han, Ning Zhang, and Ye Wang. DeepSlicing: Deep reinforcement learning assisted resource allocation for network slicing. In *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pages 1–6, 2020. doi:10.1109/GLOBECOM42002.2020.9322106.

[57] Ismail Alqerm and Basem Shihada. A cooperative online learning scheme for resource allocation in 5g systems. *2016 IEEE International Conference on Communications (ICC)*, pages 1–7, 2016.

[58] Nan Zhao, Ying-Chang Liang, Dusit Tao Niyato, Yiyang Pei, Minghu Wu, and Yunhao Jiang. Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks. *IEEE Transactions on Wireless Communications*, 18:5141–5152, 2019.

[59] Hao Song, Lingjia Liu, Jonathan D. Ashdown, and Yang Cindy Yi. A deep reinforcement learning framework for spectrum management in dynamic spectrum access. *IEEE Internet of Things Journal*, 8:11208–11218, 2021.

[60] Hai xia Peng and Xuemin (Sherman) Shen. Deep reinforcement learning based resource management for multi-access edge computing in vehicular networks. *IEEE Transactions on Network Science and Engineering*, 7:2416–2428, 2020.

[61] Santiago Paternain, Luiz FO Chamon, Miguel Calvo-Fullana, and Alejandro Ribeiro. Constrained reinforcement learning has zero duality gap. *arXiv preprint arXiv:1910.13393*, 2019.

[62] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

[63] Jakob N Foerster, Yannis M Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *arXiv preprint arXiv:1605.06676*, 2016.

[64] V. Konda and J. Tsitsiklis. Actor-Critic algorithms. In *NIPS*, 1999.

[65] Scott Fujimoto, H. V. Hoof, and D. Meger. Addressing function approximation error in Actor-Critic methods. *ArXiv*, abs/1802.09477, 2018.

[66] D. Silver, Guy Lever, N. Heess, T. Degris, Daan Wierstra, and Martin A. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[67] Juha Meinilä, Pekka Kyösti, Lassi Hentilä, Tommi Jämsä, Essi Suikkanen, Esa Kunnari, and Milan Narandžić. *Wireless World Initiative New Radio - Winner+*. Technical report, 2010.

[68] Yan Shao, Rongpeng Li, Zhifeng Zhao, and Honggang Zhang. Graph attention network-based DRL for network slicing management in dense cellular networks. *2021 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2021.

[69] Hongrui Nie, Shaosheng Li, and Yong Liu. Multi-agent deep reinforcement learning for resource allocation in the multi-objective HetNet. *2021 International Wireless Communications and Mobile Computing (IWCMC)*, pages 116–121, 2021.

[70] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[71] Cong T Nguyen, Nguyen Van Huynh, Nam H Chu, Yuris Mulya Saputra, Dinh Thai Hoang, Diep N Nguyen, Quoc-Viet Pham, Dusit Niyato, Eryk Dutkiewicz, and Won-Joo Hwang. Transfer learning for future wireless networks: A comprehensive survey. *arXiv preprint arXiv:2102.07572*, 2021.

[72] Meiyu Wang, Yun Lin, Qiao Tian, and Guangzhen Si. Transfer learning promotes 6g wireless communications: Recent advances and future challenges. *IEEE Transactions on Reliability*, 2021.

[73] Claudia Parera, Qi Liao, Ilaria Malanchini, Cristian Tatino, Alessandro E. C. Redondi, and Matteo Cesana. Transfer learning for tilt-dependent radio map prediction. *IEEE Transactions on Cognitive Communications and Networking*, 6(2):829–843, 2020. doi:10.1109/TCCN.2020.2964761.

[74] Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *J. Mach. Learn. Res.*, 8:2125–2167, 2007.

[75] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

[76] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, 2009.

[77] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

[78] Ahmad M Nagib, Hatem Abou-Zeid, and Hossam S Hassanein. Transfer learning-based accelerated deep reinforcement learning for 5G RAN slicing. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, pages 249–256. IEEE, 2021.

[79] Tianle Mai, Haipeng Yao, Ni Zhang, Wenji He, Dong Guo, and Mohsen Guizani. Transfer reinforcement learning aided distributed network slicing resource optimization in industrial IoT. *IEEE Transactions on Industrial Informatics*, 2021.

[80] Hammad Zafar, Zoran Utkovski, Martin Kasparick, and Sławomir Stańczak. Transfer learning in multi-agent reinforcement learning with double Q-networks for distributed resource sharing in V2X communication. *ArXiv*, abs/2107.06195, 2021.

[81] Renato Luís Garrido Cavalcante, Qi Liao, and Sławomir Stańczak. Connections between spectral properties of asymptotic mappings and solutions to wireless network problems. *IEEE Transactions on Signal Processing*, 67:2747–2760, 2019.

[82] Vincenzo Sciancalepore, Ilario Filippini, Vincenzo Mancuso, Antonio Capone, and Albert Banchs. A multi-traffic inter-cell interference coordination scheme in dense cellular networks. *IEEE/ACM Transactions on Networking*, 26:2361–2375, 2018.

[83] Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, C. Liu, and Dejun Yang. Experience-driven networking: A deep reinforcement learning based approach. *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 1871–1879, 2018.

[84] Dongsheng Xu, Peng Qiao, and Yong Dou. Aggregation transfer learning for multi-agent reinforcement learning. *2021 2nd International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*, pages 547–551, 2021.

[85] 3GPP, TS 28.530. Technical Specification Group Services and System Aspects; Management and orchestration; Concepts, use cases and requirements, V17.2.0, December 2021.

[86] Jeonghoon Mo and Jean Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking*, 8(5):556–567, 2000.

[87] Thomas Bonald, Laurent Massoulié, Alexandre Proutiere, and Jorma Virtamo. A queueing analysis of max-min fairness, proportional fairness and balanced fairness. *Queueing systems*, 53(1):65–84, 2006.

[88] John Ewing. *Autonomic Performance Optimization with Application to Self-Architecting Software Systems*. PhD thesis, 04 2015.

[89] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *NIPS*, 2016.

[90] Guoliang Kang, Lu Jiang, Yi Yang, and Alexander G Hauptmann. Contrastive adaptation network for unsupervised domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4893–4902, 2019.

[91] Rongpeng Li, Zhifeng Zhao, Qi Sun, Chih-Lin I, Chenyang Yang, Xianfu Chen, Minjian Zhao, and Honggang Zhang. Deep reinforcement learning for resource management in network slicing. *IEEE Access*, 6:74429–74441, 2018.

[92] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *CoRR*, abs/2009.07888, 2020. URL: https://arxiv.org/abs/2009.07888, `arXiv:2009.07888`.

[93] Ahmad M. Nagib, Hatem Abou-zeid, and Hossam S. Hassanein. Transfer learning-based accelerated deep reinforcement learning for 5G RAN slicing. *IEEE 46th LCN*, pages 249–256, 2021.

[94] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *NIPS*, 2015.

[95] Mojdeh Karbalaee Motalleb, Vahid Shah-Mansouri, Saeedeh Parsaeefard, and Onel Luis Alcaraz López. Resource allocation in an open RAN system using network slicing. *IEEE Transactions on Network and Service Management*, 20:471–485, 2023.

[96] Mathieu Leconte and et al. A resource allocation framework for network slicing. *IEEE INFOCOM*, pages 2177–2185, 2018.

[97] Liu Qiang, Choi Nakjung, and Han Tao. Constraint-aware deep reinforcement learning for end-to-end resource orchestration in mobile networks. In *IEEE ICNP*, pages 1–11, 2021.

[98] Liu Qiang, Choi Nakjung, and Han Tao. OnSlicing: Online end-to-end network slicing with reinforcement learning. In *ACM CoNEXT*, pages 141–153, 2021.

[99] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[100] Tianlun Hu, Qi Liao, Qiang Liu, Dan Wellington, and Georg Carle. Inter-cell slicing resource partitioning via coordinated multi-agent deep reinforcement learning. In *IEEE International Conference Communications (ICC)*, 2022.

[101] Yifei Shen, Yuanming Shi, Jun Zhang, and Khaled Ben Letaief. Transfer learning for mixed-integer resource allocation problems in wireless networks. *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6, 2018. URL: https://api.semanticscholar.org/CorpusID:70033937.

[102] Qingtian Zeng, Qiang Sun, Geng Chen, Hua Duan, Chao Li, and Ge Song. Traffic prediction of wireless cellular networks based on deep transfer learning and cross-domain data. *IEEE Access*, 8:172387–172397, 2020. URL: https://api.semanticscholar.org/CorpusID:222095907.

[103] Nguyen Van Huynh and Geoffrey Y. Li. Transfer learning for signal detection in wireless networks. *IEEE Wireless Communications Letters*, 11:2325–2329, 2022. URL: https://api.semanticscholar.org/CorpusID:251879154.

[104] Wouter M. Kouw. An introduction to domain adaptation and transfer learning. *CoRR*, abs/1812.11806, 2018. URL: http://arxiv.org/abs/1812.11806, `arXiv:1812.11806`.

[105] Zhiqi Yu, Jingjing Li, Zhekai Du, Lei Zhu, and Heng Tao Shen. A comprehensive survey on source-free domain adaptation, 2023. `arXiv:2302.11803`.

[106] Mohamed Akrout, Amal Feriani, Faouzi Bellili, Amine Mezghani, and Ekram Hossain. Domain generalization in machine learning models for wireless communications: Concepts, state-of-the-art, and open issues. *IEEE Communications Surveys & Tutorials*, 25:3014–3037, 2023. URL: https://api.semanticscholar.org/CorpusID:257505534.

[107] Junyang Shi, Mo Sha, and Xi Peng. Adapting wireless mesh network configuration from simulation to reality via deep learning based domain adaptation. In *Symposium on Networked Systems Design and Implementation*, 2021. URL: https://api.semanticscholar.org/CorpusID:232127966.

[108] Quan Zhou, Ronghui Zhang, Zexuan Jing, and Xiaojun Jing. Semi-supervised-based automatic modulation classification with domain adaptation for wireless iot spectrum monitoring. In *Frontiers of Physics*, 2023. URL: https://api.semanticscholar.org/CorpusID:257838140.

[109] Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *ArXiv*, physics/0004057, 2000. URL: https://api.semanticscholar.org/CorpusID:8936496.

[110] Claude E. Shannon. *Coding Theorems for a Discrete Source With a Fidelity CriterionInstitute of Radio Engineers, International Convention Record, vol. 7, 1959.*, pages 325–350. 1993. `doi:10.1109/9780470544242.ch21`.

[111] Ziv Goldfeld and Yury Polyanskiy. The information bottleneck problem and its applications in machine learning. *IEEE Journal on Selected Areas in Information Theory*, 1(1):19–38, 2020. `doi:10.1109/JSAIT.2020.2991561`.

[112] Hyewon Yang, Jeongju Jee, Girim Kwon, and Hyuncheol Park. Deep transfer learning-based adaptive beamforming for realistic communication channels. *2020 International Conference on Information and Communication Technology Convergence*

(ICTC), pages 1373–1376, 2020. URL: https://api.semanticscholar.org/CorpusID:229374594.

[113] Rahul Kumar Jaiswal, Mohamed Elnourani, Siddharth Deshmukh, and Baltasar Beferull-Lozano. Deep transfer learning based radio map estimation for indoor wireless communications. *2022 IEEE 23rd International Workshop on Signal Processing Advances in Wireless Communication (SPAWC)*, pages 1–5, 2022. URL: https://api.semanticscholar.org/CorpusID:251169549.

[114] Siavash Barqi Janiar and Ping Wang. A transfer learning approach based on integrated feature extractor for anti-jamming in wireless networks. *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–6, 2023. URL: https://api.semanticscholar.org/CorpusID:264883315.

[115] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando C Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine Learning*, 79:151–175, 2010. URL: https://api.semanticscholar.org/CorpusID:8577357.

[116] Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, Tao Qin, Wang Lu, Yiqiang Chen, Wenjun Zeng, and Philip S. Yu. Generalizing to unseen domains: A survey on domain generalization. *IEEE Transactions on Knowledge and Data Engineering*, 35(8):8052–8072, 2023. doi:10.1109/TKDE.2022.3178128.

[117] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. Domain generalization: A survey. *CoRR*, abs/2103.02503, 2021. URL: https://arxiv.org/abs/2103.02503, arXiv:2103.02503.

[118] Dayan Guan, Jiaxing Huang, Aoran Xiao, Shijian Lu, and Yanpeng Cao. Uncertainty-aware unsupervised domain adaptation in object detection. *IEEE Transactions on Multimedia*, 24:2502–2514, 2022. doi:10.1109/TMM.2021.3082687.

[119] Justin Chen, Edward Gan, Kexin Rong, Sahaana Suri, and Peter Bailis. Crosstrainer: Practical domain adaptation with loss reweighting. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning*, pages 1–10, 2019.

[120] Jayaram Raghuram, Yijing Zeng, Dolores García Martí, Rafael Ruiz Ortiz, Somesh Jha, Joerg Widmer, and Suman Banerjee. Few-shot domain adaptation for end-to-end communication, 2023. arXiv:2108.00874.

[121] Jiawei Chen, Ziqi Zhang, Xinpeng Xie, Yuexiang Li, Tao Xu, Kai Ma, and Yefeng Zheng. Beyond mutual information: Generative adversarial network for domain adaptation using information bottleneck constraint. *IEEE Transactions on Medical Imaging*, 41(3):595–607, 2022. doi:10.1109/TMI.2021.3117996.

[122] Junyang Shi, Aitian Ma, Xia Cheng, Mo Sha, and Xi Peng. Adapting wireless network configuration from simulation to reality via deep learning-based domain adaptation. *IEEE/ACM Transactions on Networking*, 2023. URL: https://api.semanticscholar.org/CorpusID:265433693.

[123] Alexander A. Alemi, Ian Fischer, and Joshua V. Dillon. Deep variational information bottleneck. *ArXiv*, abs/1612.00410, 2017. URL: https://api.semanticscholar.org/CorpusID:204922497.

[124] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. arXiv: 1312.6114.

[125] Yuzhe Yang, Kaiwen Zha, Ying-Cong Chen, Hongya Wang, and Dina Katabi. Delving into deep imbalanced regression. *ArXiv*, abs/2102.09554, 2021. URL: https://api.semanticscholar.org/CorpusID:231951565.

Acronyms

3GPP 3rd Generation Partnership Project.

5G 5th Generation Mobile Network.

AI Artificial Intelligence.

AR Augmented Reality.

BERT Bidirectional Encoder Representations from Transformers.

CDF Cumulative Distribution Function.

CNN Convolutional Neural Net.

CQI Channel Quality Indicator.

CSI Channel State Information.

CV Computer Vision.

DA Domain Adaptation.

DANN Domain Adversarial Neural Network.

DDPG Deep Deterministic Policy Gradient Algorithm.

DG Domain Generalization.

DIRP Distributed Inter-Cell Inter-Slice Resource Partition.

DL Deep Learning.

DNN Deep Neural Net.

DP Dynamic Programming.

DPG Deterministic Policy Gradient Algorithm.

DQN Deep Q-Network.

DRL Deep Reinforcement Learning.

eMBB Extreme Mobile Broadband.

GAN Generative Adversarial Net.

GD Gradient Descent.

gNB gNodeB.

GPT Generative Pre-Trained Transformer.

i.i.d Independent and Identically Distributed.

IB Information Bottleneck.

IDLA Integrated Deep Learning and Lagrangian Method.

IoT Internet of Things.

KL Kullback-Leibler.

KL-divergence Kullback-Leibler Divergence.

KPI Key Performance Indicator.

LDS Label Distribution Smoothing.

LLM Large Language Model.

LTE Long-Term Evolution.

M2M Machine-to-Machine.

MAC Media Access Control.

MADRL Multi-Agent Deep Reinforcement Learning.

MAE Mean Absolute Error.

MARL Multi-Agent Reinforcement Learning.

MDP Markov Decision Process.

MILP Mixed Integer Linear Programming.

ML Machine Learning.

MLP Multi-Layer Perceptron.

MMD Maximum Mean Discrepancy.

mMTC Massive Machine-Type Communications.

NFV Network Function Virtualization.

NLP Natural Language Processing.

O-RAN Open Radio Access Network.

OAM Operation, Administration, and Maintenance.

OFDM Orthogonal Frequency-Division Multiplexing.

PCA Principal Component Analysis.

PG Policy Gradient.

PRB Physical Resource Block.

QoE Quality of Experience.

QoS Quality of Service.

RAN Radio Access Network.

RB Resource Block.

RE Resource Element.

RL Reinforcement Learning.

SARSA State–Action–Reward–State–Action.

SDN Software-Defined Networking.

SGD Stochastic Gradient Descent.

SINR Signal-to-Interference-plus-Noise Ratio.

SL Supervised Learning.

SLA Service Level Agreement.

SON Self-Organizing Networks.

TD Temporal Difference.

TD3 Twin Delayed Deep Deterministic Policy Gradient Algorithm.

TL Transfer Learning.

TTI Transmission Time Interval.

UE User Equipment.

URLLC Ultra-Reliable Low-Latency Communication.

V2X Vehicle-to-Everything.

VAE Variational Autoencoder.

VIB Variational Information Bottleneck.

VR Virtual Reality.

XR Extended Reality.

List of Figures

List of Tables