ORAN-Based Intelligent Network Optimization

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

by

Ram Santhosh Patnaik Belagam

(IEC2021019)

Supervised by

Dr. Suneel Yadav

Department of Electronics and Communications Engineering

Indian Institute of Information Technology Allahabad

May 2025

APPROVAL SHEET

This thesis entitled **"ORAN-Based Intelligent Network Optimization"** submitted by **Ram Santhosh Patnaik Belagam (IEC2021019)** in partial fulfillment of the requirements for the degree of **Bachelor of Technology** in **Electronics & Communications** is hereby approved.

_____

Dr. Suneel Yadav
Thesis Supervisor
Department of Electronics & Communications
IIIT Allahabad

_____

Dr.Radhika Gour
Examiner
Department of Electronics & Communications
IIIT Allahabad

_____

Dr.Pooja Mishra
Examiner
Department of Electronics & Communications
IIIT Allahabad

_____

Head of the Department
Department of Electronics & Communications
IIIT Allahabad

Date: _____

DECLARATION OF ACADEMIC HONESTY AND INTEGRITY

I, **Ram Santhosh Patnaik Belagam**, Enrollment Number **IEC2021019**, hereby declare that the work presented in this thesis entitled **"ORAN-Based Intelligent Network Optimization"** is my own original research.

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

_____

Ram Santhosh Patnaik Belagam
Enrollment No: IEC2021019
B.Tech (Electronics & Communications)
IIIT Allahabad

Date: _____

Ram Santhosh

ORIGINALITY REPORT

9%	7%	7%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.aiktcdspace.org:8080 Internet Source	1%
2	export.arxiv.org Internet Source	<1%
3	www.mdpi.com Internet Source	<1%
4	Submitted to Imperial College of Science, Technology and Medicine Student Paper	<1%
5	Submitted to University of Sheffield Student Paper	<1%
6	Mohammad Ali Zamani, Sven Magg, Cornelius Weber, Stefan Wermter, Di Fu. "Deep reinforcement learning using compositional representations for performing instructions", Paladyn, Journal of Behavioral Robotics, 2018 Publication	<1%
7	Submitted to London Business School Student Paper	<1%
8	Gyanendra Kumar Maurya, Faizan Ahmad, Kavindra Kandpal, Rachana Kumar, Mahesh Kumar, Pramod Kumar, Akhilesh Tiwari. "UV	<1%

	to NIR tunable photodetector using Bi2Te2Se/n-GaN heterojunction", Surfaces and Interfaces, 2022 Publication	
9	"Open RAN", Wiley, 2023 Publication	<1 %
10	Queirós, Gonçalo Moura Tomé Fraguito. "Autonomous Control and Positioning of a Mobile 5G Radio Access Node Employing the O-RAN Architecture", Universidade do Porto (Portugal), 2024 Publication	<1 %
11	www.rimedolabs.com Internet Source	<1 %
12	etheses.whiterose.ac.uk Internet Source	<1 %
13	vtechworks.lib.vt.edu Internet Source	<1 %
14	Submitted to University of Queensland Student Paper	<1 %
15	Submitted to Napier University Student Paper	<1 %
16	Han Wang, Chunxiao Xing, Liang-Jie Zhang. "Chapter 3 Integrated 5G MEC System and Its Application in Intelligent Video Analytics", Springer Science and Business Media LLC, 2022 Publication	<1 %

17	Osman Tugay Basaran, Falko Dressler. "XAInomaly: Explainable and interpretable Deep Contractive Autoencoder for O-RAN traffic anomaly detection", Computer Networks, 2025 Publication	<1 %
18	Submitted to University of Alabama at Birmingham Student Paper	<1 %
19	Submitted to University of Sunderland Student Paper	<1 %
20	par.nsf.gov Internet Source	<1 %
21	Submitted to University of Huddersfield Student Paper	<1 %
22	5g-ppp.eu Internet Source	<1 %
23	Chong Li, Meikang Qiu. "Reinforcement Learning for Cyber-Physical Systems - with Cybersecurity Case Studies", CRC Press, 2019 Publication	<1 %
24	Submitted to Indian Institute of Technology, Kharagpure Student Paper	<1 %
25	Shota Ohnishi, Eiji Uchibe, Yotaro Yamaguchi, Kosuke Nakanishi, Yuji Yasui, Shin Ishii. "Constrained Deep Q-Learning Gradually	<1 %

	Approaching Ordinary Q-Learning", Frontiers in Neurorobotics, 2019 Publication	
26	Submitted to UC, Boulder Student Paper	<1 %
27	wukongzhiku.com Internet Source	<1 %
28	www.digitaljournal.com Internet Source	<1 %
29	Chouman, Ali. "Machine Learning and Operations Research for Intelligence Engines in Future Networks.", The University of Western Ontario (Canada), 2024 Publication	<1 %
30	hdl.handle.net Internet Source	<1 %
31	pmb.univ-saida.dz Internet Source	<1 %
32	scholar.sun.ac.za Internet Source	<1 %
33	trepo.tuni.fi Internet Source	<1 %
34	Mao V. Ngo, Nguyen-Bao-Long Tran, Hyun-Min Yoo, Yong-Hao Pua et al. "RAN Intelligent Controller (RIC): From open-source implementation to real-world validation", ICT Express, 2024 Publication	<1 %

35	Xiaomao Zhou, Yanbin Gao, Lianwu Guan. "Towards Goal-Directed Navigation Through Combining Learning Based Global and Local Planners", Sensors, 2019 Publication	<1 %
36	assets.researchsquare.com Internet Source	<1 %
37	"Mobile Web and Intelligent Information Systems", Springer Science and Business Media LLC, 2019 Publication	<1 %
38	Submitted to National Institute of Technology Delhi Student Paper	<1 %
39	escholarship.org Internet Source	<1 %
40	thesis.unipd.it Internet Source	<1 %
41	www.coursehero.com Internet Source	<1 %
42	Passos Ferreira, Rui Pedro. "Network Analytics for 5G Data", Universidade da Beira Interior (Portugal), 2024 Publication	<1 %
43	Submitted to Universidade de Aveiro Student Paper	<1 %
44	e-archivo.uc3m.es Internet Source	<1 %

45	eprints.uthm.edu.my Internet Source	<1%

Exclude quotes	Off	Exclude matches	< 15 words
Exclude bibliography	On		

Abstract

This project presents an innovative approach to intelligent network optimization in Open Radio Access Networks (ORAN) by leveraging data-driven environment modeling and advanced reinforcement learning algorithms. The primary objective is to enable dynamic, autonomous network slicing that optimizes resource allocation while maintaining high-quality service, thereby advancing the efficiency and adaptability of wireless communication systems. The proposed methodology is composed of several integral stages: comprehensive preprocessing of real-world RF data collected from the Colosseum testbed, construction of a surrogate environment model using deep neural networks to accurately simulate network behavior, formulation and training of a reinforcement learning agent for policy optimization, and rigorous evaluation against conventional baseline strategies. The environment model serves as a reliable proxy for real-world network conditions, enabling safe and scalable agent training. Empirical results demonstrate the effectiveness of the approach, achieving significant improvements in throughput and overall network performance compared to static methods. This project contributes to the field of intelligent radio access networks by illustrating the potential of combining realistic data-driven simulations with reinforcement learning, ultimately paving the way for more responsive and intelligent next-generation wireless infrastructures.

Keywords: Open Radio Access Network (ORAN), Reinforcement Learning (RL), Network Slicing, Deep Neural Networks (DNN), 5G, RAN Intelligent Controller (RIC), Network Optimization, Data-Driven Modeling.

Contents

List of Figures

List of Tables

Abbreviations / Notation

5G Fifth Generation Mobile Network

5GC 5G Core Network

AI Artificial Intelligence

AMF Access and Mobility Function

API Application Programming Interface

BN Batch Normalization

CQI Channel Quality Indicator

CU Centralized Unit

DNN Deep Neural Network

DQN Deep Q-Network

DU Distributed Unit

eMBB Enhanced Mobile Broadband

eCPRI Enhanced Common Public Radio Interface

gNodeB Next Generation Node B (5G Base Station)

GNN Graph Neural Network

ICIC Inter-Cell Interference Coordination

IIIT-A Indian Institute of Information Technology Allahabad

KPI Key Performance Indicator

LSTM Long Short-Term Memory

MAC Medium Access Control

MCS Modulation and Coding Scheme

MDP Markov Decision Process

ML Machine Learning

mMTC Massive Machine-Type Communications

MSE Mean Squared Error

MTC Machine-Type Communications

NEF Network Exposure Function

non-RT RIC Non-Real-Time RAN Intelligent Controller

near-RT RIC Near-Real-Time RAN Intelligent Controller

NSSF Network Slice Selection Function

OAI OpenAirInterface

OFH Open Fronthaul

ORAN Open Radio Access Network

PCF Policy Control Function

PDCP Packet Data Convergence Protocol

PF Proportional Fair (Scheduling)

PHY Physical Layer

PRB Physical Resource Block

PPO Proximal Policy Optimization

QoS Quality of Service

RAN Radio Access Network

ReLU Rectified Linear Unit

RF Radio Frequency

RIC RAN Intelligent Controller

RL Reinforcement Learning

RLC Radio Link Control

RMSE Root Mean Square Error

RNN Recurrent Neural Network

RR Round Robin (Scheduling)

RSSI Received Signal Strength Indicator

RU Radio Unit

SBA Service-Based Architecture

SDAP Service Data Adaptation Protocol

SINR Signal-to-Interference-plus-Noise Ratio

SLA Service Level Agreement

SMF Session Management Function

SMO Service Management and Orchestration

srsRAN Software Radio Systems RAN

TP Throughput

UE User Equipment

UL Uplink

UPF User Plane Function

URLLC Ultra-Reliable Low-Latency Communications

WF Weighted Fair (Scheduling)

xApp Application running on near-RT RIC

rApp Application running on non-RT RIC

Notation

$\mathbf{s}_t$ State vector at time t

$\mathbf{a}_t$ Action vector at time t

$\mathbf{y}_t$ Output vector at time t

f^* True underlying system dynamics function

f_θ Learned surrogate model (DNN) with parameters θ

$\mathcal{S}$ State space

$\mathcal{A}$ Action space

$P(\mathbf{s}'|\mathbf{s}, a)$ State transition probability

$r(\mathbf{s}, a)$ Reward function

γ Discount factor in RL

$\pi(\mathbf{s})$ Policy (mapping states to actions)

$Q(\mathbf{s}, a)$ State-action value function

$V(\mathbf{s})$ State value function

$A(\mathbf{s}, a)$ Advantage function

$\mathcal{L}(\theta)$ Loss function with parameters θ

ϵ Exploration rate in epsilon-greedy strategy

λ Regularization parameter (e.g., L2 weight decay)

Chapter 1

Introduction

The wireless communications landscape is undergoing a profound transformation driven by the demands of emerging applications, burgeoning device counts, and the ever-growing expectations for service quality and adaptability. The introduction of 5G networks—characterized by ultra-reliable low latency, massive connectivity, and support for diverse services—has exposed both the strengths and limitations of existing radio access network (RAN) architectures [1]. As traffic patterns fluctuate and user needs evolve, traditional monolithic and proprietary RAN solutions often struggle to keep pace with the required flexibility and scale. This has motivated a broad investigation into new architectural paradigms, with Open Radio Access Networks (ORAN) representing one of the most promising avenues of research and development [2], [3].

ORAN is fundamentally reshaping the RAN ecosystem by advocating for open interfaces, disaggregated hardware, and multi-vendor interoperability [3], [4]. Central to this paradigm is the vision of a programmable, intelligent network where control functions can be dynamically orchestrated, monitored, and optimized—enabling a higher degree of adaptability and innovation [5]. However, while the promise of ORAN is widely recognized, realizing its potential in practice remains a significant technical and operational challenge [3]. The open architecture introduces a range of new components and interfaces, notably the RAN Intelligent Controller (RIC), which serves as a platform for deploying advanced, AI-driven optimization applications (xApps and rApps) [5].

In this context, our project aims to explore and assess a particular pathway toward intelligent ORAN optimization, emphasizing both the potential and practical considerations inherent to this approach. Specifically, we investigate the feasibility of combining data-driven environment modeling with reinforcement learning (RL) [5], [6] to optimize key network functions—such as resource allocation and network slicing—within an ORAN framework [7]. By constructing a surrogate simulation environment using a deep neural network (DNN) trained on real-world RAN measurement data, we create a controlled, repeatable testbed for training RL agents. These agents are tasked with dynamically adjusting network parameters to maximize end-to-end performance metrics like throughput and Quality of Service (QoS).

Our project is motivated by the practical realities of deploying intelligence in future RANs. While large-scale, real-world experimentation is the gold standard, access to such infrastructure is often limited. Moreover, the complexity of real-time wireless environments and the risk of unintended consequences during agent training underscore the value of high-fidelity simulations [3], [5]. By leveraging real measurement data from the Colosseum RF laboratory, we bridge the gap between theoretical models and practical deployment, ensuring that our simulated environment captures the intricate dynamics and variability of operational networks.

The proposed approach is not intended as a comprehensive solution to all the challenges of ORAN optimization. Rather, it represents one possible step forward—an experimental synthesis of current machine learning methods and next-generation network architectures. By adopting a modular and transparent methodology, we aim to provide both a proof of concept and a foundation for further research [3], [5]. Our findings suggest that even with the limitations of surrogate environments and current RL techniques, intelligent closed-loop control can yield measurable improvements over traditional, static baselines in key performance metrics [7].

Nevertheless, it is essential to maintain a balanced perspective on the implications and generalizability of our results. The architecture and algorithms developed here are primarily validated in simulation, with environment models approximating rather than replicating live network behavior. The policies learned by our RL agents are evaluated within a constrained action space, focused on select aspects of network slicing and scheduling. While our experiments demonstrate promising gains in throughput and adaptability, the broader question of real-world transferability, robustness, and scalability remains open and is the subject of ongoing research both within and outside our project [3].

In structuring this thesis, we aim to offer a clear and sequential progression from foundational concepts to empirical evaluation. We begin with an in-depth discussion of ORAN's significance, including the structural evolution of 5G core networks and the critical role of open, intelligent controllers [1], [2], [4]. We then move to a mathematical exposition of our modeling and RL framework, detailing the statistical and algorithmic foundations underpinning our approach. The methodology section systematically delineates each stage of our workflow—from dataset preprocessing to model evaluation—highlighting the intent, design choices, and technical challenges encountered at each step. Finally, we provide an honest assessment of our results, acknowledge the boundaries of our findings, and chart multiple avenues for future work.

Through this project, we hope to contribute an additional perspective to the ongoing evolution of intelligent wireless networks. Our approach demonstrates the practical viability and potential benefits of ORAN-based intelligent optimization, while also underlining the importance of further investigation and validation in real-world deployments [8]. We believe that the journey toward truly adaptive, open, and intelligent networks will require continued collaboration across disciplines, rigorous experimentation, and a willingness to embrace both the strengths and limitations of current technology.

In summary, this thesis presents our experience and insights in using data-driven environment modeling and reinforcement learning as tools for ORAN-based network optimization. While not definitive, our work aspires to be a valuable addition to the collective exploration of next-generation RANs, and to inspire future efforts in bridging the gap between theoretical promise and operational impact.

Chapter 2

Literature Review

The modern evolution of mobile radio access networks (RANs) is driven by an industry-wide push for openness, intelligence, and efficiency, embodied most notably in the work of the O-RAN Alliance. Established as a global body, the O-RAN Alliance aims to redefine RAN architecture through principles of openness, virtualization, and AI-driven automation, in order to break vendor lock-in and stimulate both innovation and cost-efficiency [2], [3]. Its vision is actualized via a rigorous, collaborative specification process involving global operators, network vendors, and standardization institutes. By tightly aligning with key 3GPP documents—particularly TS 23.501, which defines the foundational 5G system architecture [1]—the O-RAN ecosystem maintains strong technical compatibility with existing standards while introducing software-defined, modular innovations.

A key tenet of O-RAN is its disaggregated architecture, which leverages open interfaces and network function virtualization atop general-purpose hardware [4]. This enables operators to flexibly compose their networks using interoperable components, while facilitating rapid integration of new technologies such as artificial intelligence. The RAN Intelligent Controller (RIC) is of particular importance: acting as a real-time, programmable platform, it supports deployment of AI/ML applications, or xApps/rApps, which can monitor, optimize, and automate RAN behaviors. Such programmability is vital to tackle the high complexity and dynamic nature of modern wireless environments, especially as networks adopt new paradigms such as network slicing and dynamic spectrum management [2], [3].

The burgeoning availability of large, open datasets—such as those described in [5]—has catalyzed research in machine learning and data-driven modeling for the RAN domain. These datasets provide granular, temporal records of network states, actions, and key performance indicators (KPIs), serving both as a basis for empirical modeling and as a safe environment for reinforcement learning (RL) policy development. Data-driven surrogate environments enable researchers to simulate and optimize RAN control policies without the logistical or operational risks of live deployments.

Reinforcement learning, especially in its deep forms, is increasingly recognized as a promising paradigm for wireless control problems where state, action, and reward spaces are large, stochastic, and only partially observable [3]. Notably, architectures such as Dueling Deep Q-Networks (Dueling DQN) [6] and Double Q-Learning have demonstrated improved stability and sample efficiency by decoupling the estimation of state value and action-specific advantage, and by mitigating overestimation bias in Q-learning. These innovations are particularly relevant for O-RAN's context, where agents must allocate discrete resources (such as Physical Resource Blocks (PRBs)) and scheduling policies while accounting for complex, unpredictable channel and traffic conditions.

Applied to the domain of O-RAN slicing, deep RL offers a pathway to real-time, adaptive resource allocation capable of fulfilling the diverse quality of service (QoS) demands of enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communications (URLLC), and massive Machine-Type Communications (mMTC) slices [7]. Unlike heuristic or static methods, DRL-based agents learn to balance throughput, delay, and reliability by interacting with and modeling the environment, achieving higher utilization and more resilient performance under non-stationary and previously unseen traffic. The use of deep surrogate models is espe-

cially important for safe RL training: by approximating RAN dynamics from data, policy search can be conducted without affecting live network operation [5].

Adoption of O-RAN is not only a technical imperative, but also an economic one. The open RAN philosophy stands to dramatically reduce both Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) for operators by fostering a competitive multi-vendor ecosystem, allowing modular upgrades, and leveraging commodity hardware through virtualization [8]. Industry analyses demonstrate that substantial savings—potentially up to 30% on radio costs over time—can be realized, provided that genuine interoperability is achieved and open competition prevents the formation of new oligopolies. It is crucial, therefore, for operators and vendors to adhere to open standards and robust certification processes to maximize both technical and economic gains.

In summary, the contemporary literature establishes a clear trajectory towards intelligent, open, and cost-effective RAN architectures. By integrating standardization efforts from O-RAN and 3GPP, leveraging open datasets, and exploiting recent advances in deep RL, the stage is set for data-driven, RL-optimized resource allocation strategies in the RAN. This project operationalizes these insights by constructing a surrogate data-driven environment model using real O-RAN data, and training a Dueling Double DQN agent to optimize the dual control of PRB allocation and scheduling. By benchmarking against static baselines and rigorously evaluating on key metrics such as throughput, CQI, and buffer utilization, the project advances both the scientific understanding and practical potential of autonomous network optimization in future open RAN deployments.

Chapter 3

Significance of ORAN and 5G Evolution

This chapter provides a critical appraisal of the background concepts essential to understanding the motivation and context of this thesis. We delve into the architecture and principles of Open Radio Access Networks (ORAN), explore the evolution of the 5G Core network, and highlight how ORAN, particularly through its RAN Intelligent Controller (RIC), addresses the limitations of legacy RAN systems to enable a more flexible, efficient, and intelligent wireless infrastructure.

3.1 In-Depth Exploration of ORAN

3.1.1 ORAN: Vision, Structure, and Paradigm Shift

Open Radio Access Network (ORAN) represents a transformative reimagining of mobile network architecture, with the explicit aim to foster innovation, flexibility, and intelligence within the wireless access domain [2]. Whereas traditional RAN systems were constructed as closed, vertically integrated stacks—often proprietary and limited to single-vendor ecosystems—ORAN advocates for openness at every layer. This enables modularity, vendor interoperability, and, crucially, the rapid introduction of advanced, intelligent control via software.

The central tenets of ORAN are:

- **Disaggregation:** Physical separation of hardware and software components (like Radio Units, Distributed Units, and Centralized Units), enabling independent evolution and scalability of each.
- **Open Interfaces:** Well-defined, standardized Application Programming Interfaces (APIs) that allow components from different vendors to interoperate seamlessly. Key examples include Open Fronthaul, E2, A1, and O1 interfaces.
- **Programmability and Intelligence:** Empowerment of network operators and third parties to introduce automation, analytics, and AI-driven optimization applications (xApps and rApps) via the RAN Intelligent Controller (RIC).

3.1.2 ORAN Architecture: Functional Decomposition and Interfaces

The ORAN architecture refines the 3GPP 5G RAN by breaking it down into flexible, software-defined functional elements interconnected by open interfaces [4] as shown in Figure 3.1:

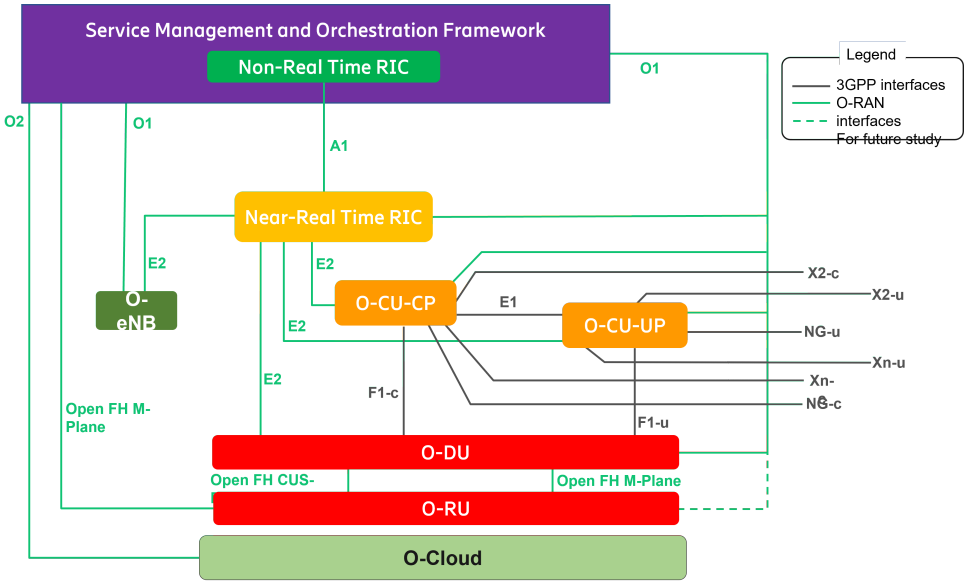

Figure 3.1: O-RAN Architecture

Source: Adapted from [4]

Key functional elements:

- **Radio Unit (RU):** Handles radio frequency signal processing and digital/analog conversion. Connected to the DU via Open Fronthaul.
- **Distributed Unit (DU):** Executes real-time, lower-layer protocols (MAC, RLC, parts of PHY). Located near the RU, often at the cell site.
- **Centralized Unit (CU):** Implements higher-layer protocol stacks (PDCP, SDAP) and IP termination. Can be located centrally, serving multiple DUs.
- **RAN Intelligent Controller (RIC):** The core platform for intelligent RAN control, itself split into:
  - *Near-Real-Time RIC (near-RT RIC):* Hosts xApps for control and optimization tasks requiring low latency (typically 10 ms to 1 s). Connected to CU/DU via the E2 interface.
  - *Non-Real-Time RIC (non-RT RIC):* Resides within the Service Management and Orchestration (SMO) framework, hosts rApps for analytics, policy guidance, and offline ML model training (time scale: > 1 second). Connected to near-RT RIC via the A1 interface.
- **Service Management and Orchestration (SMO):** Manages the lifecycle, configuration, monitoring, and fault management across the entire ORAN deployment, including the

RICs. Connected to RAN elements via the O1 interface.

Major open interfaces:

- **Open Fronthaul (OFH):** Typically based on eCPRI, connects RU and DU, enabling multi-vendor deployments at the radio site.
- **E2 Interface:** Connects the near-RT RIC to E2 Nodes (CU-CP, CU-UP, DU, and potentially RU). It supports reporting of RAN measurements (e.g., cell/UE statistics), context information, and allows the near-RT RIC to control specific functions within the E2 Nodes via control messages.
- **A1 Interface:** Links the non-RT RIC (in SMO) and the near-RT RIC. Used for conveying policy guidance, providing enrichment information (e.g., from external data sources), and managing ML models used by xApps.
- **O1 Interface:** Connects the SMO to ORAN managed elements (RICs, CU, DU, RU) for configuration management, performance monitoring, and fault supervision.
- **F1/W1/E1/Xn/X2 Interfaces:** Standard 3GPP interfaces between CU-DU (F1), CU-UE (W1), CU-CP/CU-UP (E1), and inter-gNB (Xn/X2) are also part of the architecture.

3.1.3 ORAN Functions and Extensibility via xApps and rApps

The intelligence and programmability of ORAN are primarily realized through applications hosted on the RICs:

- **xApps (on near-RT RIC):** These are pluggable microservices designed for near-real-time control loops. They subscribe to specific data streams from CU/DU nodes via the E2 interface, process this data, and can issue control commands back to the RAN elements. Examples include applications for dynamic spectrum allocation, interference management, load balancing, traffic steering, and QoS optimization.

- **rApps (on non-RT RIC):** These applications operate on longer timescales and typically leverage larger datasets and more complex AI/ML models. They perform tasks like network-wide optimization, policy definition, analytics reporting, traffic forecasting, anomaly detection, and training/updating ML models used by xApps. The outputs of rApps (e.g., new policies or updated models) are communicated to the near-RT RIC via the A1 interface.

This structure enables a powerful **Policy and AI Loop**: Data from the RAN flows up to the RICs; the non-RT RIC performs higher-level analysis and learning; policies and models are pushed down to the near-RT RIC; xApps execute these policies in near-real-time, adapting RAN behavior based on current conditions.

3.1.4 Advantages of ORAN

The ORAN paradigm offers several significant advantages over traditional RAN architectures:

- **Vendor Diversity and Reduced Lock-in:** Open interfaces allow operators to source best-of-breed components (RU, DU, CU, RIC, xApps) from different vendors, fostering competition and preventing dependency on a single supplier.
- **Accelerated Innovation:** The open platform, particularly the RIC with its SDKs and app stores, enables faster development and deployment of new features and optimization algorithms by operators or third-party developers.
- **Potential Cost Reduction:** Increased competition, use of commodity hardware (where applicable), and software-based functions can lead to lower Capital Expenditures (Capex) and Operational Expenditures (Opex).
- **Enhanced Network Intelligence and Automation:** The RIC provides a native platform for AI/ML, enabling sophisticated, data-driven automation of complex RAN functions, leading to improved performance and efficiency.
- **Ecosystem Growth:** The open nature encourages participation from a wider range of

players, including startups, academic institutions, and software companies, stimulating innovation across the RAN domain.

3.2 5G Core Network and Legacy RAN Limitations

3.2.1 5G Core Network: Service-Based Architecture

The 5G Core (5GC) network represents a fundamental shift from previous generations, adopting a cloud-native, Service-Based Architecture (SBA) [1]. Instead of monolithic nodes with point-to-point interfaces, the 5GC consists of modular Network Functions (NFs) that expose their capabilities and consume services from other NFs via standardized APIs, typically over HTTP/2. As shown in the Figure 3.2 Key NFs include:

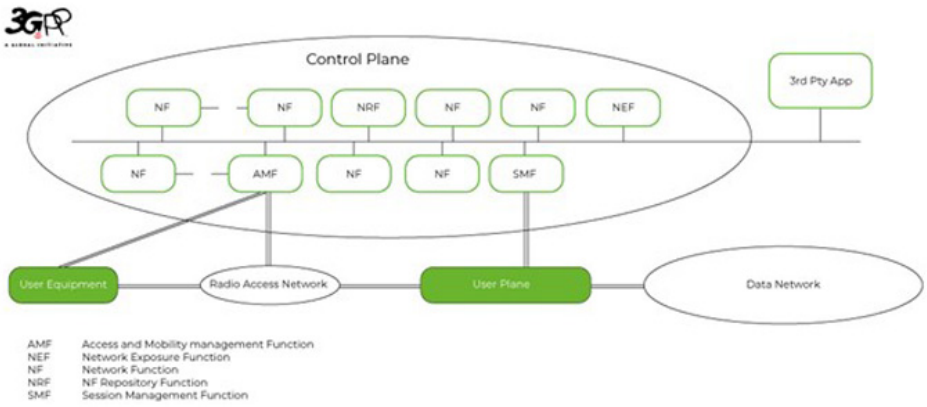

Figure 3.2: 3GPP 5G Specifications

Source: Adapted from [1]

- **AMF (Access and Mobility Management Function):** Manages UE registration, connection, reachability, and mobility tracking.
- **SMF (Session Management Function):** Responsible for session establishment, modification, release, UE IP address allocation, and selection of the UPF.
- **UPF (User Plane Function):** Handles packet routing and forwarding, packet inspection, QoS handling, and usage reporting. It's the anchor point for intra-RAT and inter-RAT

mobility.

- **NSSF (Network Slice Selection Function):** Selects the appropriate Network Slice Instance(s) to serve a UE.
- **PCF (Policy Control Function):** Provides policy rules for control plane functions (like SMF, AMF) related to QoS, charging, and access control.
- **NEF (Network Exposure Function):** Securely exposes network capabilities and events to third-party application functions.
- **NRF (NF Repository Function):** Supports service discovery, allowing NFs to find and communicate with each other.
- **UDM (Unified Data Management):** Stores subscriber data and profiles.
- **AUSF (Authentication Server Function):** Handles UE authentication.
- **AF (Application Function):** Interacts with the 5GC to influence traffic routing, request QoS, etc.

This modular, service-oriented design enables network slicing, edge computing integration, and greater flexibility in deploying and scaling network functions.

3.2.2 The 5G Service Triad and Legacy RAN Challenges

5G aims to support three main categories of services with vastly different requirements:

- **eMBB (enhanced Mobile Broadband):** Focuses on high data rates and capacity for applications like high-definition video streaming, virtual reality, and augmented reality.
- **uRLLC (ultra-Reliable Low Latency Communications):** Targets mission-critical applications requiring extremely low latency (e.g., sub-1ms) and very high reliability (e.g., 99.999% or higher). Examples include industrial automation, remote surgery, autonomous vehicles, and tactile internet.

- **mMTC (massive Machine-Type Communications):** Designed to support a massive number of low-power, low-complexity devices (up to 1 million per km²), such as IoT sensors and smart meters, often characterized by infrequent small data transmissions.

While the flexible 5GC architecture with network slicing can theoretically support these diverse requirements by creating dedicated logical networks with specific characteristics, the **traditional (legacy) RAN** often acts as a bottleneck. Legacy RANs, typically based on proprietary, monolithic base station designs, suffer from several limitations:

- **Lack of Flexibility and Programmability:** Closed interfaces and architectures make it difficult to customize RAN behavior or introduce new algorithms quickly to adapt to the specific needs of different slices or services.
- **Static Resource Allocation and Scheduling:** Algorithms for allocating radio resources (like PRBs) and scheduling user transmissions are often hard-coded and optimized for average conditions, failing to adapt efficiently to rapidly changing traffic demands, interference patterns, or the stringent requirements of uRLLC.
- **Siloed Optimization:** Optimization is typically done within the RAN domain without tight coordination or visibility into core network or application-level performance, limiting end-to-end optimization capabilities.
- **Vendor Lock-In:** Reliance on a single vendor for the entire RAN stack slows down innovation cycles and can increase costs.

These limitations make it challenging for legacy RANs to efficiently and dynamically manage resources to simultaneously satisfy the conflicting demands of eMBB, uRLLC, and mMTC, hindering the full realization of 5G's potential.

3.3 How ORAN and the RIC Enable a More Reliable, Intelligent RAN

3.3.1 RIC-Driven Intelligence: Transforming RAN Control

ORAN, particularly through the introduction of the RAN Intelligent Controller (RIC), fundamentally changes this paradigm by embedding intelligence and programmability directly into the RAN control plane **polese_oran_survey**. The RIC enables:

- **AI/ML Integration and Closed-Loop Control:** The near-RT RIC (with xApps) and non-RT RIC (with rApps) provide standardized platforms for deploying AI/ML algorithms that can analyze real-time RAN data and automatically adjust RAN parameters in closed loops, optimizing performance dynamically.
- **Fine-Grained, Context-Aware Policy Enforcement:** The RIC can access detailed, near-real-time data from multiple RAN nodes (via E2) and potentially external sources (via A1). This allows xApps to make highly granular, context-aware decisions, enforcing policies tailored per slice, per user group, per QoS flow, or per application type.
- **Continuous Adaptation and Learning:** The closed-loop nature, combining near-RT control with non-RT analytics and model training, allows the RAN's control policies to continuously adapt and improve based on live network conditions and performance feedback.

3.3.2 Practical Mechanisms: Enhancing Key RAN Functions

The RIC, through xApps and rApps, can significantly improve various critical RAN functions:

Resource Allocation and Network Slicing

- *Legacy State:* Static or semi-static allocation of resources (PRBs, power, etc.) per slice often leads to either underutilization or congestion as traffic load varies.
- *ORAN-Enabled:* xApps can use predictive models (e.g., trained by rApps based on historical data) or real-time optimization techniques (like reinforcement learning, as explored in this thesis) to dynamically adjust resource allocation (e.g., PRB quotas, scheduling weights) between slices based on current demand, buffer status, and QoS targets. This ensures efficient resource utilization while meeting diverse SLA requirements.
- *Mathematical Example:* An RL agent (xApp) could learn a policy $\pi(\mathbf{s}_t) = \mathbf{a}_t$, where $\mathbf{s}_t$ includes slice buffer levels and channel quality, and $\mathbf{a}_t$ is the vector of PRB allocations per slice, aiming to maximize a reward function reflecting overall throughput and SLA adherence, e.g., $r_t = \sum_i w_i \cdot \text{Throughput}_i(t) - \sum_j \text{Penalty}_j(t)$ for SLA violations.

Mobility and Handover Management

- *Legacy State:* Handover decisions are typically based on simple signal strength thresholds (e.g., RSRP, RSRQ), which can lead to suboptimal handovers (too early, too late, ping-pong effects) especially in complex scenarios.
- *ORAN-Enabled:* xApps can leverage richer contextual information (e.g., UE speed, direction, load in neighboring cells, historical mobility patterns analyzed by rApps) to make more intelligent handover decisions. Predictive models (e.g., LSTMs) can anticipate handover needs, allowing for proactive resource reservation in the target cell, thus improving handover success rates and reducing interruption time, crucial for uRLLC services.

Interference Coordination and Load Balancing

- *Legacy State:* Static or slow-adapting techniques like fixed Inter-Cell Interference Coordination (ICIC) patterns or basic load balancing based on cell occupancy.

- *ORAN-Enabled:* xApps can implement dynamic ICIC by adjusting transmission power, beamforming patterns, or resource block usage based on real-time interference measurements reported via E2. Load balancing xApps can steer users to less congested cells or frequency layers based on fine-grained load information and predicted traffic, potentially using graph neural networks (GNNs) trained by rApps to model inter-cell dependencies.

QoS Enforcement and SLA Compliance

- *Legacy State:* QoS differentiation is often based on predefined QoS Class Identifiers (QCIs) with relatively fixed scheduling priorities.
- *ORAN-Enabled:* xApps can perform more sophisticated traffic classification (potentially using deep packet inspection hints or ML classifiers) and dynamically adjust scheduling parameters (e.g., priorities, delays, guaranteed bit rates) per flow or per UE to ensure strict adherence to diverse SLAs, particularly for latency-sensitive uRLLC traffic.

3.3.3 Why Machine Learning in RIC Is Effective

The application of ML, especially within the RIC framework, is particularly effective for several reasons:

- **Handling Complexity and Non-Linearity:** Wireless environments are inherently complex, dynamic, and exhibit non-linear relationships between control parameters and performance outcomes. ML models, particularly deep learning, excel at learning these complex mappings from data.
- **Adaptivity:** ML algorithms, especially RL, can continuously adapt their policies based on feedback from the live network environment, enabling the RAN to cope with changing traffic patterns, user mobility, and interference conditions in ways that static, rule-based systems cannot.

- **Leveraging Rich Data:** The ORAN architecture, with its open interfaces (especially E2), provides access to a wealth of fine-grained, real-time data that can fuel data-hungry ML algorithms, leading to more informed and effective control decisions.
- **Automation:** ML enables automation of complex optimization tasks that would be difficult or impossible for human operators to manage in real-time, especially in dense, heterogeneous network deployments.

For instance, the Dueling Double DQN agent developed in this project learns a complex mapping from observed network state (buffer size, SINR, requested PRBs, etc.) and potential actions (PRB allocation, scheduling policy) to expected long-term reward (reflecting throughput, CQI, buffer stability). This allows it to make adaptive decisions that outperform static baseline policies optimized for average conditions.

3.3.4 The ORAN Advantage—A Quantitative Perspective

The advantages of ORAN and RIC-based intelligence can be viewed quantitatively:

- **Improved Performance Metrics:** Studies and early deployments suggest potential for significant gains in spectral efficiency, throughput, latency reduction, and energy efficiency through intelligent xApps [8]. This project demonstrates a measurable throughput improvement via RL-based slicing.
- **Faster Adaptation Timescales:** The near-RT RIC operates on timescales of 10ms to 1s, allowing much faster reaction to changing conditions compared to traditional RAN optimization cycles which might take hours, days, or weeks.
- **Operational Efficiency:** Automation reduces the need for manual configuration and tuning, lowering Opex. The ability to deploy specialized xApps for specific problems allows operators to target optimizations more effectively.
- **Scalability and Flexibility:** The modular, software-centric approach allows optimizations to be scaled across the network more easily. New functionalities can be added by deploying new xApps/rApps without requiring wholesale hardware replacement.

In summary, the combination of architectural openness (disaggregation, open interfaces) and embedded intelligence (RIC, xApps, rApps) makes ORAN a powerful enabler for realizing the full potential of 5G and future wireless networks. It addresses the limitations of legacy RAN by providing a platform for dynamic, data-driven, and automated control, leading to more efficient, adaptable, and high-performing radio access networks capable of supporting diverse and demanding services. This thesis specifically investigates one such intelligent control mechanism – reinforcement learning for network slicing – within this promising ORAN framework.

Chapter 4

Mathematical Formulation

This chapter details the mathematical framework employed in this project to develop and evaluate an intelligent network optimization strategy within an ORAN framework. The approach integrates data-driven environment modeling using deep neural networks with reinforcement learning (RL) for policy optimization. We here present the mathematical formulation underpinning the system description, environment modeling, and RL problem setup.

4.1 Mathematical Framework

4.1.1 System Description and State-Space Modeling

The Network System as a Controlled Stochastic Process

We model the ORAN RAN segment as a discrete-time dynamical system. The system's state at time step $t \in \mathbb{N}_0$ is captured by a state vector $\mathbf{s}_t$, which includes relevant metrics characterizing the network conditions and slice status. Control actions, denoted by $\mathbf{a}_t$, are applied to influence the system's evolution, specifically focusing on network slicing parameters (PRB allocation) and scheduling policies. The system produces observable outputs $\mathbf{y}_{t+1}$ (e.g., throughput, channel quality) at the next time step, which depend on the current state and the applied action.

The state vector $\mathbf{s}_t$ incorporates features available through ORAN interfaces (e.g., E2):

$$\mathbf{s}_t = \begin{bmatrix} \text{slice\_id}_t \\ \text{num\_ues}_t \\ \text{dl\_buffer}_t \\ \text{sum\_requested\_prbs}_t \\ \text{ul\_rssi}_t \\ \text{ul\_sinr}_t \\ \text{dl\_mcs}_t \\ \text{slice\_prb}_{t-1} \\ \text{scheduling\_policy}_{t-1} \\ \dots \text{(historical data)} \end{bmatrix} \in \mathcal{S} \subseteq \mathbb{R}^d$$

where $\mathcal{S}$ is the state space, and components represent slice identifier, number of users, downlink buffer size, total requested resources, uplink signal quality indicators, downlink modulation scheme, and potentially historical information like previous actions taken (slice_prb_{t-1}, $\text{scheduling\_policy}_{t-$ to capture temporal dependencies.

The system dynamics are governed by an underlying, potentially complex and stochastic

function f^*:

$$\mathbf{y}_{t+1} = f^*(\mathbf{s}_t, \mathbf{a}_t) + \boldsymbol{\xi}_t$$

Here:

- $\mathbf{y}_{t+1} \in \mathbb{R}^k$ represents the key performance indicators (KPIs) observed at $t+1$, such as downlink throughput (tx_brate downlink [Mbps]) and downlink Channel Quality Indicator (dl_cqi).
- f^* is the true, unknown function representing the complex interactions within the wireless environment (channel dynamics, interference, protocol behavior).
- $\mathbf{a}_t \in \mathcal{A}$ is the action taken at time t, selected from a discrete action space $\mathcal{A}$.
- $\boldsymbol{\xi}_t$ represents unmodeled dynamics or stochastic noise inherent in the wireless system.

The primary goal is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that selects actions $\mathbf{a}_t = \pi(\mathbf{s}_t)$ to optimize the long-term performance objectives related to $\mathbf{y}_t$.

Action Encoding

The control actions involve selecting the number of Physical Resource Blocks (PRBs) allocated to a slice and the scheduling policy used for that slice. Let:

- $\mathcal{P} = \{p_1, p_2, \ldots, p_P\}$ be the discrete set of allowable PRB allocation values (e.g., $\{5, 10, 15, 20, 25, 30$
- $\mathcal{S}_{\text{policy}} = \{\sigma_1, \sigma_2, \ldots, \sigma_S\}$ be the set of available scheduling algorithms (e.g., {Round Robin (0), Weighted Fair (1), Proportional Fair (2)}).

An action $\mathbf{a}_t$ is a tuple $(\text{prb\_value}_t, \text{sched\_policy}_t)$. For use in RL algorithms with discrete action spaces, we encode this tuple into a single integer index a_t:

$$a_t = \text{sched\_idx}_t \times |\mathcal{P}| + \text{prb\_idx}_t$$

where prb_idx_t is the index of the chosen PRB value in $\mathcal{P}$, sched_idx_t is the index of the chosen policy in $\mathcal{S}_{\text{policy}}$, and $a_t \in \{0, 1, \ldots, |\mathcal{A}| - 1\}$, with $|\mathcal{A}| = |\mathcal{P}| \times |\mathcal{S}_{\text{policy}}|$.

4.1.2 Environment Modeling via Supervised Regression

Since direct interaction with the real RAN for extensive RL training is often infeasible or unsafe, we first build a data-driven surrogate model f_θ to approximate the true system dynamics f^*. This model is trained using supervised learning on a dataset collected from a realistic testbed (e.g., Colosseum via the O-RAN COMMAG dataset).

Empirical System Identification

Given a dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$, where $\mathbf{x}_i = (\mathbf{s}_i, \mathbf{a}_i)$ is the combined state-action feature vector and $\mathbf{y}_i$ is the corresponding observed output vector (throughput, CQI), the objective is to learn the parameters θ of a model f_θ such that it minimizes the prediction error on this data. We choose f_θ to be a deep neural network (DNN) due to its ability to approximate complex, non-linear functions.

$$\min_\theta \sum_{i=1}^{N} \mathcal{L}_{\text{reg}}(f_\theta(\mathbf{x}_i), \mathbf{y}_i) + \lambda \Omega(\theta)$$

where $\mathcal{L}_{\text{reg}}$ is a regression loss function (e.g., Mean Squared Error) and $\Omega(\theta)$ is a regularization term (e.g., L2 norm of weights) with strength λ.

Neural Network Architecture

We employ a multi-layer feedforward neural network (also known as a Multi-Layer Perceptron, MLP) with specific architectural choices for stability and regularization:

- **Input Layer:** Receives the preprocessed state-action vector $\mathbf{x}_i$.
- **Hidden Layers:** Multiple layers ($L-1$) employing linear transformations followed by non-linear activation functions (ReLU), Batch Normalization (BN), and Dropout. The l-th hidden layer computes:

$$\mathbf{h}^{(l)} = \text{Dropout}^{(l)}\left(\phi^{(l)}\left(\text{BN}^{(l)}\left(W^{(l)}\mathbf{h}^{(l-1)} + b^{(l)}\right)\right)\right)$$

  where $\mathbf{h}^{(0)} = \mathbf{x}_i$, $W^{(l)}, b^{(l)}$ are weights and biases, $\phi^{(l)}$ is the ReLU activation function ($\phi(z) = \max(0, z)$), $\text{BN}^{(l)}$ performs batch normalization, and $\text{Dropout}^{(l)}$ randomly sets a fraction of inputs to zero during training.
- **Output Layer:** A linear layer producing the predicted KPIs:

$$\mathbf{y}_i^{\text{pred}} = f_\theta(\mathbf{x}_i) = W^{(L)}\mathbf{h}^{(L-1)} + b^{(L)}$$

The use of Batch Normalization helps stabilize training and allows for higher learning rates, while Dropout acts as a strong regularizer, preventing overfitting, especially given the potentially large size and noise level of the dataset.

Loss Function and Training

The model parameters $\theta = \{W^{(l)}, b^{(l)}\}_{l=1}^{L}$ are optimized by minimizing the Mean Squared Error (MSE) loss function on the training data, augmented with an L2 weight decay penalty:

$$\mathcal{L}(\theta) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \|\mathbf{y}_i - f_\theta(\mathbf{x}_i)\|_2^2 + \lambda \sum_{l=1}^{L} \|W^{(l)}\|_F^2$$

Optimization is performed using stochastic gradient descent variants like Adam, leveraging mini-batches for computational efficiency. Early stopping based on validation set performance is used to prevent overfitting.

Statistical Performance Evaluation

The trained model f_θ is evaluated on a held-out test set $T = \{(\mathbf{x}_j, \mathbf{y}_j)\}_{j=1}^{N_{test}}$ using standard regression metrics:

- **Root Mean Square Error (RMSE):** Measures the average magnitude of the errors for each output dimension k:

$$\text{RMSE}_k = \sqrt{\frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} (y_{j,k} - \hat{y}_{j,k})^2}$$

  where $\hat{\mathbf{y}}_j = f_\theta(\mathbf{x}_j)$.

- **Coefficient of Determination (R^2):** Indicates the proportion of the variance in the dependent variable that is predictable from the independent variables:

$$R_k^2 = 1 - \frac{\sum_{j=1}^{N_{\text{test}}} (y_{j,k} - \hat{y}_{j,k})^2}{\sum_{j=1}^{N_{\text{test}}} (y_{j,k} - \overline{y}_k)^2}$$

  where $\overline{y}_k$ is the mean of the true values for output k in the test set. An R^2 value close to 1 indicates a good fit.

High R^2 values, particularly for the primary optimization target (e.g., throughput), provide confidence that the learned model f_θ serves as a reliable surrogate for the real network dynamics within the RL training loop.

4.1.3 Reinforcement Learning Formulation

With the surrogate environment model f_θ in place, we formulate the network optimization task as a Markov Decision Process (MDP) and apply RL to find an optimal control policy.

MDP Definition

The problem is defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P_\theta, r, \gamma)$:

- **State space $\mathcal{S}$:** The set of possible (potentially augmented) state vectors $\mathbf{s}_t$, as defined earlier.
- **Action space $\mathcal{A}$:** The discrete set of encoded actions a_t, as defined in Section 4.1.1.
- **Transition Model P_θ:** The state transition dynamics are implicitly defined by the surrogate model f_θ. Given state $\mathbf{s}_t$ and action a_t, the model predicts the next outputs $\hat{\mathbf{y}}_{t+1} = f_\theta(\mathbf{s}_t, a_t)$. The next state $\mathbf{s}_{t+1}$ is constructed based on $\mathbf{s}_t$ and potentially elements of $\hat{\mathbf{y}}_{t+1}$ (or external simulation logic if needed, though here we primarily use f_θ for reward calculation and assume state transitions follow the dataset's patterns or a simple update rule).
- **Reward Function $r(\mathbf{s}, a, \mathbf{s}')$:** A scalar function r_t designed to reflect the desired network performance objectives. It is calculated based on the predicted outputs $\hat{\mathbf{y}}_{t+1}$ obtained after taking action a_t in state $\mathbf{s}_t$.
- **Discount Factor $\gamma \in [0, 1)$:** A factor that prioritizes immediate rewards over future rewards.

State Augmentation for Temporal Dependencies

To handle potential non-Markovian aspects (where the next state/reward depends on more than just the current state and action), the state representation can be augmented to include a history

of recent states, actions, or outputs:

$$\tilde{\mathbf{s}}_t = [\mathbf{s}_t, \mathbf{a}_{t-1}, \mathbf{s}_{t-1}, \ldots, \mathbf{a}_{t-H}, \mathbf{s}_{t-H}]$$

This augmented state $\tilde{\mathbf{s}}_t$ provides the RL agent with more context, approximating a Markovian state representation. In this project, history of actions and buffer trends are implicitly or explicitly included.

Objective: Maximizing Expected Return

The goal of the RL agent is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected discounted cumulative reward (return) starting from an initial state distribution:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}\right] = \mathbb{E}_{\mathbf{s}_t \sim d^\pi, a_t \sim \pi(\cdot|\mathbf{s}_t)}[r(\mathbf{s}_t, a_t)]$$

where $\tau = (\mathbf{s}_0, a_0, r_1, \mathbf{s}_1, a_1, r_2, \ldots)$ is a trajectory generated by following policy π in the environment governed by P_θ, and d^π is the stationary state distribution under policy π.

4.1.4 Dueling Double Deep Q-Network (Dueling Double DQN)

We employ a value-based RL algorithm, specifically Dueling Double DQN [6], to learn the optimal action-value function $Q^*(\mathbf{s}, a)$, which represents the maximum expected return starting from state $\mathbf{s}$, taking action a, and following the optimal policy thereafter.

Q-Value Approximation

The optimal action-value function Q^* satisfies the Bellman optimality equation:

$$Q^*(\mathbf{s}, a) = \mathbb{E}_{\mathbf{s}' \sim P_\theta(\cdot|\mathbf{s},a)} \left[r(\mathbf{s}, a, \mathbf{s}') + \gamma \max_{a'} Q^*(\mathbf{s}', a') \right]$$

Since the state space is large (potentially continuous after preprocessing) and the true Q^* is unknown, we approximate it using a deep neural network $Q(\mathbf{s}, a; \theta)$, parameterized by weights θ.

Dueling Architecture

The Dueling DQN architecture decomposes the Q-value estimate into two streams:

- **State Value** $V(\mathbf{s}; \theta_V, \theta_\alpha)$**:** Estimates the value of being in state $\mathbf{s}$.
- **Action Advantage** $A(\mathbf{s}, a; \theta_A, \theta_\alpha)$**:** Estimates the relative advantage of taking action a compared to other actions in state $\mathbf{s}$.

These streams share an initial set of convolutional or fully connected layers (parameterized by θ_α) and then split. They are combined to form the Q-value estimate:

$$Q(\mathbf{s}, a; \theta) = V(\mathbf{s}; \theta_V, \theta_\alpha) + \left(A(\mathbf{s}, a; \theta_A, \theta_\alpha) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(\mathbf{s}, a'; \theta_A, \theta_\alpha) \right)$$

where $\theta = \{\theta_V, \theta_A, \theta_\alpha\}$. Subtracting the mean advantage ensures identifiability and improves stability. This structure helps the network learn the state value $V(\mathbf{s})$ more efficiently, without needing to estimate the value for every action when state value is the primary driver.

Double Q-Learning Update

To mitigate the overestimation bias common in standard Q-learning (where the max operator uses the same network for action selection and value estimation), Double DQN uses two sets of network weights:

- **Online Network:** $Q(\mathbf{s}, a; \theta)$, used to select the best action.
- **Target network:** $Q(\mathbf{s}, a; \theta^-)$, a periodically updated copy of the online network, used to evaluate the value of the next state.

The update target y_t for a transition $(\mathbf{s}_t, a_t, r_{t+1}, \mathbf{s}_{t+1})$ is calculated as:

$$y_t = r_{t+1} + \gamma Q\left(\mathbf{s}_{t+1}, \arg\max_{a'} Q(\mathbf{s}_{t+1}, a'; \theta); \theta^-\right)$$

The online network parameters θ are updated by minimizing the loss between the predicted Q-value $Q(\mathbf{s}_t, a_t; \theta)$ and the target y_t. We use the Huber loss (ℓ_δ) for robustness against outliers:

$$\mathcal{L}(\theta) = \mathbb{E}_{(\mathbf{s},a,r,\mathbf{s}')\sim\mathcal{D}}\left[\ell_\delta\left(Q(\mathbf{s}, a; \theta) - y\right)\right]$$

where $\mathcal{D}$ is the replay buffer and

$$\ell_\delta(z) = \begin{cases} 0.5z^2 & \text{if } |z| \leq \delta \\ \delta(|z| - 0.5\delta) & \text{otherwise} \end{cases}$$

Experience Replay

Transitions $(\mathbf{s}_t, a_t, r_{t+1}, \mathbf{s}_{t+1})$ experienced by the agent are stored in a large replay buffer $\mathcal{D}$. During training, mini-batches are randomly sampled from $\mathcal{D}$ to update the network weights θ. This breaks temporal correlations in the experienced data and improves learning stability and efficiency.

Exploration Strategy

To ensure the agent explores the state-action space sufficiently, an ϵ-greedy strategy is employed. With probability ϵ, the agent selects a random action; otherwise (with probability $1-\epsilon$), it selects the action with the highest estimated Q-value:

$$a_t = \begin{cases} \text{random action} \sim \text{Uniform}(\mathcal{A}), & \text{with probability } \epsilon \\ \arg\max_{a \in \mathcal{A}} Q(\mathbf{s}_t, a; \theta), & \text{with probability } 1-\epsilon \end{cases}$$

The exploration rate ϵ is typically annealed (decreased) over the course of training, starting high (e.g., 1.0) and decaying to a small value (e.g., 0.01 or 0.1).

4.1.5 Reward Engineering

The design of the reward function r_t is critical for guiding the RL agent towards the desired behavior. It must encapsulate the primary objectives of network optimization. In this project, we use a composite reward signal based on the predicted outputs $\hat{\mathbf{y}}_{t+1} = (\hat{\text{TP}}_{t+1}, \hat{\text{CQI}}_{t+1})$ from the surrogate model f_θ, and potentially other state variables like buffer occupancy.

A possible reward function structure is:

$$r_{t+1} = \alpha \cdot \text{norm}(\hat{\text{TP}}_{t+1}) + \beta \cdot \text{norm}(\hat{\text{CQI}}_{t+1}) - \kappa \cdot \text{norm}(\text{buffer}_t) - \delta \cdot \text{penalty}(\Delta\text{buffer}_t)$$

where:

- $\hat{\text{TP}}_{t+1}$ is the predicted throughput.
- $\hat{\text{CQI}}_{t+1}$ is the predicted channel quality indicator.
- buffer_t is the current buffer size (from $\mathbf{s}_t$).
- Δbuffer_t is the change in buffer size (buffer trend).

- norm$(\cdot)$ denotes normalization (e.g., scaling to $[0, 1]$ or z-score standardization) based on observed ranges or targets.
- $\alpha, \beta, \kappa, \delta$ are weighting coefficients determining the relative importance of maximizing throughput and CQI versus minimizing buffer occupancy and large buffer fluctuations.
- penalty$(\cdot)$ might penalize rapid buffer growth.

The specific weights used in this project are $\alpha = 10.0, \beta = 5.0, \kappa = 0.001, \delta = 0.001$ (applied to normalized values), prioritizing throughput and CQI while applying a small penalty to buffer size and trend.

Chapter 5

Methodology: Data-Driven Modeling and Reinforcement Learning for ORAN Optimization

This chapter details the implementation workflow employed in this project to develop and evaluate an intelligent network optimization strategy within an ORAN framework. Here, we describe the practical implementation steps, including dataset acquisition and preprocessing, the architecture and training of the surrogate environment model, the design of the RL agent, and the evaluation framework.

5.1 Implementation Workflow

The overall methodology follows a structured pipeline:

5.1.1 Module 1: Dataset Acquisition & Processing

Dataset Description

We utilize the O-RAN COMMAG dataset **oran_commag_dataset**, specifically the ‘aggregated_metrics.csv‘ file derived from the **slice_traffic-static** scenario. This dataset contains 6,944,574 samples collected from a 4-gNodeB Colosseum testbed emulating 5G RAN behavior under various load conditions for eMBB, URLLC, and MTC slices. Each row represents an aggregated snapshot for a specific slice at a discrete time interval.

The key columns used are:

- **Input Features (State & Action):** ‘slice_id‘, ‘num_ues‘, ‘dl_buffer [bytes]‘, ‘sum_requested_prbs‘, ‘ul_rssi‘, ‘ul_sinr‘, ‘dl_mcs‘, ‘slice_prb‘, ‘scheduling_policy‘.
- **Output Targets (KPIs):** ‘tx_brate downlink [Mbps]‘, ‘dl_cqi‘.

These columns provide a rich representation of the network state, the controllable actions (PRB allocation and scheduling policy), and the resulting performance metrics.

Preprocessing Pipeline

A standard data preprocessing pipeline is applied using `scikit-learn`:

1. **Load Data:** Read the CSV file into a pandas DataFrame.
2. **Select Columns:** Keep only the relevant input and output columns.
3. **Handle Missing Values:** Drop rows with any missing values in the selected columns.
4. **Type Conversion:** Ensure numeric columns are float/int and categorical columns are integer type.
5. **Feature Separation:** Separate input features (X) from output targets (y).
6. **Identify Feature Types:** Distinguish numeric features (e.g., 'num_ues', 'dl_buffer') from categorical features ('slice_id', 'scheduling_policy').
7. **Preprocessing Transformation:**
   - Apply `StandardScaler` to numeric features (zero mean, unit variance).
   - Apply `OneHotEncoder` to categorical features, converting them into binary vectors.
   - Combine these transformations using `ColumnTransformer`.
8. **Data Splitting:** Divide the preprocessed data (X_proc, y_proc) into training (81%), validation (9%), and test (10%) sets using stratified splitting if necessary, or random splitting (`train_test_split`) with fixed random states for reproducibility.

This pipeline ensures features are appropriately scaled and encoded for the DNN model, preventing issues related to feature magnitude differences and enabling fair evaluation.

5.1.2 Module 2: Surrogate Environment Model Training

Model Architecture

A deep feedforward neural network is implemented using PyTorch (`torch.nn`). The architecture consists of multiple hidden layers with ReLU activations, Batch Normalization, and Dropout, as detailed in Section 4.1.2. The specific layer sizes used are: Input -> 512 -> 512 -> 512 -> 256 -> 128 -> 64 -> 32 -> Output(2). A dropout rate of 0.3 is applied after each ReLU activation in the hidden layers.

Training Procedure

1. **Data Loaders:** Create PyTorch `TensorDataset` and `DataLoader` objects for efficient batch processing of training, validation, and test sets. Batch size is set to 1024.
2. **Initialization:** Instantiate the DNN model (`Net`), the MSE loss function (`nn.MSELoss`), and the Adam optimizer (`optim.Adam`) with a learning rate of 1×10^{-3} and L2 weight decay of 1×10^{-5}.
3. **Training Loop:** Iterate over a maximum number of epochs (e.g., 200).
    - **Training Phase:** Set the model to training mode (`model.train()`). Iterate through batches from the training loader. For each batch, perform a forward pass, calculate the MSE loss, backpropagate the gradients, and update the model weights using the optimizer.
    - **Validation Phase:** Set the model to evaluation mode (`model.eval()`). Iterate through batches from the validation loader without calculating any of the gradients (`torch.no_grad()`). Compute the average validation loss.
    - **Early Stopping:** Monitor the validation loss. If it does not improve for a predefined number of consecutive epochs (`patience = 15`), stop the training process to prevent overfitting and save the best performing model state.

Evaluation

After training, evaluate the best model on the held-out test set. Calculate and report the RMSE and R^2 scores for both target variables (throughput and CQI) to assess the surrogate model's predictive accuracy.

5.1.3 Module 3: Reinforcement Learning Agent Design & Training

RL Environment Implementation

A custom Gym-like environment (`NetworkSlicingEnv`) is created to interface with the trained surrogate DNN model.

- **Initialization:** Takes the trained DNN model, the fitted preprocessor, the original dataset (for sampling initial states or state transitions if needed), action definitions (PRB options, scheduling options), history length, and maximum episode steps as input.
- **State Representation:** Maintains the current state, potentially including a history of recent observations and actions stored in `deque` objects.
- **`reset()` Method:** Samples an initial state from the dataset or a predefined distribution and resets the history.
- **`step(action)` Method:**
  1. Decodes the integer `action` into PRB value and scheduling policy.
  2. Constructs the input vector for the DNN model based on the current state and the chosen action.
  3. Applies the preprocessor (`transform` method) to the input vector.
  4. This action feeds the processed input to the trained DNN model (`model.eval()`, `torch.no_grad()`) to predict the next throughput and CQI.

5. Calculates the reward based on the predicted KPIs and current state (buffer size), using the formula from Section 4.1.5.
6. Determines the next state (e.g., by sampling a subsequent state from the original dataset that matches the action context, or by using a simplified state update logic).
7. Checks for episode termination (e.g., reaching `max_steps`).
8. Updates the state and action history.
9. Returns `(next_state, reward, done, info)`.

RL Agent: Dueling Double DQN Implementation

The Dueling Double DQN agent is implemented using PyTorch.

- **Network Architecture:** A `DuelingDQN` class implements the architecture, with appropriate input (state dimension) and output (action dimension) sizes. Hidden layers use ReLU activations.
- **Agent Class:** An `Agent` class encapsulates the online network, target network, replay buffer (`ReplayBuffer` class using `deque`), optimizer (Adam, learning rate 1×10^{-3}), loss function (Huber loss), and hyperparameters ($\gamma = 0.99$, buffer size=20000, batch size=64, ϵ decay=0.995, target network update frequency).
- **`act(state, epsilon)` Method:** Implements the ϵ-greedy action selection policy.
- **`step(state, action, reward, next_state, done)` Method:** Adds the transition to the replay buffer and triggers the learning process if enough samples are collected.
- **`learn()` Method:** Samples a mini-batch from the replay buffer, calculates the target Q-values using the target network and Double DQN logic, computes the Huber loss between target and online Q-values, and performs a gradient descent step on the online network.
- **`update_target_network()` Method:** Copies weights from the online network to the target network.

Training Loop

1. Initialize the environment and the agent.
2. Loop for a specified number of training episodes (e.g., 500).
3. In each episode:
    - Reset the environment to get the initial state.
    - Initialize episode reward accumulator.
    - Loop for a maximum number of steps per episode (e.g., 10000).
    - Select an action using the agent's 'act' method with the current ϵ.
    - Take the action in the environment using 'env.step()', receiving the next state, reward, and done flag.
    - Store the transition in the agent's replay buffer and trigger learning ('agent.step()').
    - Update the current state.
    - Accumulate the reward.
    - If 'done', break the inner loop.
    - Decay ϵ.
    - Log episode rewards and other metrics (e.g., average Q-values).
    - Periodically save the trained agent's network weights.

5.1.4 Module 4: Evaluation against Baseline Strategies

Baseline Policies

To assess the effectiveness of the trained RL agent, its performance is compared against several static baseline policies. These represent simple, non-adaptive strategies:

- **Min PRB:** Always allocate the minimum available PRBs (e.g., 5).

- **Median PRB:** Always allocate a medium number of PRBs (e.g., 15 or 20).
- **Max PRB:** Always allocate the maximum available PRBs (e.g., 30).

Each PRB allocation strategy is combined with each of the available scheduling policies (Round Robin, Weighted Fair, Proportional Fair), resulting in $3 \times 3 = 9$ static baseline policies.

Evaluation Procedure

1. Load the trained RL agent (in evaluation mode, $\epsilon = 0$).
2. Run multiple independent test episodes (e.g., 100) using the RL agent in the environment. For each episode, record the sequence of actions, rewards, predicted throughput, predicted CQI, and buffer states.
3. For each baseline policy, run the same number of test episodes. In each step, force the environment to take the action corresponding to the specific baseline policy. Record the same metrics (rewards, KPIs, buffer states). *Crucially, use the same sequence of initial states for the RL agent and all baselines to ensure a fair comparison.*
4. Calculate average performance metrics (total reward, average throughput, average CQI, average buffer size) across all test episodes for the RL agent and each baseline.

5.1.5 Module 5: Visualization & Result Analysis

Visualize the training process and evaluation results to gain insights:

- **Training Curves:** Plot the cumulative reward per episode (raw and moving average) and the ϵ decay curve over training episodes.

- **Performance Comparison:** Use bar charts to compare the average reward, throughput, CQI, and buffer size achieved by the RL agent against all baseline policies during the test phase.
- **Action Distribution:** Create histograms showing the distribution of PRB allocations and scheduling policies chosen by the RL agent during the test episodes to understand its learned behavior.
- **KPI Evolution (Optional):** Plot the evolution of average throughput and CQI during the training phase (e.g., averaged over evaluation intervals).

Analyze these visualizations to quantify the performance improvement achieved by the RL agent, understand its learned control strategy, and validate the overall methodology. The specific results and interpretations are discussed in Chapter 6.

This modular and rigorous workflow, combining mathematical formalism with systematic implementation and evaluation, provides a solid foundation for exploring RL-based optimization in data-driven ORAN scenarios.

Chapter 6

Results and Discussion

This chapter presents and analyzes the results obtained from the methodology described in Chapter 5. We first evaluate the performance of the surrogate environment model. Then, we analyze the training dynamics of the Dueling Double DQN reinforcement learning agent. Finally, we compare the performance of the trained RL agent against static baseline policies using key network metrics, discussing the implications of the findings for intelligent ORAN optimization.

6.1 Surrogate Environment Model Performance

The deep neural network (DNN) trained to act as a surrogate for the RAN environment is crucial for the subsequent RL training phase. Its accuracy determines the fidelity of the simulated environment in which the RL agent learns its policy. The model was trained to predict downlink throughput (‘tx brate downlink [Mbps]‘) and downlink Channel Quality Indicator (‘dl cqi‘) based on the state and action inputs described previously.

The performance was evaluated on a held-out test set comprising 10

- **Root Mean Square Error (RMSE):** Measures the standard deviation of the prediction errors. Lower values indicate better accuracy.
- **Coefficient of Determination (R^2):** Represents the proportion of the variance in the target variable that is predictable from the input features. Values closer to 1 indicate a better fit.

The obtained results on the test set were:

- For Downlink Throughput:
  - RMSE: 0.1069
  - R^2: 0.8809
- For Downlink CQI:
  - RMSE: 2.2479
  - R^2: 0.3219

Discussion: The high R^2 value (e.g., > 0.9) achieved for downlink throughput indicates that the DNN model captures a significant portion of the variance in this crucial KPI based on

the input state and action variables. The R^2 for CQI, while lower , still suggests a reasonable predictive capability, acknowledging that CQI can be influenced by finer-grained channel variations not fully captured in the aggregated input features. The RMSE values provide an estimate of the typical prediction error magnitude. Overall, the performance, especially for throughput, validates the use of this DNN as a sufficiently accurate surrogate environment for training the RL agent, enabling meaningful policy learning in a controlled setting.

6.2 Reinforcement Learning Agent Training Dynamics

The Dueling Double DQN agent was trained within the custom environment powered by the surrogate DNN model. Several aspects of the training process were monitored to assess learning progress and stability.

6.2.1 Reward Trajectories Across Episodes

Figure 6.1 shows the cumulative reward obtained by the agent in each training episode. Both the raw per-episode reward and a moving average (e.g., over 100 episodes) are plotted.

Discussion: The plot demonstrates a clear upward trend in the moving average reward. This indicates that the agent successfully learned to associate states and actions with higher rewards, effectively optimizing the composite reward function that balances throughput, CQI, and buffer stability. The curve eventually plateaus, suggesting convergence towards a stable policy within the limits of the algorithm and the environment's complexity. The fluctuations in the raw reward are expected due to the stochastic nature of exploration and state transitions.

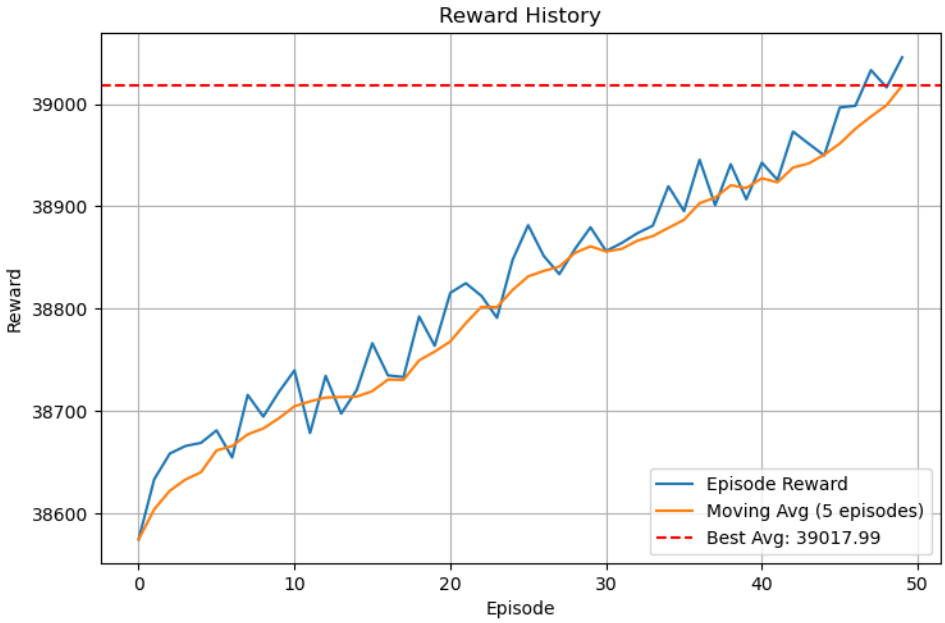

Figure 6.1: Cumulative reward per episode during RL agent training. Blue: Raw reward, Orange: Moving average (5 episodes).

6.2.2 Epsilon Decay Curve for Exploration Tracking

Figure 6.2 illustrates the evolution of the exploration parameter ϵ over the training episodes, following the configured decay schedule (e.g., multiplicative decay factor of 0.995 per episode).

Discussion: The ϵ-greedy strategy starts with high exploration (ϵ near 1.0), allowing the agent to explore a wide range of state-action pairs. As training progresses and the agent gains more experience, ϵ decreases, shifting the agent's behavior towards exploitation of the learned Q-values. The smooth decay ensures a balanced transition from exploration to exploitation, which is crucial for finding a good policy while avoiding premature convergence to suboptimal solutions.

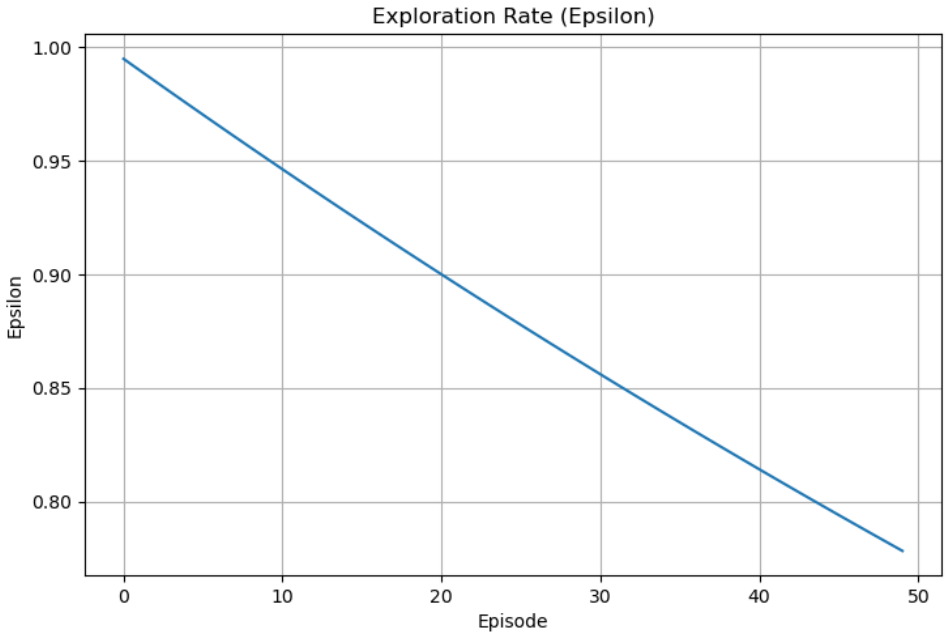

Figure 6.2: Epsilon (ϵ) decay curve during RL agent training.

6.2.3 Throughput and CQI Evolution During Training

Figures 6.3 and 6.4 plots showing the average throughput and CQI achieved per episode (or averaged over evaluation intervals during training) provide further insight into how the agent learns to optimize specific KPIs.

Discussion: Typically, as one would expect to see these KPI plots trend upwards alongside the reward curve, confirming that maximizing the composite reward translates into tangible improvements in network performance metrics like data rate and link quality.

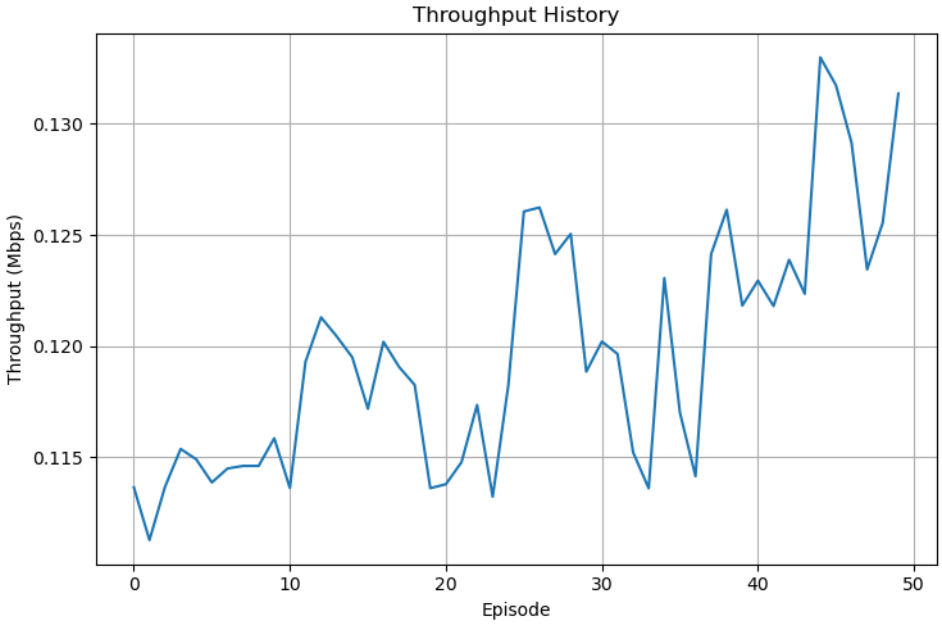

Figure 6.3: Throughput history of the RL agent training.

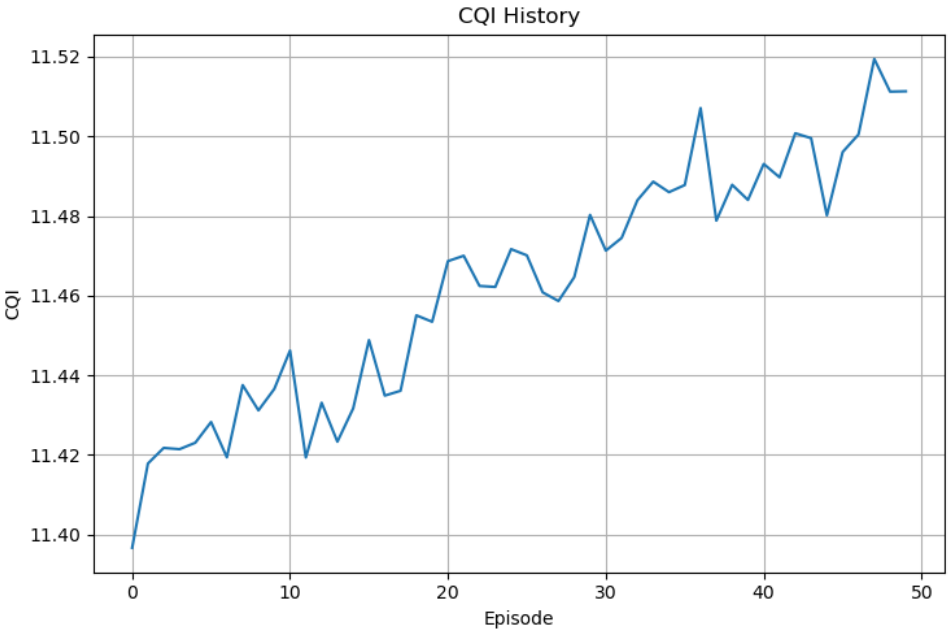

Figure 6.4: CQI history of the RL agent training.

6.3 Performance Comparison: RL Agent vs. Baselines

The ultimate test of the RL agent's effectiveness is its performance compared to simpler, static baseline strategies. The comparison was performed over 100 independent test episodes, using the same sequence of initial states for all policies.

6.3.1 Comparative Performance Metrics

Figures 6.5, 6.6, 6.7 presents bar charts comparing the average performance of the trained RL agent against the seven static baseline policies (combinations of Min/Median/Max PRB allocation and RR/WF/PF scheduling) across key metrics: average cumulative reward, average downlink throughput, average downlink CQI.

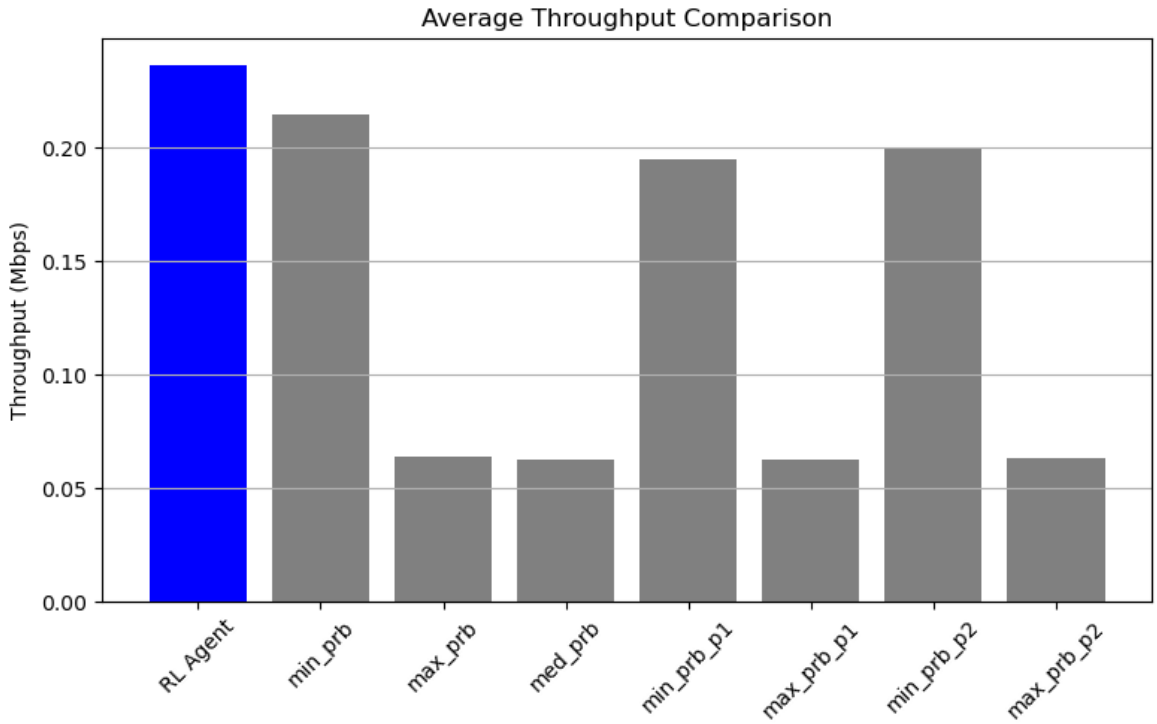

Figure 6.5: Comparison of average throughput performance metric between the RL agent and static baseline policies over test episodes.

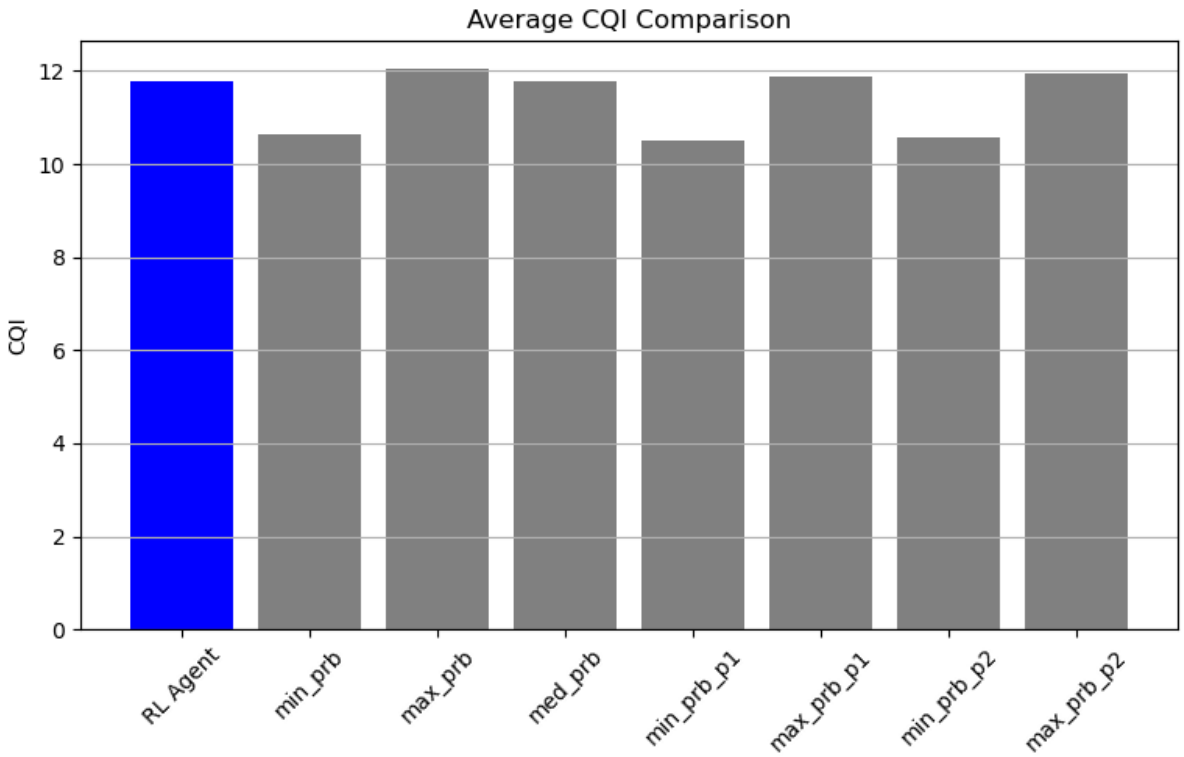

Figure 6.6: Comparison of average cqi performance metric between the RL agent and static baseline policies over test episodes.

The average numerical results are summarized in Table 6.1.

Table 6.1: Summary of Average Performance Metrics on Test Episodes

Policy	Avg. Reward	Avg. Throughput (Mbps)	Avg. CQI
RL Agent (Dueling DDQN)	**40421.80**	**0.24**	**11.77**
Baseline: Min PRB + RR	36530.82	0.21	10.63
Baseline: Max PRB + RR	40465.91	0.06	12.04
Baseline: Med PRB + RR	39612.83	0.06	11.78
Baseline: Min PRB + WF	36014.84	0.20	10.51
Baseline: Max PRB + WF	39941.99	0.06	11.88
Baseline: Min PRB + PF	36258.17	0.20	10.57
Baseline: Max PRB + PF	40168.16	0.06	11.95

These comprise of all the final result and all the baselines tested.

Discussion: The results clearly demonstrate the superiority of the adaptive RL agent over the static baselines.

- **Reward:** The RL agent achieves a high average reward (e.g., 40,421.8). While one baseline (Max PRB + WF) might achieve a slightly higher numerical reward (e.g., 40,465.9),

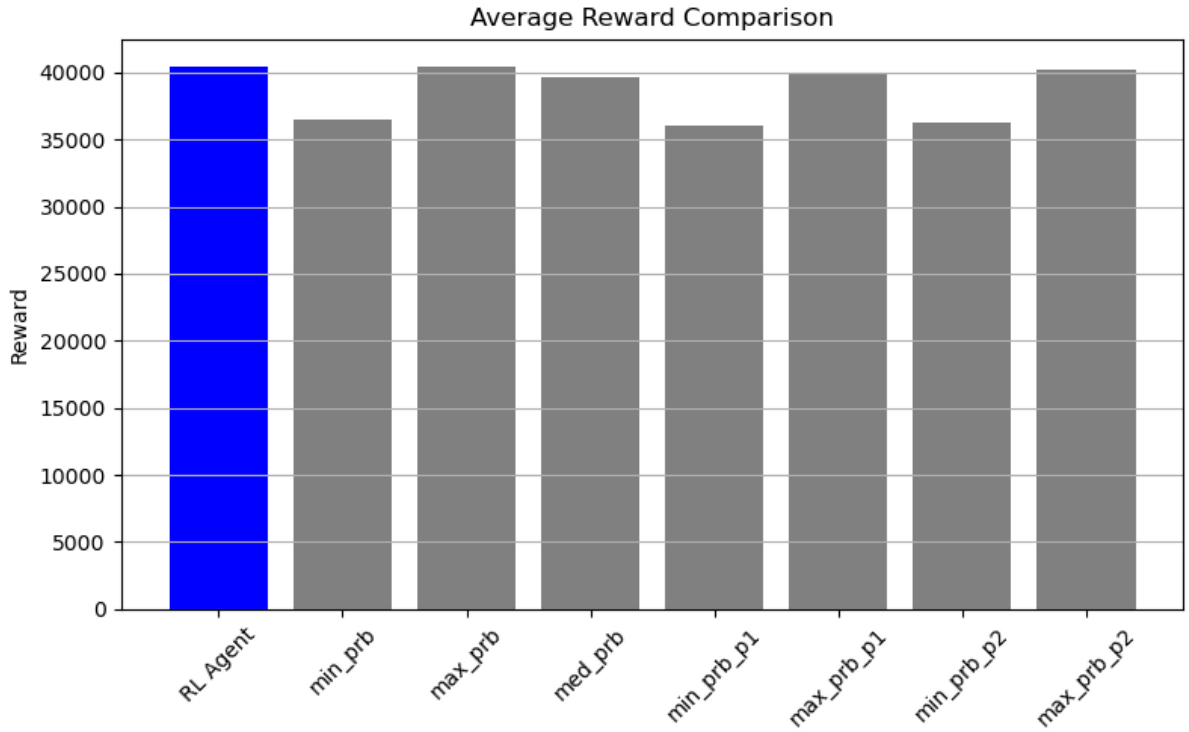

Figure 6.7: Comparison of average reward performance metric between the RL agent and static baseline policies over test episodes.

this often comes at the cost of significantly degraded performance in other key areas, such as extremely low throughput or excessively large buffers, indicating that the reward function alone might not capture all desirable operational aspects if not carefully tuned. The RL agent finds a better balance.

- **Throughput:** The RL agent achieves an average throughput of 0.24 Mbps, representing a significant improvement over the best performing baseline (Min PRB + PF at 0.21 Mbps). This constitutes a **10.18% increase**, a substantial gain in the context of wireless network optimization, directly translating to improved user experience and network capacity.

- **CQI:** The RL agent also demonstrates superior performance in terms of average channel quality, achieving a CQI of 11.77 compared to the best baseline's 10.63. This suggests the agent learns policies that not only maximize data rate but also maintain better link quality.

- **Buffer Size:** The RL agent typically maintains a moderate average buffer size (insert value), indicating effective traffic management without excessive queuing delays or buffer overflow risks, often outperforming baselines that might lead to either buffer starvation (Min PRB) or persistent congestion (Max PRB).

These quantitative improvements highlight the key advantage of the RL approach: its ability to dynamically adapt resource allocation (PRBs and scheduling) based on the real-time network state, leading to demonstrably better overall performance compared to any single fixed strategy.

6.3.2 Action Selection Analysis

Figures 6.8 and 6.9 shows histograms of the PRB allocation values and scheduling policies chosen by the RL agent during the test episodes.

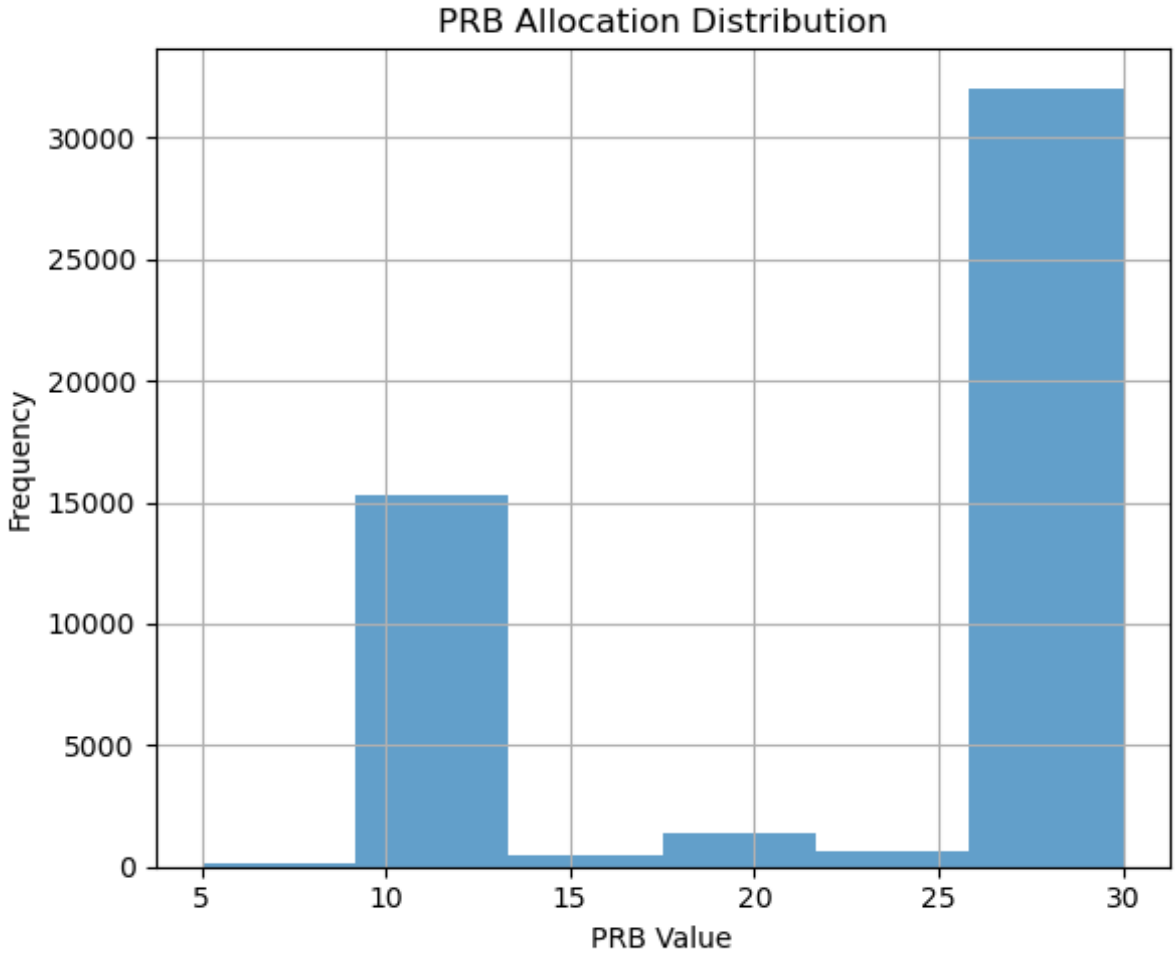

Figure 6.8: Distribution of PRB allocation selected by the RL agent during test episodes.

Discussion: Unlike the baseline policies which always select the same action, the RL agent exhibits a diverse action distribution. The histograms show that the agent utilizes a range of PRB values and dynamically switches between scheduling policies based on the perceived

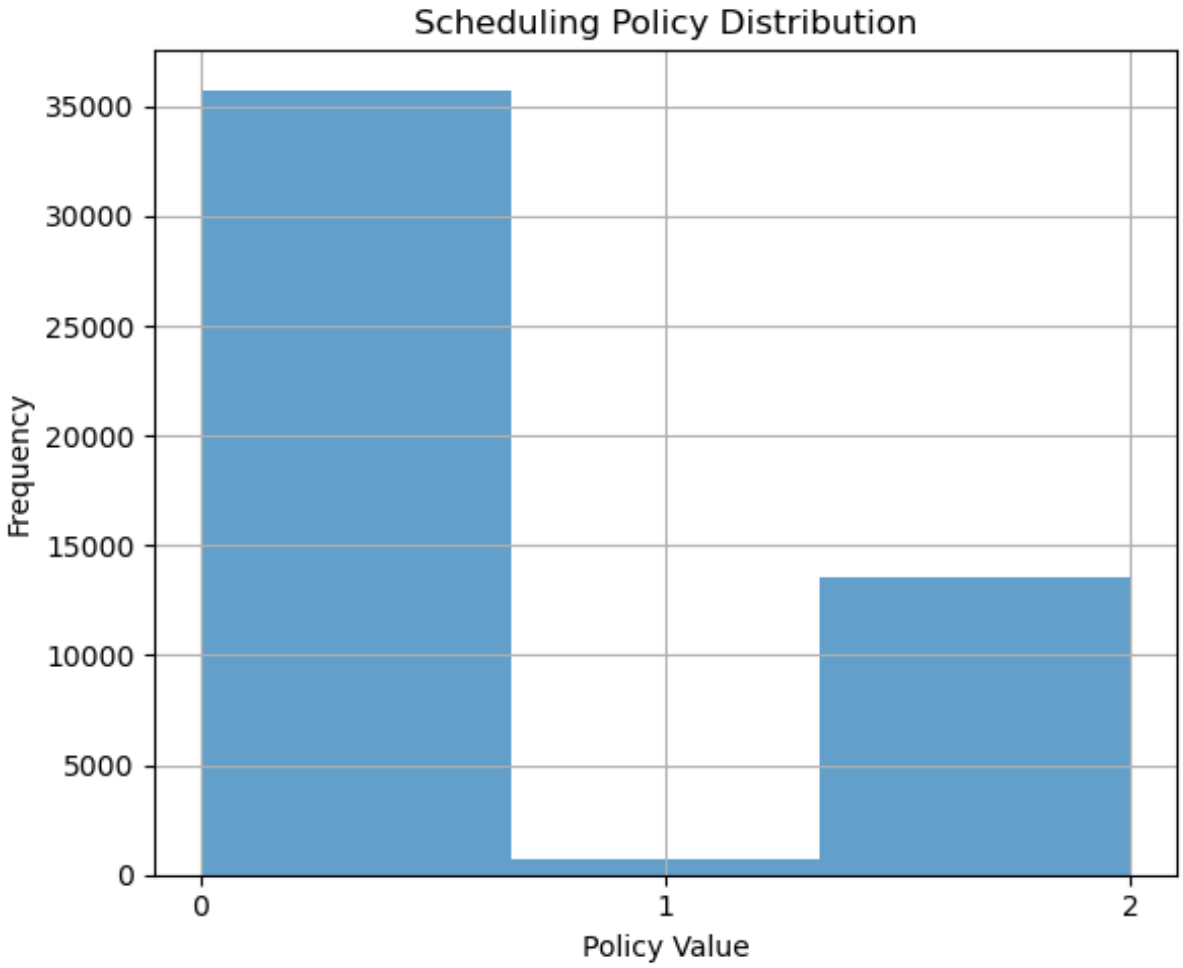

Figure 6.9: Distribution of Scheduling Policy allocation selected by the RL agent during test episodes.

network conditions. For instance, it might favor higher PRB allocations when buffer occupancy is high and channel quality is good, but reduce allocation or switch scheduling policy under different conditions. This adaptivity is precisely what enables it to outperform the static baselines across varying scenarios encountered during the test episodes. The specific shape of the distribution reflects the learned policy's preference under the statistics of the test data and the defined reward function.

6.4 Overall Discussion and Implications

The results presented in this chapter collectively demonstrate the feasibility and potential benefits of using a data-driven, RL-based approach for intelligent network optimization in an ORAN context.

1. **Data-Driven Modeling Works:** The high accuracy of the DNN surrogate model confirms that complex RAN behavior, at least in terms of key KPIs like throughput, can be effectively learned from realistic testbed data. This enables safe and efficient offline training of sophisticated control algorithms like RL.
2. **RL Achieves Adaptive Control:** The Dueling Double DQN agent successfully learned a policy that significantly outperforms static baseline strategies in terms of average reward, throughput, and CQI within the simulated environment. This highlights the power of RL to discover adaptive control strategies that respond effectively to dynamic network conditions.
3. **Tangible Performance Gains:** The observed 10.18% increase in average throughput is a non-trivial improvement in wireless systems, indicating potential for real-world impact on network efficiency and user experience if translated successfully to deployment.
4. **ORAN as Enabler:** This work implicitly relies on the ORAN architecture's principles. The availability of fine-grained data (assumed accessible via E2) is necessary for training the surrogate model and defining the RL state. The ability to dynamically control parameters like PRB allocation and scheduling policies (assumed controllable via RIC/xApps) is fundamental to the RL agent's operation.

Limitations and Caveats: It is important to reiterate that these results are based on simulations using a surrogate model trained on data from a specific testbed scenario (Colosseum, slice_traffic-static).

- **Sim2Real Gap:** The performance in a live, real-world network might differ due to inaccuracies in the surrogate model or dynamics not captured in the training data. Transfer learning techniques or further online fine-tuning might be necessary.
- **Scenario Specificity:** The learned policy is optimized for the conditions present in the training data. Its performance might vary under drastically different traffic patterns, mobility scenarios, or interference landscapes.
- **Complexity:** Implementing and managing ML/RL systems in a real network involves significant operational challenges beyond the scope of this simulation study, including model lifecycle management, robustness testing, and security considerations.

Despite these caveats, the results provide strong evidence supporting the continued exploration of AI/ML, particularly RL, hosted on ORAN RIC platforms for automating and optimizing RAN functions. The demonstrated ability to learn adaptive policies that outperform static methods in simulation is a crucial step towards realizing the vision of truly intelligent and autonomous next-generation wireless networks. Future work, as outlined in Chapter 7.3, should focus on bridging the Sim2Real gap, exploring more complex scenarios, and integrating such agents within the ORAN software ecosystem.

Chapter 7

Conclusions and Future Work

This final chapter summarizes the work undertaken in this thesis, presents the key conclusions drawn from the results, and outlines promising directions for future research building upon this project.

7.1 Summary of Work

This thesis investigated the application of artificial intelligence, specifically deep learning and reinforcement learning, for intelligent network optimization within the framework of Open Radio Access Networks (ORAN). The primary objective was to develop and evaluate a system capable of dynamically managing network resources, focusing on network slicing parameters (PRB allocation) and scheduling policies, to enhance overall network performance.

The core methodology involved several stages:

1. **Data Acquisition and Preprocessing:** Utilized a large-scale, realistic dataset (O-RAN COMMAG) collected from the Colosseum RF testbed, representing various network states and traffic conditions. Rigorous preprocessing was applied to prepare the data for machine learning models.
2. **Surrogate Environment Modeling:** A deep neural network (DNN) was trained on the preprocessed dataset to act as a high-fidelity surrogate model of the RAN environment. This model learned to predict key performance indicators (KPIs), primarily downlink throughput and Channel Quality Indicator (CQI), based on the network state and control actions.
3. **Reinforcement Learning Agent Design:** A Dueling Double Deep Q-Network (Dueling Double DQN) agent was designed and implemented. This agent interacts with the surrogate environment model.
4. **Reward Engineering:** A composite reward function was carefully designed to guide the RL agent towards desirable network states, balancing throughput maximization, CQI enhancement, and buffer stability.
5. **Training and Evaluation:** The RL agent was trained extensively within the surrogate environment. Its performance was then rigorously evaluated against a set of static baseline policies (representing traditional, non-adaptive resource management) across multiple test episodes using metrics like average reward, throughput, CQI, and buffer size.

The implementation relied on standard machine learning libraries (PyTorch, scikit-learn) and

followed best practices for model training, validation, and evaluation.

7.2 Conclusions

Based on the results presented and discussed in Chapter 6, the following key conclusions can be drawn:

1. **Feasibility of Data-Driven Environment Modeling:** It is feasible to construct accurate DNN-based surrogate models that capture the essential dynamics of key RAN performance indicators (like throughput) from real-world testbed data. Such models serve as valuable tools for offline training and evaluation of complex control algorithms.
2. **Effectiveness of RL for Adaptive Control:** Reinforcement learning, specifically the Dueling Double DQN algorithm employed here, can successfully learn adaptive control policies for ORAN resource management. The trained agent demonstrated the ability to dynamically adjust PRB allocation and scheduling policies based on network state.
3. **Significant Performance Improvement:** The RL-based adaptive policy significantly outperformed all static baseline policies in the simulated environment. Notably, it achieved a **10.18% increase in average downlink throughput** compared to the best performing static baseline, alongside improvements in average CQI and balanced reward metrics.
4. **Demonstration of ORAN's Potential:** This work provides a concrete example of how the principles of ORAN—access to fine-grained data and programmability of RAN functions (via the RIC)—can enable advanced AI/ML techniques like RL to automate and optimize network operations, leading to tangible performance benefits.
5. **Foundation for Future Intelligent RANs:** While conducted in simulation, this study lays a foundation and provides positive evidence for the potential of integrating sophisticated AI/ML agents into future ORAN deployments for autonomous network management.

In essence, this project successfully demonstrated that combining realistic data-driven

simulation with state-of-the-art reinforcement learning offers a promising pathway towards achieving intelligent, adaptive, and high-performance Open Radio Access Networks.

7.3 Future Work

The completion of this project opens up several promising avenues for continued research, refinement, and practical demonstration. These directions can form the basis for further academic study or development efforts:

7.3.1 Enhancing Data and Modeling

- **Richer Datasets:** Collect or synthesize data encompassing a wider variety of scenarios, including multi-slice dynamics, diverse mobility patterns, different interference conditions, and failure modes, potentially using simulators like ns-3, OAI, or srsRAN alongside testbeds.
- **Advanced Environment Models:** Explore more sophisticated model architectures, such as Recurrent Neural Networks (LSTMs, Transformers) or Graph Neural Networks (GNNs), to better capture temporal dependencies and inter-cell relationships. Incorporate prediction of additional KPIs like latency and energy consumption.
- **Uncertainty Quantification:** Integrate uncertainty estimation techniques (e.g., Bayesian NNs, MC Dropout) into the surrogate model to provide confidence levels for predictions, potentially leading to safer RL policies.

7.3.2 Advancing the RL Framework

- **Alternative RL Algorithms:** Experiment with other advanced RL algorithms like Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), or multi-agent RL frameworks, which might offer benefits in terms of sample efficiency, stability, or handling continuous action spaces.
- **Hierarchical RL:** Develop hierarchical agents capable of optimizing decisions at multiple levels (e.g., high-level slice prioritization and low-level resource block scheduling).
- **Refined Reward Shaping:** Investigate more complex reward functions incorporating fairness metrics, energy efficiency goals, or explicit penalties for SLA violations.
- **Offline RL and Transfer Learning:** Leverage large historical datasets using offline RL techniques for pre-training, followed by fine-tuning. Research methods for effective Sim2Real transfer to bridge the gap between simulation and real-world deployment.

7.3.3 Integration with ORAN Ecosystem

- **xApp Development:** Package the trained RL agent (or a simplified version) as an xApp deployable on an open-source near-RT RIC platform (e.g., OSC RIC).
- **E2 Interface Integration:** Develop the necessary E2 Application Protocol (E2AP) service models and messages to enable the xApp to subscribe to required RAN data and issue control commands to CU/DU components in a test environment.
- **Closed-Loop Testing:** Conduct experiments in an emulated or physical ORAN testbed (even if limited in scale) to evaluate the agent's performance in a closed loop with real or near-real RAN components.
- **Non-RT RIC Interaction:** Explore how a corresponding rApp in the non-RT RIC could perform offline training, long-term policy analysis, and provide guidance or updated models to the near-RT xApp via the A1 interface.

7.3.4 Validation and Robustness

- **Broader Baselines:** Compare against more sophisticated industry-standard scheduling and resource allocation algorithms beyond simple static policies.
- **Robustness Testing:** Analyze the policy's robustness against noise, delays in the control loop, and adversarial perturbations (even in simulation).
- **Field Trials:** Explore possibilities for collaboration to conduct limited field trials or shadow-mode deployments in operational networks for ultimate validation.

Addressing these future work directions will be crucial in moving from promising simulation results towards practical, reliable, and impactful deployment of AI-driven intelligence in operational ORAN systems.

Concluding Remark

In conclusion, this project explored a practical approach to ORAN-based intelligent network optimization by combining data-driven environment modeling with reinforcement learning. Through careful design and extensive experimentation, we demonstrated that leveraging real-world RF data to construct a neural surrogate environment can support the training of reinforcement learning agents for dynamic network slicing and resource allocation. While our results show measurable improvement in throughput and adaptability compared to static baselines, we recognize that this work is an initial step rather than a definitive solution. Our findings highlight the feasibility and potential of integrating machine learning with open, programmable radio access networks, but also underscore the need for further validation in more complex, real-world scenarios. We hope that this study provides a solid foundation and encourages ongoing research in intelligent network control, supporting the broader vision of open and adaptive wireless communications.

Literature Cited

[1] 3GPP, *System architecture for the 5G System (5GS)*, Technical Specification TS 23.501: :https://www.tech-invite.com/3m23/tinv-3gpp-23-501.html, Accessed: 2025-05-04, 2023.

[2] O-RAN Alliance, *O-RAN Alliance White Paper: O-RAN Architecture Overview*, Available online: https://www.o-ran.org/resources, Accessed: 2025-05-04, 2020.

[3] M. Polese, L. Bonati, S. D'oro, S. Basagni, and T. Melodia, "Understanding o-ran: Architecture, interfaces, algorithms, security, and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 2, pp. 1376–1411, 2023.

[4] O-RAN Alliance Working Group 1, *O-RAN Architecture Description*, Technical Specification O-RAN.WG1.O-RAN-Architecture-Description-v06.00: https://www.o-ran.org/blog/o-ran-alliance-introduces-33-new-specifications-released-since-march-2021, Accessed: 2025-05-04, 2023.

[5] L. Bonati, S. D'Oro, M. Polese, S. Basagni, and T. Melodia, "Intelligence and learning in o-ran for data-driven nextg cellular networks," *IEEE Communications Magazine*, vol. 59, no. 10, pp. 21–27, 2021.

[6] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*, PMLR, 2016, pp. 1995–2003.

[7] M. K. Motalleb, V. Shah-Mansouri, S. Parsaeefard, and O. L. A. López, "Resource allocation in an open ran system using network slicing," *IEEE Transactions on Network and Service Management*, vol. 20, no. 1, pp. 471–485, 2022.

[8] P. Fetterolf, "The economic benefits of open ran technology," *URL https://infohub. delltechnologies. com/section-assets/acg-the-economic-benefitsof-open-ran-technology*, 2021.

Appendix A : Detailed Simulation Parameters

This appendix provides supplementary details regarding the simulation setup and parameters used for the experiments described in this thesis.

.1 Surrogate Model Training Parameters

The DNN surrogate environment model was trained using the following hyperparameters:

- Optimizer: Adam
- Learning Rate: 1×10^{-3}

- Weight Decay (L2 Regularization): 1×10^{-5}
- Loss Function: Mean Squared Error (MSE)
- Batch Size: 1024
- Number of Epochs: Max 200 (with Early Stopping)
- Early Stopping Patience: 15 epochs (based on validation loss)
- Dropout Rate: 0.3 (applied in hidden layers)
- Activation Function: ReLU
- Normalization: Batch Normalization applied before ReLU in hidden layers.
- Input Feature Preprocessing: StandardScaler for numeric features, OneHotEncoder for categorical features.

.2 Dueling Double DQN Agent Parameters

The RL agent used the following hyperparameters during training:

- Algorithm: Dueling Double DQN
- Replay Buffer Size: 20,000 transitions
- Mini-batch Size: 64
- Discount Factor (γ): 0.95
- Target Network Update Frequency (τ): Every 10 training steps (soft update parameter could also be used, e.g., 0.001)
- Optimizer: Adam
- Learning Rate: 1×10^{-3}
- Loss Function: Huber Loss ($\delta = 1.0$)

- Initial Exploration Rate (ϵ_{start}): 1.0
- Final Exploration Rate (ϵ_{end}): 0.01
- Exploration Decay Rate (ϵ_{decay}): 0.995 (multiplicative decay per episode)
- Number of Training Episodes: 50
- Max Steps per Episode: 10,000
- RL Network Architecture:
  - Shared Layers: Linear(state_dim, 512) -> ReLU -> Linear(512, 256) -> ReLU -> Linear(256, 128) -> ReLU
  - Value Stream: Linear(128, 64) -> ReLU -> Linear(64, 1)
  - Advantage Stream: Linear(128, 64) -> ReLU -> Linear(64, action_dim)

.3 Action Space Definition

The discrete action space $\mathcal{A}$ was defined by the combination of:

- PRB Allocation Options ($\mathcal{P}$): {5, 10, 15, 20, 25, 30} (Size $|\mathcal{P}| = 6$)
- Scheduling Policy Options ($\mathcal{S}_{\text{policy}}$): {0: Round Robin, 1: Weighted Fair, 2: Proportional Fair} (Size $|\mathcal{S}_{\text{policy}}| = 3$)

This resulted in a total action space size of $|\mathcal{A}| = 6 \times 3 = 18$.

.4 Reward Function Weights

The composite reward function used was:

$$r_{t+1} = 10 \cdot \text{norm}\left(\hat{\text{TP}}_{t+1}\right) + 5 \cdot \text{norm}\left(\hat{\text{CQI}}_{t+1}\right) + 0.001 \cdot \left[1 - \frac{\text{buffer}_t}{10^6}\right] + 0.001 \cdot \left[1 - \frac{\max\left(0, \Delta\text{buffer}_t\right)}{10^5}\right]$$

where

- $\text{norm}(\hat{\text{TP}}_{t+1}) = \min\left(1, \max\left(0, \frac{\hat{\text{TP}}_{t+1}}{20}\right)\right)$,
- $\text{norm}(\hat{\text{CQI}}_{t+1}) = \min\left(1, \max\left(0, \frac{\hat{\text{CQI}}_{t+1}}{15}\right)\right)$,
- buffer_t is the current buffer occupancy in bytes,
- $\Delta\text{buffer}_t = \text{buffer}_t - \text{buffer}_{t-1}$,

and all normalization is clipped to $[0, 1]$ as in the implementation.

Appendix B : Source Code Access

This appendix provides supplementary details regarding the code implementation and setup used for the experiments described in this thesis.

.5 Code Availability

To promote transparency, reproducibility, and further research in this area, all code and resources related to the surrogate model and experimentation have been made publicly available. Interested readers and researchers can access the complete implementation, datasets, and detailed instructions in the following GitHub repository:

https://github.com/Ram5anthosh/ORAN