

UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Generative Agents for Network Management and Protocol Design

Permalink

<https://escholarship.org/uc/item/0t78x55m>

ISBN

9798171612627

Author

KWON, DAE CHEOL

Publication Date

2026-10-26

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA SAN DIEGO

Generative Agents for Network Management and Protocol Design

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Electrical Engineering (Communication Theory and Systems)

by

Dae Cheol Kwon

Committee in charge:

Professor Xinyu Zhang, Chair
Professor Zhiting Hu
Professor Laurence Milstein
Professor Yuanyuan Shi

2026

Copyright

Dae Cheol Kwon, 2026

All rights reserved.

The Dissertation of Dae Cheol Kwon is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2026

DEDICATION

To my family, whose love, patience, and support made this dissertation possible.

TABLE OF CONTENTS

Dissertation Approval Page	iii
Dedication	iv
Table of Contents	v
List of Figures	viii
List of Tables	xi
Acknowledgements	xii
Vita	xiv
Abstract of the Dissertation	xv
Chapter 1 Introduction	1
1.1 Motivation	1
1.2 Challenges	3
1.3 Thesis Statement	5
1.4 Dissertation Contributions	5
1.5 Dissertation Organization	6
Chapter 2 Background and Related Work	8
2.1 Wireless Network Heterogeneity and Protocol Design	8
2.2 Open Radio Access Network	10
2.3 xApps and RAN Control	11
2.4 xApp Conflicts in Open RAN	12
2.5 Cellular Configuration and Misconfiguration	13
2.6 Large Language Models and Generative Agents	14
2.7 Summary	15
Chapter 3 Generative Agents for Communication Protocol Design	16
3.1 Introduction	16
3.2 CP-AgentNet System Design	20
3.2.1 CP-AgentNet Architecture	20
3.2.2 CP-AgentNet Workflow	21
3.2.3 Autonomous Design	27
3.2.4 Stability of CP-AgentNet	29
3.3 Protocol Use Cases with CP-AgentNet	30
3.3.1 LLMA	30
3.3.2 CPTCP	33
3.4 Evaluation	34

3.4.1	Simulation Setup	34
3.4.2	LLMA results	36
3.4.3	CPTCP results	43
3.5	Discussion	45
3.6	Chapter-Specific Related Work	46
3.6.1	MAC protocols through RL	46
3.6.2	LLM-empowered Agents	46
3.6.3	LLMs with Reinforcement Learning	47
3.6.4	LLMs for communication network	47
3.7	Summary	47
Chapter 4	Autonomous xApp Synthesis for Open RAN	49
4.1	Introduction	49
4.2	Background	55
4.2.1	O-RAN Architecture and E2-Based Control	55
4.2.2	xApp Development Workflow	56
4.3	Auto-xApp Design	57
4.3.1	System Overview	57
4.3.2	Intent Translation via Schema-Guided Iteration	59
4.3.3	Dataset Analysis	61
4.3.4	Context-Aware Mapping via RAG	62
4.3.5	Capability Validation	64
4.3.6	xApp Code Generation	66
4.4	Evaluation	67
4.4.1	Testbed Setup	67
4.4.2	Use Case 1: Rule-Based GBR Enforcement	69
4.4.3	Use Case 2: Learning-Based Anomaly Detector	70
4.4.4	Use Case 3: DRL-Based PRB Slicing	77
4.5	Discussion and Limitations	83
4.6	Chapter-Specific Related Work	84
4.6.1	Performance-Driven xApps in O-RAN	84
4.6.2	LLMs in Communication Networks	85
4.7	Summary	86
Chapter 5	Generative-Agent-Based Conflict Detection and Mitigation in Open RAN	87
5.1	Introduction	87
5.2	Background and Problem Setting	91
5.2.1	Conflicts in ORAN App Configuration	91
5.2.2	SHAP Analysis for Feature Importance	91
5.3	System Design	92
5.3.1	ORCA Architecture	92
5.3.2	Environment Encoder (EE)	94
5.3.3	Feature Importance Extractor (FIE)	95
5.3.4	Feature Importance Reasoner (FIR)	97

5.3.5	Conflict Detection and Mitigation (CD&M)	103
5.4	Evaluation	104
5.4.1	Simulation Setup	104
5.4.2	Results	105
5.5	Discussion	113
5.6	Related Work	113
5.6.1	Conflict mitigation in ORAN	113
5.6.2	LLMs for communication network	114
5.7	Summary	114
Chapter 6	Proactive Reasoning for Implicit Misconfiguration Elimination	116
6.1	Introduction	116
6.2	Constraint Taxonomy	121
6.3	PRIME Design	122
6.3.1	System Overview	122
6.3.2	Parameter Extractor	124
6.3.3	Cross-Reference-Augmented Semantic Labeling	125
6.3.4	Pair Generator	127
6.3.5	Constraint Extractor	128
6.3.6	Runtime Enforcement	131
6.4	Evaluation	131
6.4.1	Methodology	131
6.4.2	Operational Impact	133
6.5	Related Work	137
6.5.1	Misconfiguration in Cellular Networks	137
6.5.2	Conflict Detection and Mitigation in O-RAN	139
6.5.3	LLM Applications in Telecom	140
6.6	Conclusion	141
Chapter 7	Conclusion and Future Directions	142
7.1	Conclusion	142
7.2	Lessons Learned	144
7.3	Limitations	145
7.4	Future Directions	146
7.5	Final Remarks	147
	Bibliography	148

LIST OF FIGURES

Figure 3.1.	Overall workflow of CP-AgentNet	21
Figure 3.2.	Structure and prompt of PSA that synthesizes multiple strategies by resolving conflicts and eliminating redundancy.	26
Figure 3.3.	Structure and prompt of ASI. Implementation is delegated to the programming assistant by the node or observer agent.	27
Figure 3.4.	LLMA nodes in homogeneous environment	37
Figure 3.5.	Heteronode with 3 ALOHA and 1 TDMA nodes.	38
Figure 3.6.	Heteronode in a dynamic environment.	38
Figure 3.7.	α -fairness throughput in a complex heterogeneous environment. C, EB, and FW represent CSMA node, EB-ALOHA node, and FW-ALOHA node, respectively.	40
Figure 3.8.	Explainable CP-AgentNet	43
Figure 4.1.	Auto-xApp vision: automatic generation of deployable xApps from user requirements and network KPI data.	51
Figure 4.2.	Auto-xApp framework.	57
Figure 4.3.	Intent Translator with schema-guided iteration.	59
Figure 4.4.	URLLC delay exhibits a sharp performance cliff around 31–32 PRBs.	74
Figure 4.5.	PRB allocation landscape across eMBB, URLLC, and MTC slices. Markers denote operating points: $\times$ Static (33/33/33), $*$ Sweep-Opt (40/40/20), $\blacklozenge$ DRL.	75
Figure 4.6.	Step-by-step comparison of Static vs. DRL xApp across KPIs. Dashed gray line: SLA threshold. Dashed green line: Sweep-Opt reference.	76
Figure 4.7.	Performance comparison across Static, Sweep-Opt, and DRL xApps.	76
Figure 4.8.	Per-slice CDF comparison for eMBB throughput, URLLC delay, and MTC throughput across static allocation, sweep optimum, and the DRL xApp.	82
Figure 5.1.	Conflict detection methods do not hold in unseen environments.	89
Figure 5.2.	Overall architecture of ORCA, consisting of FIE, EE, FIR, and CD&M.	93

Figure 5.3.	Detailed structure of the FIE.	95
Figure 5.4.	FIR module infers new importance using β -Reasoning Cascade and external knowledge retrieved via SA-RAG.	97
Figure 5.5.	Feature importance is derived by SHAP analysis (left), while new feature importance is obtained via β -propagation (right). β values are propagated through intermediate metrics.	98
Figure 5.6.	Conventional hybrid RAG (left) vs SA-RAG (right).....	102
Figure 5.7.	Comparison of derived feature importance against ground-truth (ACTUAL) values. Closer alignment indicates higher estimation accuracy.	103
Figure 5.8.	Comparison of cosine error and NRMSE. “Ext” denotes the extrapolation baseline.	107
Figure 5.9.	Conflict scenario between ES and TM.	108
Figure 5.10.	Conflict scenario between HM and MRO. In the top subplots, solid lines show the number of handovers and dashed lines show the number of RLFs.	108
Figure 5.11.	Conflict scenario between TM and DM.	110
Figure 5.12.	(b) and (c) show the accuracy of feature importance estimation for throughput and average delay, respectively. ORCA (full system) is compared against Beta (without SA-RAG) and SA-RAG (without the β -Reasoning Cascade).	111
Figure 5.13.	Example from the β -Reasoning Cascade (step 3)	112
Figure 6.1.	Two examples of implicit misconfigurations. Each parameter is individually valid; the violation lies in the hidden inter-parameter relation.	118
Figure 6.2.	Configuration risks evolve across cellular generations: handover loops in 3G/4G, missed SCells under carrier aggregation and dual connectivity in 5G/4.5G, and hidden inter-parameter dependencies introduced by programmable O-RAN xApps.....	118
Figure 6.3.	Overall architecture of PRIME. The taxonomy (§??) is the spine that the four middle stages reference; the LLM is invoked only in Steps 2 and 5 (shaded), the remaining stages being deterministic.	123
Figure 6.4.	A C1 constraint record. A C5 record replaces <code>param_b</code> and <code>direct_inequality</code> with a referenced entity and a co-presence relation.....	130

Figure 6.5.	C1 ($\theta_{B1} > \theta_{A2}$). Without the constraint the 5G SCell oscillates and throughput collapses; with it the SCell is stable and throughput is sustained.	134
Figure 6.6.	C4 (A1/A2 metric mismatch). RSRP and RSRQ diverge; without the constraint the measurement state conflicts and throughput drops, with a 7% improvement once the metrics are unified.	135
Figure 6.7.	C5 (RLF detection). Without RadioLinkMonitoringConfig the UE never detects out-of-sync and its view of the connection diverges from the network's; enforcing RLM co-presence with the RLF timers restores out-of-sync detection and call recovery.	136
Figure 6.8.	C5 (inter-frequency mobility). Without measGapConfig the UE cannot measure inter-frequency neighbors in CONNECTED, so each cell boundary triggers an RLF blackout; enforcing co-presence enables CONNECTED-mode handovers without service interruption.	138

LIST OF TABLES

Table 3.1.	(N)RMSE of all scenarios. T, A, and H represent TDMA, slotted ALOHA, and heteronode, respectively.	39
Table 3.2.	The impact of each agent. 1: No Agent, 2: Strategy + Node, 3: Observer + Node, 4: Strategy + Observer + Node	41
Table 3.3.	The impact of using an LLM ranker.	41
Table 3.4.	Comparison of computation time and RMSE performance across smaller LLM variants	43
Table 3.5.	Total throughput (Kbps) with homogeneous TCP	44
Table 3.6.	Homogeneous TCP under different RTTs.	44
Table 3.7.	Throughput and fairness with heterogeneous TCP	45
Table 4.1.	Overall Anomaly Detection Performance on SpotLight Dataset	72
Table 4.2.	Per-Scenario Performance (Precision / Recall / F1).....	73
Table 4.3.	Deployment Capability Comparison	73
Table 4.4.	Performance comparison: static baseline, sweep optimum, and auto-generated DRL xApp.....	81
Table 5.1.	Overview of Applications and Controlled Parameters.....	104
Table 5.2.	Average cosine error and NRMSE across all 32 unseen environments and 4 KPIs.	106
Table 6.1.	Constraint taxonomy. Each category is one structural form an implicit inter-parameter dependency can take.	122
Table 6.2.	Representative oracle entries, one per PRIME taxonomy category.	132
Table 6.3.	Recall of PRIME on the 94-entry partial oracle. The real-world category is split by configuration scope: single-cell entries are within PRIME’s design; cross-cell and topology-dependent entries are explicitly out of scope.	132

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor and committee chair, Professor Xinyu Zhang, for his guidance, support, and encouragement throughout my Ph.D. studies. His insight and mentorship have shaped both the direction and the quality of this dissertation.

I am also grateful to my committee members, Professor Professor Zhiting Hu, Professor Laurence Milstein, and , Yuanyuan Shi for their time, feedback, and valuable advice during my doctoral study and dissertation process.

I would like to thank my collaborators, labmates, friends, and colleagues for the many discussions, suggestions, and encouragement that supported this research. Their help has been invaluable throughout the development of the work presented in this dissertation.

Finally, I am deeply thankful to my family for their constant love, patience, and support. This dissertation would not have been possible without them.

Chapter 3, in part, is a reprint of the material as it appears in Dae Cheol Kwon and Xinyu Zhang, “CP-AgentNet: Autonomous and Explainable Communication Protocol Design Using Generative Agents,” in *Proceedings of the 2025 IEEE 33rd International Conference on Network Protocols (ICNP)*, 2025. The dissertation author was the primary investigator and first author of this paper.

Chapter 4, in part, has been submitted for publication of the material as it may appear in Dae Cheol Kwon and Xinyu Zhang, “Auto-xApp: Autonomous xApp Synthesis for Open RAN via Generative Agents,” submitted to *IEEE International Conference on Network Protocols (ICNP)*, 2026. The dissertation author was the primary investigator and first author of this material.

Chapter 5, in part, is a reprint of the material as it appears in Dae Cheol Kwon and Xinyu Zhang, “Open RAN Conflict Agents: Detecting and Mitigating xApp Conflicts with Generative Agents,” in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2026. The dissertation author was the primary investigator and first author of this paper.

Chapter 6, in part, has been submitted for publication of the material as it may appear in

Dae Cheol Kwon and Xinyu Zhang, “PRIME: Proactive Reasoning for Implicit Misconfiguration Elimination with Generative Agents,” submitted to *IEEE Conference on Computer Communications (INFOCOM)*, 2027. The dissertation author was the primary investigator and first author of this material.

VITA

2006	Bachelor of Science in Electronics Engineering, Sogang University
2008	Master of Science in Electronics, Sogang University
2008–2021	Principal Engineer, Protocol Development Team, System LSI, Samsung Electronics
2021–2025	Ph.D. Student in Electrical Engineering (Communication Theory and Systems), University of California San Diego
2025–2026	Ph.D. Candidate in Electrical Engineering (Communication Theory and Systems), University of California San Diego
2026	Doctor of Philosophy in Electrical Engineering (Communication Theory and Systems), University of California San Diego

PUBLICATIONS

Dae Cheol Kwon and Xinyu Zhang, “CP-AgentNet: Autonomous and Explainable Communication Protocol Design Using Generative Agents,” in *Proceedings of the 2025 IEEE 33rd International Conference on Network Protocols (ICNP)*, 2025.

Dae Cheol Kwon and Xinyu Zhang, “Open RAN Conflict Agents: Detecting and Mitigating xApp Conflicts with Generative Agents,” in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2026.

Dae Cheol Kwon and Xinyu Zhang, “LiteFBD: Lightweight CNN Design with Federated Block-wise Knowledge Distillation for Federated Learning,” in *Proceedings of the 2025 IEEE 102nd Vehicular Technology Conference (VTC2025-Fall)*, 2025.

Dae Cheol Kwon and Xinyu Zhang, “Auto-xApp: Autonomous xApp Synthesis for Open RAN via Generative Agents,” submitted to *IEEE International Conference on Network Protocols (ICNP)*, 2026.

Dae Cheol Kwon and Xinyu Zhang, “PRIME: Proactive Reasoning for Implicit Misconfiguration Elimination with Generative Agents,” submitted to *IEEE Conference on Computer Communications (INFOCOM)*, 2027.

ABSTRACT OF THE DISSERTATION

Generative Agents for Network Management and Protocol Design

by

Dae Cheol Kwon

Doctor of Philosophy in Electrical Engineering (Communication Theory and Systems)

University of California San Diego, 2026

Professor Xinyu Zhang, Chair

This dissertation investigates how large-language-model-based generative agents can support autonomous network management in complex wireless systems. Modern wireless networks are increasingly heterogeneous, programmable, and difficult to manage manually. As network components, control applications, and configuration parameters interact in increasingly complex ways, future network management requires systems that can adapt to changing environments, coordinate decisions across independently developed components, and generate executable control logic from high-level objectives.

This dissertation advances the thesis that autonomous network management can be realized by decomposing complex wireless networking tasks into domain-grounded generative-

agent workflows that reason over demonstrations, environments, user intents, and specifications to produce adaptive strategies, coordinated decisions, and executable network logic. The dissertation develops this idea across network management and protocol design. First, it studies how generative agents can design adaptive and explainable communication protocols from limited demonstrations. Second, it studies how generative agents can synthesize deployable Open RAN xApps from high-level user requirements. Third, it studies how generative agents can detect and mitigate conflicts among independently developed xApps under dynamic network environments. Finally, it studies how generative agents can reason over protocol specifications to identify implicit configuration constraints and prevent misconfigurations.

Together, these studies show that generative agents can provide a reasoning and synthesis layer for autonomous network management, reducing manual effort while improving adaptability, coordination, and generation in next-generation wireless networks.

Chapter 1

Introduction

1.1 Motivation

Modern wireless networks are becoming increasingly complex. They are no longer composed of a single protocol, a single class of devices, or a fixed operating environment. Instead, diverse protocols, devices, software components, vendors, and dynamic network conditions interact within shared wireless systems. This evolution creates new opportunities for programmability and intelligence, but it also introduces new forms of system-level complexity. A component may operate correctly in isolation, yet the overall network may still behave unexpectedly when multiple components interact. In this sense, individual correctness does not guarantee system-level harmony.

This complexity appears across both protocol design and network management. At the protocol level, heterogeneous wireless systems must support coexistence among different access mechanisms, traffic patterns, and communication strategies. A protocol that performs well under one condition may become inefficient or unstable when the environment changes or when other protocols compete for shared resources. At the management level, emerging architectures such as Open Radio Access Network (O-RAN) disaggregate the RAN into multiple programmable components and allow independently developed xApps to control radio resources. This openness enables flexibility and innovation, but it also increases the risk of conflicts, hidden dependencies, and misconfigurations when independently developed components are integrated.

To address this complexity, this dissertation studies autonomous network management. In this dissertation, autonomous network management refers to the ability of wireless systems to support network management and protocol design with reduced human intervention, even when environments change, multiple components interact, and network design and configuration become increasingly complex. This vision is organized around three capabilities: adaptability, coordination, and generation. Adaptability enables network behavior to adjust to changing environments. Coordination enables independently developed components to operate together without harmful conflicts or unexpected interactions. Generation enables high-level goals, demonstrations, specifications, and observations to be transformed into executable strategies, control logic, and deployable software artifacts.

Achieving these capabilities remains difficult. Adaptability is challenging because wireless environments evolve over time, while learning or redesigning protocols for every new condition can be costly and slow. Coordination is challenging because cross-component dependencies are often implicit; conflicts or misconfigurations may appear only after components are integrated. Generation is challenging because operators typically express goals at a high level, whereas wireless systems require precise and executable control logic. Existing approaches only partially address these challenges. Rule-based methods require extensive manual design and are difficult to scale across diverse conditions. Deep reinforcement learning and other data-driven methods can provide adaptability, but they often require environment-specific training data and may generalize poorly to unseen conditions. Conventional optimization and learning methods are also not naturally designed to interpret natural-language requirements, reason over protocol specifications, synthesize executable implementations, or identify hidden dependencies.

This dissertation argues that LLM-powered generative agents provide a promising reasoning and synthesis layer for autonomous network management. Generative agents combine language understanding, reasoning, planning, code generation, memory, tool use, and external knowledge retrieval. When embedded in domain-grounded workflows, they can connect human intent, wireless domain knowledge, technical specifications, runtime observations, and executable

network behavior. The focus of this dissertation is therefore not on using generative agents as generic AI tools, but on designing structured workflows that specialize them for wireless networking tasks.

The dissertation develops this idea through four systems. CP-AgentNet studies adaptability by using demonstrations and online observations to generate reusable protocol strategies for heterogeneous environments. Auto-xApp studies generation by translating high-level network goals into deployable O-RAN xApp software. ORCA studies coordination by reasoning from known environments to unseen environments to detect and mitigate potential xApp conflicts. PRIME further supports coordination and reliability by extracting implicit configuration constraints from network specifications to prevent misconfigurations. Together, these systems show how domain-grounded generative-agent workflows can support adaptive strategies, coordinated decisions, and executable network logic for autonomous network management.

1.2 Challenges

Realizing autonomous network management requires addressing three major challenges: adaptability, coordination, and generation.

Challenge 1: Adaptability in heterogeneous wireless environments. Wireless environments change over time due to variations in traffic demand, channel conditions, device behavior, mobility, and coexistence with other protocols. A protocol that performs well in one environment may fail to achieve similar performance when the environment changes. Traditional protocol design relies heavily on expert knowledge and repeated manual tuning. Deep reinforcement learning can automate parts of this process, but it typically requires substantial training data, careful neural architecture selection, hyperparameter tuning, and retraining when the environment changes. These limitations make it difficult to support protocol design in heterogeneous and dynamic settings. Autonomous network management therefore requires adaptive mechanisms that can generate, refine, and reuse protocol strategies across changing environments while

keeping the decision process understandable to human designers.

Challenge 2: Coordination across independently developed components. Modern wireless networks increasingly consist of components developed by different vendors, operators, or third-party developers. O-RAN makes this issue especially important because multiple xApps may coexist in the same RAN environment and control related resources. Each xApp may be correct in isolation, but their combined behavior may still create direct, indirect, or implicit conflicts. Similarly, configuration parameters may be individually valid but unsafe in combination due to hidden dependencies across procedures, states, metrics, cells, or time. Since these dependencies are often implicit, conflicts or misconfigurations may not become visible until the system is integrated or deployed. Autonomous network management therefore requires reasoning methods that can detect conflicts and unsafe configurations before they degrade network performance.

Challenge 3: Generation of executable network logic from high-level intent. Network operators usually express goals at a high level, such as improving throughput, reducing delay, optimizing resource allocation, or maintaining service-level objectives. However, wireless systems require precise and executable control logic. In O-RAN, for example, a deployable xApp must identify relevant key performance measurements, map user terminology to standard service-model definitions, verify platform capabilities, subscribe to telemetry streams, construct valid control messages, and integrate with the near-real-time RIC. A high-level objective cannot be directly executed unless it is translated into these concrete system-level artifacts. Autonomous network management therefore requires an end-to-end generation pipeline that can transform user intent into deployable and standards-compliant network software.

These challenges share a common difficulty: they require reasoning across demonstrations, environments, user intents, specifications, domain knowledge, system constraints, and runtime behavior. Conventional optimization and learning methods are powerful for narrowly defined tasks, but they are not naturally designed to connect all of these sources of information. This motivates domain-grounded generative-agent workflows as a foundation for autonomous

network management.

1.3 Thesis Statement

This dissertation claims that autonomous network management can be realized by decomposing complex wireless networking tasks into domain-grounded generative-agent workflows that reason over demonstrations, environments, user intents, and specifications to produce adaptive strategies, coordinated decisions, and executable network logic.

1.4 Dissertation Contributions

This dissertation makes the following contributions.

First, this dissertation introduces generative-agent-based communication protocol design. We develop CP-AgentNet, a framework that uses generative agents as autonomous decision-makers for communication protocol design. CP-AgentNet addresses the limitations of handcrafted and deep-reinforcement-learning-based protocols by using multi-agent role specialization, few-shot demonstrations, self-reflection, and autonomous strategy implementation. Through protocol design examples such as multiple access and TCP adaptation, this work demonstrates that generative agents can design explainable communication strategies for heterogeneous network environments.

Second, this dissertation develops autonomous xApp synthesis for O-RAN. We present Auto-xApp, a structured framework that translates natural-language operator requirements into deployable O-RAN xApps. Instead of directly generating code from a prompt, Auto-xApp decomposes xApp construction into intent translation, semantic mapping, capability checking, and code synthesis. This enables the generation of xApps that are aligned with O-RAN specifications, compatible with platform capabilities, and executable in a real near-real-time RIC environment.

Third, this dissertation proposes generative-agent-based xApp conflict detection and

mitigation. We develop ORCA, a conflict management framework for O-RAN environments where independently developed xApps may interact in direct, indirect, or implicit ways. ORCA uses feature importance reasoning to identify how control parameters affect key performance indicators, and it employs generative agents to infer how feature importance changes under unseen network environments. This enables fine-grained and context-aware conflict mitigation beyond static historical profiles.

Fourth, this dissertation introduces proactive reasoning for implicit misconfiguration prevention. We develop PRIME, a framework for identifying hidden configuration dependencies before they manifest as runtime failures. PRIME reads cellular protocol specifications, semantically labels configurable parameters, reduces the candidate dependency space using a taxonomy of implicit constraints, and derives configuration constraints through structured reasoning. This shifts misconfiguration discovery from reactive field measurement toward proactive specification-grounded prevention.

Finally, this dissertation establishes a unified vision of autonomous wireless intelligence. Across protocol design, xApp generation, conflict management, and misconfiguration prevention, the dissertation shows that generative agents can provide capabilities that are difficult to obtain from conventional optimization or learning alone: interpreting intent, reasoning over domain knowledge, synthesizing executable artifacts, explaining decisions, and identifying hidden dependencies. Together, these contributions move wireless systems toward more autonomous, explainable, and reliable design and management.

1.5 Dissertation Organization

The remainder of this dissertation is organized as follows.

Chapter 2 presents background and related work on wireless protocol design, O-RAN architecture, xApp development, xApp conflict management, cellular configuration, and generative-agent-based reasoning.

Chapter 3 presents CP-AgentNet, a generative-agent-based framework for autonomous and explainable communication protocol design. This chapter focuses on how generative agents can replace or complement conventional handcrafted and deep-reinforcement-learning-based protocol design methods.

Chapter 4 presents Auto-xApp, a framework for synthesizing deployable O-RAN xApps from high-level user requirements. This chapter shows how generative agents can bridge the gap between operator intent and standards-compliant xApp implementation.

Chapter 5 presents ORCA, a generative-agent-based framework for detecting and mitigating conflicts among independently developed xApps. This chapter focuses on conflict reasoning under unseen and dynamically changing O-RAN environments.

Chapter 6 presents PRIME, a framework for proactive identification of implicit configuration dependencies in cellular networks. This chapter shows how generative agents can reason over protocol specifications to prevent hidden misconfigurations before deployment.

Chapter 7 concludes the dissertation by summarizing the main findings, discussing limitations, and outlining future directions toward fully autonomous wireless network design and management.

Chapter 2

Background and Related Work

This chapter provides the background necessary to understand the technical contributions of this dissertation. The dissertation studies generative agents for autonomous wireless intelligence across two broad directions: communication protocol design and Open RAN management. Accordingly, this chapter first reviews the challenges of protocol design in heterogeneous wireless environments. It then introduces the Open Radio Access Network (O-RAN) architecture, xApps, and the practical challenges of developing and deploying xApps. Next, it discusses conflicts among independently developed xApps and configuration-dependent misconfigurations in cellular networks. Finally, it introduces large language models, generative agents, retrieval-augmented generation, and related methods that support reasoning and synthesis in the systems developed in this dissertation.

2.1 Wireless Network Heterogeneity and Protocol Design

Modern wireless networks are increasingly heterogeneous. Multiple access technologies, protocol stacks, traffic types, and deployment scenarios coexist in the same operational environment [182]. Wi-Fi, cellular networks, private networks, and emerging programmable radio access architectures may share spectrum, infrastructure, or application workloads [125, 182]. In such environments, communication protocols must not only optimize performance within a single network, but also coexist with other protocols and adapt to changing network conditions [181].

Traditional communication protocols are usually designed through expert-driven engineering and standardization [3]. This process has produced reliable and widely deployed systems, but it also requires substantial human effort. A protocol designer must understand the network objective, identify relevant state variables, define an action space, design control rules, and evaluate the resulting behavior under diverse scenarios. When the environment changes, the protocol may need to be redesigned or manually tuned [171]. This becomes particularly difficult when multiple protocols interact, such as when different medium access schemes coexist [182] or when congestion-control algorithms compete over shared network paths [33, 63].

Machine learning, especially deep reinforcement learning (DRL), has been explored as an alternative approach to protocol design [98, 102]. In DRL-based designs, network nodes learn decision policies from interaction with an environment [63, 99, 182]. Such methods can outperform handcrafted rules in specific scenarios [99, 182], but they also introduce several limitations. They often require large amounts of training data, careful selection of neural network architectures, hyperparameter tuning, and retraining when the environment changes [135, 172]. In addition, their learned policies are often difficult to interpret, which complicates debugging, verification, and deployment in real systems [132].

These limitations motivate new approaches for autonomous protocol design. In this dissertation, protocol design is treated not only as a numerical optimization problem but also as a reasoning problem. A protocol designer must reason about objectives, states, actions, environmental changes, and coexistence with other protocols. Generative agents provide a promising mechanism for such reasoning because they can interpret high-level instructions [20], use few-shot examples [20], generate strategies [119], revise decisions through reflection [138], and produce explanations in natural language [169]. Chapter 3 develops this direction through CP-AgentNet.

2.2 Open Radio Access Network

Open Radio Access Network (O-RAN) is an emerging architecture that aims to make cellular networks more open, programmable, and interoperable [111, 125]. Unlike traditional RAN deployments, where hardware and software components are often tightly integrated by a single vendor, O-RAN promotes disaggregation through standardized interfaces [15, 125]. This enables network operators to combine components from different vendors and to introduce software-driven intelligence into the RAN control loop [14].

A central component of O-RAN is the RAN Intelligent Controller (RIC). The RIC is divided into two logical control domains: the non-real-time RIC and the near-real-time RIC [111, 125]. The non-real-time RIC operates at longer timescales and is typically responsible for policy management, model training, analytics, and high-level optimization [112]. The near-real-time RIC operates at shorter timescales and hosts control applications that monitor and adapt RAN behavior [113].

The near-real-time RIC communicates with RAN nodes through the E2 interface [113, 125]. This interface enables applications in the RIC to subscribe to network measurements and issue control actions. E2 service models define the semantics of monitoring and control. For example, E2SM-KPM specifies key performance measurements such as throughput, PRB usage, and radio-quality indicators [115], while E2SM-RC specifies control primitives for modifying RAN behavior [116]. A RAN node advertises its supported monitoring and control capabilities through E2 setup procedures, and an xApp can only use the metrics and actions exposed by the target platform [113].

O-RAN programmability creates a new opportunity for intelligent network management [14]. Instead of relying only on fixed vendor-defined control logic, operators can deploy applications that optimize resource allocation [35], mobility, energy efficiency, interference management, traffic steering, and anomaly detection [124, 125]. However, this openness also introduces new forms of complexity. Applications must be correctly integrated with service

models, platform capabilities, and runtime constraints [51]. Moreover, multiple independently developed applications may interact through shared resources and performance metrics [4, 17]. These issues motivate the Open RAN management contributions developed in this dissertation.

2.3 xApps and RAN Control

An xApp is a near-real-time control application deployed on the near-real-time RIC [113, 125]. It receives telemetry from RAN nodes, executes decision logic, and may issue control commands through standardized interfaces [124]. Examples include xApps for radio resource allocation [35], mobility optimization, load balancing, energy saving, interference mitigation, anomaly detection, and network slicing [16, 124].

The basic workflow of an xApp consists of monitoring, decision-making, and control [16, 124]. In the monitoring stage, the xApp subscribes to relevant measurements through the E2 interface [115]. In the decision stage, it processes the received measurements and determines whether a control action is needed. In the control stage, it constructs and sends a valid control message to the RAN node [116]. This workflow may be implemented using rule-based logic, classical optimization, supervised learning, unsupervised learning, or reinforcement learning [102, 124].

While this control model is powerful, it also makes xApp development difficult [51]. A developer must understand not only the control objective but also O-RAN service models, measurement names, control styles, ASN.1 message structures, RIC software development kits, and platform-specific capabilities [16, 115, 116]. A high-level objective such as “improve throughput for a slice” must be translated into concrete monitoring metrics, target scopes, control parameters, trigger conditions, and message encodings [101]. Minor mismatches between the intended control logic and the actual service-model semantics can cause runtime failures or unsupported control attempts [51].

This dissertation addresses xApp development in Chapter 4. Rather than treating xApp

implementation as a manual engineering task, Auto-xApp studies how generative agents can translate high-level requirements into structured intent [101], align that intent with O-RAN specifications [74], verify platform capabilities, and synthesize deployable xApp code [25].

2.4 xApp Conflicts in Open RAN

The openness of O-RAN allows multiple xApps to coexist on the same near-real-time RIC [113, 125]. These xApps may be developed by different vendors or teams, and each may optimize a different network objective. Although this modularity accelerates innovation, it can also create conflicts [4, 114]. Since xApps share the same underlying RAN resources and influence common performance metrics, the action of one xApp may interfere with the objective of another [17].

xApp conflicts can appear in several forms [4, 114]. A direct conflict occurs when multiple xApps attempt to control the same parameter or resource in incompatible ways. An indirect conflict occurs when different parameters controlled by different xApps influence the same KPI [6]. An implicit conflict occurs when xApps optimize different metrics or control different parameters, but their actions still interfere through hidden dependencies in the network [4]. These conflicts are difficult to detect because they may depend on traffic load, channel conditions, UE mobility, cell density, and other environmental factors [17, 62].

Existing conflict-management methods often rely on historical data or statistical profiles [17, 163]. Such methods may work when the deployment environment is similar to the training or profiling environment. However, RAN conditions are dynamic [124]. A conflict profile derived under one traffic pattern, load condition, or mobility regime may not hold under another. As a result, conflict detection must become context-aware and capable of generalizing to unseen environments [6, 186].

Chapter 5 addresses this problem through ORCA. ORCA uses feature importance [91, 132] to reason about how control parameters affect KPIs and uses generative agents to infer how

this importance changes when the network environment shifts. This enables more fine-grained and adaptive conflict detection than methods that rely only on static historical profiles [4, 163].

2.5 Cellular Configuration and Misconfiguration

Cellular networks expose a large number of configuration parameters that govern radio access behavior [2, 3]. These include measurement thresholds, hysteresis offsets, reporting configurations, timers, priorities, cell-selection rules, and handover-related parameters [3, 78]. Such parameters determine how user equipment discovers cells, evaluates radio quality, reports measurements, maintains connectivity, and performs mobility procedures [3].

Although individual parameters may be valid according to a specification, combinations of parameters can still lead to undesirable behavior [187]. For example, two thresholds may create an overlap region where contradictory actions are triggered [78]. Timers may be configured in an unsafe order. Parameters used in different RRC states may be inconsistent with one another [187]. A configuration stage may omit information required by a later stage [187]. These failures are difficult to detect because the dependency is often implicit rather than explicitly stated as a single rule [187]. Analogous problems are well documented in wired networks, where individually valid router configurations combine into network-wide misconfigurations [96].

Prior measurement studies have shown that configuration-dependent failures can cause handover loops [78, 80], missed secondary cells [86, 87], problematic handoffs [78, 153], and throughput degradation [87, 187]. However, such failures are often discovered after deployment, when abnormal behavior is observed in the field [79, 97]. This reactive approach is increasingly insufficient as cellular networks become more complex and programmable. In O-RAN environments, xApps may continuously tune parameters at runtime [35, 124], which can further increase the risk of hidden dependencies and unsafe combinations [114].

Chapter 6 develops PRIME to address this issue. PRIME studies how generative agents can reason over cellular specifications [2,3] to identify implicit dependencies among configuration

parameters before deployment. Instead of waiting for failures to appear in operational traces [79, 187], PRIME aims to build a reusable constraint database that can be used to flag unsafe configurations proactively.

2.6 Large Language Models and Generative Agents

Large language models (LLMs) are neural models trained on large-scale text and code corpora [20, 25, 159]. They can generate natural language, summarize documents, answer questions, write code [25], and perform multi-step reasoning when provided with suitable prompts and context [167, 169]. While LLMs were initially used primarily as text-generation systems, they are increasingly used as components of larger agentic systems [119, 178].

A generative agent is an LLM-based system that can perceive inputs, maintain context, plan actions, use tools, consult memory, retrieve external knowledge, and produce outputs that affect an environment or workflow [119, 134, 165, 178]. In this dissertation, generative agents are used not merely to generate text, but to support reasoning and synthesis tasks in wireless systems [10, 95]. They interpret user requirements [101], reason over protocol behavior, synthesize control logic [25], retrieve relevant domain knowledge [74], identify dependencies, and generate explanations.

Several techniques are important for making generative agents reliable in technical systems. Few-shot prompting provides examples that guide the agent’s behavior without full model retraining [20]. Chain-of-thought-style reasoning can structure multi-step analysis [167, 169], although it must be grounded carefully to avoid unsupported conclusions [158]. Retrieval-augmented generation (RAG) supplies relevant external documents to reduce hallucination and align outputs with domain knowledge [42, 74]. Multi-agent role specialization decomposes complex tasks into smaller roles, such as observation, strategy generation, verification, and implementation [54, 76, 173]. Programmatic tool use allows agents to generate and execute code for tasks that require precise computation or structured processing [41, 134].

Despite their potential, generative agents also have limitations. Their outputs can be inconsistent, incomplete, or hallucinated if they are not properly grounded [60, 64]. They may struggle with precise numerical reasoning [37], long documents [85], or strict interface requirements [51]. Domain-specific benchmarks show that this gap persists for telecom and O-RAN knowledge in particular [154]. Therefore, the systems in this dissertation combine generative reasoning with structured workflows, deterministic validation, external knowledge retrieval [74], and domain-specific constraints. This design principle appears across CP-AgentNet, Auto-xApp, ORCA, and PRIME.

2.7 Summary

This chapter reviewed the background for the dissertation. Modern wireless networks are becoming heterogeneous, programmable, and increasingly difficult to design and manage manually [125]. Protocol design requires adaptability and explainability [182]. O-RAN introduces xApps as programmable control applications, but xApp development requires careful alignment with specifications and platform capabilities [51]. Independently developed xApps can conflict, especially under changing network environments [4, 17]. Cellular configuration parameters can also interact through hidden dependencies that lead to misconfigurations [78, 187].

These challenges motivate a common direction: wireless systems need a reasoning and synthesis layer that can connect human intent, technical specifications, domain knowledge, executable control logic, and runtime behavior [10, 172]. The following chapters develop this direction through generative-agent-based protocol design, xApp synthesis, xApp conflict management, and proactive misconfiguration prevention.

Chapter 3

Generative Agents for Communication Protocol Design

3.1 Introduction

This chapter addresses the first technical component of the dissertation: generative-agent-based communication protocol design. While later chapters focus on Open RAN management, this chapter studies whether generative agents can directly design communication protocols for heterogeneous environments. The chapter is based on CP-AgentNet, a framework that uses LLM-powered agents as autonomous and explainable decision-makers for protocol design.

Conventional communication protocols are meticulously established through standardization processes. However, challenges arise from the explosive demands for data traffic and ever-increasing network heterogeneity. Typically designed for a single type of network, standard communication protocols lack the adaptability to coexist in diverse networks. Enabling operations across heterogeneous networks concurrently necessitates the creation of a new protocol for each coexistence scenario, demanding significant expertise and effort.

DRL (deep reinforcement learning) has emerged as a powerful tool for devising communication protocols, which are essentially decision-making algorithms executed by the agents, i.e., network nodes. Despite their potential, DRL methods have several drawbacks. First, choosing an appropriate neural network architecture and tuning hyperparameters are crucial yet often require tedious trial-and-error experimentation. Moreover, architectures optimized for a specific

environment often generalize poorly to unseen conditions, necessitating costly redesign or re-training [45, 183], thus limiting practical adaptability. Second, the decision-making process in DRL models is often considered a “black box” [50], making it difficult to interpret or verify the rationale behind actions. Third, DRL models are data hungry, requiring not only large-scale data collection but also labor-intensive labeling and preprocessing. Data acquisition may introduce non-trivial overhead, potentially disrupting normal network operations.

Recently, generative agents—specifically those empowered by large language models (LLMs), hereafter referred to as LLM-agents—have gained attention for their ability to reason, plan, and act in interactive environments. However, their potential in communication protocol design remains largely unexplored. While recent work such as NADA [48] leverages LLMs to automate the design of DRL agents, these approaches remain fundamentally constrained by the inherent limitations of DRL itself.

This chapter presents CP-AgentNet¹, the first framework designed to create communication protocols using LLM-agents as autonomous decision-makers. CP-AgentNet addresses the inherent limitations of both handcrafted and DRL-based protocols. By leveraging the in-context learning capabilities of LLMs, these LLM-agents can quickly interpret and adapt to instructions provided through context or interaction sequences, eliminating the need for explicit retraining when conditions change. Our framework requires crafting only a few demonstrations, capitalizing on the strong generalization abilities of LLM-agents to efficiently guide their training. This significantly reduces the effort involved in selecting neural network architectures, tuning hyperparameters, and gathering, labeling, and preprocessing large datasets. Additionally, since the outputs generated by LLM-agents are in natural language, they can be easily interpreted and understood by humans, greatly enhancing interpretability and facilitating straightforward interactions.

However, despite substantial advantages of using LLM-agents, their use still presents significant challenges. First and foremost, although the LLM-agent is capable of handling tasks

¹All prompts used in CP-AgentNet are available at: <https://github.com/BigIron0315/CP-AgentNet>

across various disciplines, a single LLM-agent struggles to manage multiple tasks simultaneously [61]. In particular, an agent in this framework must make transmission decisions while concurrently monitoring the network environment, adding complexity to its operational tasks. To address this, we employ multi-agent role-play, an approach that has already proven effective in domains such as programming and mathematical reasoning. Harnessing multiple agents is advantageous, as each agent plays a distinct role, thereby simplifying complex tasks [34]. Additionally, this approach significantly enhances explainability by making the decision-making process traceable.

Second, establishing an autonomous design framework using LLM-agents presents unique challenges, primarily due to the absence of explicit optimization methods such as gradient descent, which are commonly used in traditional DRL to systematically improve policies. Existing LLM-agent frameworks often compensate for this limitation by relying heavily on explicit human feedback [104, 174], thereby requiring substantial user effort. To address this fundamental challenge, we introduce *Progressive Strategy Augmentation (PSA)*, an approach specifically designed for LLM-agents. Rather than updating parameters via gradient-based optimization, PSA utilizes a structured self-reflection mechanism [139] to incrementally refine strategies. Guided by carefully crafted prompts, the LLM-agent autonomously reviews previous episodes, identifies underlying reasons for suboptimal outcomes, and progressively revises its strategies. The refined strategies obtained through multiple self-reflection episodes are subsequently aggregated, and redundant or conflicting components are systematically identified and removed. In essence, PSA serves as a symbolic counterpart to gradient descent, leveraging the generative reasoning and in-context learning capabilities inherent to LLMs. Another method we employ for autonomous design is *Autonomous Strategy Implementation (ASI)*. LLM responses can often be unpredictable, especially when prompts solicit open-ended design strategies. ASI plays a critical role in such cases by leveraging the LLM’s programming capabilities to autonomously interpret and implement the generated strategies, thereby maintaining autonomy without requiring human intervention.

Lastly, to address the inherent variability of LLMs, which stems from their probabilistic token prediction, we employ an LLM ranker [128] to enhance consistency and mitigate planning errors. Inspired by The Wisdom of Crowds [149]—which suggests that large groups often outperform individual experts in problem-solving, decision-making, and prediction—we apply this principle to improve CP-AgentNet’s stability. The LLM ranker evaluates multiple output candidates and selects the most reliable option as the final output. This approach significantly enhances the consistency of our framework.

As a use case of CP-AgentNet, we designed the LLMA (LLM-agents-based multiple access) protocol for heterogeneous networks. To verify the effectiveness of the LLMA protocol, we conducted comprehensive simulations across various scenarios. Specifically, we first established the ideal operation of an AWARE node—a hypothetical node equipped with complete knowledge of the environment. We then compared the performance of the LLMA node to that of the AWARE node by measuring RMSE (root mean square error). As a benchmark, we also assessed the performance of a DLMA node, which employs the DRL-based method proposed in [183]. Our results indicate that the LLMA node performs closer to the ideal than the DLMA node. Remarkably, in dynamic environments where nodes using different protocols intermittently join or leave, the LLMA node adapts more swiftly and exhibits a significantly lower RMSE value of 0.0476—just 21.0% of the RMSE of 0.227 resulting from the DLMA node.

In addition, we designed CPTCP (CP-Agent-based Transmission Control Protocol) to coexist with different types of TCP algorithms. There are mainly two types of TCP algorithms: loss-based and delay-based TCP. We show that CPTCP enables effective coexistence with both types of TCP algorithms. Moreover, this demonstrates that our framework is not limited to designing a specific protocol.

In summary, this chapter makes the following contributions:

- We introduce CP-AgentNet, the first framework designed to create communication protocols using LLM-agents as autonomous decision-makers. CP-AgentNet employs multi-

agent role-play to efficiently process tasks and facilitate the explainable design of communication protocols.

- We propose PSA and ASI to facilitate autonomous design. These enable LLM-agents to develop strategies and implement them to optimize network performance, without the need for human intervention.
- We have developed LLMA and CPTCP, specifically designed for a heterogeneous environment using CP-AgentNet. We demonstrate that nodes using these protocols can efficiently coexist with other nodes operating different protocols.

3.2 CP-AgentNet System Design

3.2.1 CP-AgentNet Architecture

CP-AgentNet comprises a CP-agent and three distinct types of memory. The CP-agent includes multiple agents such as a strategy agent, an observer agent, and a node agent, along with a programming assistant, each assigned specific roles to perform tasks. These agents are empowered by the LLM, each using tailored prompts suited to their individual roles.

Strategy Agent. The strategy agent is responsible for developing and refining strategies. It advises the node agent on selecting appropriate actions based on trajectory data and environmental changes and suggests methods to escape from suboptimal actions.

Node Agent. The primary role of the node agent is to determine the actions. Although the node agent generally adheres to the strategy, it relies on the observer agent for assistance in making the final decisions. The node agent adapts its action every T slots, a period which we refer to as the *query period*. Once established, the action is maintained for the duration of the query period.

Observer Agent. The observer agent diligently monitors the environment, focusing on the convergence of the action, any environmental changes, and notable occurrences. Environmental changes occur when new nodes join the network or existing nodes leave the network, or when other nodes change their parameters. Although the observer agent may not identify the specific

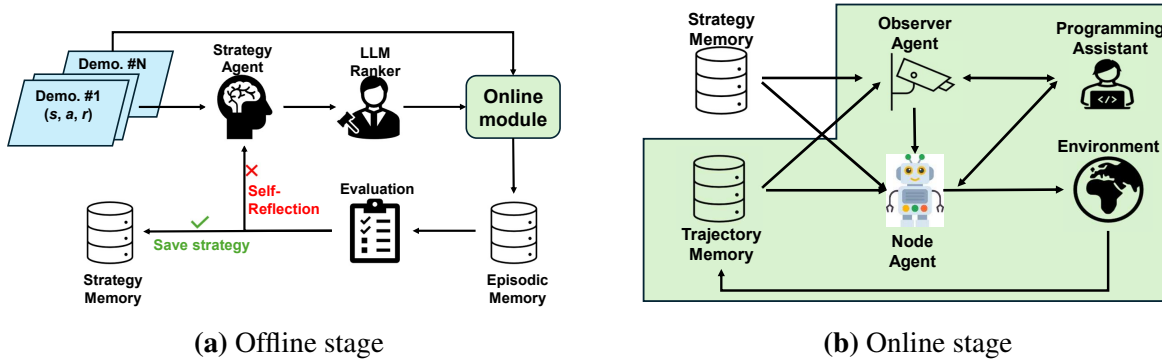


Figure 3.1. Overall workflow of CP-AgentNet

changes, it is adept at detecting when changes occur. Notable occurrences can be any anomalies or outliers that the observer agent detects. For instance, if a specific slot is consistently overused or not used at all, the observer agent identifies this irregularity. Upon identifying such environmental changes or irregularities, the observer agent assists the node agent in adjusting its action accordingly.

Programming Assistant. Given that LLMs are not adept at complex mathematical tasks, we employ a programming assistant built upon LLMs. This agent supports other agents by solving mathematical problems using advanced programming skills. It transforms tasks into programming scripts and returns the executed results to the initiating agent. Additionally, this agent is integral to ASI, a process where the CP-Agent autonomously implements strategies.

Memory. CP-AgentNet employs three types of memories: strategy memory, episodic memory, and trajectory memory. As their names suggest, each memory type is dedicated to storing strategies, episodic data, and trajectories, respectively.

3.2.2 CP-AgentNet Workflow

The operation of CP-AgentNet consists of two stages: an offline stage and an online stage, as illustrated in Fig. 3.1. In the offline stage, LLM-agents develop strategies that can be effectively utilized during the online stage. In the online stage, LLM-agents determine the optimal action based on these strategies.

Generating Few-Shot Demonstrations

To enable CP-AgentNet to learn communication strategies efficiently, we generate few-shot demonstrations leveraging simulation-based action-reward sampling and heterogeneous protocol representations. This process ensures that the LLM-agent can generalize across different network conditions and protocols with minimal training data. While our framework designed MAC protocol and TCP as use cases, applying it to other protocols would require generating separate few-shot demonstrations tailored to their characteristics.

Simulation-Based Demonstration Generation

Instead of relying on large-scale datasets, we construct few-shot demonstrations through targeted simulation by leveraging in-context learning and generalization capabilities of LLM-agent. This process does not require domain expertise, but only a basic understanding of the underlying protocol is sufficient. Given an environment characterized by state space S and action space A , we sample a set of actions and derive the corresponding rewards from the simulation. Specifically, for each scenario, we:

- Sample K actions, $a_1, a_2, \dots, a_K$, from the defined action space A . (e.g., in a slotted ALOHA setup with $N = 3$ nodes, $K = 11$ actions can be uniformly sampled from the range $[0, 1]$ with step size 0.1 representing transmission probabilities)
- Evaluate each action in the simulated environment to obtain the corresponding reward values, $r_1, r_2, \dots, r_K$.
- Construct state-action-reward tuples (s, a, r) , where $s \in S$, $a \in A$, and r is obtained from simulation feedback.
- Store these tuples as part of the few-shot demonstration set $\mathcal{D}$ to be utilized in in-context learning for the LLM-agent.

Heterogeneous Demonstration Generation

To ensure robustness across diverse network protocols, we generate demonstrations for both MAC and TCP protocols. Our framework constructs protocol-specific few-shot demonstrations that capture variations in communication behaviors.

For MAC protocols, we create total 4 demonstrations:

- Three protocol-specific demonstrations for CSMA, TDMA, and ALOHA.
- One dynamic demonstration that adapts to changing network conditions, ensuring adaptability to real-world variations.

For TCP protocols, we generate total 3 demonstrations:

- Two protocol-specific demonstrations for Loss-base TCP and Delay-based TCP.
- One dynamic demonstration to account for variations in congestion and network dynamics.

Strategy generation

Strategy generation focuses solely on developing effective strategies based on the provided few-shot demonstrations. Rather than learning from a vast dataset, our approach enables LLM agents to infer strategies by analyzing various heterogeneous protocols and environmental scenarios. Through this process, the agent formulates strategies that can be effectively applied during the online adaptation stage. By distilling knowledge from limited demonstrations, the strategy generation phase ensures efficient adaptation while significantly reducing training time. We define a strategy π as a function that maps the state space S to the action space A , i.e., $\pi : S \rightarrow A$. Given a set of few-shot demonstrations $\mathcal{D} = (s_i, a_i, r_i, s'_i)_{i=1}^N$, the optimal strategy is generated by maximizing the expected rewards over the sampled actions:

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{(s,a,r) \sim \mathcal{D}} [R(s, a)] \quad (3.1)$$

where $R(s, a)$ represents the reward function. Unlike traditional reinforcement learning, which requires extensive training, CP-AgentNet enables LLM-agents to leverage few-shot demonstrations and their generalization capabilities to construct an initial strategy.

Refining Strategies

After generating an initial strategy, its effectiveness is evaluated by measuring the expected reward:

$$J(\pi) = \mathbb{E}_{s \sim P, a \sim \pi(s)} [R(s, a)] \quad (3.2)$$

where P represents the state distribution. If the obtained performance does not meet the predefined optimal threshold J_{opt} , then the strategy undergoes a refinement process. This is accomplished through self-reflection [139], enabling the LLM-agent to analyze its past decisions and adjust accordingly. The mathematical formulation conceptually represents a gradient-descent process, in which each candidate strategy is iteratively adjusted until it approaches its local optimum, ensuring convergence toward a good strategy. The refinement process updates the strategy iteratively:

$$\pi^{(t+1)} = \pi^{(t)} + \lambda \nabla J(\pi^{(t)}) \quad (3.3)$$

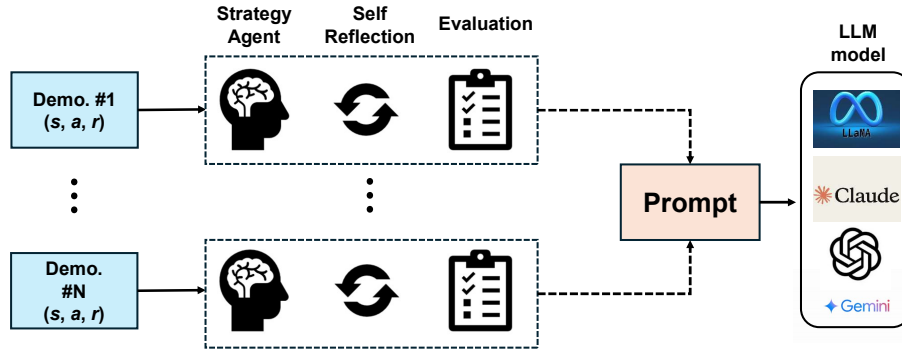
where t represents the iteration index and λ is the step size for adjusting the policy. The process continues until the strategy reaches the desired performance:

$$J(\pi^{(t)}) \geq J_{\text{opt}} \quad \text{or} \quad t = N_{\text{max}} \quad (3.4)$$

By iteratively refining the strategy, CP-AgentNet ensures that suboptimal policies are corrected, allowing for efficient adaptation to dynamic environments. The refinement step enables the agent to incorporate feedback, adjust its decision-making process, and improve strategy performance without requiring extensive online learning. An example prompt for strategy generation.

Online Adaptation

Once strategies are learned in the offline stage, the LLM-agent dynamically selects actions during the online stage while adapting to environmental variations. This adaptation is supported by the observer agent, which monitors the node agent’s actions, evaluates environmental conditions, and assists in decision-making. During execution, the node agent follows the pre-learned strategy π^* , while the observer agent provides real-time feedback O_t to guide adaptation. Instead of directly modifying the action, the observer adjusts the decision-making process by supplying contextual insights to the node agent, ensuring robust responsiveness to dynamic conditions. Notably, in the online stage, the strategy agent and episodic memory are inactive, and the system operates solely based on the pre-learned strategies and real-time observer feedback.



(a) Structure of PSA

Prompt Used in PSA

The strategy agent developed for the node agent $\{n\}$ individual strategies based on separate demonstrations as listed below:

- Strategy 1: ...
- ...
- Strategy n: ...

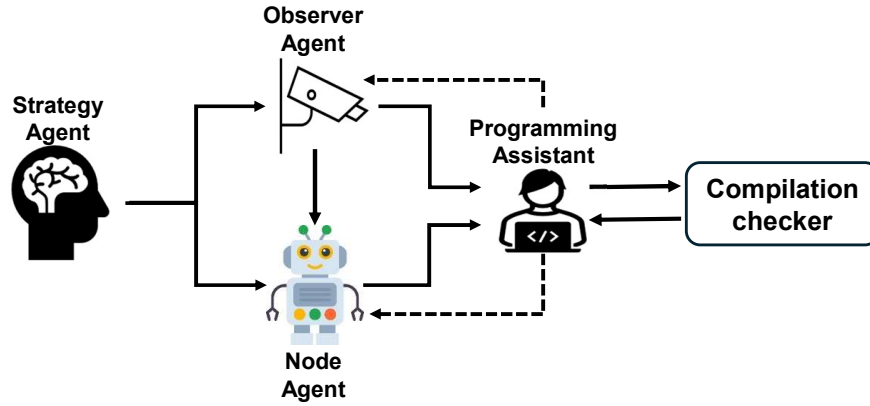
Your task now is to create a **combined strategy** that:

- Incorporates essential elements from all provided strategies.
- Eliminates redundancy and overlap efficiently.
- Maintains the general applicability of the combined strategy to different environments, where optimal actions vary, but the action-reward relationship remains similar.

Ensure that no individual strategy is omitted from the final combined version—only remove redundant or repetitive parts. Provide your combined strategy clearly and succinctly.

(b) Prompt used in PSA

Figure 3.2. Structure and prompt of PSA that synthesizes multiple strategies by resolving conflicts and eliminating redundancy.



(a) Structure of ASI

Prompt Used in ASI

Strategy to escape from suboptimal action is $\{ \dots \}$.

Based on the provided strategy, create a Python function named $\{function_name\}$.

- Take the initial action as input.
- Return a final action $\{range\}$ that aligns with your strategy to escape suboptimal decisions.
- Clearly implement your strategy logic, ensuring it can be generalized to other environments.

Provide only the Python code for the function without additional explanations or comments.

(b) Prompt used in ASI

Figure 3.3. Structure and prompt of ASI. Implementation is delegated to the programming assistant by the node or observer agent.

3.2.3 Autonomous Design

To enable fully autonomous protocol design, CP-AgentNet must not only generate strategies but also evaluate and manage them without relying on explicit optimization techniques such as gradient descent or human feedback. After a strategy is generated and refined through

self-reflection, the system must decide whether to retain, discard, or integrate it into its current strategy set. To support this, we propose Progressive Strategy Adaptation (PSA), an iterative mechanism that manages the evolution of the strategy set. As illustrated in Fig. 3.2, each few-shot demonstration leads to a candidate strategy, and PSA determines how to update the strategy set accordingly. This update is defined as:

$$\Pi^{(t+1)} = \left(\Pi^{(t)} \cup \{\pi_{\text{new}}\} \right) \setminus \{\pi_{\text{obsolete}}\} \quad (3.5)$$

Here, π_{new} is a newly generated strategy, and π_{obsolete} represents redundant or conflicting entries filtered out during self-assessment. While this process resembles batch-based learning in ML, CP-AgentNet achieves adaptation with only a few demonstrations. By iteratively applying PSA, CP-AgentNet constructs a concise, high-quality strategy set that supports scalable protocol design.

In addition, since the LLM’s responses can be unpredictable, CP-AgentNet requires an autonomous mechanism to implement generated strategies without human intervention. To this end, we introduce Autonomous Strategy Instantiation (ASI), in which the LLM generates executable functions that are stored and reused throughout the system’s operation. This design avoids redundant queries and ensures efficient execution without manual involvement. As illustrated in Fig. 3.3, the strategy agent develops high-level strategies for both the node agent and the observer agent. If a strategy requires assistance from the programming assistant, the node or observer agent consults the programming assistant. Upon request, the programming assistant implements the strategy and checks for compile errors using an external tool. If an error is detected, the assistant iteratively corrects it. Once validated, the resulting function is invoked by the original requester as needed. For example, if the strategy agent suggests the ϵ -greedy algorithm for an exploration-exploitation trade-off, the node agent implements this with the assistance of a programming assistant, which translates it into a function that operates autonomously at runtime. This end-to-end process not only enables autonomous execution

but also enhances robustness; by combining the previously described compilation checker and function generation with the LLM ranker (see §3.2.4), ASI achieves significantly improved reliability.

3.2.4 Stability of CP-AgentNet

Despite their advantages, LLMs’ inherently inconsistent outputs can compromise the stability of systems like CP-AgentNet. This inconsistency arises from their fundamental mechanism of predicting the next token’s probability distribution, even at zero temperature. Such unpredictability can degrade CP-AgentNet’s performance or lead to operational failures. To address this, we implement an LLM ranker [128], inspired by *The Wisdom of Crowds* [149]. The LLM-agent queries the LLM twice, reversing the order of inputs to mitigate the influence of input sequencing on response generation. Both responses are then evaluated by another LLM acting as a judge, which selects the most optimal strategy autonomously.

Querying the LLM with Different Input Orders

Let Q be the original query from the CP-agent. To reduce bias from input ordering, we construct two variations: $Q_1 = Q$ and $Q_2 = \text{ReverseOrder}(Q)$. For each query, the LLM generates a response: $R_1 = \text{LLM}(Q_1)$ and $R_2 = \text{LLM}(Q_2)$, where R_i represents the strategy proposed by the LLM.

Selecting the Best Strategy Using Another LLM as a Judge

Instead of manually defining evaluation criteria, we employ another LLM as a strategy judge J_{LLM} , which compares two candidate strategies and returns the preferred one as $R^* = J_{\text{LLM}}(R_1, R_2)$. The selection is entirely autonomous, relying on the judge LLM’s ability to infer which response is more optimal.

Feasibility in Offline Processing

While the LLM ranker significantly improves decision consistency and overall reliability, it also increases processing time. However, since the LLM ranker is primarily utilized during the

offline stage—where time constraints are less stringent—its impact on real-time performance is minimal. Specifically, the ranker is applied during the formulation of transmission strategies rather than during the time-sensitive online stage. The effectiveness of the LLM ranker will be demonstrated in Section 3.4.

3.3 Protocol Use Cases with CP-AgentNet

In this section, we describe how we tailor the CP-AgentNet framework to design specific protocols. As use cases, we develop LLMA, a multiple access protocol, and CPTCP, a transmission control protocol, both designed for heterogeneous environments using CP-AgentNet.

3.3.1 LLMA

Heterogeneous networks

We consider heterogeneous networks where multiple nodes transmit data packets through a shared channel, fundamentally adhering to the assumptions outlined in DLMA [183]. We assume that the time slots are synchronized among these nodes, each using different MAC protocols. Each node initiates its data packet transmission at the beginning of a slot and completes it within the same slot. Every node consistently has a data packet to transmit. If more than one node transmits a data packet using the same channel simultaneously, a collision occurs, rendering those transmissions unsuccessful. We consider heterogeneous network scenarios where at least one “heteronode” coexists with slotted ALOHA or TDMA nodes. A slotted ALOHA node transmits a data packet with a fixed probability q in each time slot, while a TDMA node transmits data packets at X specific slots out of a frame composed of ten slots. The term “heteronode” refers to nodes such as AWARE node, DLMA node, or LLMA node. The AWARE node, considered an ideal solution, has complete knowledge of the environment, including the number of slotted ALOHA and TDMA nodes and their parameters, represented by q and X . The optimal behavior of the AWARE node can be derived as shown in [179]. Consequently, our goal is to enable the LLMA node to perform as closely as possible to the AWARE node. A DLMA node utilizes the

DLMA protocol, which is developed using a DRL method, to make transmission decisions. It can listen to the channel and observe whether other nodes' transmissions are successful or if the channel is idle. Similarly, an LLMA node employs the LLMA protocol, which is developed through our CP-AgentNet framework. All assumptions for LLMA are the same as for DLMA unless specified otherwise.

Additionally, we consider more complex scenarios where the heteronode coexists not only with TDMA and ALOHA nodes but also with CSMA (carrier-sense multiple access), EB (exponential backoff window)-ALOHA, and FB (fixed window)-ALOHA nodes. The CSMA node generates a random value of w within the range $[0, W-1]$, and senses the channel is busy or not. If it is sensed as idle, w is decreased by one, otherwise it is frozen. If this value reaches to zero, the CSMA node transmits data. Only the CSMA node has the capability to sense the channel, and the sensing time is negligible compared to the packet length. An FW-ALOHA node generates a random value of w within the range $[0, W-1]$ after transmission, it then waits for w slots for the next transmission. An EB-ALOHA node's operation is basically the same as that of an FW-ALOHA node. However, an EB-ALOHA node doubles its window size upon a collision with other nodes, up to a maximum window size of $2^m W$. In these scenarios, only DLMA and LLMA nodes are used as heteronodes, as we do not derive the ideal behavior for the AWARE node.

Objective function of heteronode

We set the heteronode objective to maximize α -fairness throughput with $\alpha = 1$, rather than solely maximizing total throughput. The α -fair function is

$$g_{\alpha}(x) = \begin{cases} \log x, & \alpha = 1, \\ \frac{x^{1-\alpha}}{1-\alpha}, & \text{otherwise,} \end{cases} \quad (3.6)$$

and the overall objective is $f(\vec{x}) = \sum_{i=1}^N g_{\alpha}(x_i)$, where N is the number of nodes and x_i is the throughput of node i . Setting $\alpha = 0$ maximizes total throughput, while $\alpha = 1$ maximizes proportional fairness.

LLMA design

The goal of LLMA is to optimize the action of a node agent based on strategies and real-time environmental feedback. In this context, the action determines the likelihood of transmitting in each time slot, setting the transmission probability for each slot. At time step t , the action a_t is selected from continuous range $[0, 1]$. The observation space at time t , denoted as o_t , includes ‘S’ for ‘SUCCESSFUL’, ‘C’ for ‘COLLIDED’, or ‘I’ for ‘IDLE’. It is important to note that this observation o_t is distinct from the observer feedback O_t , where O_t represents higher-level real-time feedback used by the LLM agent to adjust its action. In this setup, the observation space directly serves as the state space S . Rewards from the environment, given after an action, are defined as follows: $r_{t+1} = 1$ if o_t is ‘S’, and $r_{t+1} = 0$ if o_t is either ‘C’ or ‘I’. Assuming there are N nodes in the network, the node agent receives an N dimension vector of rewards from the environment. Each element of the vector represents the transmission result of one particular node. The actions taken by the node agent and the rewards from the environment are essentially the same as those in DLMA [183].

The node agent adapts its action every T slots, referred to as the *query period*. For decisions to remain valid, this period should be shorter than the coherence time—defined here as the average duration over which the active user set remains stable. Since user arrivals and departures typically occur on the order of several seconds, a 1-second query period is a practical. During this time, the accumulated actions and rewards, along with the observations, are stored in the trajectory memory. Before the node agent adapts the action at the end of the query period, the observer agent monitors three aspects: the convergence of the plan, any environmental changes, and any notable occurrences. With assistance from the programming assistant, if the observer agent determines that the transmission has converged, the node agent ceases adaptation and

implements the strategy to escape from suboptimal as described in §3.2.3. If the observer agent detects any environmental changes, it notifies the node agent to adjust the plan accordingly. When coexisting with a TDMA node, if the LLMA node’s observer agent notices that specific slots are consistently used, it advises the node agent to avoid these slots. Through collaboration with the observer agent and the programming assistant, the node agent decides the action for each slot.

3.3.2 CPTCP

Heterogeneous TCPs

We consider heterogeneous TCP scenarios in which CPTCP can coexist with different TCP algorithms. There are mainly two types of TCP algorithms: loss-based and delay-based TCP. Fairness issues arise when a loss-based TCP flow coexists with a delay-based TCP flow, as the loss-based TCP reactively reduces its congestion window size, while the delay-based TCP proactively reduces it [21]. Although previous studies have attempted to overcome this problem, our goal is to mitigate it by leveraging the in-context learning capability of the LLM, ultimately enabling CPTCP to effectively coexist with both loss-based and delay-based TCP flows without the need for hand-crafted design requiring human effort.

Objective function of CPTCP

Since we focus on fairness when coexisting with different types of TCP flows, our objective is to maximize Jain’s fairness index, which is defined:

$$f(x_1, \dots, x_N) = \frac{(\sum_{i=1}^N x_i)^2}{N \sum_{i=1}^N x_i^2} \quad (3.7)$$

where x_i is the throughput of i th flow and N is the number of flows. In a completely fair scenario, $f(\vec{x}) = 1$. Conversely, if a single TCP flow monopolizes the link, $f(\vec{x}) = \frac{1}{N}$. Thus, the closer $f(\vec{x})$ is to 1, the greater the fairness; conversely, the closer $f(\vec{x})$ is to $\frac{1}{N}$, the lower the fairness.

CPTCP Design

The design procedure for CPTCP follows the same foundational principles as LLMA, with modifications tailored to its specific objectives. Accordingly, the objective of CPTCP is to optimize the action of a node agent based on strategies and real-time environmental feedback. In this context, the action determines the congestion window. At time step t , the action a_t represents the congestion window size, which is a positive integer within a predefined range. Specifically, we define the action space as $a_t \in [1, C_{\max}]$, where $C_{\max}$ is the maximum congestion window size, determined by factors such as the receiver’s advertised window and network conditions. While the theoretical limit can be large in high-bandwidth networks, practical implementations impose constraints based on these factors. While the theoretical maximum congestion window can be large in high-bandwidth networks, practical implementations impose constraints based on these factors. The observation space at time t , denoted as o_t , consists of two key metrics: a , the number of received acknowledgments (ACKs), and r , the round-trip time (RTT), forming a vector representation $o_t = [a, r]$. The environment provides rewards following each action, $r_{t+1} = \log(a) - \beta r$. Here, the logarithmic function promotes fairness in throughput allocation, while β serves as a tunable parameter balancing throughput and RTT trade-offs, and is empirically set to 0.5 in our experiments. If the observer agent detects any environmental changes, it notifies the node agent, which then adapts the action accordingly. The final decision is made based on the updated conditions, ensuring an optimized response to dynamic network environments.

3.4 Evaluation

3.4.1 Simulation Setup

To evaluate the effectiveness of CP-AgentNet, we conducted experiments with LLMA and CPTCP. CP-Agent is powered by GPT-4o, with the temperature value set to zero to minimize inconsistencies.

LLMA

We focus on heterogeneous networks incorporating ALOHA and TDMA protocols, alongside heteronodes such as AWARE nodes, DLMA nodes, and LLMA nodes. Since our MAC protocol evaluation already includes direct quantitative comparisons against both an ideal baseline (AWARE node) and a representative DRL-based method, additional comparisons against other methods are unnecessary to sufficiently validate the performance of CP-AgentNet. Specifically, within these networks, only one type of heteronode coexists with nodes operating under TDMA or ALOHA protocols. Each frame consists of ten time slots, each lasting 1ms. Specifically for TDMA nodes, we consider a scenario where only one TDMA node utilizes slots 3 and 5, denoted as $X = 2$. For ALOHA nodes, we set the transmission probability to $q = 0.2$ except for the massive nodes case. For a massive nodes scenario, q is set to 0.02. Moreover, to demonstrate the robustness of LLMA nodes within complex heterogeneous networks, we conduct additional tests involving nodes that utilize CSMA, FW-ALOHA, and EB-ALOHA. We set the contention window size W to 4 for FW-ALOHA. For EB-ALOHA, $W_{\min} = 2$ and $W_{\max} = 32$ with a maximum backoff stage m of 4. For CSMA, we follow the IEEE 802.11 standard with $W_{\min} = 16$ and $W_{\max} = 1024$. The maximum backoff stage is set to 7, but the last two stages are capped at $W_{\max}$. Following the setting in DLMA [183], we use throughput as the performance metric in this chapter. However, since the objective is not to maximize total throughput but to maximize α -fairness throughput for heteronodes, we use RMSE as an additional metric to measure how closely the behavior approximates the ideal operation of the AWARE node.

CPTCP

We first evaluate the throughput of homogeneous TCP configurations, followed by assessments of various combinations involving different TCP algorithms. We utilize a network capacity of 1 Mbps, with throughput and Jain’s fairness index serving as the evaluation metrics. Since the primary goal of our CPTCP evaluation is to demonstrate its capability to coexist with diverse TCP variants, we benchmark CPTCP against representative loss-based (TCP CUBIC [46]),

delay-based (TCP BBR [23]), and Cx-TCP [21] algorithms.

3.4.2 LLMA results

We initially define various scenarios to assess the robustness of LLMA nodes, which are detailed in Table 3.1 along with their respective RMSE results. In the following, we will review these scenarios and analyze the corresponding outcomes.

Homogeneous scenario

We evaluate multiple LLMA nodes in configurations of 2, 4, and 8, and compare the results with the ideal behavior of the AWARE node. After conducting separate simulations with AWARE nodes in the same configurations, we plot the throughput of one AWARE node alongside the LLMA nodes for comparison in Fig. 3.4. The throughput of the LLMA nodes closely aligns with the ideal behavior, demonstrating that the emerging behavior of the LLMA nodes matches the optimal behavior.

Heterogeneous scenario

To demonstrate that LLMA nodes can coexist with nodes using different protocols, we explore various combinations involving TDMA, ALOHA nodes, and heteronodes. The illustrations in Fig. ??, arranged from left to right, depict an AWARE node, a DLMA node, and an LLMA node. Although the LLMA node's behavior visually appears similar to that of the AWARE nodes, we used RMSE to objectively confirm this similarity. The results for all other scenarios are presented in the Table 3.1. Except for one scenario (1H+1T), the RMSE values of the LLMA node are smaller than those of the DLMA node, indicating that the LLMA node's operation is closer to that of the AWARE node. In scenarios where multiple heteronodes coexist with different nodes, the RMSE values of LLMA nodes are approximately 49.4% and 6.8% lower than those of DLMA nodes in the 2A+2H and 1T+2A+3H scenarios, respectively. In summary, our evaluations in heterogeneous scenarios, similar to those examined in [183], demonstrate that

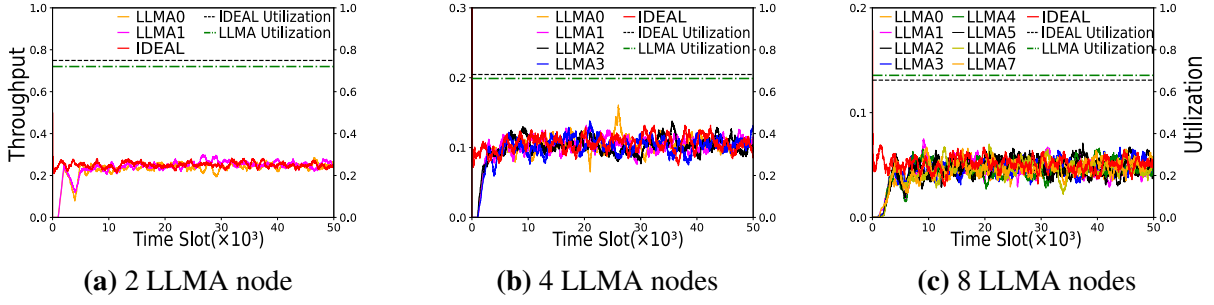


Figure 3.4. LLMA nodes in homogeneous environment

the CP-AgentNet framework manages heterogeneous network environments more effectively than traditional DRL methods.

Complex Heterogeneous scenario

We assess whether the LLMA node can function effectively in more complex scenarios. Initially, we increase the number of ALOHA nodes to 20 to test the robustness of the LLMA node, setting the parameter $q = 0.02$ for these ALOHA nodes. In this scenario of increased complexity, the LLMA node achieves near-optimal performance, evidenced by an RMSE value of 0.044, which is lower than 0.052 observed for the DLMA node. Additionally, we introduce nodes using various other protocols, such as CSMA, EB-ALOHA, and FW-ALOHA. Since the optimal behavior of the AWARE node in coexistence with ALOHA, TDMA, CSMA, EB-ALOHA, and FW-ALOHA nodes has not been derived, we directly compare the performance of LLMA nodes against DLMA nodes by assessing the sum of α -fairness throughput. The α -fairness throughput of each node and the sum of α -fairness throughput are presented in Fig. 3.7. To calculate α -fairness throughput, we multiply the throughput by 100 to avoid negative values when applying the logarithm. Given our goal to maximize the sum of α -fairness throughput, these figures demonstrate that the LLMA node outperforms the DLMA node by approximately 17.7%, 29.1%, 14.1%, and 19.1%, respectively. These results indicate that the LLMA protocol, designed through the CP-AgentNet, can effectively coexist in more complex heterogeneous environments.

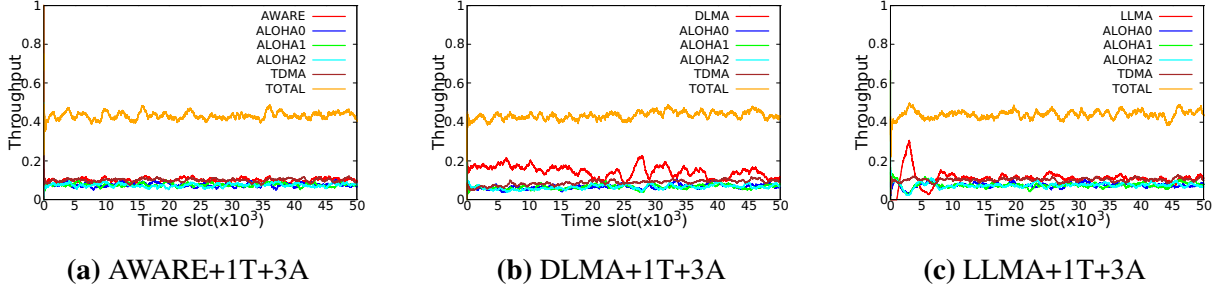


Figure 3.5. Heteronode with 3 ALOHA and 1 TDMA nodes.

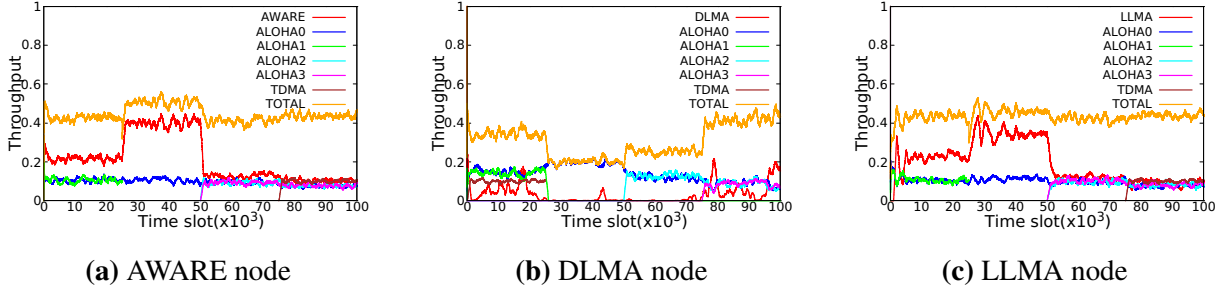


Figure 3.6. Heteronode in a dynamic environment.

Dynamic scenario

We evaluate the adaptability of the LLMA node in a dynamic setting to assess its responsiveness to environmental changes. This scenario starts with two slotted ALOHA nodes and one heteronode (2A+1H). At frame 2500, one ALOHA node exits the network (1A+1H), followed by the entry of two new ALOHA nodes at frame 5000 (3A+1H). The scenario progresses with the addition of one TDMA node at frame 7500 (1T+3A+1H), concluding at frame 10000. In this dynamic environment, as depicted in Fig. ??, the operation of the LLMA node closely mirrors the ideal behavior. The RMSE value for the LLMA node is 0.044, which is approximately 5 times lower than 0.227 with the DLMA node, demonstrating that the LLMA, designed using the CP-AgentNet framework, effectively adapts to environmental changes. This significant difference arises because LLMA does not depend on the model architecture for different numbers of nodes. Unlike traditional DRL methods, the LLMA does not rely on a specific model architecture, thereby enhancing its scalability.

Table 3.1. (N)RMSE of all scenarios. T, A, and H represent TDMA, slotted ALOHA, and heteronode, respectively.

Combination		RMSE		NRMSE	
		DLMA	LLMA	DLMA	LLMA
Single-heteronode	1T+1H	0.1409	0.1908	0.1761	0.2385
	1A+1H	0.1175	0.0768	0.3422	0.2028
	2A+1H	0.1192	0.0491	0.5573	0.2298
	3A+1H	0.0931	0.0468	0.7323	0.368
	4A+1H	0.1044	0.0433	0.6884	0.2647
	1T+1A+1H	0.0909	0.0681	0.2822	0.2117
	1T+3A+1H	0.0544	0.0403	0.5363	0.3977
Multi-heteronode	2A+2H	0.0537	0.0272	0.4517	0.2287
	1T+2A+3H	0.019	0.0177	0.2925	0.2728
Massive	20A+1H	0.0522	0.044	0.6809	0.5873
Dynamic	Dynamic	0.227	0.0476	1.0706	0.2203

Ablation Study

Impact of CP-Agent. Although we have demonstrated the superior performance of the LLMA, it could be argued that this success stems from the pretrained knowledge of LLM. To demonstrate that the exceptional performance is specifically attributable to our CP-AgentNet, we conduct targeted assessments. Initially, we simulated the system using only the LLM under the same conditions as those used in CP-AgentNet. Subsequently, we evaluated the impact of each agent within the framework. Given the central role of the node agent in decision-making, we could not omit it from our simulations; therefore, we first ran simulations with the node agent paired with the strategy agent, and then with the observer agent, to isolate and assess their individual contributions. As shown in Table 3.2, in static tests, the RMSE values for both the LLMA node and the ALOHA node are similar between “1” and “3” configurations, as well as between “2” and “4”. This suggests that the strategy agent significantly influences the performance of both the LLMA and ALOHA. Conversely, the RMSE value for the TDMA node is notably affected

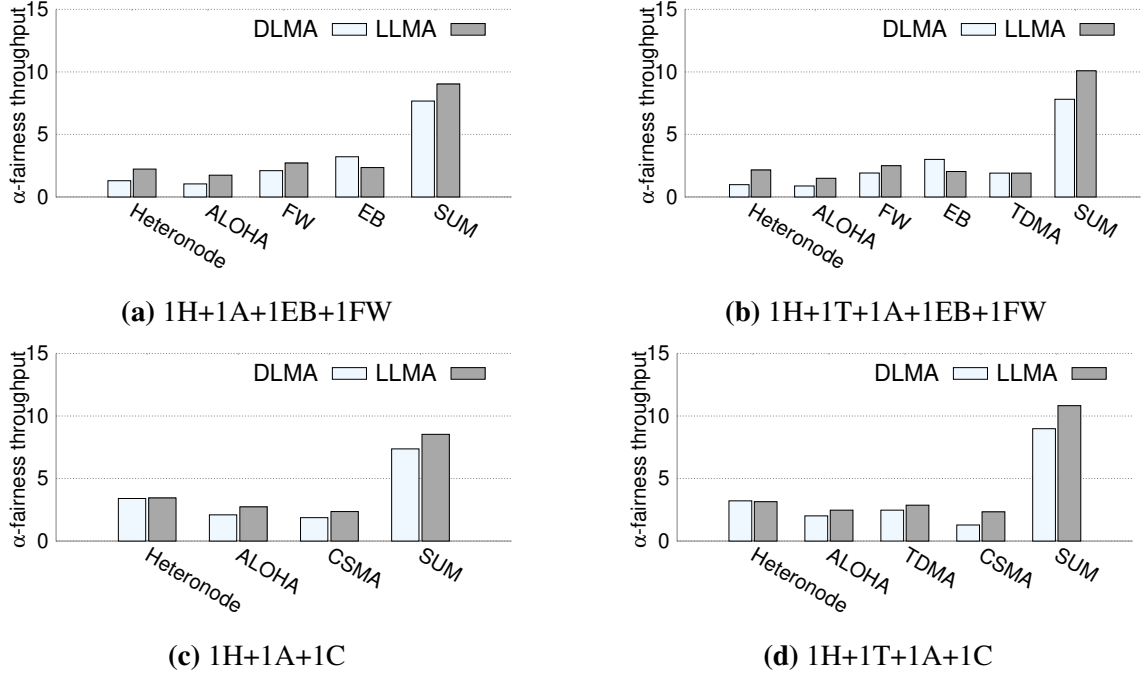


Figure 3.7. α -fairness throughput in a complex heterogeneous environment. C, EB, and FW represent CSMA node, EB-ALOHA node, and FW-ALOHA node, respectively.

by the observer agent, because the node agent avoids the time slots used by the TDMA node with the assistance of the observer agent. In dynamic scenarios, the observer agent impacts not only the behavior of the TDMA node but also that of the LLMA and ALOHA nodes, as it detects environmental changes and assists the node agent in quickly adapting to new conditions. These experiments have demonstrated the impact of each agent, with the superior performance originating from the CP-AgentNet framework.

Effectiveness of LLM ranker. We evaluate the effectiveness of the LLM ranker. Specifically, during the offline stage, the strategy agent utilizes the LLM ranker to formulate transmission strategies. We conduct five simulations for each strategy, both with and without the LLM ranker, comparing the resultant RMSE values and their variances to assess which approach more closely approximates ideal behavior. In the online stage, we implement a transmission strategy that was formulated using the LLM ranker. Additionally, we explore scenarios where the node agent either uses or does not use the ranker to determine transmission probabilities. As shown in Table 3.3, both the average RMSE and standard deviation are lower when the strategy agent employs

Table 3.2. The impact of each agent. 1: No Agent, 2: Strategy + Node, 3: Observer + Node, 4: Strategy + Observer + Node

	Static			Dynamic		
	LLMA	ALOHA	TDMA	LLMA	ALOHA	TDMA
1	0.138	0.036	0.058	0.154	0.04	0.064
2	0.04	0.015	0.031	0.089	0.022	0.033
3	0.155	0.039	0.011	0.153	0.041	0.015
4	0.04	0.015	0.013	0.048	0.016	0.01

Table 3.3. The impact of using an LLM ranker.

Phase	Without Ranker		With Ranker	
	RMSE	STD	RMSE	STD
Offline learning	0.0636	0.036	0.0404	0.008
Online execution	0.0362	0.007	0.0346	0.007

the LLM ranker, indicating enhanced stability and consistency. However, the impact of the LLM ranker is less critical for the node agent during the online stage. This reduced impact is because the node agent’s decisions largely depend on the predefined strategy, which is relatively simpler than strategy formulation.

Explainability

There is no established consensus on the definition of explainability in machine learning, nor on how it should be measured [103]. Explainable AI (XAI) focuses on demystifying the decision-making processes of ML models—understanding how and why decisions are made [105]. One of the most straightforward approaches to achieving explainability is to employ decision trees [103]. In our framework, multi-agent interaction facilitates the generation of a decision tree, rendering the decision-making process transparent and interpretable. Fig. 3.8a is generated by the LLM to illustrate how the action is devised. The observer agent monitors the environment and informs the node agent that slot 3 and 5 are fully utilized. Consequently, the node agent avoids these slots and determines the action for each slot. This process is clearly

depicted in the decision tree, enabling users to easily understand how the decisions are made. Moreover, the LLM facilitates the *ability to question*, enhancing comprehension of the decisions for laypeople and enabling them to explore ‘what-if’ scenarios— a capability not available with traditional DRL methods. In our scenario, the LLM can respond to queries like ‘What if the utilization rate of slot 6 is 1?’ without the need for additional training. The response is illustrated in Fig. 3.8b, demonstrating the capability to facilitate relevant questioning. In addition, this can help assess whether the LLM’s explanations are consistent with its actual decision-making. Observing responses to controlled hypothetical inputs allows users to gauge how well the model’s reasoning aligns with expected behavior.

Computational Overhead

We evaluate the computational overhead of LLMA nodes and find that LLMA requires 19.1 seconds to process decisions for 1,000 frames (10 seconds of simulation time). While not yet ready for real-time decision-making, this result demonstrates strong potential for near-future deployment, particularly with the continued advancement of lightweight personal LLMs. To illustrate this potential, we compare the computation time of smaller models such as GPT-4o-mini and GPT-4.1-nano. Note that while the offline phase is conducted using GPT-4o, the online phase can be replaced with smaller models. As shown in Table 3.4, GPT-4o-mini and GPT-4.1-nano require 11.3 and 10.1 seconds, respectively, while achieving comparable performance. Although CP-AgentNet relies on high-end LLMs during the offline phase for strategy generation, these results suggest that the online execution phase can operate efficiently with smaller models. This separation between strategy generation and execution significantly reduces the computational burden on individual agents, enabling efficient operation even in resource-constrained environments.

Table 3.4. Comparison of computation time and RMSE performance across smaller LLM variants

	Combination	LLMA (4o)	LLMA (4o-mini)	LLMA (4.1-nano)
RMSE	1T+1A+1H	0.0681	0.0592	0.0720
	1T+3A+1H	0.0403	0.0508	0.0521
	Dynamic	0.0476	0.0492	0.0517
Runtime (s)	—	19.1	11.3	10.1

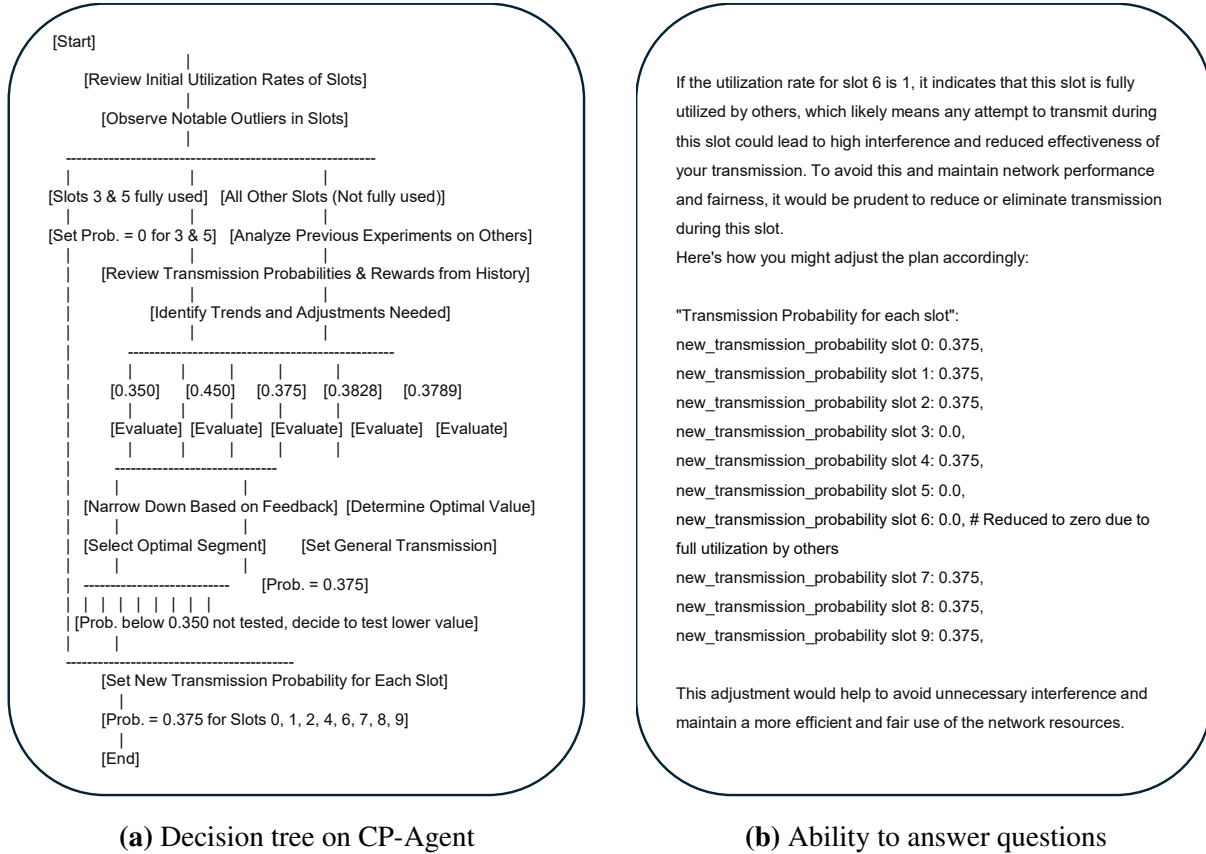


Figure 3.8. Explainable CP-AgentNet

3.4.3 CPTCP results

Homogeneous scenario

We evaluate multiple CPTCP flows in configurations of 2, 3, and 4 in a homogeneous environment, comparing the total throughput results with TCP CUBIC [46] and TCP BBR [23]. As shown in Table 3.5, although CPTCP does not overwhelmingly outperform CUBIC and

Table 3.5. Total throughput (Kbps) with homogeneous TCP

number of flows	CUBIC	BBR	CPTCP	Pure LLM
2	758.4	793.6	813.6	543.2
3	858.3	942.4	868.4	679.1
4	883.0	932.7	890.1	584.0

Table 3.6. Homogeneous TCP under different RTTs.

RTT (ms)	Algorithm	Throughput (Kbps)	Jain’s fairness Index
100	CPTCP	890.1	0.999
	BBR	932.7	0.998
200	CPTCP	652.4	0.996
	BBR	724.4	0.997
Mixed	CPTCP	732.8	0.970
	BBR	746.0	0.884

BBR, these results demonstrate that comparable TCP performance can be achieved through CP-AgentNet. Importantly, when the LLM is used without CP-Agent, performance consistently ranks as the lowest, regardless of the number of flows. These results indicate that the performance of the CPTCP does not stem solely from the pretrained LLM.

We further evaluate CPTCP against BBR, a congestion control algorithm that estimates bottleneck bandwidth and minimum RTT to maximize throughput while minimizing latency, to evaluate performance under varying RTTs as shown in Table 3.6. While BBR achieves slightly higher throughput, CPTCP demonstrates superior fairness, particularly in mixed RTT environments. This improvement in fairness stems from the design of the reward function, which explicitly incorporates both RTT and packet loss during training to guide the decision-making process.

Heterogeneous scenario

To demonstrate that CPTCP can coexist with different types of TCP algorithms, we assess its performance in a heterogeneous TCP environment. As shown in Table 3.7, when CUBIC and

Table 3.7. Throughput and fairness with heterogeneous TCP

TCP algorithm		Throughput (Kbps)		Jain’s fairness index
flow1	flow2	flow1	flow2	
CUBIC	CUBIC	391.2	367.2	0.999
BBR	BBR	398.8	394.8	1.000
CPTCP	CPTCP	398.6	415.0	1.000
CUBIC	BBR	575.2	210.5	0.823
CUBIC	Cx-TCP	441.2	338.8	0.976
BBR	Cx-TCP	362.2	387.7	0.999
CUBIC	CPTCP	372.1	417.7	0.997
BBR	CPTCP	438.7	341.3	0.984

BBR coexist, a fairness issue arises, resulting in a low Jain’s fairness index of 0.823. However, when CPTCP coexists with either CUBIC or BBR, it does not interfere with the other TCP algorithm, achieving Jain’s fairness indices of 0.997 and 0.984. These results are comparable to those achieved by Cx-TCP [21], indicating that CPTCP can effectively coexist with both types of TCP algorithms. Furthermore, its applicability extends beyond TCP coexistence scenarios.

3.5 Discussion

Scalability—While CP-AgentNet has not yet been evaluated in large-scale real-world deployments, the framework is designed with scalability in mind. Its modular, agent-based architecture supports distributed operation and can be extended to larger and more complex network scenarios. However, the current 1-second query period may not meet the latency requirements of real-time systems, highlighting the need to optimize decision latency to support time-sensitive applications. Despite this limitation, CP-AgentNet offers a flexible and extensible foundation for autonomously designing communication protocols in heterogeneous networks.

3.6 Chapter-Specific Related Work

3.6.1 MAC protocols through RL

A MAC protocol utilizing QMIX [131], a renowned multi-agent reinforcement learning algorithm, has been developed to coexist with legacy systems like CSMA/CA [45]. This protocol exhibits adaptability in dynamic settings, but requires reinitialization with different weight parameters as node numbers change. In contrast, the DRL-based MAC protocols [180, 183] are designed to operate in heterogeneous networks without requiring explicit knowledge of the number of nodes or their parameters. However, their performance remains highly dependent on the underlying neural network architecture, which is often optimized for a fixed node count—limiting their robustness and adaptability in dynamic environments.

3.6.2 LLM-empowered Agents

Generative Agents [118] elicit emerging behaviors and collaboration within assemblies of agents as societal groups. Autonomous software development has improved through the collaborative use of multiple LLM agents [27, 34, 53, 59, 76, 126, 168]. These systems utilize various LLM agents, assigning specific tasks to each to reduce complexity and facilitate collaboration in accomplishing these tasks. Notably, MetaGPT [53] and AgentVerse [27] simulate a human-like software development process, enabling efficient role-playing for each agent. Additionally, improvements in mathematical accuracy using LLM agents have been explored [36, 88]. While these LLM-agents are typically built on top of LLMs accessed via the internet, personal LLM-agents [77] can also be deployed on-device using LLM [175]. CP-AgentNet operates under the assumption that personal LLM-agents can be utilized, although in practice, we employ GPT-4o via the internet.

3.6.3 LLMs with Reinforcement Learning

Recent research has explored various ways of integrating reinforcement learning with LLMs [19,56,58,73,121,142,148,151]. When2Ask [56] method involves using RL to control the frequency of queries to the LLM, optimizing for cost-effective interaction between the agent and the LLM. LLARP [151] employs a pre-trained, frozen LLM as the core neural network within a DRL framework. This setup enhances the agent’s generalization capabilities for embodied tasks through the use of LLM. ICPI [19] iteratively updates the contents of the prompt, which serves as the foundation for its policy, through trial-and-error interactions within an RL environment. This approach does not employ gradient-based methods; instead, the focal point of learning is the prompt content itself.

3.6.4 LLMs for communication network

Following the trend of leveraging LLMs for specialized disciplines, there is a growing movement to adopt LLMs in the field of communication networks [11,22,24,52,66,93,100,122]. The work in [11] fine-tunes LLMs with telecommunication-specific language to identify working groups within 3GPP. [52,66,122] utilize LLMs as QA assistants to interpret 3GPP standards. Collectively, these approaches primarily focus on classification of working groups or QA tasks, indicating that the use cases of LLMs in the communication domain remain quite limited.

3.7 Summary

This chapter introduced CP-AgentNet, a framework using generative agents to autonomously design communication network protocols, enhancing explainability. Using CP-AgentNet, we designed the LLMA and CPTCP for heterogeneous environments. Our simulations show that nodes using LLMA or CPTCP coexist efficiently with nodes using different protocols. This work sets the stage for broader applications of generative agents in protocol design and solving complex network problems.

Connection to the dissertation.

This chapter demonstrates that generative agents can be used not only as text-generation tools, but also as autonomous reasoning components for wireless protocol design. The next chapters shift from protocol design to Open RAN management, where the same high-level idea—using generative agents as a reasoning and synthesis layer—is applied to xApp generation, xApp conflict management, and proactive misconfiguration prevention.

Chapter 3, in part, is a reprint of the material as it appears in Dae Cheol Kwon and Xinyu Zhang, “CP-AgentNet: Autonomous and Explainable Communication Protocol Design Using Generative Agents,” in *Proceedings of the 2025 IEEE 33rd International Conference on Network Protocols (ICNP)*, 2025. The dissertation author was the primary investigator and first author of this paper.

Chapter 4

Autonomous xApp Synthesis for Open RAN

This chapter addresses the second component of autonomous wireless intelligence studied in this dissertation: automatic generation of Open RAN control applications. Building on the previous chapter’s investigation of generative agents for communication protocol design, this chapter moves from protocol-level decision making to RAN-management software. Specifically, it studies whether generative agents can transform high-level operator requirements into deployable, standards-compliant xApps that execute in a real O-RAN environment.

4.1 Introduction

Open Radio Access Network (O-RAN) has emerged as a transformative architecture for cellular systems by decoupling traditional monolithic RAN stacks into interoperable and programmable components. A key enabler of this flexibility is the RAN Intelligent Controller (RIC), which hosts modular control applications known as xApps. Operating in the near-real-time control loop, xApps continuously ingest telemetry such as key performance metrics (KPMs) and issue control actions over standardized interfaces (e.g., E2) to optimize network behavior. Through xApps, operators can implement a wide range of intelligent functionalities, including anomaly detection, traffic steering, interference mitigation, mobility optimization, and network slicing control. This architectural openness shifts innovation from vendor-locked hardware

upgrades to software-driven intelligence, allowing network operators and third-party developers to rapidly introduce new optimization logic into operational networks. As 5G and beyond networks become increasingly heterogeneous and service-driven, xApps are expected to play a central role in enabling adaptive, data-driven, and service-aware RAN control.

Despite their promise, developing and deploying xApps in practice remains complex and error-prone. Existing research primarily focuses on designing better-performing algorithms—such as more accurate anomaly detectors or more efficient schedulers—while the end-to-end generation and integration of xApps into operational O-RAN systems is often treated as an engineering afterthought. In reality, xApp development requires deep familiarity with O-RAN architecture, service models (e.g., E2SM-KPM, E2SM-RC), ASN.1-based message encoding, subscription procedures, and strict interface compliance. A functional xApp must correctly register with the RIC, subscribe to appropriate telemetry streams, parse standardized reports, satisfy real-time execution constraints, and generate control messages that conform to protocol specifications. Even minor mismatches in message formats, metric granularity (e.g., per-UE vs. cell-level), or control semantics can cause runtime failures. As a result, translating high-level network intents into deployable, specification-compliant xApps remains a significant bottleneck to O-RAN innovation.

While most prior work concentrates on improving the performance of individual xApps, only a limited number of efforts address the engineering complexity of xApp development itself. Platforms such as CoO-RAN [123] provide programmable experimental environments that enable the prototyping and evaluation of machine learning–based xApps in closed-loop RAN control scenarios. These systems lower the barrier for experimentation by integrating RIC components with open-source RAN implementations and facilitating data-driven control evaluation. More recently, xDevSM [39] introduces a flexible framework that abstracts E2 Service Model (E2SM) handling behind developer-friendly APIs, simplifying ASN.1 encoding/decoding and improving interoperability across different RAN stacks. By modularizing service model logic and providing reusable libraries, xDevSM significantly reduces the protocol-level burden

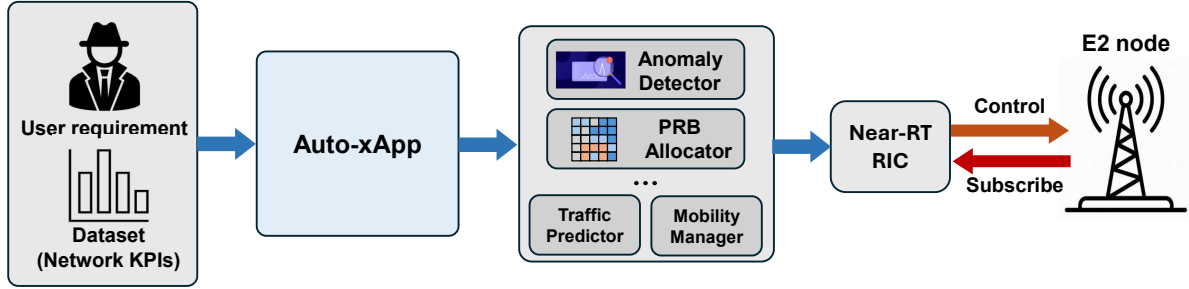


Figure 4.1. Auto-xApp vision: automatic generation of deployable xApps from user requirements and network KPI data.

placed on xApp developers.

However, these approaches primarily streamline implementation mechanics rather than addressing the higher-level challenge of transforming operator intent into deployable, specification-compliant xApps. Developers are still required to manually design control logic, determine compatible service model configurations, and ensure correctness across RIC–RAN interactions. The semantic gap between high-level network objectives (e.g., “optimize resource allocation across network slices to maintain QoS”) and low-level E2 procedures remains largely unaddressed. As O-RAN deployments scale and diversify, this gap becomes increasingly problematic: even small mismatches in metric granularity, action definitions, or service model versions can lead to silent failures or degraded control behavior. Bridging this gap requires not only abstraction of protocol details, but also systematic automation of xApp design, validation, and deployment. Addressing this missing layer is essential to unlock the full programmability promised by O-RAN.

This chapter presents **Auto-xApp**, an end-to-end pipeline that automatically generates deployable xApps from natural language requirements while ensuring their correct integration and operation within O-RAN systems (Figure 4.1). The O-RAN Alliance AI/ML framework [107] identifies supervised learning, unsupervised learning, and reinforcement learning as the principal ML paradigms for xApp intelligence, while conventional xApps rely on rule-based or heuristic control logic [38, 81]. Auto-xApp targets three paradigms that span this spectrum: (i) rule-based control logic that maps observed metrics to deterministic actions, (ii) supervised and

unsupervised learning models trained on offline datasets for tasks such as anomaly detection and classification, and (iii) deep reinforcement learning (DRL) agents that optimize long-term objectives through closed-loop interaction with the RAN. These paradigms encompass a broad range of real-world near-RT RIC applications, including threshold-based traffic optimizers, data-driven anomaly detectors, and adaptive resource allocators. Supporting this range is critical for practical automation, as different operational scenarios require distinct control strategies depending on data availability, optimization objectives, and latency constraints. Automating xApp generation across these paradigms is fundamentally challenging, as it requires bridging a multi-layer gap between high-level operator intent and executable, standards-compliant xApps. Auto-xApp addresses this by decomposing the automation process into coordinated modules, each targeting a specific barrier.

First, user requirements are often vague, incomplete, and expressed using non-standard terminology, leaving critical design elements unspecified—such as which KPMs to monitor, which control parameters to adjust, or how conflicting objectives should be prioritized. To address this, the *Intent Translator* employs schema-guided iteration: a predefined workflow schema specifies the required fields and structural constraints for xApp construction, and the system iteratively parses user input, detects missing or ambiguous elements, and generates targeted clarification questions until a complete specification is produced.

Second, even when requirements are fully specified, user-provided terms and dataset feature names frequently mismatch O-RAN specifications. Informal expressions such as “signal strength” or “throughput” do not directly correspond to E2SM-KPM identifiers, and user-specified value ranges may use different units or encodings than those defined in 3GPP standards. The *Context-Aware Mapper* resolves these mismatches by employing Retrieval-Augmented Generation (RAG) over an O-RAN knowledge base to translate informal terms and dataset features into specification-compliant identifiers, and to align value representations with standardized encodings. This component serves both the *Intent Translator* and the *Dataset Analyzer*, which inspects available datasets to determine the applicable learning paradigm and extract structural

schema needed for model configuration.

Third, not all monitoring metrics and control parameters are exposed by a given RAN implementation through the E2 interface, rendering some objectives infeasible despite being logically valid and semantically aligned. The *Capability Checker* validates the mapped specification against the actual platform capabilities discovered from E2 SETUP REQUEST messages, rejecting configurations that reference unsupported metrics or control actions before they reach code generation.

Finally, translating a validated specification into deployable code requires correctly composing heterogeneous system knowledge—spanning E2 subscription procedures, control message construction, RIC SDK integration, and paradigm-specific logic for rule-based, supervised/unsupervised learning, or DRL approaches. The *xApp Generator* synthesizes deployment-ready xApp code by combining LLM-driven design with parameterized code templates, producing implementations that integrate with the target RIC platform and conform to the validated specification. These components are supported by a shared *Knowledge Base* comprising O-RAN specification entries indexed for RAG retrieval, a platform capability database, and the workflow schema that drives the iterative elicitation process.

To evaluate Auto-xApp, this chapter instantiates three representative xApps corresponding to the three supported design paradigms: a rule-based control xApp, a learning-based inference xApp, and a DRL-based optimization xApp. These use cases are deployed on a real O-RAN testbed integrating OSC near-RT RIC with srsRAN, validating that automatically generated xApps can be successfully synthesized, compiled, registered, and executed through the E2 interface without manual intervention.

As a first use case, we generate a rule-based PRB allocation xApp that guarantees a minimum throughput for a GBR user in a two-UE scenario (one GBR, one best-effort). When the GBR user’s channel quality degrades and its throughput drops below the 20 Mbps target, the generated xApp dynamically adjusts the PRB quota to reallocate resources from the best-effort user. The results show that the throughput constraint is restored while maintaining stable system

behavior, demonstrating that Auto-xApp can correctly translate operator intent into functional control logic and enforce it through standards-compliant RIC–RAN interactions.

As a second use case, we generate a learning-based anomaly detection xApp that maps available telemetry to model inputs, constructs the training and inference pipeline, and deploys the resulting inference logic through the RIC environment. This use case demonstrates that Auto-xApp can support data-driven xApps beyond static rule-based control. As a third use case, we generate a DRL-based PRB slicing xApp that converts high-level SLA requirements into a reward function, selects a DRL formulation, trains the agent from testbed-collected data, and deploys the learned policy for closed-loop resource allocation. Together, these use cases show that Auto-xApp generalizes across rule-based, learning-based, and DRL-based xApp paradigms rather than being tailored to a single control strategy.

This chapter makes the following contributions:

- We design and implement Auto-xApp, an end-to-end automation pipeline that bridges this gap through schema-guided intent translation, context-aware semantic mapping, capability validation, and automated xApp code synthesis. The framework supports rule-based, learning-based, and DRL-based xApp paradigms while preserving specification compliance.
- We implement Auto-xApp on a practical O-RAN testbed integrating OSC near-RT RIC and srsRAN, extending the RAN control interface where necessary to validate feasibility. The generated xApps successfully perform E2 subscription, control message construction, and closed-loop execution without manual intervention.
- We evaluate Auto-xApp using rule-based, learning-based, and DRL-based xApps for resource control and performance optimization scenarios, demonstrating correct deployment, runtime stability, and effective control behavior. Our results show that natural-language-driven automation can produce functional and standards-compliant xApps in real systems.

4.2 Background

4.2.1 O-RAN Architecture and E2-Based Control

O-RAN disaggregates the traditional base station into modular components—the O-CU, O-DU, and O-RU—and introduces the RAN Intelligent Controller (RIC) as a programmable control entity. The near-real-time (near-RT) RIC hosts third-party applications called xApps that monitor network conditions and issue control decisions to RAN nodes in a closed-loop manner, operating at sub-second time scales (10–1000 ms). Communication between the near-RT RIC and RAN nodes occurs through the standardized E2 interface, which serves as the primary interface for runtime telemetry and actuation.

The E2 interface supports service models (E2SMs) that define both monitoring and control semantics. E2SM-KPM specifies key performance measurements such as throughput (`DRB.UETpd1`), PRB utilization (`RRU.PrbUsedD1`), and radio quality indicators (RSRP, CQI). E2SM-RC defines control primitives that allow modification of specific RAN parameters, such as slice-level PRB quota allocation. An xApp subscribes to selected metrics through an E2 subscription request, receives periodic indication messages from the E2 node containing the requested measurements, and issues control requests when its decision logic determines an action is needed. All interactions must conform to ASN.1-based message encodings defined in the service model specifications.

Upon connection establishment, the E2 node advertises its supported capabilities through a RAN Function Definition carried in the E2 SETUP REQUEST message. This definition enumerates the available metrics for E2SM-KPM and the supported control styles and actions for E2SM-RC, enabling the RIC to determine what monitoring and control operations are feasible on a given RAN platform. This architectural boundary is fundamental: an xApp can only monitor metrics and modify parameters that are explicitly exposed through the implemented E2 service models. Any control objective that references unsupported metrics or actions is infeasible regardless of its algorithmic validity.

Although O-RAN includes additional interfaces—A1 for policy guidance from the non-RT RIC and O1 for management-plane configuration via the SMO—neither provides a direct near-real-time actuation path from xApps to RAN nodes. Consequently, the effective boundary for runtime xApp operation is defined by the E2 interface and the service models implemented by the E2 node.

4.2.2 xApp Development Workflow

From a developer’s perspective, an xApp is not merely a control algorithm but a fully integrated RIC application that must interact correctly with standardized interfaces and runtime services. Beyond implementing decision logic, a developer must register the xApp with the near-RT RIC, construct E2 subscription requests using correct service model identifiers and report styles, parse periodic indication messages by decoding ASN.1 structures into usable data, and generate properly formatted control messages that conform to the semantics of the target E2SM. Even minor inconsistencies in metric identifiers, unit encodings, or action definitions can result in silent failures or rejected control requests.

The development process therefore begins with determining which monitoring metrics and control parameters are relevant to the target objective and confirming that they are supported by the underlying RAN implementation. This requires inspecting E2 node capabilities, understanding service model specifications, and mapping high-level objectives to specification-compliant identifiers—a process that demands significant domain expertise in both O-RAN standards and the target platform.

For learning-based xApps, the complexity increases further. Developers must define input feature mappings from monitored metrics to model inputs, design training pipelines, manage model loading and inference execution within near-real-time constraints, and ensure that model outputs are translated into valid control parameters. These specification-driven integration tasks are orthogonal to the control algorithm itself, yet they constitute a significant portion of the engineering effort required to deploy an operational xApp.

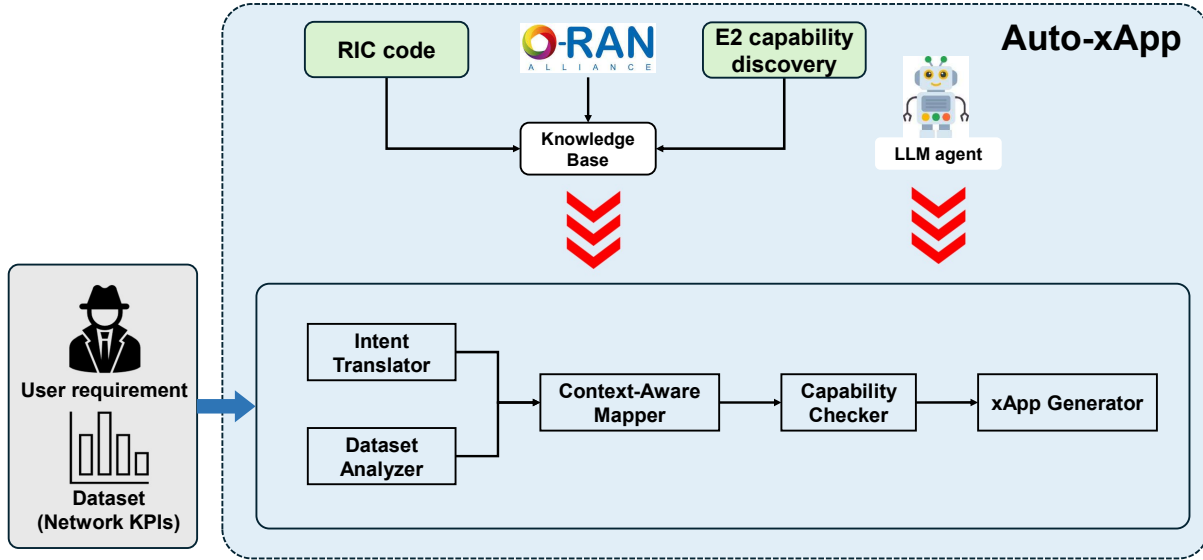


Figure 4.2. Auto-xApp framework.

This layered complexity—spanning requirement interpretation, specification alignment, capability verification, and deployment integration—motivates the need for systematic automation that bridges the gap between high-level operator intent and executable, standards-compliant xApps.

4.3 Auto-xApp Design

4.3.1 System Overview

Auto-xApp is an automated framework that translates high-level network requirements into deployable O-RAN xApps. Given a natural language description of network objectives and, when learning-based control strategies are selected, a corresponding training dataset, Auto-xApp systematically generates executable xApp implementations that comply with O-RAN E2 interfaces and operate within the near-RT RIC environment. As illustrated in Figure 4.2, Auto-xApp addresses the challenges of ambiguous requirements, semantic mismatches with O-RAN specifications, and platform capability constraints through a structured multi-stage pipeline that progressively refines intent, validates feasibility, and synthesizes deployable implementations. Throughout the pipeline, LLM agents augmented with a domain-specific knowledge base are

employed for tasks requiring semantic understanding—such as intent parsing, terminology disambiguation, DRL system design, and code generation—while deterministic logic handles schema validation and capability checking. Through retrieval-augmented generation (RAG), the knowledge base grounds LLM outputs in O-RAN specifications and platform-specific constraints rather than relying solely on pretrained knowledge.

The workflow begins with two parallel input paths. The *Intent Translator* (§4.3.2) converts natural language requirements into structured intent representations through schema-guided iterative elicitation. An LLM agent extracts structured fields from the user’s input, a gap detector identifies missing information against a predefined schema, and clarification questions are generated and posed iteratively until all required fields are fully specified. In parallel, when a training dataset is provided, the *Dataset Analyzer* (§4.3.3) inspects its structure to extract a dataset profile including column names, data types, and value ranges, and classifies column roles into state features, actions, and rewards.

The *Context-Aware Mapper* (§4.3.4) then aligns user-defined terms and dataset feature names with standardized O-RAN terminology. Operating as a shared component invoked by both the Intent Translator and the Dataset Analyzer, it employs RAG over an O-RAN knowledge base to retrieve candidate specification entities, uses an LLM agent with intent context to disambiguate among candidates, and resolves unit and encoding discrepancies between user-facing values and specification-level reported values. The mapped configuration is subsequently validated by the *Capability Checker* (§4.3.5), which verifies whether the required monitoring metrics and control actions are supported by the target RAN platform. The checker loads a capability database populated by parsing the RAN Function Definitions from E2 SETUP REQUEST messages, and rejects configurations that reference unsupported control actions before they reach code generation. Finally, the *xApp Generator* (§4.3.6) synthesizes executable xApp code from the validated specification, producing deployment-ready implementations that integrate with the near-RT RIC.

All pipeline components are supported by three knowledge artifacts: an O-RAN specifica-

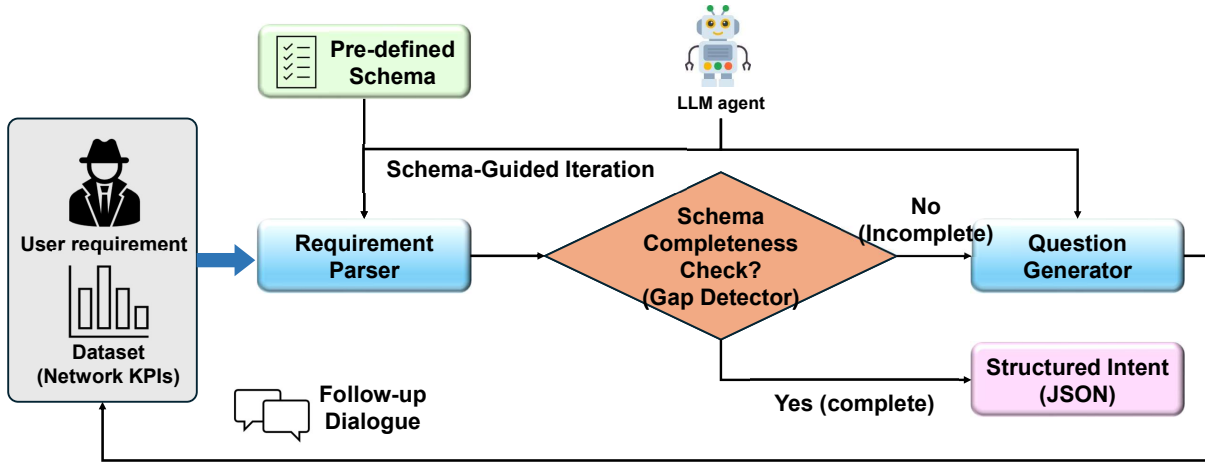


Figure 4.3. Intent Translator with schema-guided iteration.

tion knowledge base indexed in a vector store, used by the Context-Aware Mapper for RAG-based terminology alignment; a platform capability database discovered from E2 SETUP REQUEST messages, used by the Capability Checker for feasibility validation; and a RIC SDK code base containing reference xApp implementations, used by the xApp Generator to guide code generation.

4.3.2 Intent Translation via Schema-Guided Iteration

A fundamental challenge in automating xApp development is bridging the gap between high-level operational requirements and the precise specifications needed for code generation. Users may describe objectives using informal natural language—for example, “maximize throughput by allocating PRBs (Physical Resource Blocks)” or “prioritize video streaming users”—that are often incomplete, ambiguous, or lacking implementation-level details such as target metrics, control parameters, trigger conditions, and deployment constraints. Directly converting such inputs into executable xApp logic leads to inconsistent or infeasible designs.

To address this challenge, Auto-xApp introduces an *Intent Translator* that converts natural language requirements into structured intent specifications through a process we term *schema-guided iterative elicitation*. Rather than performing a single-pass translation, the Intent Translator employs an LLM agent within a multi-turn dialogue loop, progressively refining

the user’s intent against a predefined *xApp workflow schema*. This schema explicitly defines the structured fields required for xApp construction, organized into sections including control objective (goal, target metric, optimization direction), monitoring configuration, control action (action type and parameters), trigger conditions, target scope, and metadata.

The Requirement Parser leverages an LLM agent to extract structured information from the user’s natural language description. The LLM is provided with the complete workflow schema, which defines all fields necessary for xApp construction. It maps relevant elements from the user’s input to the corresponding schema fields. LLMs are particularly effective for this task, as schema guided information extraction from unstructured text is one of their core strengths [152, 176].

The extracted fields are subsequently validated by a Gap Detector, which performs a deterministic comparison against the schema’s required and conditionally required fields. It identifies any missing mandatory information, such as monitored metrics, selected DRL algorithm, or associated hyperparameters, and evaluates conditional dependencies. For example, a threshold value becomes mandatory when the trigger type is threshold based. If a training dataset is provided, flagged fields may either be resolved through user clarification or automatically inferred from the dataset profile.

For any remaining missing fields, a Question Generator, powered by the LLM agent, produces targeted clarification prompts. For instance, it may request the user to select a DRL algorithm such as PPO, DDPG, or SAC, specify hyperparameters including learning rate or discount factor, define which metrics should be monitored, or choose automatic configuration based on the dataset profile. A Conversation Manager coordinates this interactive refinement loop by maintaining dialogue state across turns and routing user responses back to the parser with context from previous extractions.

In subsequent turns, the parser updates only the specific field under discussion while preserving previously extracted values through a structured merge operation. This iterative process continues until the Gap Detector verifies that all required schema fields have been fully

specified. When the user explicitly defers certain decisions, default values defined in the schema are applied to complete the specification.

The output of the Intent Translator is a complete, machine interpretable intent specification in the form of a structured JSON workflow populated with all schema defined fields. This specification serves as the foundation for the subsequent stages of Auto-xApp. By enforcing schema completeness through iterative, LLM driven elicitation before downstream processing, this step prevents ambiguity from propagating into dataset analysis, specification mapping, and capability validation. As a result, it establishes a reliable transition from human readable requirements to executable system level design inputs.

4.3.3 Dataset Analysis

The O-RAN AI/ML workflow requires that ML models undergo validation prior to deployment [107]. Consequently, even for paradigms that support online learning, a training dataset is required for pre-training and validation before the xApp can be deployed on the near-RT RIC. Auto-xApp assumes that the user provides this dataset, whether collected from RAN deployments or publicly available sources. However, a raw dataset alone is insufficient for automated code generation: the xApp Generator must know the role of each column (state feature, action, or reward), the dimensionality of the observation and action spaces, and the value ranges of controllable parameters in order to produce a correctly configured training pipeline and inference loop.

To extract this information, Auto-xApp incorporates a *Dataset Analyzer* that inspects the provided data and produces a dataset profile summarizing its format, dimensionality, and column roles. The intended paradigm is specified by the user during the intent elicitation stage. For DRL datasets, the analyzer identifies the action column and reward column by examining column names against domain-specific keyword sets (e.g., columns containing action or reward), and classifies the remaining columns as state features, excluding transition indicators (next_* prefixed columns) and episode termination flags. It also computes the action value range to

inform action space configuration during code generation, and checks whether the dataset contains transition tuples and episode boundaries needed for experience-replay-based training. For supervised/unsupervised learning datasets, all numeric columns are treated as input features, with common label columns (e.g., `label`, `anomaly`) excluded from the feature set. For DRL datasets where column roles are not identifiable through naming conventions, users can specify them explicitly during the intent elicitation stage.

Limitations. The Dataset Analyzer is designed for structured tabular datasets and relies on semantically meaningful column names to infer roles. Although column names are not required to follow strict specification compliant terminology, they are expected to reflect common semantic conventions (e.g., indicating actions, rewards, or labels through intuitive naming). Datasets with opaque or arbitrary column names, or with roles encoded implicitly rather than explicitly represented in the table, may require manual role specification during the intent elicitation stage.

4.3.4 Context-Aware Mapping via RAG

Following intent elicitation and dataset analysis, Auto-xApp must reconcile differences between three independently defined domains: user-specified objectives expressed in informal terminology, dataset features with implementation-specific column names, and standardized O-RAN metrics and control parameters defined in service model specifications such as E2SM-KPM and E2SM-RC. These domains frequently employ inconsistent naming conventions and abstraction levels. For instance, a user may refer to “throughput” while the dataset contains a column named `DRB.UEThpD1`, which the O-RAN specification formally defines as the per-UE downlink data radio bearer throughput measured in kbps. Similarly, the same high-level term can resolve to different specification entities depending on context: “throughput” maps to `DRB.UEThpD1` in a downlink video streaming scenario but to `DRB.UEThpU1` in an uplink context. Static keyword matching cannot resolve such context-dependent ambiguities.

To address this, Auto-xApp introduces a *Context-Aware Mapper* as a shared component invoked by both the Intent Translator and the Dataset Analyzer. Given inputs from either

source—vague terms extracted from user requirements or column names from dataset analysis—the mapper produces precise O-RAN specification-compliant identifiers along with associated metadata such as service model, measurement unit, granularity, and applicable value ranges.

The mapping process employs RAG and proceeds in three phases. In the first phase (*semantic retrieval*), the mapper queries a vector database to retrieve candidate O-RAN entities that are semantically related to the input term. The vector database is built from a structured O-RAN knowledge base that encodes E2SM-KPM metrics, E2SM-RC control parameters, scheduling algorithms, RAN platform capabilities, and explicit terminology mappings between common user expressions and specification-defined identifiers. Each knowledge entry is converted into a text document that includes the entity name, description, associated context keywords, and applicable use cases. These documents are embedded and indexed in a ChromaDB vector store, enabling semantic similarity search rather than exact keyword matching—for example, a query for “data rate” can retrieve DRB.UETHPD1 even though the term does not appear in the metric name.

In the second phase (*contextual disambiguation*), the mapper invokes an LLM agent to select the most appropriate candidate from the retrieved set. The LLM is provided with the full user intent context—including the optimization goal, target direction (e.g., downlink vs. uplink), use case (e.g., video streaming), and any previously resolved mappings—and is tasked with selecting the candidate that best fits the operational scenario. For example, given the retrieved candidates DRB.UETHPD1 and DRB.UETHPU1 for the query “throughput,” the LLM disambiguates to DRB.UETHPD1 when the context indicates a downlink video streaming optimization. This phase also determines the applicable service model (e.g., E2SM-KPM for monitoring, E2SM-RC for control actions) and outputs a confidence score reflecting the mapping quality.

In the third phase (*value alignment*), the mapper resolves unit and encoding discrepancies between user-facing representations and specification-level reported values. O-RAN metrics are reported using encoding schemes defined in 3GPP specifications that often differ from the units operators use in practice. For instance, RSRP is reported as an integer index in the

range 0–97 per 3GPP TS 36.133, where the conversion to physical units follows $\text{RSRP}_{\text{dBm}} = \text{index} - 140$; thus a user-specified threshold of -60 dBm corresponds to a reported index of 80. Similarly, `DRB.UEThpD1` is reported in kbps, requiring conversion when users express throughput targets in Mbps. The mapper maintains a metric metadata registry that stores the user-facing unit, the reported unit, the valid value range, and bidirectional conversion functions for each metric, enabling automatic translation of user-specified thresholds and dataset values into specification-compliant representations.

The output of the Context-Aware Mapper is a standardized specification that includes the resolved O-RAN metric or control parameter identifiers, the associated service model, and any converted threshold values. When invoked from the Intent Translator, this output replaces vague terms in the structured workflow with precise, specification-compliant references (e.g., `monitored_metrics: ["throughput"]` becomes `monitored_metrics: ["DRB.UEThpD1"]`). When invoked from the Dataset Analyzer, the mapper identifies which dataset columns correspond to specification-defined observables, establishing the link between available training data and the monitoring or control entities required for xApp operation. By centralizing terminology resolution, value alignment, and context-aware disambiguation in a single shared component, the Context-Aware Mapper prevents inconsistent or invalid mappings from propagating into downstream capability validation and code generation stages.

4.3.5 Capability Validation

Even after aligning user intent and dataset semantics with standardized O-RAN terminology, an xApp design may remain infeasible in practice due to limitations of the underlying RAN implementation. In O-RAN systems, monitoring metrics and controllable parameters are exposed to the near-RT RIC only through supported E2 service models, and the specific set of available metrics and actions varies across RAN vendors and software versions. Consequently, not all logically valid optimization objectives can be realized as executable xApps on a given deployment.

To ensure deployability, Auto-xApp incorporates a *Capability Checker* that validates whether the mapped monitoring metrics and control parameters are actually supported by the target RAN environment. This validation relies on a *capability database* that records the exact metrics and actions exposed by the E2 node, obtained through two complementary discovery methods.

For monitoring capabilities, the available E2SM-KPM metrics are extracted by parsing the *RAN Function Definition* carried in the E2 SETUP REQUEST message, which the E2 node transmits to the near-RT RIC upon connection establishment [109]. Per the O-RAN E2AP specification, this message contains a RANfunction-Item for each supported service model, encapsulating the RAN Function OID and a definition structure that enumerates all advertised measurement types. For our srsRAN-based testbed, this discovery yields 19 E2SM-KPM metrics spanning radio quality indicators (e.g., CQI, RSRP), resource utilization counters (e.g., RRU.PrbUsedD1), and throughput measurements (e.g., DRB.UEThpD1).

For control capabilities, the available E2SM-RC actions are similarly extracted from the RAN Function Definition, which encodes the supported RIC Control Styles and their associated Control Actions with per-action parameter definitions [110]. In our testbed, srsRAN advertises Action 6 (slice-level PRB quota) as the supported control action. The discovered capabilities are exported to a structured JSON database that the Capability Checker loads at runtime.

Given a mapped intent specification, the Capability Checker performs validation by verifying each requested metric and control action against the capability database. Metric availability is checked through direct and partial string matching against the E2SM-KPM metric list; control action availability is checked against the E2SM-RC action catalog. The checker classifies the overall result into one of four statuses: *available* (all requirements satisfied), *partially available* (some metrics missing but control actions present), *not available* (required control actions missing), or *alternative available* (a substitute exists). Missing control actions prevent the pipeline from proceeding, as the xApp would be unable to actuate its decisions; missing monitoring metrics trigger a warning but do not block generation if alternative observables can serve the

optimization objective.

By explicitly enforcing capability awareness before code synthesis, Auto-xApp guarantees that generated xApps reference only metrics and actions that the target RAN platform actually supports. The validated configuration produced at this stage forms the final specification passed to the xApp Generator.

4.3.6 xApp Code Generation

Given a validated specification comprising the structured intent, dataset schema, and confirmed capabilities, the *xApp Generator* produces deployable xApp source code. Auto-xApp supports three generation paradigms—rule-based, supervised learning (DL), and deep reinforcement learning (DRL)—each employing a different combination of LLM-driven and programmatic code synthesis.

For the **rule-based** paradigm, the LLM agent directly generates a complete xApp implementation. It receives the validated workflow specification, the structured control rules extracted during intent translation and a reference xApp that demonstrates the target platform’s E2 subscription and control APIs. The LLM synthesizes a self-contained xApp that implements the specified conditional logic while conforming to the platform-specific interfaces (e.g., xAppBase for FlexRIC or OSC RIC).

For the **supervised/unsupervised learning** paradigm, code generation is split into two stages. In the first stage, an LLM-based *Learning Design Agent* analyzes the task description, dataset characteristics, and deployment constraints to recommend a model, its hyperparameters, and a training strategy. If the user has already specified these during intent elicitation, the design agent incorporates their choices directly; otherwise, the agent provides a recommendation that the user may accept, modify, or override. In the second stage, a programmatic *Learning Code Generator* produces a training script and a deployment xApp through parameterized templates that incorporate the selected model and its configuration. The model instantiation code uses runtime introspection to automatically adapt scoring and prediction calls to whichever model is

configured, enabling support for off-the-shelf ML models without model-specific templates.

For the **DRL** paradigm, code generation is similarly split into two stages. In the first stage, an LLM-based *DRL Design Agent* takes the workflow specification, available KPMs, and dataset schema, and produces a complete DRL system design: the algorithm selection (e.g., DDPG, PPO, DQN) with rationale, the state space composition from available metrics, the action space configuration derived from control parameter ranges, the reward function expressed as executable Python code, and the training hyperparameters. If the user has already specified the algorithm and hyperparameters during intent elicitation, the design agent incorporates their choices directly; otherwise, the agent recommends them while the LLM still designs the remaining components (state space, reward function). In the second stage, a programmatic *DRL Code Generator* consumes this design and produces a multi-file xApp package through parameterized templates: a Gymnasium-compatible training environment, a training script using Stable-Baselines3, a deployment xApp with E2 subscription and control loops, a model loader, configuration files, and documentation. The templates are parameterized by the workflow-derived settings including reporting period, monitoring granularity, control granularity, and trigger conditions.

Across all three paradigms, the generated code is directly executable on the target RIC platform. Each xApp package includes the complete source files, dependency specifications, configuration, and documentation required for deployment.

4.4 Evaluation

4.4.1 Testbed Setup

We implement and evaluate Auto-xApp on a software-defined 5G O-RAN testbed built around the open-source srsRAN project [141]. The testbed comprises three major subsystems: a standalone 5G core network, a RAN with over-the-air transmission, and a near-real-time RIC [140] hosting the generated xApps. The gNodeB (gNB) runs on an Ubuntu 22.04 workstation. Over-the-air radio transmission is handled by a USRP X310 software-defined radio (SDR) unit

connected to the host machine operating in 5G NR band n78 (3.5 GHz TDD). Commercial off-the-shelf (COTS) user equipment (UE) devices with programmable USIM cards (sysmoISIM-SJA2 [150]) serve as end users and are configured to attach to specific network slices via their NSSAI parameters. Auto-xApp is powered by an LLM agent based on o4-mini [117], serving as the backbone of the Intent Translator, Dataset Analyzer, and xApp Generator throughout the automated development pipeline. The backend is implemented in Python 3.10, with ChromaDB providing vector storage and semantic retrieval over O-RAN domain knowledge, including O-RAN Alliance specifications and prior xApp implementation references.

We evaluate Auto-xApp across three use cases, each corresponding to one of the supported xApp generation paradigms: rule-based control, learning-based inference, and deep reinforcement learning (DRL)-based optimization.

Rule-based xApp. We task Auto-xApp with generating a deterministic control xApp from a natural language operator intent, without any training data. This use case validates the end-to-end pipeline, verifying that Auto-xApp correctly interprets operator requirements and generates a conformant xApp that enforces the specified control policies via the E2 interface.

Learning-based xApp. We validate the learning-based generation pipeline using the SpotLight [145] anomaly detection dataset. Auto-xApp automatically selects and trains an appropriate model from the dataset characteristics. We report F1 score under two feature configurations: full features (combining platform and E2 radio metrics) and E2-only features.

DRL-based xApp. We evaluate the DRL-based xApp on a network slicing task involving PRB allocation across three O-RAN slices: eMBB, URLLC, and MTC. The agent is trained on Scenarios B and C and evaluated on Scenario A as an out-of-distribution generalization test. Performance is measured against per-slice SLA targets (eMBB downlink throughput, URLLC packet delay, MTC throughput) and benchmarked against an offline sweep optimum serving as an upper-bound reference.

4.4.2 Use Case 1: Rule-Based GBR Enforcement

To validate the rule-based xApp generation paradigm, we task Auto-xApp with a GBR (Guaranteed Bit Rate) enforcement use case. The natural language requirement specifies: *“Monitor per-slice DL throughput for a two-slice deployment. If the GBR slice throughput drops below a target threshold, decrease PRB quota of non-GBR slice. Release PRBs back when throughput recovers.”*

Setup. We configure a two-slice testbed with one GBR user (SST=1) and one non-GBR user (SST=2), each assigned to a separate network slice. To create dynamic channel conditions, we deploy a GNU Radio Companion (GRC) channel emulator between the gNB and the GBR UE’s ZMQ path, injecting controllable AWGN noise on the downlink. The BE user operates on a clean channel throughout. Both UEs run continuous iperf3 DL traffic to compete for PRB resources. The xApp subscribes to per-UE DL throughput via E2SM-KPM and issues per-slice PRB quota adjustments via E2SM-RC Action 6.

Generated xApp logic. From the natural language intent, Auto-xApp’s Paradigm Classifier routes the requirement to the rule-based code generator, which produces a threshold-based control loop with the following logic: (i) if GBR throughput $< T_{\text{gbr}}$, increase the GBR slice PRB quota by a configurable step Δ and decrease the BE quota accordingly; (ii) if GBR throughput $> T_{\text{gbr}} + T_{\text{hyst}}$, release PRBs back to the BE slice; (iii) enforce minimum and maximum PRB bounds per slice to prevent starvation. The generated code includes hysteresis ($T_{\text{hyst}} = 1000$ kbps) to suppress oscillation and a configurable control interval (default 2 s) to avoid overreacting to transient fluctuations.

Results. We evaluate the xApp under a repeatable noise injection scenario: after an initial clean phase, we increase the GBR user’s channel noise amplitude from 0 to 0.3 over 60 seconds, hold for 120 seconds, then restore the clean channel. Without the xApp, the GBR user’s throughput drops from approximately 25 Mbps to below 5 Mbps and remains degraded for the entire noise period. With the xApp active (threshold $T_{\text{gbr}} = 5000$ kbps, step $\Delta = 5\%$),

the control loop detects the throughput degradation within one monitoring interval and begins reallocating PRBs from the BE slice to the GBR slice. Over approximately 4–6 control cycles (8–12 seconds), the GBR slice PRB quota increases from the initial 50% to 70–80%, recovering GBR throughput above the target threshold. When the noise is removed, the xApp gradually releases PRBs back to the BE slice over several cycles due to the hysteresis mechanism, restoring the balanced allocation.

This experiment demonstrates that Auto-xApp can generate a functionally correct, deployable rule-based xApp from a natural language specification. The generated code correctly implements threshold monitoring, bidirectional PRB reallocation, bounds enforcement, and hysteresis—all derived automatically from the intent without manual coding.

4.4.3 Use Case 2: Learning-Based Anomaly Detector

Dataset We evaluate Auto-xApp using the SpotLight dataset [146], the largest publicly available Open RAN anomaly detection dataset. SpotLight was collected from an enterprise-scale 5G Open RAN deployment using the Janus telemetry framework, comprising over 100 million data points at 100ms granularity. The dataset includes four anomaly scenarios: MAC layer contention, radio interference, network contention, and mixed anomalies.

The publicly released SpotLight dataset contains 206 RAN-layer features. We preprocess these by applying variance-based selection (top-50 base features) and Exponentially Weighted Moving Average (EWMA) smoothing with spans of 5, 10, and 20, yielding 200 features as input to our model. We note that SpotLight’s full pipeline uses 600+ KPIs, including 466 platform-layer metrics such as thread scheduling events and Precision Time Protocol (PTP) synchronization logs collected via extended Berkeley Packet Filter (eBPF) instrumentation. These platform-layer metrics are not observable through standardized O-RAN E2 interfaces and thus cannot be replicated in a deployable xApp, motivating our focus on the radio-layer subset.

Baselines We compare Auto-xApp against SpotLight and seven baseline methods reported in the original work:

- **SpotLight** [147]: Multi-stage deep generative pipeline (JVGAN + MRPI + CausalNex) using 600+ features.
- **TranAD** [156]: Transformer-based reconstruction method.
- **MADGAN** [75]: GAN-based multivariate anomaly detection.
- **VAE-LSTM** [84]: Hybrid variational autoencoder with LSTM.
- **GDN** [31]: Graph neural network-based detection with explainability.
- **LSTM-AE/LSTM-PRED** [68]: Autoencoder and prediction-based LSTM methods.
- **Z-Score**: Statistical baseline using sliding window z-score.

Auto-xApp Configurations We evaluate four Auto-xApp configurations spanning two feature sets and two model selection strategies. The *LLM* configurations are fully automated: the Auto-xApp agent analyzes the dataset, selects the anomaly detection algorithm, and tunes hyperparameters without human intervention. The *Manual* configurations replace the LLM’s model selection with Local Outlier Factor (LOF) [18], a density-based anomaly detection algorithm that identifies outliers by comparing the local density of a sample to that of its neighbors; this serves as a human-expert upper bound on model selection quality.

- **Auto-xApp Full (LLM)**: Uses all 200 RAN features; model and hyperparameters selected automatically by the LLM agent (Isolation Forest).
- **Auto-xApp Full (Manual)**: Uses all 200 RAN features with manually selected LOF.
- **Auto-xApp E2-Only (LLM)**: Uses only 8 E2-compatible features (derived from 2 base metrics via EWMA expansion); model and hyperparameters selected automatically by the LLM agent (Isolation Forest).
- **Auto-xApp E2-Only (Manual)**: Uses only 8 E2-compatible features with manually selected LOF.

Table 4.1. Overall Anomaly Detection Performance on SpotLight Dataset

Method	Model	Features	Precision	Recall	F1
SpotLight [147]	JVGAN+MRPI	600+	95.5%	94.5%	94.9%
Auto-xApp Full (Manual)	LOF	200	83.7%	94.6%	88.7%
Auto-xApp Full (LLM)	Isolation Forest	200	79.3%	95.0%	86.2%
Auto-xApp E2-Only (Manual)	LOF	8	70.0%	97.8%	81.1%
Auto-xApp E2-Only (LLM)	Isolation Forest	8	67.1%	96.5%	78.7%
TranAD [156]	Transformer	600+	71.0%	92.0%	80.0%
MADGAN [75]	GAN	600+	68.0%	88.0%	77.0%
VAE-LSTM [84]	Hybrid	600+	48.0%	93.0%	63.0%
GDN [31]	Graph NN	600+	30.0%	66.0%	41.0%
LSTM-AE [68]	Autoencoder	600+	12.0%	96.0%	21.0%
LSTM-PRED [155]	Prediction	600+	13.0%	94.0%	21.0%
Z-Score	Statistical	600+	13.0%	81.0%	21.0%

Following SpotLight, we report precision, recall, and F1 score. For RIC deployment validation, we use OSC RIC with srsRAN 5G as the E2 node. The xApp subscribes to E2SM-KPM Report Service Style 1 and performs real-time inference on incoming indication messages.

Overall Performance

Table 4.1 presents the overall anomaly detection performance across all methods. Auto-xApp Full (Manual) with LOF achieves 88.7% F1 with 83.7% precision and 94.6% recall, outperforming all baselines except SpotLight’s complex multi-stage pipeline. The LLM-recommended Isolation Forest achieves 86.2% F1, demonstrating effective automated model selection.

Notably, Auto-xApp achieves competitive performance using only 200 features compared to 600+ used by baselines. The E2-Only configurations with merely 8 features achieve 78.7–81.1% F1, outperforming MADGAN (77.0%) while being the *only* configuration deployable on standard O-RAN infrastructure.

Per-Scenario Analysis

Table 4.2 presents per-scenario performance metrics. Auto-xApp demonstrates consistent performance across all anomaly types, with particularly strong results on MIXED scenarios (92.5–

Table 4.2. Per-Scenario Performance (Precision / Recall / F1)

Method	MAC	RADIO	NETWORK	MIXED
SpotLight	100/93/96.3	95/93/93.9	94/92/92.9	95.5/94.5/95.0
Auto-xApp Full (Manual)	91.0/93.5/92.3	77.3/93.9/84.8	80.3/91.1/85.4	86.0/100/92.5
Auto-xApp Full (LLM)	74.8/92.8/82.8	72.9/96.3/83.0	71.4/91.4/80.2	98.2/99.6/98.9
Auto-xApp E2-Only (Manual)	71.8/91.3/80.4	65.0/99.7/78.7	57.1/100/72.7	86.1/99.9/92.5
Auto-xApp E2-Only (LLM)	60.1/86.3/70.8	65.0/99.8/78.8	57.2/99.7/72.7	86.0/100/92.5

Table 4.3. Deployment Capability Comparison

Method	E2SM-KPM	RIC Deploy	Code Gen
SpotLight	55	55	Manual
Auto-xApp (Full)	55	55	Auto
Auto-xApp (E2-Only)	51	51	Auto

98.9% F1). The E2-Only configuration shows graceful degradation, maintaining competitive performance despite using 96% fewer features.

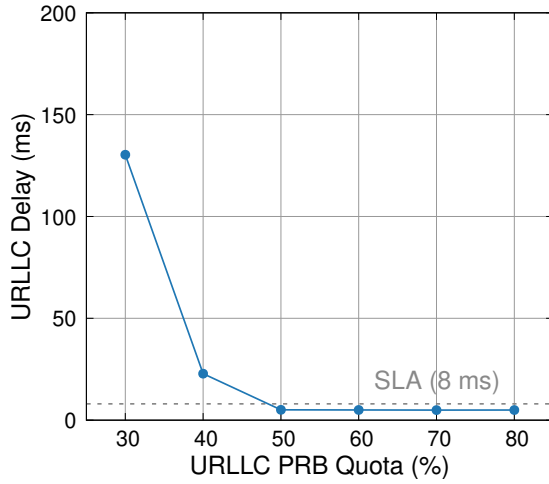
The performance gap between Full and E2-Only configurations varies by scenario: MIXED shows minimal degradation while MAC exhibits larger drops (92.3% $\rightarrow$ 80.4% for LOF). This aligns with expectations, as MAC-layer anomalies may require features not exposed via current E2SM-KPM implementations.

RIC Deployment Validation

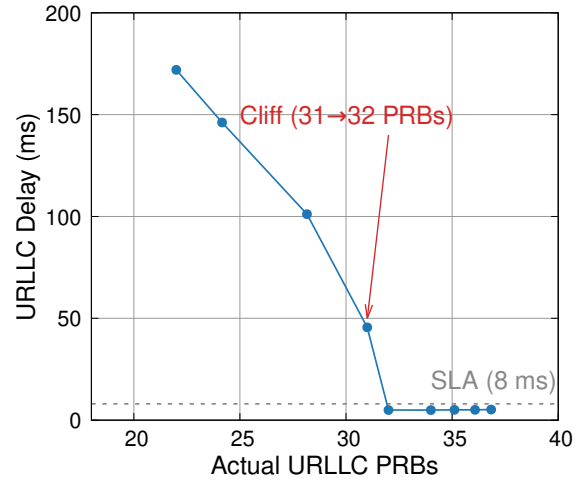
To validate end-to-end deployment, we deployed the Auto-xApp generated xApp on OSC RIC with srsRAN as the E2 node. Table 4.3 summarizes the deployment capabilities across methods.

The generated xApp successfully:

1. Connects to OSC RIC via RMR messaging
2. Sets E2SM-KPM RAN function ID
3. Subscribes to KPM Report Service Style 1
4. Receives real-time indication messages



(a) Delay vs. PRB Quota Setting



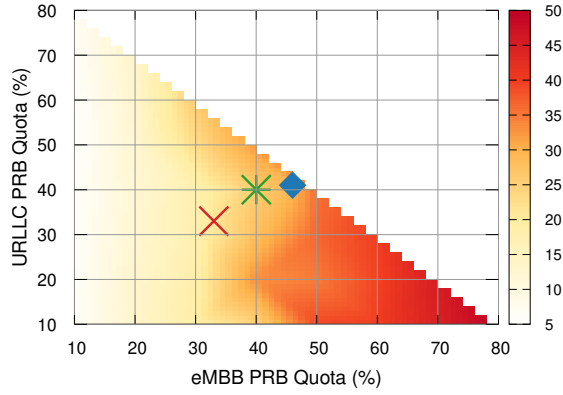
(b) Delay vs. Actual PRB

Figure 4.4. URLLC delay exhibits a sharp performance cliff around 31–32 PRBs.

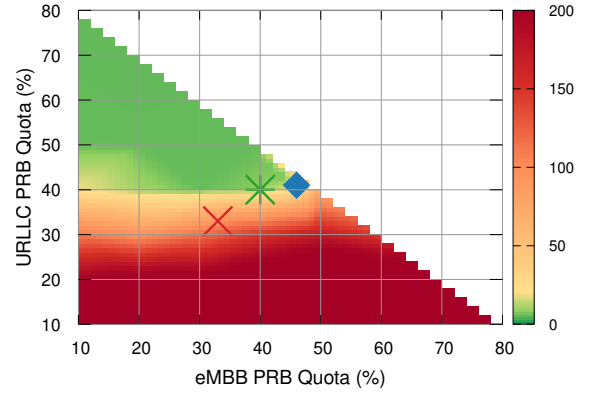
5. Applies reverse feature mapping (E2 metric names → training feature names)
6. Computes EWMA preprocessing with state management
7. Performs model inference and anomaly scoring

These results confirm that Auto-xApp generates fully deployable xApps without manual code modification. While the Full configuration cannot be directly deployed due to E2SM-KPM interface constraints, the E2-Only configuration operates directly over standard O-RAN interfaces, demonstrating that Auto-xApp can generate xApps deployable on any compliant O-RAN RIC without vendor-specific instrumentation.

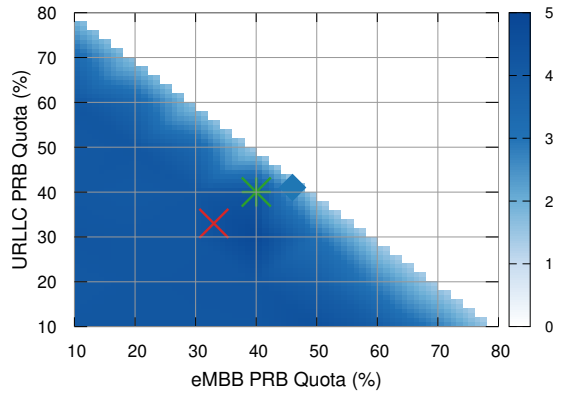
Limitations. Auto-xApp generates xApps using off-the-shelf ML models from standard libraries (scikit-learn, PyTorch, TensorFlow), covering a wide range of algorithms from classical ML to deep learning. Custom cascaded architectures (e.g., SpotLight’s JVGAN+MPRI multi-stage pipeline) require paper-specific implementation that is beyond Auto-xApp’s automated generation scope.



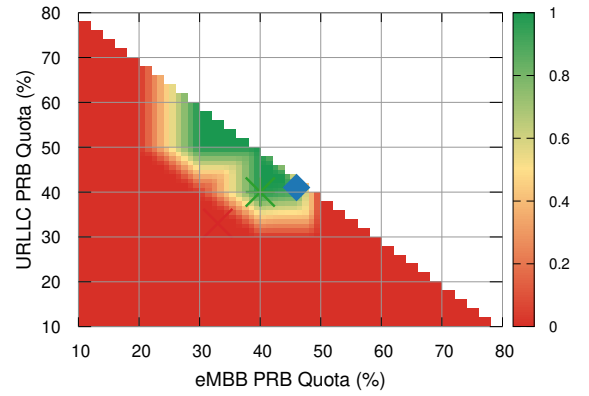
(a) eMBB Throughput (Mbps)



(b) URLLC Delay (ms)

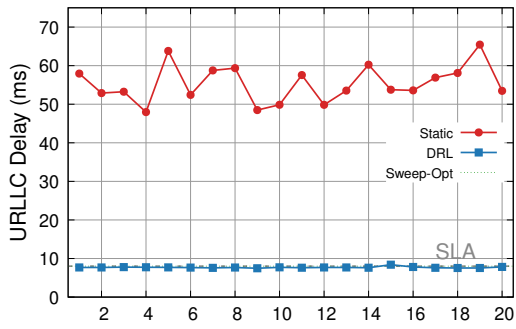


(c) MTC Throughput (Mbps)

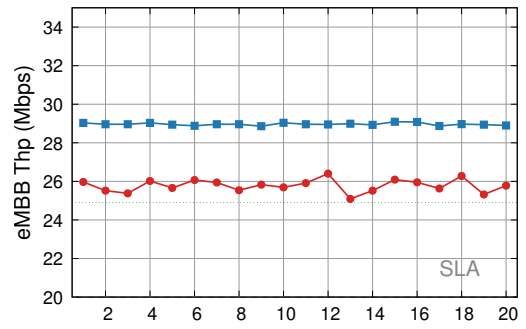


(d) Joint SLA Satisfaction

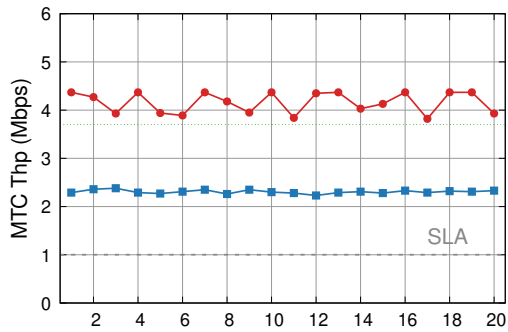
Figure 4.5. PRB allocation landscape across eMBB, URLLC, and MTC slices. Markers denote operating points: $\times$ Static (33/33/33), $*$ Sweep-Opt (40/40/20), $\diamond$ DRL.



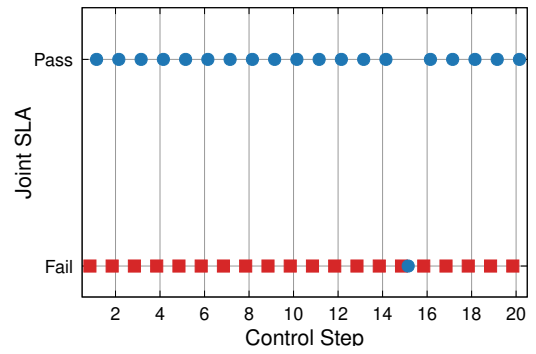
(a) URLLC Delay



(b) eMBB Throughput

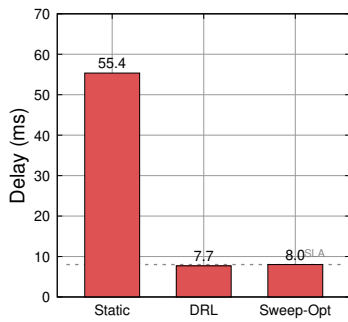


(c) MTC Throughput

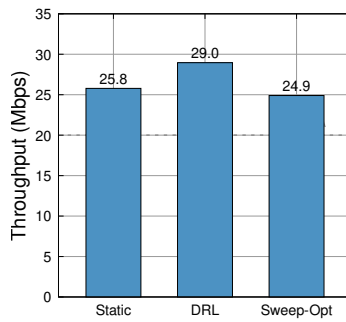


(d) Joint SLA

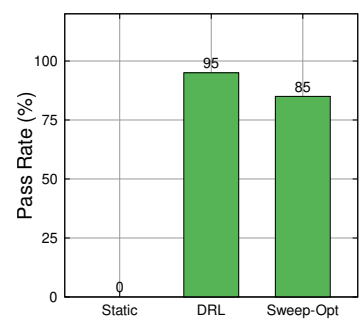
Figure 4.6. Step-by-step comparison of Static vs. DRL xApp across KPIs. Dashed gray line: SLA threshold. Dashed green line: Sweep-Opt reference.



(a) URLLC Delay



(b) eMBB Throughput



(c) Joint SLA

Figure 4.7. Performance comparison across Static, Sweep-Opt, and DRL xApps.

4.4.4 Use Case 3: DRL-Based PRB Slicing

To evaluate the DRL xApp generation capability of Auto-xApp, we consider a network slicing scenario in which a DRL agent dynamically allocates radio resources across three concurrent slices with competing Quality of Service (QoS) requirements.

Scenario and Traffic Configuration

We configure the testbed with three network slices, each serving a distinct service type: enhanced Mobile Broadband (eMBB, SST=1), Ultra-Reliable Low-Latency Communication (URLLC, SST=2), and massive Machine-Type Communication (MTC, SST=3). Each slice is assigned one dedicated UE generating continuous downlink traffic representative of its service class. The eMBB UE generates high-throughput traffic to saturate its allocated bandwidth, the URLLC UE sends periodic small-packet flows with strict latency requirements, and the MTC UE produces moderate-rate periodic transmissions. The gNB operates on a 20 MHz channel providing 106 usable PRBs, which must be partitioned among the three slices to simultaneously satisfy their respective SLA targets: eMBB throughput exceeding 20 Mbps, URLLC RLC-layer delay below 8 ms, and MTC throughput exceeding 1 Mbps.

PRB Quota vs. Actual Allocation

A key subtlety in O-RAN resource control is the distinction between the *PRB quota* configured via E2SM-RC and the *actual PRB allocation* realized by the MAC scheduler. The PRB quota, set through E2SM-RC Action 6, specifies a soft upper bound on the percentage of PRBs each slice may use. However, the actual number of PRBs allocated to a slice at any given scheduling instant depends on the interaction between competing slices' demands, the scheduler's internal prioritization logic, and the instantaneous channel conditions. As a result, the same quota percentage can yield different actual allocations under different traffic conditions.

We observe this discrepancy directly in our measurements: a 40% URLLC quota yields 37 actual PRBs when eMBB is set to 40%, but only 31 PRBs when eMBB is raised to 50%. This

indirect mapping between the configured control parameter and the realized allocation makes the resource management problem fundamentally more challenging than a simple partitioning task. The DRL agent must learn this mapping implicitly by observing both the quota it sets (its action) and the actual PRB usage reported via E2SM-KPM (part of its state), adapting its quota settings to achieve the desired allocation despite the scheduler’s non-transparent behavior.

It is important to note that the PRB quota is the *only* control knob available to xApps through the standard E2SM-RC interface for per-slice resource management. The MAC scheduler’s internal scheduling decisions—such as per-UE PRB assignments and scheduling order—operate at the real-time control loop (sub-millisecond) and are not exposed through the near-RT RIC’s E2 interface. Consequently, the xApp must influence resource distribution indirectly by adjusting slice-level quotas, rather than directly controlling per-PRB assignments.

Offline Environment Characterization

Before deploying the DRL agent, we characterize the allocation landscape through an exhaustive PRB sweep to identify the best static configuration and to understand the structure of the optimization problem. We systematically test all 36 valid PRB quota configurations at 10% granularity (with each slice receiving at least 10%), collecting per-slice throughput, URLLC delay, and actual PRB usage over multiple measurement intervals per configuration. The E2SM-KPM indication reports are used to measure per-slice DL throughput (DRB.UETHPD1), RLC-layer delay (DRB.RLCSduDelayD1), and actual PRB utilization (RRU.PrbUsedD1).

The sweep results, visualized as heatmaps in Figure 4.5, reveal several important characteristics of the allocation space. First, eMBB throughput scales approximately linearly with eMBB PRB quota, while MTC throughput exhibits a similar linear relationship with its quota. Second, URLLC delay exhibits a sharp nonlinear cliff: delay remains below 8 ms when the URLLC slice receives at least 32 actual PRBs, but rises catastrophically—from 5 ms to over 100 ms—when allocation drops below this threshold (Figure 4.4). This phase-transition behavior is consistent with queuing theory: once offered load exceeds slice capacity, packets accumulate

in the RLC buffer and delay grows rapidly.

The URLLC cliff further underscores the gap between PRB quota and actual allocation. When plotted against the configured quota percentage (Figure 4.4a), the cliff appears as a gradual transition between 30% and 50%. However, when plotted against the actual number of PRBs allocated (Figure 4.4b), the cliff is revealed to be a sharp step between 31 and 32 PRBs—a single PRB difference separating 45 ms delay from 5 ms delay. This distinction highlights why the DRL agent must reason about the indirect effect of its quota actions on actual resource allocation, rather than treating the quota as a direct allocation.

Among all 36 configurations tested, the best-performing static allocation is 40/40/20 (eMBB/URLLC/MTC), which we refer to as the *sweep optimum*. The limited granularity of the 10%-step sweep means the true optimal allocation likely lies between the tested grid points, in a region that the discrete sweep cannot explore.

DRL-Based xApp Generation

The LLM agent determines that the problem structure (continuous state space, continuous action space, multi-objective SLA reward) is best suited to *Deep Deterministic Policy Gradient* (DDPG) [83], a model-free actor-critic algorithm designed for continuous action spaces. The LLM agent automatically determines the network architecture based on the inferred state dimensionality (9 features: per-slice throughput, URLLC delay, and per-slice PRB usage) as a medium-complexity problem, selecting two fully connected hidden layers of 128 neurons each with ReLU activations (`net_arch=[128, 128]`). The agent is trained in offline mode using the 36-configuration sweep dataset pre-loaded into the replay buffer, with `learning_starts=0` to train immediately without online exploration. Key hyperparameters selected by the LLM are: learning rate $\alpha = 10^{-4}$, discount factor $\gamma = 0.99$, soft target update rate $\tau = 0.005$, and batch size 64. Gaussian exploration noise with standard deviation $\sigma = 10$ PRB units (decaying to 1) is applied during action selection. The model is trained for 50000 gradient steps via `stable-baselines3` [129].

The training dataset for the DRL agent is collected directly from the testbed by running the 36-configuration sweep and recording state transitions. Each sample consists of a state vector (per-slice throughput, URLLC delay, and per-slice actual PRB usage), the applied action (PRB quota percentages), the resulting next state, and a scalar reward. The reward function is automatically generated by the LLM from the user’s natural language SLA requirements, combining normalized per-slice SLA satisfaction:

$$r = \frac{1}{3} \min\left(\frac{\text{thp}_{\text{eMBB}}}{T_{\text{eMBB}}}, 1\right) + \frac{1}{3} \max\left(1 - \frac{d_{\text{URLLC}}}{D_{\text{URLLC}}}, 0\right) + \frac{1}{3} \min\left(\frac{\text{thp}_{\text{MTC}}}{T_{\text{MTC}}}, 1\right) \quad (4.1)$$

where $T_{\text{eMBB}} = 40$ Mbps, $D_{\text{URLLC}} = 8$ ms, and $T_{\text{MTC}} = 2$ Mbps are the per-slice SLA thresholds extracted directly from the user’s intent. This demonstrates that Auto-xApp correctly translates high-level SLA requirements into a well-structured reward function without any manual engineering.

The DRL agent is trained offline using the collected dataset as a replay buffer. The actor and critic networks are implemented as multi-layer perceptrons using the Stable-Baselines3 library. Training proceeds for 50000 gradient steps with a learning rate of 3×10^{-4} and a batch size of 256. Crucially, the entire training pipeline—from environment definition to agent architecture and hyperparameter selection—is automatically generated by Auto-xApp’s DRL code generator based on the natural language requirement and the collected dataset, without manual intervention.

Live Deployment Results

We evaluate the trained DRL agent by deploying it on the near-RT RIC and comparing its performance against two baselines: the naïve equal-split allocation (33/33/33) and the sweep optimum (40/40/20) identified through the offline characterization. Each method is run for 20 consecutive control intervals (each 15 s), and per-step KPMs are recorded via E2SM-KPM.

Figure 4.6 shows the step-by-step comparison. The DRL agent converges to a fixed

Table 4.4. Performance comparison: static baseline, sweep optimum, and auto-generated DRL xApp.

Metric	Static 33/33/33	Sweep-Opt 40/40/20	DRL 46/41/11
URLLC delay (ms)	55.4	8.0	7.7
eMBB throughput (Mbps)	25.8	24.9	29.0
MTC throughput (Mbps)	4.2	3.7	2.3
Joint SLA compliance	0%	85%	95%

allocation of 46/41/11 (eMBB/URLLC/MTC), which it applies consistently across all 20 steps. Compared to the equal-split baseline, the DRL agent reduces URLLC delay by $7\times$ (from 55.4 ms to 7.7 ms), bringing it well below the 8 ms SLA threshold. Simultaneously, eMBB throughput improves by 12% (from 25.8 to 29.0 Mbps) because the agent allocates a larger share to eMBB than the equal split. MTC throughput decreases from 4.2 to 2.3 Mbps, reflecting the agent’s prioritization of eMBB and URLLC, but remains above the 1 Mbps SLA target.

The most striking result is the comparison of joint SLA compliance, summarized in Table 4.4. The equal-split baseline fails the URLLC SLA on every step (0% joint compliance). The sweep optimum (40/40/20)—the best configuration identified through the exhaustive 36-point search—achieves only 85% joint compliance, because its average URLLC delay of 8.0 ms sits precisely at the SLA boundary and fluctuates above it. The DRL agent, by contrast, achieves 95% joint compliance (19 out of 20 steps), with only one marginal violation at 8.4 ms.

The DRL agent’s advantage stems from its ability to operate in a continuous action space. The coarse-grained sweep at 10% resolution tests 36 discrete configurations but cannot explore the space between grid points. The configuration 40/40/20 lands exactly at the URLLC cliff boundary, where a difference of 0.3 ms in average delay separates 39% from 95% SLA compliance. The DRL agent, unconstrained by grid resolution, discovers a nearby but finer-grained allocation at 46/41/11 that clears the cliff with sufficient margin. A hypothetical exhaustive search at 1% resolution would require over 5,000 configurations—impractical on real testbed hardware—yet the DRL agent finds a comparable operating point automatically through

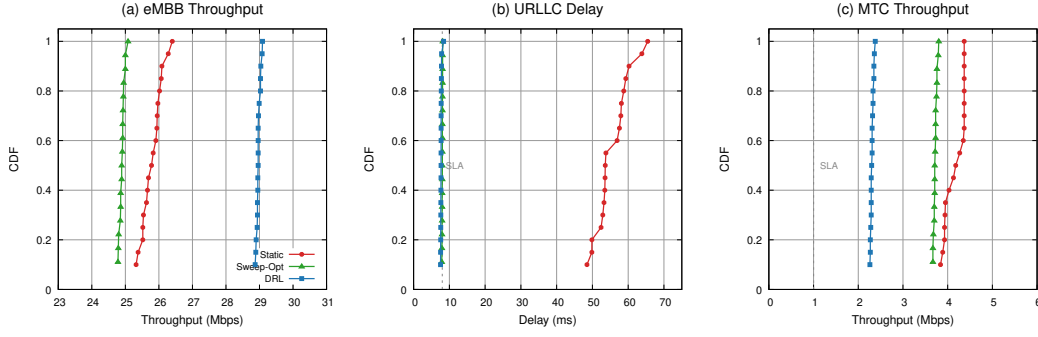


Figure 4.8. Per-slice CDF comparison for eMBB throughput, URLLC delay, and MTC throughput across static allocation, sweep optimum, and the DRL xApp.

gradient-based optimization.

The cumulative distribution functions (CDFs) in Figure 4.8 provide further insight into the per-slice performance distributions. The URLLC delay CDF (the URLLC-delay panel of Figure 4.8) most clearly illustrates the distinction: the static baseline’s delay distribution is concentrated around 48–66 ms, entirely above the SLA threshold; the sweep optimum’s distribution clusters tightly around 7.8–8.1 ms, straddling the 8 ms boundary; while the DRL agent’s distribution falls between 7.5–8.4 ms with the majority below the threshold. For eMBB throughput (the eMBB-throughput panel of Figure 4.8), the DRL agent’s distribution is shifted to the right of both the static baseline and the sweep optimum, reflecting the agent’s decision to allocate 46% to eMBB rather than 33% or 40%. The MTC throughput CDF (the MTC-throughput panel of Figure 4.8) shows the trade-off: the DRL distribution is shifted left compared to the other methods, but remains entirely above the 1 Mbps SLA threshold.

These results demonstrate that Auto-xApp’s DRL code generator can produce a functional DRL agent that, when trained on testbed data and deployed on a live O-RAN platform, discovers a resource allocation policy that outperforms the best static configuration identified through exhaustive coarse-grained search—achieving 95% joint SLA compliance compared to the sweep optimum’s 39%. The agent accomplishes this without any manual tuning, domain expertise, or knowledge of the underlying performance cliff, relying solely on the automatically generated

reward function and the training data collected from the testbed.

Note. Although the evaluation in this chapter uses slice-level PRB quota control (Action 6), Auto-xApp is not limited to a single control action. The framework supports any E2SM-RC action advertised in the capability database. To verify this extensibility, we implemented an additional control action (Action 7, TX power control) in srsRAN and confirmed that the Capability Checker correctly discovers and validates it through the same E2 SETUP REQUEST discovery mechanism.

4.5 Discussion and Limitations

Auto-xApp demonstrates that generative agents can automate a substantial portion of the xApp development workflow, but several limitations remain. First, Auto-xApp is bounded by the capabilities exposed by the target E2 node and service-model implementation. A user objective may be semantically valid with respect to the O-RAN specification, but infeasible on a particular platform if the required KPMs or control actions are not advertised through E2. Auto-xApp addresses this through capability checking, but it cannot synthesize control paths that the underlying platform does not support.

Second, the quality of generated xApps depends on the correctness and completeness of the knowledge artifacts used by the framework, including the O-RAN specification index, platform capability database, workflow schema, and reference RIC code base. Retrieval-augmented generation reduces hallucination by grounding the LLM in relevant documentation, but it does not eliminate the need for deterministic validation and testbed-level verification.

Third, Auto-xApp currently targets representative rule-based, learning-based, and DRL-based paradigms. These paradigms cover a broad set of xApp designs, but highly customized architectures, proprietary telemetry formats, or multi-component control systems may still require manual adaptation. Extending Auto-xApp to support richer composition patterns, multi-xApp workflows, and cross-platform portability remains an important future direction.

Finally, Auto-xApp focuses on generating individual xApps. Once multiple generated or independently developed xApps coexist in the same RAN environment, their control actions may interact or conflict. This limitation motivates the next chapter, which studies conflict detection and mitigation among xApps using generative agents.

4.6 Chapter-Specific Related Work

4.6.1 Performance-Driven xApps in O-RAN

A substantial body of recent work leverages the near-real-time (near-RT) RIC to design xApps that improve specific RAN performance objectives. These studies demonstrate the practical feasibility of closed-loop optimization in O-RAN and highlight the flexibility of the E2-based control architecture.

RAN slicing and resource allocation. Several works develop reinforcement learning (RL) and deep RL-based xApps for dynamic network slicing and resource management. Latency-aware slicing via DRL has been explored to jointly optimize throughput and delay under realistic RIC integration constraints [130]. The REAL framework demonstrates experimental RL-enabled xApps operating with OSC RIC and srsRAN for closed-loop slice optimization [13]. ORANSlice presents an open-source slicing platform enabling programmable slice-level control through xApps [28]. Additional studies investigate SLA-driven QoS slicing [185], actor-critic-based resource allocation [69], and scheduler parameter tuning through O-RAN control loops [144]. Collectively, these works focus on improving slice isolation, latency guarantees, and spectrum efficiency through learning-driven control.

Traffic steering and spectrum optimization. Beyond slicing, xApps have been proposed for policy-based traffic steering and load balancing [143], hierarchical learning for traffic steering [47], and mobility-aware dynamic spectrum allocation through coordinated rApp/xApp control [43]. Prediction-driven slice control using LSTM-based traffic forecasting has also been studied to proactively adapt resource allocation decisions [89]. These works show how near-RT

intelligence can enhance spectrum utilization and user QoS through data-driven adaptation.

Mobility management. Several efforts extend xApps to mobility and handover optimization. Customized handover strategies implemented as xApps enable adaptive HO decision-making beyond static threshold configurations [160]. Meta-learning-based handover frameworks further improve generalization across varying radio conditions [65].

Operational intelligence and energy efficiency. xApps have also been used for operational monitoring and anomaly mitigation. SpotLight presents an explainable anomaly detection framework for Open RAN telemetry [147]. Learning-based interference mitigation has been integrated into xApps to improve throughput and reliability under adverse radio conditions [8]. Energy-aware xApps for RU sleep control and energy-efficient resource management demonstrate substantial reductions in power consumption while maintaining QoS [7, 82, 90].

4.6.2 LLMs in Communication Networks

Large language models (LLMs) have demonstrated strong capabilities in structured reasoning, tool use, and program synthesis, which have motivated their adoption in engineering and systems domains [53, 59, 127, 165]. Rather than serving solely as generative text models, recent LLM-based systems increasingly operate as autonomous agents capable of multi-step reasoning, external tool interaction, and iterative refinement.

Within the networking domain, early efforts primarily investigate LLMs for document-centric tasks, such as question answering over telecom standards, specification summarization, terminology classification, and knowledge graph construction [22, 52, 66, 93]. These approaches leverage LLMs as semantic analyzers to extract or organize information from technical corpora but do not directly integrate them into network control loops.

More recently, researchers have begun exploring LLM-driven automation in communication systems. Emerging studies examine how LLMs can assist with network management, intent interpretation, and system configuration [11, 24, 100]. In parallel, agent-based LLM frameworks enable models to decompose complex tasks into intermediate reasoning steps and interact with

structured system representations [27, 76]. Such approaches suggest the potential for LLMs to participate in higher-level orchestration of communication networks.

4.7 Summary

This chapter presented Auto-xApp, a structured automation framework for synthesizing deployable O-RAN xApps from high-level requirements. We identified the fundamental gap between operator intent and standards-compliant xApp deployment, arising from requirement ambiguity, semantic mismatches with O-RAN service models, and platform-specific capability constraints. Auto-xApp addresses this gap through schema-guided intent formalization, retrieval-grounded specification alignment, E2 capability validation, and paradigm-aware code generation supporting rule-based, supervised, and DRL-based control strategies. The implementation on OSC near-RT RIC with srsRAN demonstrates that automatically generated xApps can successfully perform E2 subscription, telemetry processing, and closed-loop control without manual integration. By systematizing the transformation from intent to executable xApps, Auto-xApp moves O-RAN closer to practical, scalable programmability and lowers the barrier to deploying intelligent RAN control applications. This chapter showed how generative agents can synthesize deployable xApps from high-level requirements. The next chapter considers what happens after multiple xApps are developed and deployed: their objectives and control actions may conflict, requiring context-aware conflict detection and mitigation.

Chapter 4, in part, has been submitted for publication of the material as it may appear in Dae Cheol Kwon and Xinyu Zhang, “Auto-xApp: Autonomous xApp Synthesis for Open RAN via Generative Agents,” submitted to *IEEE International Conference on Network Protocols (ICNP)*, 2026. The dissertation author was the primary investigator and first author of this material.

Chapter 5

Generative-Agent-Based Conflict Detection and Mitigation in Open RAN

This chapter presents ORCA, a generative-agent-based framework for detecting and mitigating conflicts among independently developed xApps in Open RAN. The previous chapter studied how generative agents can synthesize deployable xApps from high-level requirements. Once such xApps are deployed together, however, their control decisions may interact through shared RAN resources, overlapping control parameters, or coupled key performance indicators. This chapter addresses the next stage of autonomous Open RAN management: reasoning about these interactions and deciding whether xApp control requests should be accepted, rejected, or constrained under the current network environment.

5.1 Introduction

The ORAN paradigm promotes a disaggregated, virtualized, and multi-vendor mobile network architecture, enabling flexibility and innovation through standardized open interfaces. Central to this design are the RAN Intelligent Controllers (RICs)—non-Real-Time (non-RT) and near-Real-Time (near-RT)—which host independently developed software applications (rApps and xApps) for network optimization. While openness drives innovation, it can also introduce unintended interactions or conflicts among applications that share a common base of network resources.

Recent efforts in conflict detection primarily rely on statistical methods. For instance, the state-of-the-art PACIFISTA method [30] uses empirical cumulative distribution functions (ECDFs) to quantify conflict severity and decides whether to allow the coexistence of applications before deployment. Another approach [9] uses statistical correlation to predict KPI impacts from configuration changes, preemptively deciding whether to apply the configuration changes. Despite these advances, several fundamental limitations persist. First, existing methods lack fine-grained identification of conflicting parameters, forcing coarse-grained decisions where entire change sets are accepted or rejected wholesale. Second, their reliance on static historical data makes them ineffective in unseen environments.

Fig. 5.1a elucidates such limitations by experimenting with PACIFISTA [30]. Two xApps control physical resource block (PRB) allocation with different objectives: xApp A maximizes eMBB throughput, while xApp B minimizes URLLC delay. We compare the ECDFs (y-axis) of eMBB PRB allocations (x-axis) across two environments with *different cell loads and traffic patterns*. In one environment (left), their ECDFs are similar, indicating compatible behavior. In the other (right), they diverge significantly, showing that xApp coexistence breaks under environmental changes, as also noted in [30]. PACIFISTA addresses this by collecting conditioned statistical profiles for each environment, which allows it to predict the impact of resource usage with potential conflicts. Fig. 5.1b shows PACIFISTA’s prediction of throughput under network environment dynamics. We trained a model using data from different environments with varying parameters (e.g., TxPower, traffic load). The left and right subfigures show the prediction results by including and excluding one particular (unseen) environment, respectively. We observe poor generalization to unseen conditions, which also hampers PACIFISTA’s ability of conflict detection under environmental changes. These experiments motivate the need for a framework that can adapt to new environments without requiring retraining or collecting statistical profiles for each xApp.

In response, we introduce Open RAN Conflict Agents (ORCA), a framework to employ LLM agents for conflict detection. To enable fine-grained conflict identification, ORCA estimates

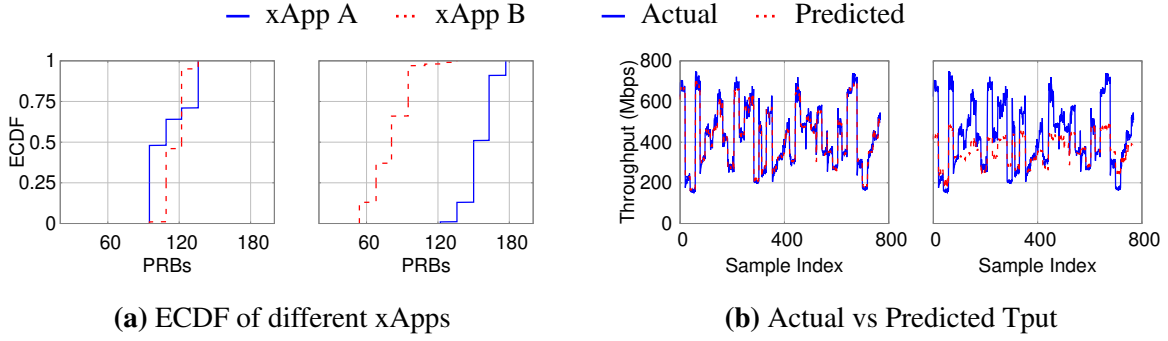


Figure 5.1. Conflict detection methods do not hold in unseen environments.

feature importance, defined as the sensitivity of a KPI to variations in individual control parameters or network metrics. ORCA computes this using SHapley Additive exPlanations (SHAP) [92], a well established method for estimating feature importance. However, directly applying SHAP to the full set of ORAN parameters is impractical due to high dimensionality of the parameter space. ORCA addresses this via a *two-stage pre-selector*: an autoencoder removes irrelevant features, and an LLM agent recovers those critical for KPI prediction. This pre-selector focuses SHAP on the most relevant subset for efficient and accurate feature attribution.

The second challenge, adapting to dynamic and evolving network environments, is considerably more complex. While one might rerun SHAP analysis in a new environment, this would require a large and diverse volume of log data to produce stable and meaningful feature attributions, which can take hours to days to collect and can easily expire as network condition changes. ORCA addresses this limitation by employing generative LLM agents to infer feature importance in unseen environments without requiring such extensive data. Given SHAP values (feature importance) from a known environment, ORCA guides the LLM agents to reason about how feature importance might shift under new conditions. Unfortunately, the LLM often produces heuristic rather than quantitatively grounded outputs. ORCA overcomes this limitation through a *β -Reasoning Cascade*, a structured process that supports symbolic reasoning for deriving robust and interpretable feature importance.

Another challenge is the risk of hallucination, which may result from missing domain

knowledge or, even when the knowledge exists, from the LLM’s internal reasoning errors or faulty synthesis. To mitigate hallucinations and maintain stability, we propose Segmented Active RAG (SA-RAG)—a dynamic retrieval method that adapts queries as reasoning evolves and targets domain-specific knowledge sources such as 3GPP specifications, keeping the LLM grounded in context-relevant information throughout the process.

We carefully constructed 32 categories of network environments to capture a broad and representative range of variations across key factors such as control parameters, traffic patterns, cell load, channel conditions, and UE mobility. While not exhaustive, this set reflects meaningful diversity across realistic operational conditions, providing strong coverage for evaluating the generalization capability of our framework. We compared ORCA with two baselines: Extrapolation, and Pure LLM, by using SHAP’s results as ground-truth. We found that ORCA achieved the most accurate estimates, reducing cosine error by 60.7% over Extrapolation and NRMSE by 65.1% over Pure LLM. We also tested scenarios involving resource conflicts, where we compared ORCA with PACIFISTA [30] and a prediction-based method inspired by [9]. Our experiments indicate that ORCA enables fine-grained, context-aware control by selectively rejecting actions that threaten critical KPIs.

This chapter makes the following contributions:

- We propose ORCA, a fine-grained ORAN conflict detection framework designed for unseen and dynamically changing network environments.
- We design a two-stage pre-selector scheme, which combines dimension reduction techniques with LLM agents for generalizable conflict analysis. In addition, we present the β -Reasoning Cascade and SA-RAG, two novel mechanisms that empower LLM agents to perform accurate multi-step reasoning, by effectively retrieving ORAN specific domain knowledge.
- We verify the effectiveness of ORCA across diverse network environments, in comparison with state-of-the-art baselines.

5.2 Background and Problem Setting

5.2.1 Conflicts in ORAN App Configuration

Consider an ORAN deployment consisting of a set L xApps: $\mathcal{X} = \{x_1, x_2, \dots, x_L\}$, M control parameters: $\mathcal{P} = \{p_1, p_2, \dots, p_M\}$, and N KPIs: $\mathcal{K} = \{k_1, k_2, \dots, k_N\}$. Different xApps often target diverse objectives (i.e., different KPIs) and control different sets of parameters, but fundamentally they are sharing a common pool of network resources. So their operations can unintentionally interfere with each other. The ORAN Alliance specification [1] classifies such *conflicts* among xApps into three categories.

(i) **Direct conflict:** Occurs when two or more xApps control the same parameters or compete for the same RAN resources.

(ii) **Indirect conflict:** Occurs when multiple xApps control different parameters to optimize the same metric.

(iii) **Implicit conflict:** Occurs when multiple xApps control different parameters to optimize different metrics, yet their actions inadvertently interfere with each other.

ORCA mitigates all conflict types by detecting potential parameter conflicts before xApps apply them.

5.2.2 SHAP Analysis for Feature Importance

Understanding the contribution of parameters to KPIs is critical for fine-grained conflict detection. The most rigorous way to quantify this contribution is to perturb the target parameter and measure its impact on the KPI across all possible combinations of other parameters. This forms the core principle of the original Shapley value framework [136]. Let $\mathcal{F}$ denote the selected feature set (parameter set). The feature i 's importance ϕ_i is formally defined as:

$$\phi_i = \sum_{S \subseteq \mathcal{F} \setminus \{i\}} \frac{|S|!(|\mathcal{F}| - |S| - 1)!}{|\mathcal{F}|!} [f(S \cup \{i\}) - f(S)], \quad (5.1)$$

where f denotes the expected output of the surrogate model. Here, feature importance is interpreted as a weighted average of the marginal contributions of feature i across all possible subsets $S \subseteq \mathcal{F}$. However, computing exact Shapley values is infeasible in practice for two reasons. First, without a trained model, perturbing parameters requires applying each change to a real system or simulator to observe its effect on the KPI, which is extremely costly. Second, the number of possible parameter subsets grow exponentially with the number of parameters, making exhaustive evaluation impractical.

SHAP [92] addresses both challenges by using a trained model to efficiently approximate Shapley values and by applying sampling-based methods like Kernel SHAP to reduce computational complexity. We apply SHAP to quantify how changes in parameters affect KPIs. In line with SHAP terminology, we treat parameters as “features”, and refer to the degree of contribution as “feature importance”. This enables fine-grained estimation of individual parameter’s impact, critical for identifying potential conflicts between xApps.

However, SHAP validity fundamentally depends on model accuracy. SHAP values reliably reflect feature importance within the environment the model was trained on, where the environment refers to external operating conditions such as traffic load, channel condition, and mobility status, but may become unreliable under unseen conditions. Consequently, when network environments change, reusing SHAP values from prior conditions can lead to incorrect feature impact assessments. Maintaining reliable conflict detection in dynamic environments requires generating or inferring updated SHAP values that reflect the current context.

5.3 System Design

5.3.1 ORCA Architecture

As illustrated in Fig. 5.2, ORCA consists of four key modules: the Environment Encoder (EE), the Feature Importance Extractor (FIE), the Feature Importance Reasoner (FIR), and the Conflict Detection and Mitigation (CD&M) module. EE processes raw contextual features

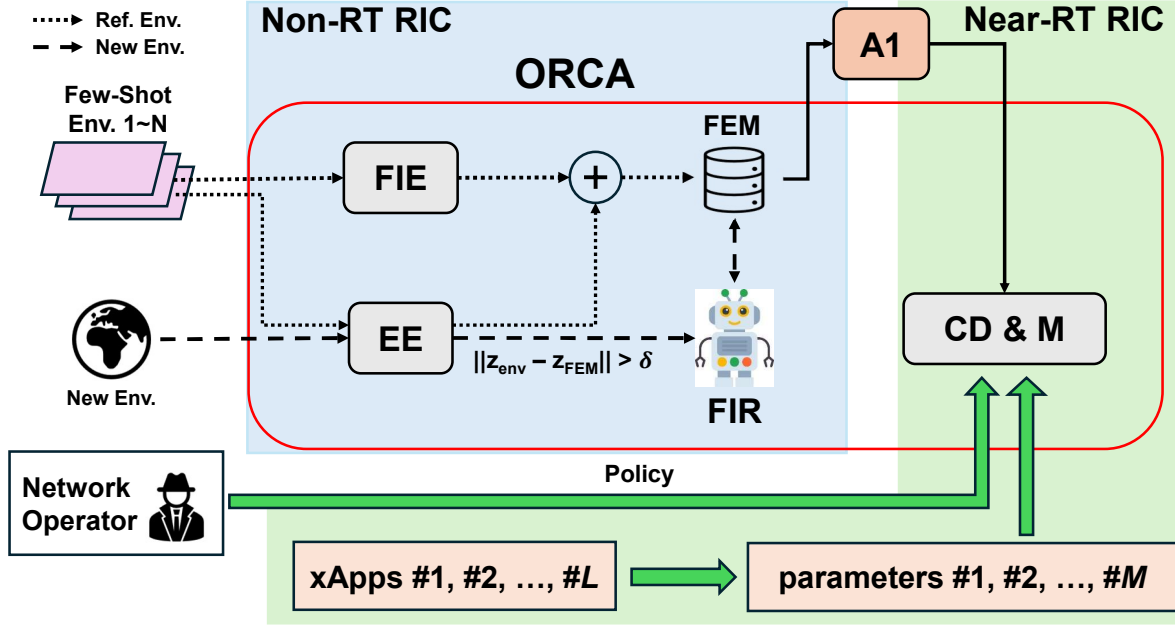


Figure 5.2. Overall architecture of ORCA, consisting of FIE, EE, FIR, and CD&M.

and transforms them into a compact environment embedding. FIE applies SHAP analysis to extract each feature’s importance, while FIR uses LLM agents to derive updated feature importance. Finally, the CD&M module detects and mitigates conflicts based on the resulting feature importance.

ORCA first ingests few-shot ORAN logs from known environments (“shot” refers to the number of environments), which we refer to as *reference environments*. The logs include control parameters set by xApps or network operator (e.g., TxPower), metrics (e.g., SNR), and KPIs (e.g., throughput). Since conflict detection is performed by the network operator, we reasonably assume that such logs are readily available.

EE and FIE are applied to each reference environment. EE generates a compact embedding of the environment context, while FIE performs SHAP analysis to compute the importance of each control parameter. Both the embeddings and the corresponding feature importance vectors are stored in the Few-Shot Environment Memory (FEM) and passed to the CD&M module via the A1 interface [108]. To detect environmental shifts, EE is executed periodically at a fixed interval T , producing an embedding vector $\mathbf{z}_{\text{env}}$ for the current environment. This embedding

is compared against embeddings $\mathbf{z}_{\text{FEM}}$ stored in the FEM. If $\|\mathbf{z}_{\text{env}} - \mathbf{z}_{\text{FEM}}\| > \delta$, where δ is a predefined threshold, ORCA infers that a previously unseen environment has emerged.

Upon detecting a new environment, FIR is invoked to derive updated SHAP values that reflect the new network conditions. These updated feature importance vectors are then stored in the FEM and passed to the CD&M module via the A1 interface. The CD&M module continuously monitors control parameter change requests from xApps, predicts their impact on KPIs using the updated SHAP values, and decides whether to approve or reject each request.

The overall system design reflects practical deployment considerations within ORAN. EE, FIE, and FIR are designed to reside in the non-RT RIC, as they rely on historical log data and do not require low-latency execution. In contrast, the CD&M module, which performs live conflict mitigation, is deployed in the near-RT RIC to satisfy latency constraints. This division of responsibilities aligns well with execution times. The FIR takes approximately 33 seconds to compute new feature importance values, which is acceptable given that environmental changes in cellular networks typically occur over timescales of hours. Meanwhile, the CD&M module performs only lightweight arithmetic over a small number of features, enabling decision times below one millisecond, meeting near-RT latency needs.

5.3.2 Environment Encoder (EE)

The network environment is dynamic and evolves over time. To capture this variability, we employ an EE that maps the current environment into a compact embedding summarizing key contextual attributes. The encoder processes features such as cell-level traffic load, channel condition, slice configuration, and mobility status to produce a representation of the network state. These features are typically accessible to the network operator through standard O1, with cell-level metrics changing slowly enough that reporting intervals of tens of seconds provide sufficient context for environment detection. This embedding serves as contextual grounding for downstream reasoning tasks such as β -propagation, enabling generalization to previously unseen environments.

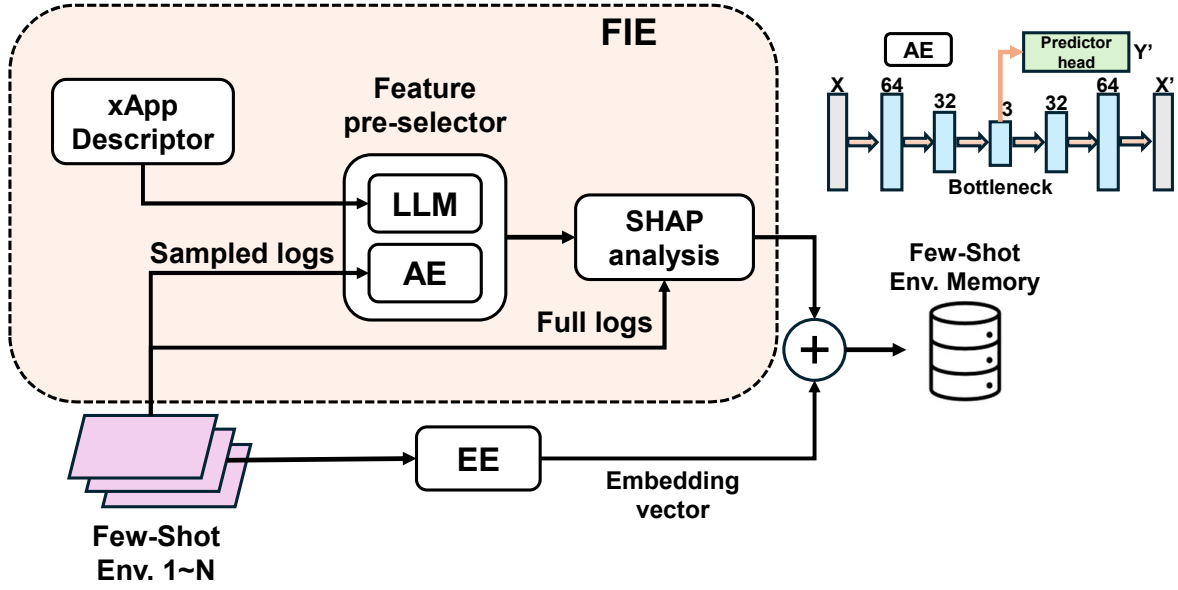


Figure 5.3. Detailed structure of the FIE.

5.3.3 Feature Importance Extractor (FIE)

Accurate feature-importance extraction is vital for conflict management, as it shows how each parameter influences KPIs. FIE employs SHAP [92] to extract feature importance in the reference environments. However, applying SHAP directly to ORAN data, which often contain hundreds of features, faces two major challenges. First, SHAP’s computational complexity grows exponentially with input size: irrelevant features force the explainer to evaluate many unnecessary coalitions, severely slowing analysis. Second, training a reliable surrogate model on high-dimensional data with numerous irrelevant inputs risks overfitting or demands substantially more samples; a poorly performing model yields misleading SHAP values.

To address these challenges, we introduce a feature pre-selector combining a supervised autoencoder (AE) and an LLM agent. The AE is trained on a sampled subset of logs (e.g., 20% of the ORAN logs) to rapidly filter out irrelevant features prior to SHAP analysis. We empirically observed that unsupervised training leads to inaccurate feature importance ordering, as the latent representation is optimized solely for input reconstruction rather than for preserving KPI-relevant feature relationships. Therefore, we augment the AE with a KPI-prediction head to guide the

latent representation toward KPI-relevant feature relationships. As illustrated in Fig. 5.3, the encoder consists of three fully connected layers and a decoder mirrors this structure in reverse to reconstruct the original input vector. In parallel, a predictor head estimates the KPI directly from the bottleneck. Training minimizes a joint loss:

$$L(\theta) = \lambda \|\hat{\mathbf{x}} - \mathbf{x}\|_2^2 + (1 - \lambda) (\hat{y} - y)^2 + \alpha \sum_{i,j} |w_{ij}^{(7)}|,$$

where $\mathbf{x} \in \mathbb{R}^D$ is the input feature vector and $\hat{\mathbf{x}}$ is its reconstruction. The scalar y is the ground truth KPI (e.g., throughput), and $\hat{y}$ is the predicted KPI. The matrix $W^{(7)}$ denotes the weight matrix of the first linear layer in the predictor head. The term $\lambda = 0.3$ balances the reconstruction loss and KPI prediction loss, while $\alpha = 10^{-4}$ controls the sparsity regularization. The total trainable parameters of the model are denoted by θ . The reconstruction loss encourages the model to preserve features necessary for reconstructing the overall input structure, while the KPI prediction loss focuses the model on features that directly influence the target KPI.

During training, features that contribute to neither reconstruction nor KPI prediction objectives are naturally suppressed by the AE. After training, we compute the gradient of the loss with respect to each input feature to quantify their influence. Features whose gradient magnitudes fall below a relative threshold (e.g., 10% of the top-ranked feature) are discarded. The final output of this stage is a reduced set of selected features, prioritized for their joint contribution to both input reconstruction and KPI prediction. However, this process may inadvertently eliminate infrequent yet important features.

To address this limitation, we incorporate an LLM agent guided by the developer-provided xApp descriptors, which specify the controllable parameters and the targeted KPIs. The LLM uses these descriptors as structured constraints and applies domain knowledge to identify parameters that have a documented or plausible causal relationship with the target KPIs, even if they are underrepresented in the sampled data. It cross-references the xApp’s targets with the full parameter set to recover important parameters that may have been excluded earlier, ensuring that

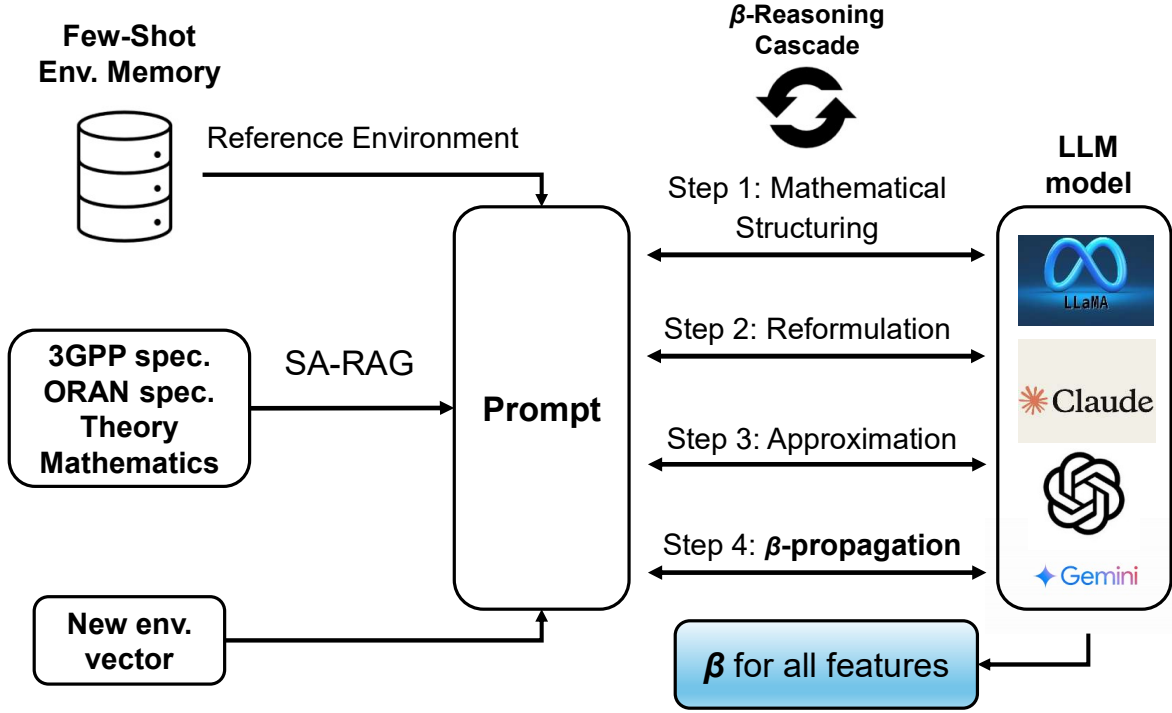


Figure 5.4. FIR module infers new importance using β -Reasoning Cascade and external knowledge retrieved via SA-RAG.

critical features are not overlooked. This ensures that all relevant parameter–KPI relationships are retained for SHAP analysis.

After pre-selection, we train a surrogate model using the reduced feature set and perform SHAP analysis to compute each feature’s contribution to the KPI. Since exact computation is intractable (see §5.2.2), we adopt approximation methods such as Kernel SHAP [92] using standard SHAP libraries. SHAP is applied to both control parameters and network metrics. The resulting values, along with the environment embedding, are stored in the FEM.

5.3.4 Feature Importance Reasoner (FIR)

Although we have feature importance of reference environment, deriving SHAP values for a new, unseen environment remains challenging. Conventional methods like extrapolation, regression, or ML prediction are inadequate: extrapolation fails under non-linear dynamics, and regression or ML requires extensive SHAP supervision across diverse conditions. A natural

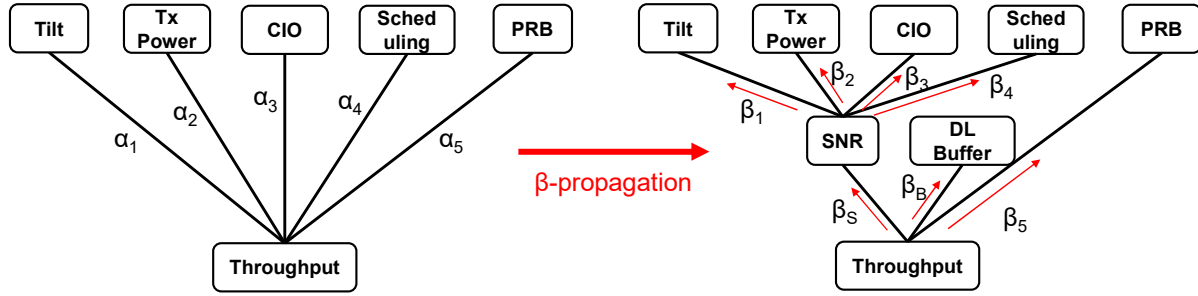


Figure 5.5. Feature importance is derived by SHAP analysis (left), while new feature importance is obtained via β -propagation (right). β values are propagated through intermediate metrics.

alternative is principled mathematical reasoning, as a domain expert might do.

Suppose the target KPI is throughput and the features are TxPower, PRB, DL_buffer, and antenna tilt. Since throughput cannot be directly expressed as a function of these parameters, experts often use analytic formula such as Shannon’s capacity and recursively reason over lower-level features. Feature importance is then estimated using partial derivatives to assess KPI sensitivity. When closed-form models are unavailable or too complex, experts rely on approximations based on feature distributions. However, in systems like ORAN, this process becomes intractable due to high dimensionality and implicit dependencies. Even if an approximation is available, validating its applicability under distributional shift remains a major challenge. We address this by delegating the structured expert reasoning process to an LLM agent, implemented through the β -Reasoning Cascade.

β -Reasoning Cascade

One might expect a powerful LLM to derive feature importance through symbolic reasoning by manipulating mathematical relationships. However, without structured guidance, LLMs often fail to perform this reliably. To address this, FIR uses a structured reasoning process, the ***β -Reasoning Cascade***. Rather than following a rigid pipeline, FIR leverages the LLM agent’s reasoning capabilities to dynamically construct, transform, and validate logic at each stage. Unlike human experts, who struggle to repeat symbolic analysis across many environments, the

LLM agent efficiently scales this reasoning by combining symbolic manipulation with domain-informed generalization. As shown in Fig. 5.4, the reasoning framework is composed of 4 steps. At each step, the LLM agent leverages its reasoning based on prompt, domain specific and pretrained knowledge. The necessary domain specific knowledge is retrieved via SA-RAG (see §5.3.4).

Step 1: Mathematical Structuring. Each KPI's definition is provided to the LLM agent via prompt. For example:

- *Throughput*: total amount of user data successfully transmitted to all UEs in a cell per second (Mbps)
- *RLFs*: events where SNR remains below a defined threshold γ for a specified time duration T

Together with the given information and knowledge, the LLM agent formulates a symbolic expression of the KPI as a function of abstract system variables, such as

$$\text{Throughput} = \min(\text{System Capacity}, \text{Traffic Demand}).$$

. In this step, the primary source of domain knowledge is often communication theory, as observed in our experiments. However, for certain stochastic KPIs (e.g., RLFs), LLM agent determines that no deterministic symbolic form exists and therefore skips Step 1, proceeding directly to Step 2.

Step 2: Reformulation. The symbolic expression is then rewritten in terms of measurable parameters, including network metrics and control parameters. For example, system capacity may be expressed as a function of Avg_SNR and PRB_num, while traffic demand may be approximated using DL_buffer. For KPIs where Step 1 is skipped, the LLM proceeds directly to Step 2, identifying dominant measurable parameters that serve as the basis for probabilistic modeling. In this step, the primary source of domain knowledge is typically 3GPP and ORAN specifications.

Step 3: Approximation. Although Step 2 yields a reformulated expression in terms of measurable parameters, the resulting equation is often not in closed form. This is particularly common for network KPIs, which are frequently expressed as probabilities or expectations. Moreover, partial derivatives can be applied to the expression to assess the sensitivity of the KPI to different control parameters. This often results in intractable or non-analytic terms. In such cases, mathematical approximation techniques are employed to convert the expression into a closed form. In this step, the primary source of domain knowledge is mathematics.

Step 4: β -Propagation. We use the importance shift ratio, denoted by β , instead of directly deriving the absolute SHAP values denoted by ϕ for the new environment. β describes how the importance of a feature changes relative to a reference environment when evaluated in a new environment. This design choice is motivated by the fact that approximations are applied at multiple stages of the reasoning process—particularly in step 2 and 3, where the KPI is reformulated and simplified into closed-form expressions. If we attempt to compute the absolute SHAP values directly in the new environment, the accumulated approximation error can lead to inaccuracies. In contrast, computing the importance shift ratio β helps mitigate this issue: since both the numerator and denominator undergo the same symbolic transformations, approximation errors tend to “cancel out” in the ratio. Furthermore, to enable robust reasoning within the LLM agent, we use normalized SHAP values α instead of raw values ϕ , as ϕ is often proportional to the KPI. While SHAP is not mathematically equivalent to a partial derivative, it can be conceptually approximated, as both quantify the local sensitivity of the model output with respect to a given input feature. In particular, the magnitude of the partial derivative serves as a reasonable proxy for the normalized SHAP value:

$$\alpha_i^{\text{ref}} = \frac{|\phi_i^{\text{ref}}|}{\sum_{j \in \mathcal{F}} |\phi_j^{\text{ref}}|} \approx \frac{\left| \frac{\partial \text{KPI}}{\partial i} \right|_{\text{ref}}}{\sum_{j \in \mathcal{F}} \left| \frac{\partial \text{KPI}}{\partial j} \right|_{\text{ref}}} \quad (5.2)$$

$$\beta_i = \frac{\alpha_i^{\text{new}}}{\alpha_i^{\text{ref}}} \approx \frac{\left| \frac{\partial \text{KPI}}{\partial i} \right|_{\text{new}} / \sum_{j \in \mathcal{F}} \left| \frac{\partial \text{KPI}}{\partial j} \right|_{\text{new}}}{\left| \frac{\partial \text{KPI}}{\partial i} \right|_{\text{ref}} / \sum_{j \in \mathcal{F}} \left| \frac{\partial \text{KPI}}{\partial j} \right|_{\text{ref}}} \quad (5.3)$$

In the example of throughput, β for features such as PRB can be directly derived using Eq. (5.3), since PRB appears explicitly in the throughput formulation. However, other features such as TxPower, or antenna tilt, influence throughput indirectly through intermediate variables like SNR. In such cases, their impact is embedded within the shift ratio of the intermediate metric. For instance, β_{TxPower} is implicitly captured by β_{SNR} , and must be resolved recursively to recover the original feature’s contribution. As illustrated in Fig. 5.5, this recursive tracing of influence from intermediate variables back to original features is referred to as β -propagation. The actual values of β are conditioned on the current environment, and are computed based on the low-dimensional embedding produced by the EE. We emphasize again that, although the reasoning process follows a structured four-step framework, each step is autonomously performed by the LLM agent, not through fixed logic. The degree of reasoning varies: Steps 1 requires light symbolic mapping, whereas Step 2, Step 3, and 4 demand deeper inference, including approximation and distribution-aware validation.

Once the β_i values for all features are reasoned by FIR, we compute the new normalized feature importance by applying the shift ratio, $\alpha_i^{\text{new}} = \beta_i \cdot \alpha_i^{\text{ref}}$. We then renormalize to ensure that the normalized importance values sum to one: $\tilde{\alpha}_i^{\text{new}} = \frac{\alpha_i^{\text{new}}}{\sum_{j \in \mathcal{F}} \alpha_j^{\text{new}}}$. Finally, we rescale the renormalized importance values to obtain the absolute SHAP values by multiplying with the KPI value of the new environment:

$$|\phi_i^{\text{new}}| = \tilde{\alpha}_i^{\text{new}} \cdot \left(\frac{\text{KPI}^{\text{new}}}{\text{KPI}^{\text{ref}}} \right) \cdot \sum_j |\phi_j^{\text{ref}}| \quad (5.4)$$

Note on KPI applicability: reasoning pipeline is particularly effective for KPIs like throughput and delay, where domain knowledge aligns well with observable relationships in the data, enabling the model to reason effectively about control parameter impact. For more stochastic

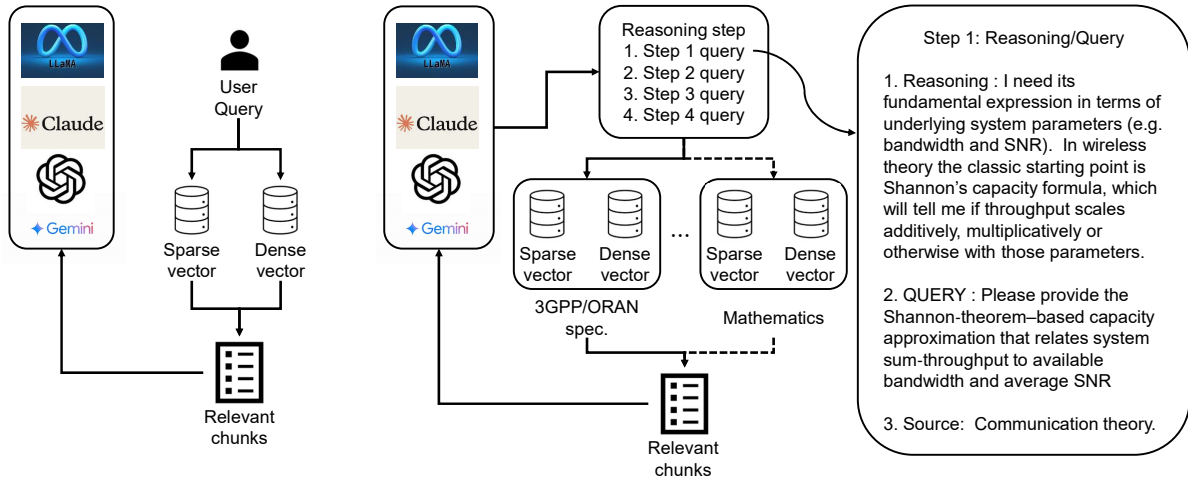


Figure 5.6. Conventional hybrid RAG (left) vs SA-RAG (right).

or higher-layer KPIs (e.g., reliability or user satisfaction), additional domain modeling or assumptions may be required to capture indirect dependencies.

Knowledge-Augmented Reasoning

Another challenge in reasoning feature importance is hallucination, where an LLM generates incorrect or unfounded outputs. To mitigate such risks, we provide domain knowledge through a hybrid RAG framework combining sparse and dense retrieval. Sparse retrieval locates exact keyword matches while dense retrieval captures semantic similarity and retrieves relevant information even when the query and document use different terminologies or phrasing. By combining both, hybrid RAG ensures comprehensive and contextually accurate knowledge grounding for the LLM. However, conventional hybrid RAG methods are passive, meaning that retrieval queries are either manually written or generated from the initial prompt without adapting to intermediate reasoning needs or environmental context. As a result, they often retrieve loosely relevant information, making them insufficient for multi-step reasoning tasks.

To address this limitation, we propose Segmented Active RAG (SA-RAG), a reasoning-aware RAG framework in which the LLM dynamically generates retrieval queries during the reasoning process. As shown in Fig. 5.6, the LLM formulates queries at each stage based on

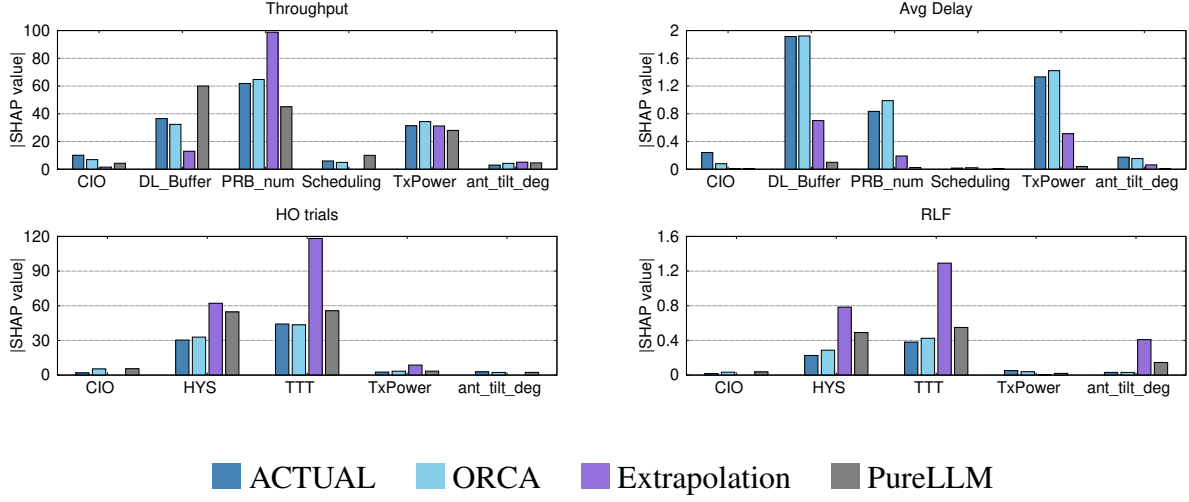


Figure 5.7. Comparison of derived feature importance against ground-truth (ACTUAL) values. Closer alignment indicates higher estimation accuracy.

the current reasoning objective. For instance, it may first retrieve Shannon’s capacity formula to relate throughput, bandwidth, and SNR, then refine queries to focus on parameter behaviors or constraints. While the LLM may have learned such knowledge during pretraining, it can produce incorrect or inconsistent outputs due to reasoning errors. To mitigate this, our framework explicitly instructs the LLM to retrieve external domain knowledge via SA-RAG, enabling it to combine pretrained and retrieved knowledge for more accurate decisions.

During the β -Reasoning Cascade, relevant knowledge spans 3GPP and ORAN specifications, mathematics, communication theory, and system-level information. Mixing these domains in a single retrieval pool often leads to irrelevant results. Thus, SA-RAG segments knowledge sources and routes each query to the most appropriate domain, improving retrieval precision.

5.3.5 Conflict Detection and Mitigation (CD&M)

As mentioned earlier, a major advantage of extracting feature importance is that it enables network operators to make policy-driven decisions by estimating the impact of feature changes on KPIs. This process begins with quantifying the expected KPI change resulting from a proposed parameter adjustment. We begin by interpreting the SHAP value ϕ_i as the expected absolute

change in the KPI due to perturbing feature i : $\phi_i \approx \mathbb{E}[|\Delta \text{KPI}_i|]$. Assuming local linearity of the KPI with respect to feature i , $\Delta \text{KPI}_i \approx \frac{\partial \text{KPI}}{\partial i} \cdot (i - i_{\text{baseline}})$, where i_{baseline} denotes the baseline value of feature i in the reference environment. It represents the operating point at which the SHAP value ϕ_i was originally computed. Taking the expectation of the absolute value, we obtain $\phi_i \approx \left| \frac{\partial \text{KPI}}{\partial i} \right| \cdot \varepsilon_i$ where $\varepsilon_i := \mathbb{E}[|i - i_{\text{baseline}}|]$ is the expected magnitude of deviation from the average value of feature i in the reference environment. We use ε_i from the reference environment since the new SHAP value is derived from the reference distribution of feature i . Now, given a specific proposed change Δi in feature i , the expected impact on the KPI is:

$$\Delta \text{KPI}_i \approx \frac{\phi_i}{\varepsilon_i} \cdot \Delta i \quad (5.5)$$

When parameter changes are requested by xApps, the impact on the KPI can be estimated using Eq. (5.5). A conflict is declared if $\Delta \text{KPI} > \delta$, where the δ is a predefined threshold by the network operator. Depending on the predefined network operator policy and prioritization rules, the proposed parameter change is either accepted or rejected accordingly.

Table 5.1. Overview of Applications and Controlled Parameters

Application	Objective	Control Time Scale (slots)	Controlled Parameters
Throughput maximization (TM)	Maximize throughput	200	PRBs, TxPower
Energy Saving (ES)	Reduce energy consumption	200	TxPower
Delay Minimization (DM)	Minimize end-to-end delay	200	PRBs, scheduling
Handover Management (HM)	Reduce number of handovers	500	TTT, Hysteresis
Mobility Robustness Optimization (MRO)	Reduce RLFs (radio link failures)	500	Antenna Tilt, CIO

5.4 Evaluation

5.4.1 Simulation Setup

ORCA is powered by an LLM agent based on o4-mini [117], serving as the reasoning engine for conflict detection. We evaluate ORCA using a MATLAB-based 5G simulator with the 5G Toolbox. While this is a simple simulator, it provides sufficient flexibility to model key aspects of RAN slicing, including per-slice PRB allocation, traffic generation, and KPI logging. The setup includes two gNBs and up to 100 UEs, operating over 100 MHz (273 PRBs)

partitioned between eMBB and URLLC slices. ORCA treats cross-cell influences as part of the environment; for example, if an xApp controls only cell A, we monitor the KPIs of cell A, whereas if it controls both A and B, we monitor the combined KPIs of both cells. Although our setup includes only two gNBs, this design naturally extends to scenarios with a larger number of cells. Moreover, we define only one eMBB slice and one URLLC slice, but each slice is not limited to just a single flow. Instead, it can represent a group of multiple traffic flows that have similar service requirements. We measure four KPIs: eMBB downlink throughput, URLLC average delay, number of handovers (HOs), and radio link failures (RLFs). UE mobility includes back-and-forth movement, random direction changes, and stationary behavior. URLLC traffic follows a Poisson process (1–5 packets/s), while eMBB users generate continuous traffic to simulate high throughput.

We applied Kernel SHAP [92] on models including XGBoost [26] and MLP to derive ground truth feature importance. As benchmarks, we used extrapolation and a pure LLM agent. Regression and ML-based predictions require large labeled datasets across diverse environments, which contradicts our goal of generalization to unseen conditions.

We quantify accuracy using cosine error and normalized root mean square error (NRMSE). Cosine error captures the directional similarity of predicted and ground-truth importance vectors, while NRMSE measures the magnitude difference. Together, they evaluate both relative and absolute accuracy. We also evaluate ORCA’s conflict management with various xApps as indicated in Table 5.1. We benchmark against PACIFISTA [30] and a prediction-based method, inspired by [9].

5.4.2 Results

Our evaluation consists of two parts: (i) the accuracy of feature importance, measured by NRMSE and cosine error against ground truth, and (ii) conflict detection and mitigation, evaluated through KPI performance. The latter inherently involves trade-offs across KPIs, which vary depending on the network operator’s priorities. *Therefore, rather than merely presenting*

statistical KPIs, we showcase why ORCA makes certain decisions under conflicting control requests. Since improving all KPIs simultaneously is infeasible, our focus is on explainable, priority-aware decision-making.

Accuracy of feature importance

We create an additional set of 32 network environments by systematically perturbing control parameters and environmental factors. From a single reference environment, both ORCA and the pure LLM agent are provided with the true feature importance in a one-shot manner. In contrast, the extrapolation-based method requires at least two known environments to function, so we provide it with additional ground-truth importance values accordingly. Fig. 5.7 presents inferred SHAP values for each parameter in one unseen environment. As can be visually confirmed, ORCA most closely matches the actual feature importance. For all 32 environments, each method generates its own feature importance estimates, which are compared against the ground truth. As shown in Fig. 5.8 and Table 5.2, ORCA achieves the lowest error across all KPIs, outperforming the best baseline by 60.7% in cosine error and 65.1% in NRMSE.

Conflict Detection and Mitigation

We first consider a conflict scenario between the ES and TM xApps. The ES xApp requests a reduction in TxPower to conserve energy. In Fig. 5.9a, all methods accept the change, as the expected impact on throughput is minimal in this environment. From our experiments, this situation tends to occur when eMBB traffic demand is relatively low or the cell is lightly loaded, resulting in sufficient PRB allocation to the eMBB slice. In such cases, a moderate TxPower reduction does not significantly impact throughput. However, as shown in Fig. 5.9b, in a new

Table 5.2. Average cosine error and NRMSE across all 32 unseen environments and 4 KPIs.

	Extrapolation	Pure LLM	ORCA(ours)
Cosine error	0.061	0.103	0.024 (↓60.7% vs EXT)
NRMSE	0.589	0.300	0.105 (↓65.1% vs LLM)

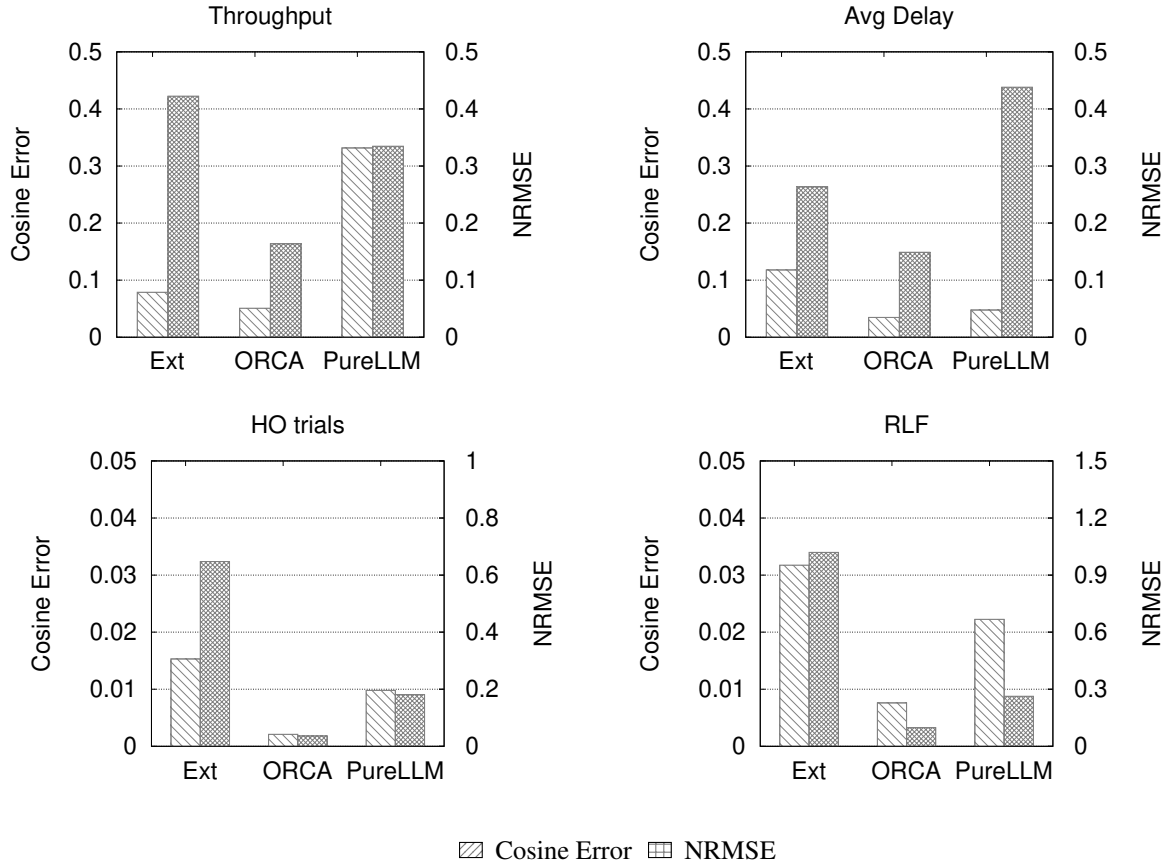


Figure 5.8. Comparison of cosine error and NRMSE. “Ext” denotes the extrapolation baseline.

environment with higher traffic load or near-full PRB utilization, ORCA correctly rejects the TxPower reduction, anticipating substantial throughput degradation under these conditions.

We then consider a conflict scenario between the HM and MRO functionalities. PACI-FISTA does not allow the coexistence of conflicting xApps in this environment; therefore, only prediction-based methods are considered for comparison. In Fig. 5.10a, both methods reject the parameter change, as increasing the hysteresis is predicted to raise the number of RLFs. However, in a different environment (Fig. 5.10b), while the prediction-based method still rejects the change, ORCA accepts it. As a result, ORCA reduces the number of ping-pong handovers while maintaining the RLF rate. We observed this behavior particularly when UE mobility was reduced, such that RLFs were less sensitive to hysteresis adjustments. In our simulation, we

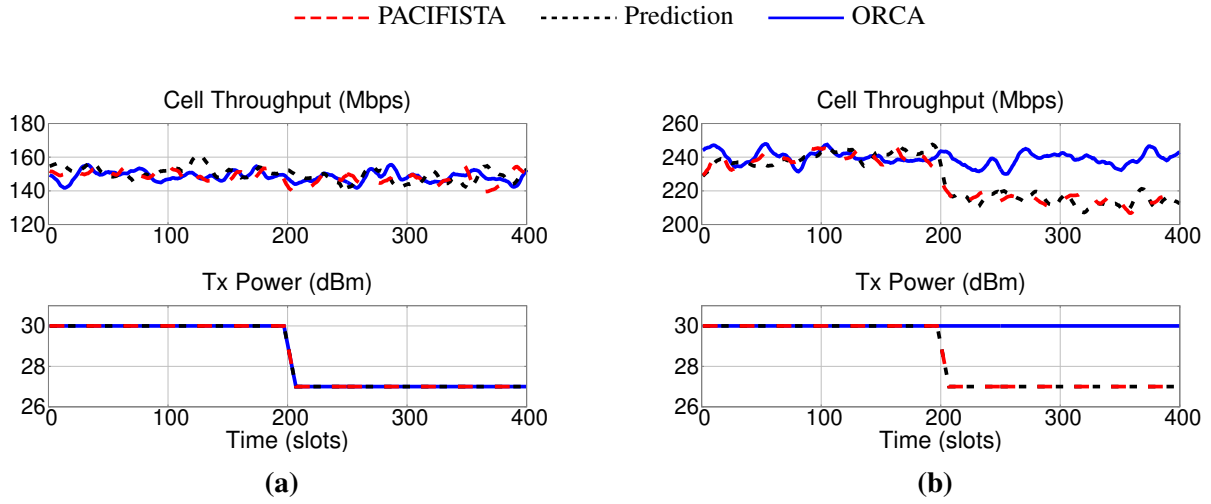


Figure 5.9. Conflict scenario between ES and TM.

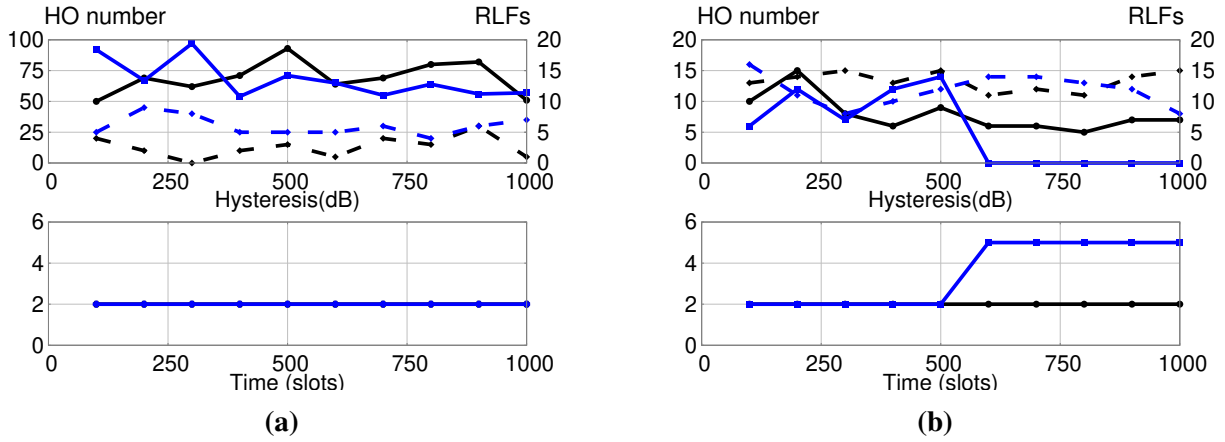


Figure 5.10. Conflict scenario between HM and MRO. In the top subplots, solid lines show the number of handovers and dashed lines show the number of RLFs.

assume the ORAN system does not directly observe UE mobility, but SNR measurements from sounding reference signals (SRS) are available. FIR leverages the variation of SNR to infer the feature importance of hysteresis under such conditions.

Finally, we examine a conflict scenario between the TM and DM xApps. The TM xApp requests an increase in both TxPower and PRBs to boost throughput. In Fig. 5.11a, all methods accept the change, as the expected impact on delay is relatively minor. We observed that when URLLC traffic is low and PRBs are sufficiently provisioned to URLLC, the DM xApp also tends to accept additional PRBs for eMBB without causing conflict. However, in a new environment (Fig. 5.11b), while PACIFISTA accepts the change, both the prediction-based method and ORCA detect potential conflicts. The prediction-based method rejects all requested changes, whereas ORCA makes a finer decision: it rejects the PRB increase because allocating more PRBs to eMBB reduces availability for URLLC, but accepts the TxPower increase.

Ablation study

Impact of the feature preselector To evaluate the impact of feature preselector, we compared two scenarios: one using only real control and metric features ($n = 8$), and another with 64 additional dummy features ($n = 72$). Since SHAP becomes infeasible at high dimensionality, so we sampled 2,000 subsets for both cases.¹ As shown in Fig. 5.12a, dummy features absorbed 11.5% of total SHAP importance despite having no causal relevance. While individual SHAP values were small, their cumulative effect was non-negligible. We note that the 11.5% dummy feature share reflects the variance inherent in the sampling-based Kernel SHAP method at the chosen evaluation budget, and is expected to decrease further with larger sample sizes. In contrast, the preselector removed dummy features, resulting in more stable and concentrated importance on key variables, confirming reduced noise and preventing SHAP dilution across

¹For $n = 8$, we intentionally oversampled beyond the full subset space of $2^8 = 256$ to match the experimental conditions of the large- n case. For $n = 72$, where the full subset space (2^{72}) is computationally infeasible, we heavily undersampled to 2,000 evaluations to enable direct runtime comparison.

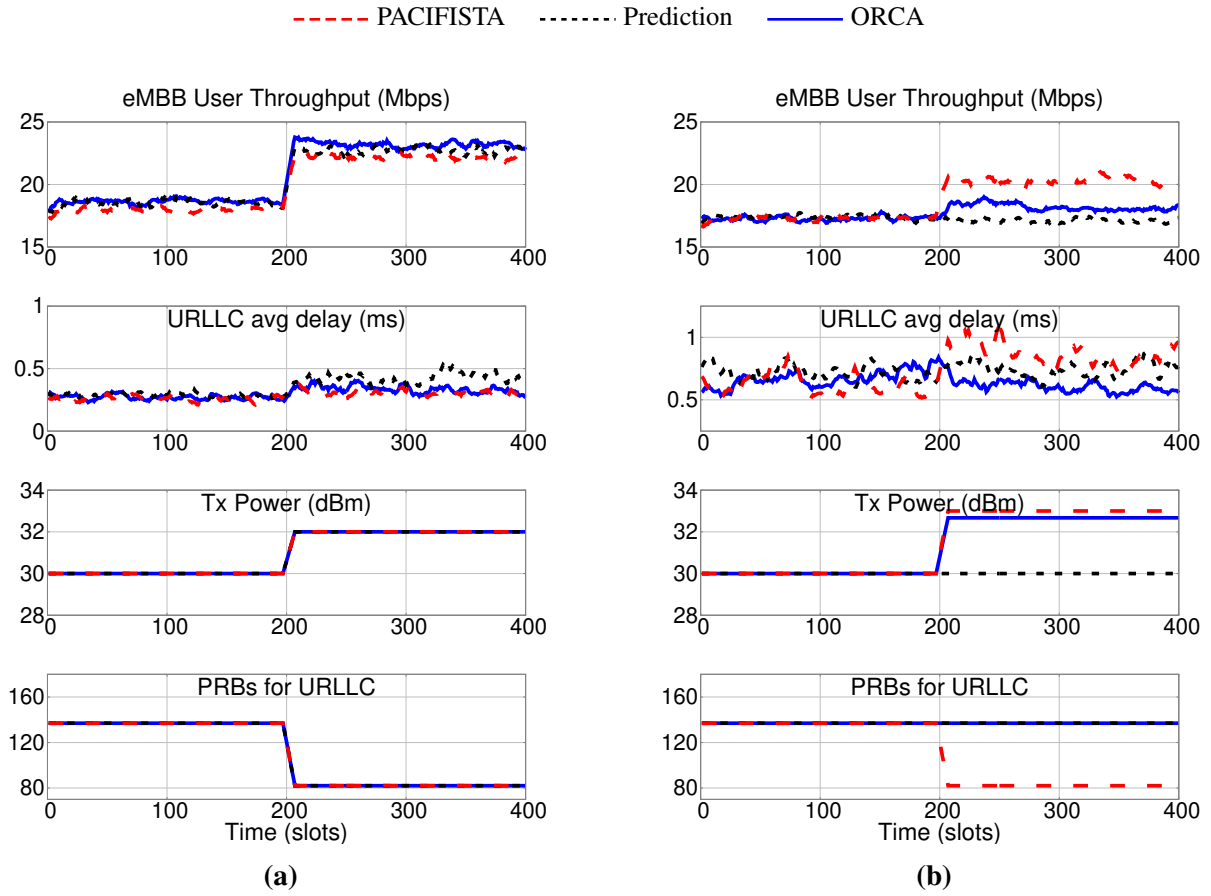


Figure 5.11. Conflict scenario between TM and DM.

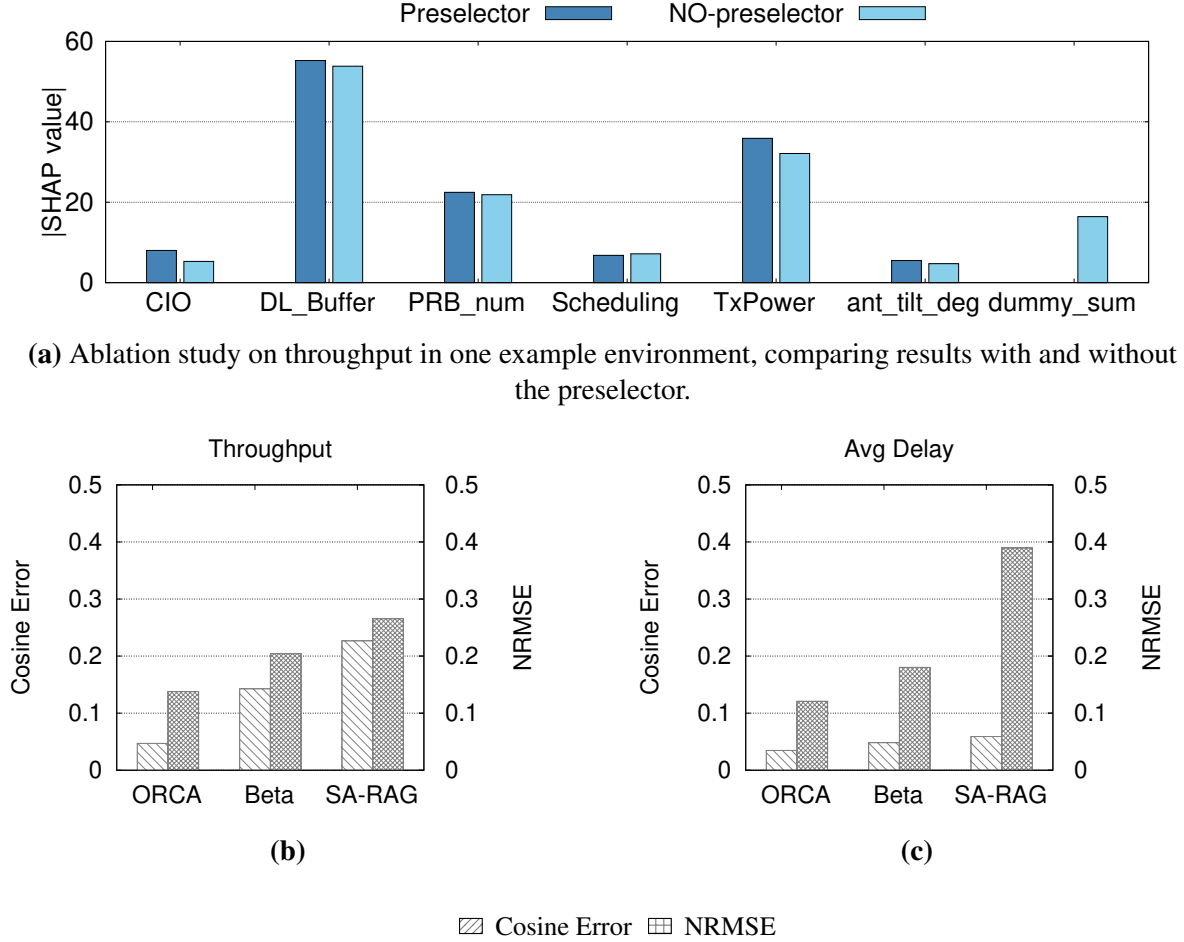


Figure 5.12. (b) and (c) show the accuracy of feature importance estimation for throughput and average delay, respectively. ORCA (full system) is compared against Beta (without SA-RAG) and SA-RAG (without the β -Reasoning Cascade).

irrelevant features. We then compared the SHAP computation time. Despite using the same number of subset evaluations, SHAP analysis took 576s for $n = 72$, compared to 28s for $n = 8$, both using only 800 log samples. While the absolute computation time depends on hardware, the relative increase of approximately 20 times remains consistent across hardware. We attribute this to the larger model size and the increased masking overhead associated with the larger feature set, both of which substantially raise the per-sample computation cost during SHAP analysis.

We also evaluated whether important features can be reinstated when they are discarded by AE. For RLFs, `ant_tilt_deg` was removed by the AE as part of the dimensionality reduction

β -Reasoning Cascade (part of step 3)

1. Probabilistic formulation

Let $C = \min(A, B)$. A perturbation in A affects C only if $A < B$; similarly for B . The importance of A is defined as

$$X := \mathbb{E}[|\partial C / \partial A|] = P(A < B)$$

$$Y := \mathbb{E}[|\partial C / \partial B|] = P(B < A)$$

Letting $D = A - B$,

$$P(A < B) = F_D(0), \quad P(B < A) = 1 - F_D(0),$$

2. Closed-form ratio approximation

To simplify, we approximate win probabilities using the means $a = \mathbb{E}[A]$, $b = \mathbb{E}[B]$. Define

$$\Delta\gamma := \frac{a'}{b'} \bigg/ \frac{a}{b}.$$

Then estimate

$$\beta_A := \frac{X'}{X} \approx \frac{1}{\Delta\gamma}, \quad \beta_B := \frac{Y'}{Y} \approx \Delta\gamma.$$

Thus, if A grows relative to B , B 's importance increases by that factor, and A 's decreases reciprocally.

Figure 5.13. Example from the β -Reasoning Cascade (step 3)

process. However, LLM agent rescues it. Although `ant_tilt_deg` exhibited low importance with SHAP analysis in one environment, `ant_tilt_deg` represents the primary actuator of the MRO xApp. This inclusion ensures completeness of the control-to-KPI mapping and preserves explainability for future scenarios and conflict detection.

Impact of FIR To validate the effectiveness of the proposed scheme in FIR, we conduct an experiment by disabling SA-RAG and β -Reasoning Cascade. As illustrated in Fig. 5.12, full ORCA reduces the cosine error and NRMSE by 47.0% and 32.5%, respectively, compared to when SA-RAG is disabled. Even without SA-RAG, the LLM agent can still produce feature-importance estimates; however, cosine error and NRMSE increase, indicating less reliable estimates. In contrast, when the β -Reasoning Cascade is removed, the performance closely resembles that of the pure LLM agent. This is primarily because, without β -reasoning, the actively generated queries are overly abstract, making it difficult to retrieve sufficiently relevant

knowledge.

Reasoning of FIR

We present a partial response from the FIR module at step 3 in Fig. 5.13. This example not only demonstrates the internal reasoning process of the LLM agent but also shows how SA-RAG is utilized to ground the reasoning with external knowledge. The first part explains the LLM’s logical derivation, followed by the application of SA-RAG for retrieving relevant supporting information.

5.5 Discussion

Interpretable Conflict Management. A key observation from our evaluation is that conflict resolution inherently involves trade-offs across KPIs. For example, in the ES-TM scenario, ORCA rejects a TxPower reduction to preserve eMBB throughput, at the cost of higher energy consumption. This highlights ORCA’s strength in providing interpretable, fine-grained decisions, allowing operators to understand conflict causes and manage competing objectives. Unlike black-box approaches, ORCA enables explicit balancing of KPIs based on current priorities.

5.6 Related Work

5.6.1 Conflict mitigation in ORAN

Some DRL-based approaches adopt multi-agent deep Q-learning to foster cooperation among xApps. While effective in controlled settings, they face scalability issues and assume xApps can be retrained jointly, which is impractical in modular, vendor-agnostic O-RAN. xApp Distillation [38] mitigates this by merging pre-trained xApps via policy distillation into a single DQN-based xApp, but this sacrifices modularity and requires re-distillation as conditions evolve. The scheduler in [29] selects xApps using A2C based on current environments but assumes all possible environments are known during training, thus limiting generalization.

The framework in [5] classifies conflicts as direct, indirect, or implicit and resolves them via a Conflict Resolution agent that prioritizes actions. Extensions such as [161, 164] optimize parameter configurations based on utility mappings from KPIs but depend on fixed-environment data, reducing robustness to change. Statistical methods collect ECDFs under conditioned environments [30], using distance metrics to assess conflict severity and deployment compatibility. Others cluster cells by configuration and traffic patterns to predict KPI impacts and preemptively cancel harmful requests [9]. However, these approaches rely heavily on prior environmental assumptions, limiting their applicability in dynamic or unseen scenarios.

5.6.2 LLMs for communication network

LLMs have shown strong effectiveness across diverse tasks, including reasoning, embodied decision-making, and code generation. They are now applied across a wide range of disciplines including engineering, computer systems and social science [55, 57, 118, 127, 133, 165, 189, 190]. Recent research has begun exploring how LLMs can empower communication networks. Early studies focus on tasks like technical question answering over telecom standards, classification of working groups, and document tagging [11, 22, 24, 52, 66, 93, 100, 122]. In parallel, LLM agents have emerged, enabling models to reason through complex problems using intermediate steps, external tools, or symbolic representations. They show strong potential in dynamic environments, especially for autonomous software development [27, 34, 53, 59, 76, 126, 168]. Recent works [?, 70] employed LLM agents for network protocol design. Unlike this line of research, we elicit the agent’s reasoning capability via a guided multi-step inference enabled by a β -Reasoning Cascade and SA-RAG, allowing it to actively analyze and reason about system behavior with well-grounded domain knowledge.

5.7 Summary

We present ORCA, the framework leveraging generative LLM agents to detect and mitigate xApp conflicts in ORAN, achieving up to 60.7% lower cosine error and 65.1% lower

NRMSE than baselines. By combining symbolic reasoning with domain knowledge, ORCA delivers fine-grained conflict mitigation and showcases the potential of LLM-driven reasoning for future network optimization.

Chapter 5, in part, is a reprint of the material as it appears in Dae Cheol Kwon and Xinyu Zhang, “Open RAN Conflict Agents: Detecting and Mitigating xApp Conflicts with Generative Agents,” in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2026. The dissertation author was the primary investigator and first author of this paper.

Chapter 6

Proactive Reasoning for Implicit Misconfiguration Elimination

6.1 Introduction

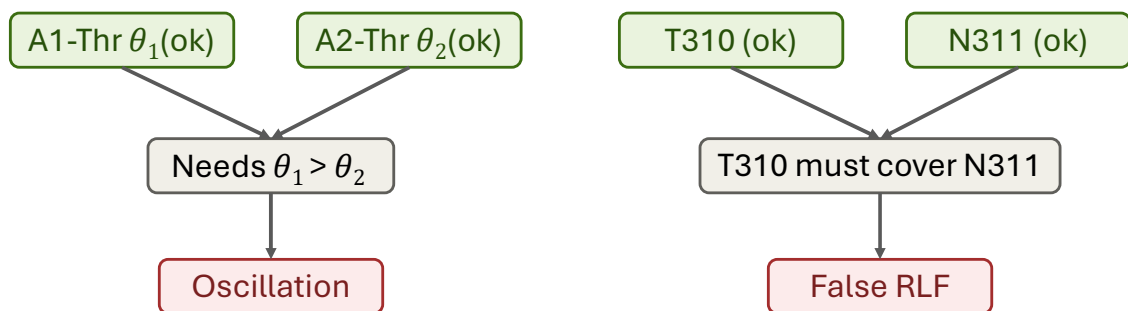
This chapter addresses the final component of autonomous wireless intelligence studied in this dissertation: configuration safety. The preceding chapters showed how generative agents can design communication protocols, synthesize O-RAN xApps, and reason about conflicts among independently developed xApps. This chapter moves one level deeper, from conflicts among control applications to hidden dependencies among the configuration parameters that those applications or network operators may modify. The goal is to prevent unsafe configurations before they are deployed, rather than diagnosing them only after failures appear in operational networks.

Cellular networks are increasingly governed by a large and evolving configuration surface. These parameters are essential for adapting radio access behavior, but their effects are rarely independent. Parameters that are valid individually may still interact within shared RRC procedures, influence sequential decisions, or affect the same runtime variable, as illustrated in Fig. ???. As a consequence, a network can satisfy every individual configuration rule while still exhibiting undesirable behavior, and the risk compounds as networks grow denser, more heterogeneous, and more frequently reconfigured at runtime.

Prior studies have shown that such configuration risks are not merely theoretical. In

3G/4G networks, configuration errors can trap devices in persistent handover loops or steer them toward inferior cells [78, 120, 184]. In 5G and 4.5G, this problem becomes more complex: carrier aggregation and dual connectivity allow multiple cells to serve a device concurrently, and the primary cell may rewrite measurement objects, reporting events, and threshold conditions as the serving cell set evolves. These dynamics introduce dependencies not only across neighboring cells but also across configuration states over time. Recent work has shown that such dependent RRC misconfigurations can stem from structural dependencies across states and cross-parameter dependencies within a state, leading to persistent loops, missed secondary cells, problematic handoffs, and severe throughput degradation in operational networks [187]. Open RAN is likely to accelerate this trend by moving radio-control decisions into xApps, where deep learning or reinforcement learning policies may continuously tune parameters at runtime. This programmability improves adaptability, yet it also makes it easier for independently developed control logic to create hidden dependencies, conflicting updates, or unsafe parameter combinations, as illustrated in Fig. ??.

These findings reveal the severity of configuration-dependent failures, but they also expose a fundamental limitation in how such failures are discovered today. Existing studies identify misconfigurations primarily through empirical measurement: researchers collect traces from operational networks, visit locations where abnormal behavior occurs, reconstruct RRC state transitions, and then analyze the parameter dependencies responsible for the observed failures. This approach is invaluable for understanding real-world behavior, but it is inherently reactive. A misconfiguration must first be deployed, triggered under specific radio and mobility conditions, and observed through field measurements before its root cause can be analyzed. As the configuration space expands and control logic becomes increasingly automated, relying on post-deployment measurement campaigns alone is unlikely to keep pace with the number of possible parameter interactions. What is needed is a proactive capability: given a set of configurable parameters and their procedural semantics, the system should reason about implicit dependencies and flag unsafe combinations before they manifest in operational networks.



(a) A1 and A2 thresholds are individually valid, but $\theta_1 < \theta_2$ opens an overlap region that causes oscillation.

(b) T310 and N311 are individually valid, but T310 shorter than the N311 evaluation window causes false RLF.

Figure 6.1. Two examples of implicit misconfigurations. Each parameter is individually valid; the violation lies in the hidden inter-parameter relation.

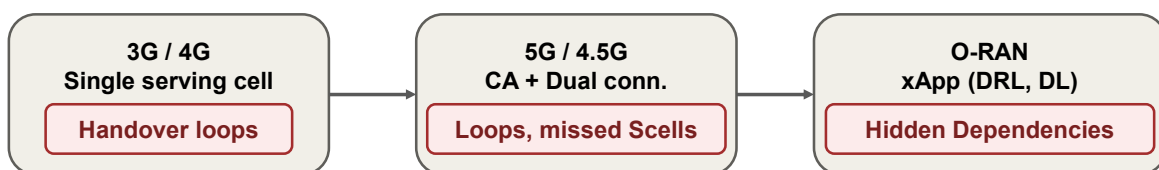


Figure 6.2. Configuration risks evolve across cellular generations: handover loops in 3G/4G, missed SCells under carrier aggregation and dual connectivity in 5G/4.5G, and hidden inter-parameter dependencies introduced by programmable O-RAN xApps.

This chapter presents PRIME, a framework that proactively reasons about implicit inter-parameter dependencies and eliminates the misconfigurations they cause before deployment. Rather than inferring problems from measurements of a live network, PRIME reads the 3GPP specifications directly and reasons about the implicit constraints that govern how parameters may be combined. Recently, large language models (LLMs) have demonstrated a strong reasoning ability over complex technical specifications, making them a natural fit for surfacing dependencies that are stated only implicitly in natural-language prose. However, simply prompting an LLM to find misconfigurations does not work, and realizing PRIME presents non-trivial challenges.

First, there is no clear target for discovering implicit configuration constraints. The specification does not define the types of inter-parameter relationships that should be considered constraints, nor does it provide representative examples of such relationships. Without concrete constraint categories and examples, asking an LLM to discover implicit dependencies becomes an open-ended task: the model has no clear criterion for determining which parameter relationships are genuine constraints and which are merely semantically related. To address this challenge, PRIME develops a systematic taxonomy that organizes implicit dependencies into five categories, C1–C5, and provides a well-defined target for constraint discovery.

Second, an LLM extracts implicit constraints unreliably. Identifying a constraint requires two steps: interpreting the precise meaning of each parameter, and then reasoning about how the two relate. A generic prompt performs both poorly, misreading parameters and inferring spurious relationships. Chain-of-thought (CoT) [170] prompting is a well-established technique for improving LLM reasoning on multi-step problems. However, we find that CoT alone yields little improvement on this task. This is consistent with prior observations that CoT structures a model’s reasoning without guaranteeing that its conclusions are grounded or faithful to the source [157, 166]; in our setting, step-by-step prompting supplies neither the domain semantics each parameter requires nor the specific derivation a constraint demands. PRIME addresses this by labeling each parameter with semantics drawn from the specification and guiding the LLM through explicit reasoning steps, so that constraint derivation operates over structured labels

rather than raw prose.

Third, the search space is prohibitively large. Because an implicit constraint may involve more than two parameters, the 12,090 parameters extracted from the Release 17 LTE and 5G NR RRC specifications create 2^{12090} possible combinations. PRIME first shows that the targeted constraints can be decomposed into pairwise relationships, reducing the space to more than 73 million pairs. It then uses semantic labeling to retain only pairs whose state, metric, direction, target, and type match a constraint-category signature, reducing the space to 29,137 candidate pairs.

We evaluate PRIME on the Release 17 LTE and 5G NR RRC specifications using a partial oracle of 94 dependencies that we construct from three complementary sources. Because implicit constraints are not explicitly documented, no complete ground truth is available. We therefore derive the oracle from real-world misconfigurations reported in measurement studies, conformance constraints specified in 3GPP TS 36.523 and TS 38.523, and synthetic ground-truth constraints obtained through manual inspection of TS 36.331 and TS 38.331 independently of the PRIME pipeline. Across this oracle, PRIME recovers 90 of the 94 dependencies, achieving 95.74% recall.

In summary, the main contributions of this paper are:

(i) We design PRIME, the first framework that proactively detects implicit inter-parameter misconfigurations in 3GPP RRC using an LLM agent that reasons directly over the specifications, eliminating these errors before deployment without requiring any measurement of a live network.

(ii) We systematically construct a structured taxonomy of implicit inter-parameter dependencies in 3GPP RRC, characterizing them into five categories (C1–C5) derived from the structural elements of the specification. The taxonomy turns an open-ended misconfiguration search into detection against defined targets and provides per-category signatures that narrow the candidate pair space for tractable LLM-based extraction.

(iii) We construct a 94-entry partial oracle from three complementary sources to enable evaluation despite the absence of established ground truth for implicit RRC dependencies. Using

this oracle, we show that PRIME recovers 95.74% of confirmed dependencies and identifies 433 distinct constraints.

6.2 Constraint Taxonomy

A straightforward approach to finding potential inter-parameter misconfigurations in a cellular standard is to prompt an LLM to read the specification and report any dependencies it identifies. However, without a clear target, the LLM cannot determine which relationships should be treated as genuine constraints or whether an entire class of dependencies has been missed. PRIME therefore requires a taxonomy of implicit constraint types.

Rather than constructing the taxonomy from known examples, we derive it systematically from RRC configuration rules. This is challenging because the standard does not provide a consolidated list of such rules. We therefore analyze RRC information elements (IEs) and identify four recurring configuration elements: the parameter definition, procedure usage, trigger condition, and scope. Although these elements are dispersed across the specification—for example, procedure usage appears in configuration sequence descriptions, trigger conditions are reflected in thresholds, timer behavior, and radio-metric conditions, and scope is defined by the applicable RRC state or configuration context—they can be reconstructed into complete configuration rules. The reconstructed rules exhibit a small set of recurring structural attributes that govern parameter relationships. Each recurring attribute serves as the basis for one category in our taxonomy, resulting in five implicit constraint types (C1–C5): opposite actions on the same metric within a state, opposite actions across states, coupled timers or counters, decisions involving different radio metrics, and dependencies between sequential configuration stages.

C1 (opposite-action) arises when two rules act on the same metric in the same state but in opposite directions; if their thresholds are set independently they can define an overlap region in which both fire, causing oscillation, as with the A1 and A2 event thresholds on serving-cell RSRP. **C2 (cross-state)** arises when the same quantity is bounded in two different RRC states,

Table 6.1. Constraint taxonomy. Each category is one structural form an implicit inter-parameter dependency can take.

	Category	Example
C1	Opposite-action	A1 vs A2 thresholds (serving RSRP)
C2	Cross-state	Idle selection vs connected-mode link maintenance
C3	Timing	Time-to-trigger vs T310
C4	Cross-metric	A3 (RSRP) vs A6 (RSRQ) offset for one decision
C5	Sequential-stage	Measurement config omits a frequency the next stage needs

such as idle-mode cell selection and connected-mode link maintenance, with the two bounds set without regard to one another. **C3 (timing)** arises when two timers or counters are temporally coupled and their durations must be ordered, as with time-to-trigger and the radio-link-failure timer T310. **C4 (cross-metric)** arises when a single decision is governed by thresholds on different radio metrics, for instance an RSRP-based and an RSRQ-based threshold configured independently, so that a strong reading on one metric need not imply an acceptable reading on the other. **C5 (sequential-stage)** arises when one configuration stage produces an output that a later stage requires but does not receive, such as a measurement configuration that omits a frequency the cell-selection stage expects. These five categories define the targets PRIME must extract. The taxonomy is systematic but not intended to be exhaustive. It captures the recurring dependency forms identified from the structural attributes of the current RRC specifications and can be extended as new dependency forms emerge. Future releases may introduce new procedures, metrics, or inter-procedure couplings that are not covered by the present categorization.

6.3 PRIME Design

6.3.1 System Overview

As illustrated in Fig. 6.3, PRIME consists of four modules arranged as an offline pipeline that transforms 3GPP RRC specifications into a reusable *constraints database*. The pipeline draws on two inputs: the RRC specifications themselves (TS 38.331 for NR and TS 36.331

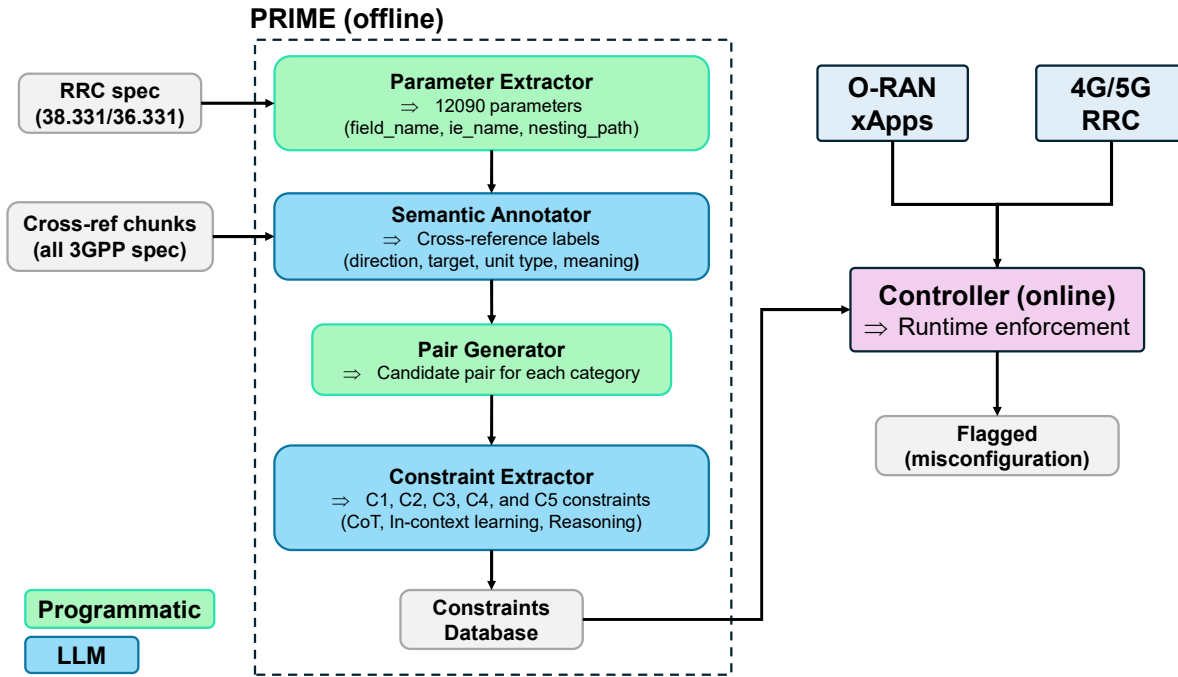


Figure 6.3. Overall architecture of PRIME. The taxonomy (§??) is the spine that the four middle stages reference; the LLM is invoked only in Steps 2 and 5 (shaded), the remaining stages being deterministic.

for LTE), which define the configurable parameters, and a corpus of cross-reference chunks gathered from the wider 3GPP specification suite, which supplies the behavioral context required to interpret those parameters correctly.

The Parameter Extractor parses the RRC spec into a structured inventory of 12,090 configurable parameters, each represented as a $\langle \text{field_name}, \text{ie_name}, \text{nesting_path} \rangle$ tuple. The Semantic Annotator then enriches every parameter with the relevant cross-referenced chunks and assigns it a set of semantic labels—such as its direction, target, unit type, and meaning—that expose the structural properties our constraint taxonomy reasons over. Using these labels, the Pair Generator produces, for each constraint category, the candidate parameter pairs whose labels jointly match that category’s signature, reducing the otherwise intractable space of more than 73 million possible parameter pairs to a small, category-specific candidate set. Finally, the Constraint Extractor derives the implicit constraint underlying each candidate pair through category-specific chain-of-thought reasoning with in-context examples, producing the C1–C5

constraints that populate the constraints database.

Crucially, the output of PRIME is the constraints database, a reusable artifact rather than a runtime component. We treat enforcement as the responsibility of the network operator: at runtime, an online controller validates a deployed configuration against the database and flags any violation as a misconfiguration. The configurations under scrutiny may originate from automated O-RAN xApps or from conventional 4G/5G RRC configuration, both of which the same database can serve. We implement this enforcement path to demonstrate how the database is used and to evaluate its effectiveness (Section 6.4), but we emphasize that it is a *consumer* of PRIME, not part of PRIME itself.

6.3.2 Parameter Extractor

The goal of the Parameter Extractor is to convert the RRC specifications into a complete and canonical inventory of every configurable parameter, which serves as the common substrate that all subsequent modules reason over. The defining requirement is *completeness*: a parameter that is missed here is invisible to every downstream module, so any constraint involving it is unrecoverable regardless of how well the later stages perform. The completeness of extraction therefore sets a hard ceiling on the set of constraints PRIME can possibly discover.

The challenge is that the ASN.1 does not present parameters as a flat list. It is a deeply nested grammar in which information elements (IEs) contain other IEs, a field's type is frequently another IE or a CHOICE rather than a primitive, and the same field name recurs across unrelated IEs. Recovering a complete and unambiguous inventory of leaf parameters therefore requires more than enumerating named fields: the hierarchy must be traversed in full so that no configurable field is missed, type references must be resolved so that each field's true domain is known, and each leaf must be given an identity that distinguishes it from like-named fields elsewhere in the grammar.

The framework of the Parameter Extractor follows directly from this challenge and comprises two components. The **ASN.1 parser** traverses the grammar to recover every field

together with its type, value range, and optionality; parsing the complete module set, rather than sampling, is what guarantees the completeness the inventory demands. The **IE linker** then resolves type references and flattens the nested grammar into a fully qualified `nesting_path`, which gives every leaf field a unique identity even when the same name recurs in different IEs, and resolves polymorphic types into their concrete domains. This last step is not mere bookkeeping: it is what later exposes, for example, that `a1-Threshold` is typed as a `MeasTriggerQuantity CHOICE` over `RSRP`, `RSRQ`, and `SINR`, a fact that directly governs how the Pair Generator and Constraint Extractor must treat its metric.

The module emits each parameter as a $\langle \text{field_name}, \text{ie_name}, \text{nesting_path} \rangle$ tuple, together with the formal attributes (value range and optionality) recovered from the ASN.1, for a total of 12,090 leaf parameters. We take the *leaf field* (rather than the enclosing IE) as the unit of extraction, because the value-relationship categories (C1–C4) are properties of individual configurable values. Since the sequential-stage dependency category (C5) instead reasons about IE *presence*, the linker additionally retains the IE optionality structure (which IEs are optional or conditionally present) layered over the leaf inventory. The result is a single structural representation that supports both value-level and presence-level reasoning across the remaining modules.

6.3.3 Cross-Reference-Augmented Semantic Labeling

The parameter extractor provides the structural candidates needed for constraint analysis, but its output alone is not sufficient for the stages that follow. Extracted parameters are often represented only by bare identifiers and local schema structure, without the semantic information needed to determine their role in a protocol procedure. For example, from `a1-Threshold` and `a2-Threshold` alone, a reasoner cannot reliably infer which condition each threshold governs, which radio metric it is associated with, or whether the corresponding event is triggered by an increasing or decreasing metric value. Without this semantic information, neither the pairing of candidates by taxonomy signature nor the subsequent constraint reasoning has any grounding for

distinguishing true hidden dependencies from superficially related parameters.

PRIME therefore introduces an explicit semantic labeling stage before constraints are derived. For each parameter, the annotator assigns structured labels corresponding to the attributes used by the constraint taxonomy, including the associated procedure, RRC state, radio metric, threshold direction, the parameter’s functional role in the procedure. These labels transform low-level configuration identifiers into semantically grounded objects that can be compared by later stages.

However, labeling parameters from their local RRC descriptions alone is insufficient. In 3GPP specifications, a parameter may be declared in an RRC specification while its operational behavior is defined in a different document. For instance, `drxInactivityTimer` is declared in TS 38.331, but its functional meaning is specified by the DRX procedure in the MAC specification TS 38.321, where it determines how long the UE remains active after data activity. Therefore, relying only on the local declaration in TS 38.331 can produce incomplete or incorrect labels, especially for parameters whose semantics are distributed across multiple specifications.

To address this, the annotator labels each parameter using the descriptions it found across the specifications, not only the document in which it is declared. We split the full set of 3GPP specifications into chunks and, for each parameter, retrieve every chunk that mentions it by name. Because a parameter’s name recurs across documents, these chunks include not only its declaration but also the passages elsewhere that describe its behavior. The language model then reads the retrieved chunks collectively and assigns the parameter’s semantic labels, including its direction, unit type, meaning, and the other attributes the taxonomy reasons over. The output of this stage is a set of parameters augmented with semantic labels. These labels provide the common representation used by subsequent stages: the Pair Generator tests taxonomy-specific signatures over the labels, and the Constraint Reasoner reasons about candidate parameter pairs with explicit knowledge of each parameter’s protocol role, metric, and directionality.

6.3.4 Pair Generator

The goal of the Pair Generator is to emit, for each taxonomy category, the candidate parameter pairs whose labels jointly match that category’s signature. These are the input the Constraint Reasoner reasons over, and because that reasoning invokes a language model once per candidate, the candidate count determines whether the pipeline is tractable.

A constraint can in principle bind any number of parameters. With $n = 12,090$, the subsets of size at least two number

$$\sum_{k=2}^n \binom{n}{k} = 2^n - n - 1 \approx 2^{12090},$$

which cannot be enumerated. The taxonomy permits two successive reductions, both essential. The first restricts the unit from arbitrary subsets to pairs, leaving $\binom{n}{2} \approx 73$ million candidates. The second filters those pairs by category signatures, leaving roughly 29,000.

From subsets to pairs.

In general, examining only parameter pairs is not sufficient for arbitrary constraints, as some rules may involve three or more parameters. PRIME does not attempt to cover every possible multi-parameter rule. Instead, we show that all constraints in the C1-C5 taxonomy are binary-decomposable. Therefore, even when a constraint involves several parameters, it can be fully represented and verified through a set of pairwise conditions.

The argument differs for numeric and structural constraints. The ordering categories (C1–C3) relate commensurable values, and numeric ordering is transitive. For example, if $\theta_A > \theta_B$ and $\theta_B > \theta_C$, then $\theta_A > \theta_C$ follows automatically. Thus, checking only the consecutive pairs is sufficient to enforce the complete ordering. The structural categories (C4–C5) are binary-decomposable for different reasons. A cross-metric constraint (C4) requires related measurement events to use a consistent radio metric. For example, if a group of events is intended to make comparable mobility decisions based on RSRP, configuring one event to use RSRQ introduces

an inconsistency. Verifying this property only requires checking that every pair of related events uses the same metric, allowing constraints over an arbitrary number of events to be decomposed into pairwise consistency checks. A sequential configuration constraint (C5) requires that a parameter be accompanied by every prerequisite parameter on which it depends. If parameter A depends on both B and C , the rule is satisfied by verifying the two independent conditions (A, B) and (A, C) . More generally, a parameter with multiple prerequisites decomposes into a collection of pairwise dependency checks. Therefore, every constraint covered by the C1–C5 taxonomy can be represented and verified through pairwise conditions, even when the original rule involves multiple parameters.

From pairs to signature-matched candidates.

Even restricted to pairs, 73 million candidates cannot each be sent to a language model. The Pair Generator applies the taxonomy a second time, now as a filter on labels rather than a structural reduction. Each category is expressed as a signature over the labels assigned in §6.3.3: a direction, a target or decision, a metric, a unit type, and an RRC state. A first sub-stage removes pairs inconsistent with every category signature using cheap label comparisons over direction, target, and unit type. A second sub-stage generates candidates separately for each category, retaining only the pairs whose labels jointly satisfy that category’s signature, for example two thresholds on the same metric and state but opposite directions for C1, or two coupled timers for C3. Together these reduce the 73 million pairs to approximately 29,000 candidates, a reduction of over $2,500\times$. The filter is sound with respect to the assigned labels: a pair is dropped only when its labels fail a condition the category requires, so no pair whose labels match a category is discarded, and the candidate set is passed to the Constraint Reasoner.

6.3.5 Constraint Extractor

The goal of the Constraint Extractor is to decide, for each candidate pair the Pair Generator emits, whether an implicit constraint genuinely holds between the two parameters, and when

one does, to record it in a form a runtime check can evaluate directly. This is the stage that converts the taxonomy from a statement of which dependencies can exist into a database of the dependencies that do.

Recovering such a constraint is a reasoning task that an undirected prompt performs unreliably. It decomposes into two steps that must both succeed: reading the precise meaning of each parameter, namely the quantity it governs, the state it applies in, and the direction of its threshold, and then reasoning about how the two interact to form a hidden dependency. A generic prompt handles neither step well, misreading parameters and asserting relationships that do not exist, and plain chain-of-thought prompting does not close the gap: it organizes how the model reasons but neither grounds what it reads from the specification nor specifies which derivation a given dependency demands. We quantify this shortfall against a chain-of-thought-only baseline in §6.4.

The Constraint Extractor addresses both gaps by reasoning over the semantic labels from §?? rather than raw specification text, and by guiding that reasoning with the structure the taxonomy already supplies. Because the Pair Generator has assigned every candidate a category, the constraint that could relate its two parameters is known up to its specific values: a C1 pair must resolve to a threshold ordering on a shared metric, a C3 pair to an ordering on two durations, and so on. The extractor exploits this by issuing one focused prompt per candidate pair, carrying the two parameters’ labels, the category and its definition, the category-specific reasoning steps the derivation should follow, and in-context examples of a confirmed constraint and a rejected pair in that category. Processing each pair in isolation keeps the reasoning shallow and avoids the degradation that arises when many pairs share a single context, while the category-specific steps and examples direct the model toward the one derivation the category requires rather than an open-ended search. The model returns a decision on whether the constraint holds and, when it does, the constraint in the structured form below; pairs it judges spurious are discarded.

```

{
  "category": "C1",
  "param_a": "a1-Threshold",
  "param_b": "a2-Threshold",
  "source_spec": "TS 38.331",
  "direct_inequality": "a1-Threshold >= a2-Threshold",
  "violation": "a1-Threshold < a2-Threshold",
  "description": "A1 and A2 trigger on serving-cell RSRP
    in opposite directions; if a1-Threshold is set below
    a2-Threshold the two events overlap over a range of
    RSRP values and the cell oscillates."
}

```

Figure 6.4. A C1 constraint record. A C5 record replaces `param_b` and `direct_inequality` with a referenced entity and a co-presence relation.

Each confirmed constraint is stored as a record carrying its category, the two parameters it relates, the source specification, the direct inequality that must hold between their values, a natural-language description of the dependency, and the violation condition that negates the inequality (Fig. 6.4). Two refinements matter for downstream checking. First, an inequality may embed a constant fixed by the specification rather than relate two configurable values alone. The timing constraint between the radio-link-failure timer T310 and the in-sync evaluation count, for example, carries the per-evaluation period from the specification inside the inequality, so the runtime check evaluates it as a formula rather than a bare comparison. Second, C5 dependencies concern IE presence rather than values, so a C5 record replaces the second parameter and the inequality with a single referenced entity and a co-presence relation, recording that one IE's presence requires another's. The outcome is a single record schema that expresses every category as a directly checkable predicate.

6.3.6 Runtime Enforcement

PRIME outputs a database of implicit inter-parameter constraints that network operators can use to validate configurations before or after deployment. For example, an operator may reject a proposed configuration that violates a constraint, return the violated constraint as feedback to the requester, or accept the configuration while raising an alarm for further review. In our evaluation, we implement a prototype checker to demonstrate that the database can be used for runtime validation. However, PRIME does not prescribe a specific enforcement policy; the operator determines how each detected violation should be handled.

6.4 Evaluation

6.4.1 Methodology

Database under test. We run PRIME over the NR and LTE RRC specifications (TS 38.331 and TS 36.331). The Parameter Extractor recovers 12,090 leaf parameters; the Pair Generator reduces the roughly 73 million possible parameter pairs to about 29,000 category-consistent candidates; and the Constraint Extractor confirms 433 constraints from those candidates. This 433-constraint database is the artifact all subsequent measurements are taken against.

Partial oracle. Measuring extraction quality requires a ground truth of constraints that should be recovered. We construct a partial oracle of 94 implicit constraints and show examples in (Table 6.2), drawn from three sources: *(i)* misconfigurations documented in real-world 4G and 5G measurement studies of operational networks [78, 187], *(ii)* requirements that the 3GPP specifications state explicitly — including 3GPP UE conformance test specifications (TS 38.523, TS 36.523) and inter-IE correctness conditions stated in TS 38.331 and TS 36.331 procedure text — and that a configuration can therefore violate, and *(iii)* synthetic constraints derived by manual inspection of TS 38.331 and TS 36.331 specification text, constructed independently of PRIME’s pipeline output. Each entry is reduced to a checkable relationship between named parameters or IEs and assigned the taxonomy category it falls under. The oracle is partial by construction: it

Table 6.2. Representative oracle entries, one per PRIME taxonomy category.

ID	Cat.	Constraint	Source	Found
O1	C1	$\theta_{B1} > \theta_{A2}$: 5G SCell add/remove overlap loop	Real-world	o
O2	C2	threshServingLowP < a1-Threshold (IDLE vs. CONNECTED serving-good consistency)	Synthetic (TS 38.331 §5.2.4)	o
O3	C3	T310 must allow N311 in-sync recovery counts within evaluation period	Conformance (TS 38.523 §7.3)	o
O4	C4	a1-Threshold and a2-Threshold on the same cell must use the same triggerQuantity	Synthetic (TS 38.331 §5.5.4)	o
O5	C5	RLF-TimersAndConstants requires RadioLinkMonitoringConfig with at least one RLM-RS resource	Conformance (TS 38.523 §5.4)	o

Table 6.3. Recall of PRIME on the 94-entry partial oracle. The real-world category is split by configuration scope: single-cell entries are within PRIME’s design; cross-cell and topology-dependent entries are explicitly out of scope.

Oracle source	Total	Found	Recall (%)
Conformance	27	26	100.00
Real-world (single-cell)	3	3	100.00
Real-world (cross-cell)	3	0	0.00
Synthetic	61	61	100.00
Combined	94	90	95.74
In-scope only	90	90	100.00

captures known, spec-stated, and synthetic cases, not the full space of implicit dependencies, so recall against it is a lower bound on coverage rather than a complete recall figure.

PRIME extracts 433 constraints from the two specifications and recovers 90 of the 94 oracle entries, a recall of 0.96. The recovered entries span the taxonomy, covering the opposite-action and cross-state value constraints, the timing constraints, the cross-metric constraint, and the sequential-stage co-presence constraints, which confirms that the per-category reasoning derives constraints across all five categories rather than only the easiest.

The four missed entries fall into three classes, each outside what PRIME is designed to recover from a single specification. Entry 3 requires comparing thresholds configured on two

different neighboring cells, whereas PRIME reasons within one configuration and so cannot pair parameters across cells. Entry 4 has the shape of a sequential-stage constraint but depends on the set of frequencies actually reachable at a deployment site, which is runtime topology the specification does not state. Entries 5 and 17 concern a single parameter whose absence triggers an unsafe default; with no second parameter or IE to pair against, they do not fit the pairwise and co-presence model the taxonomy uses. None of the four is a reasoning failure on an in-scope pair, which is the failure mode the design is meant to avoid; they instead mark the boundary of the intra-configuration, specification-only setting.

6.4.2 Operational Impact

We inject the recovered constraints’ violating configurations into system-level simulations built on the MATLAB 5G Toolbox with TR 38.901 channel models (UMa NLOS path loss, log-normal shadowing, and TDL-A small-scale fading), and compare the deployed configuration when it violates the constraint (*Without PRIME*) against the same configuration corrected to satisfy it (*With PRIME*).

Opposite-action (C1). Setting the 5G SCell addition threshold below the removal threshold ($\theta_{B1} = -115 \text{ dBm} < \theta_{A2} = -96 \text{ dBm}$) opens an overlap region in which the SCell is added and immediately removed. Figure 6.5 shows the resulting behavior: without the constraint the SCell state oscillates continuously between 4G-only and 4G+5G, and downlink throughput collapses and fluctuates as the secondary carrier is repeatedly torn down. Enforcing $\theta_{B1} > \theta_{A2}$ holds the SCell stable and restores sustained throughput.

Cross-metric (C4). Keying the A1 stop-measurement event on RSRP while keying the A2 start-measurement event on RSRQ lets both fire at once, because RSRP can read acceptable while RSRQ degrades under interference. Figure 6.6 shows the two metrics diverging against their thresholds, the measurement state entering a conflict whenever they disagree, and throughput dropping during each conflict. Requiring A1 and A2 to share a metric removes the conflict and yields a 7% throughput improvement over the trace.

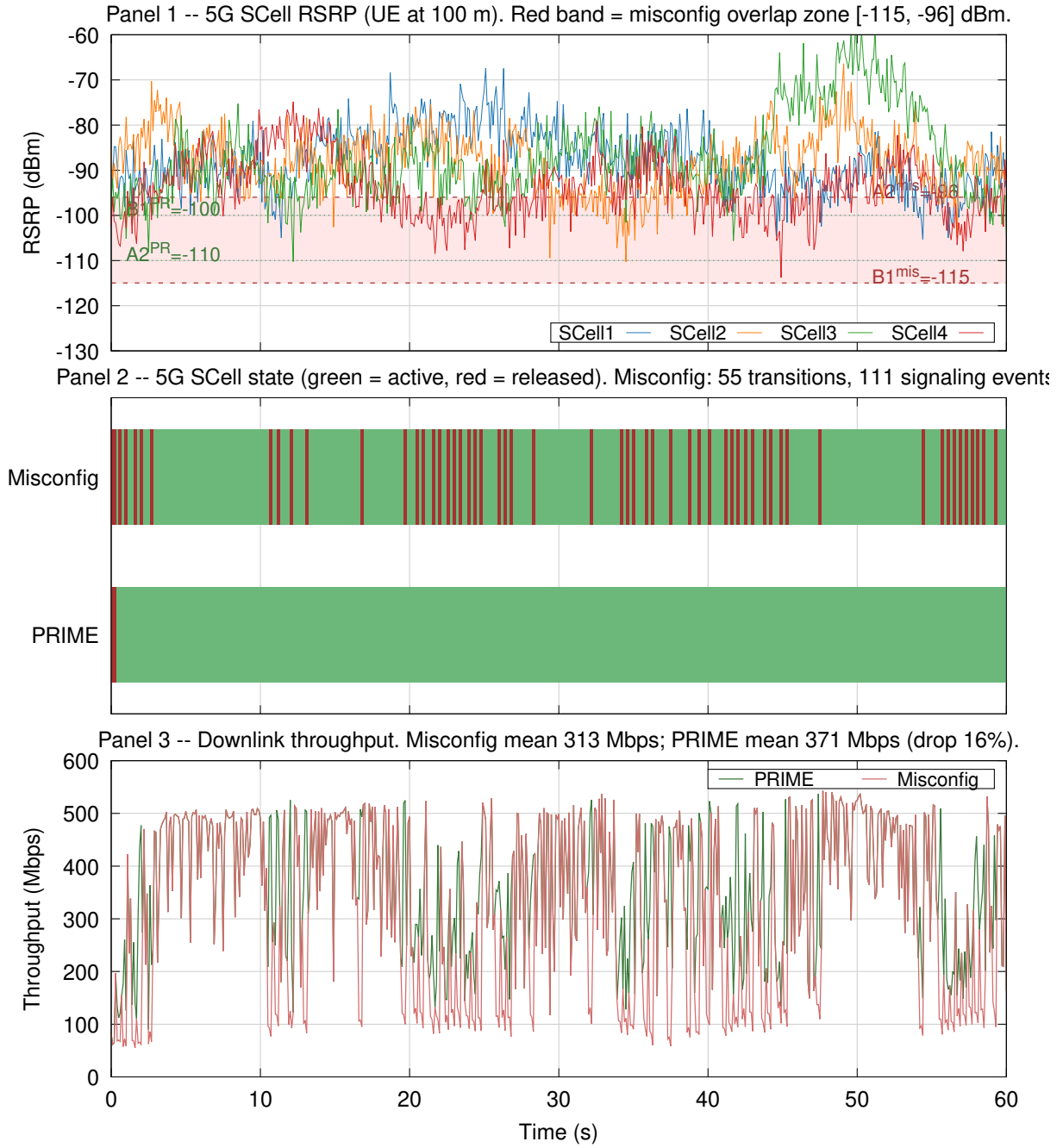


Figure 6.5. C1 ($\theta_{B1} > \theta_{A2}$). Without the constraint the 5G SCell oscillates and throughput collapses; with it the SCell is stable and throughput is sustained.

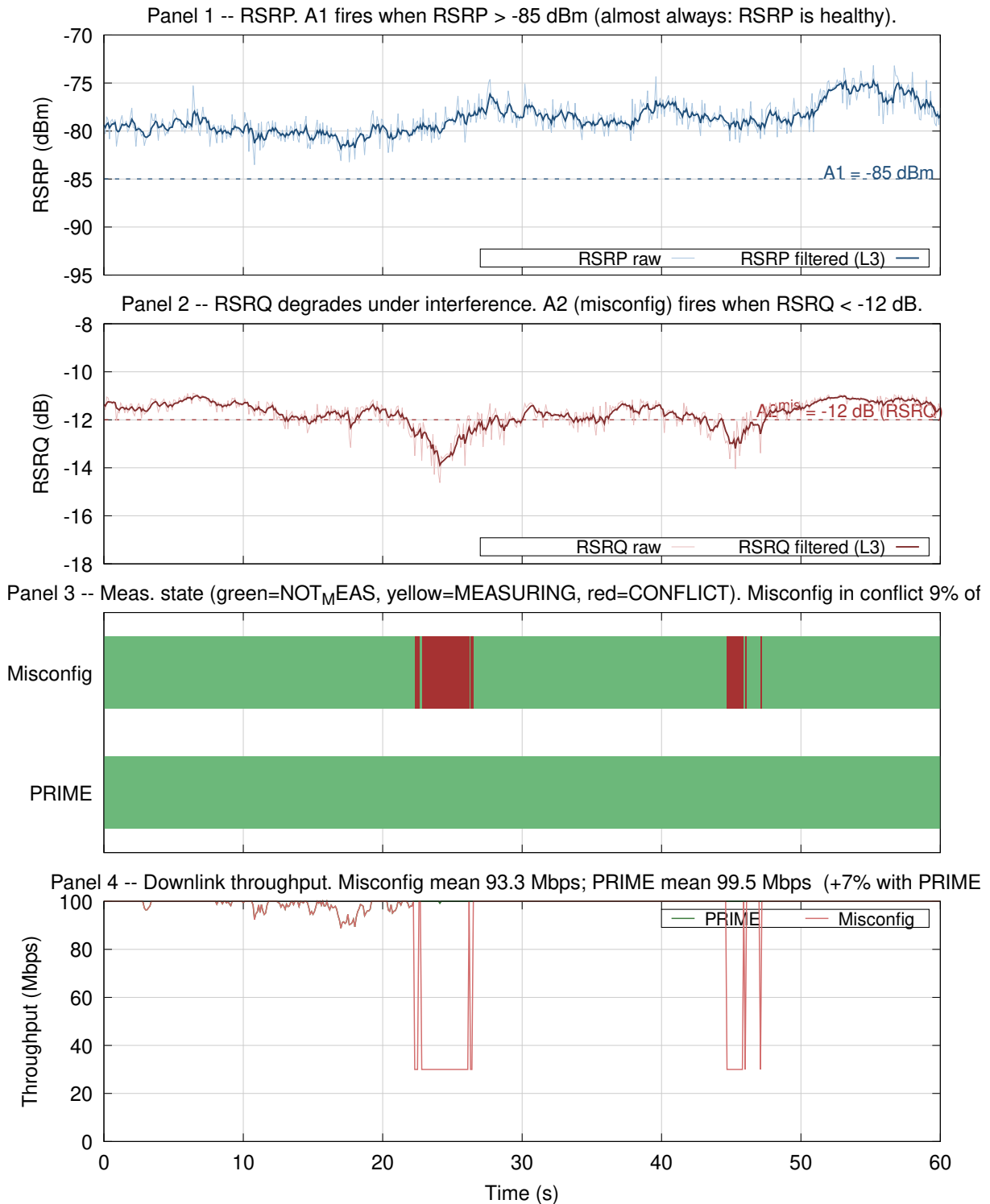


Figure 6.6. C4 (A1/A2 metric mismatch). RSRP and RSRQ diverge; without the constraint the measurement state conflicts and throughput drops, with a 7% improvement once the metrics are unified.



Figure 6.7. C5 (RLF detection). Without RadioLinkMonitoringConfig the UE never detects out-of-sync and its view of the connection diverges from the network's; enforcing RLM co-presence with the RLF timers restores out-of-sync detection and call recovery.

Sequential-stage — RLF detection (C5). Configuring `RLF-TimersAndConstants` (T310, N310, N311) without also configuring `RadioLinkMonitoringConfig` leaves the UE physical layer with no reference signals to monitor, so out-of-sync indications are never generated and RLF is never declared. Figure 6.7 shows the silent failure mode across a 30-minute trace with six blockage events: at $t \approx 987$ s a deep fade drives RSRP below Q_{out} longer than the network’s bearer-inactivity timeout, so the network releases the bearer while the UE, lacking RLM, remains in the *connected* state; from that point on the UE’s and network’s views of the connection diverge silently and all subsequent calls fail. Enforcing co-presence of RLM with the RLF timers restores out-of-sync detection and re-establishment after each fade; call drop rate falls from 42% to 15% and service availability rises from 52% to 87%.

Sequential-stage — inter-frequency mobility (C5). Configuring an inter-frequency `MeasObject` without the required `measGapConfig` prevents the UE from measuring inter-frequency neighbors in the CONNECTED state, so at each cell boundary the UE has no handoff candidate; RLF eventually fires and the UE recovers via IDLE-mode cell selection before re-attaching. Figure 6.8 shows the resulting blackouts across a 350-second drive past five cells, evaluated over 50 Monte Carlo runs with independent channel realizations: each cell boundary produces a blackout averaging 6.3 s (T310 expiry plus stochastic cell-scan, RACH, and RRC setup) during which throughput drops to zero. Enforcing co-presence of `measGapConfig` enables CONNECTED-mode handovers with ~ 70 ms gaps; mean throughput rises from 71 to 114 Mbps and cumulative service interruption falls from 30.8 s to 0.3 s over the trace.

6.5 Related Work

6.5.1 Misconfiguration in Cellular Networks

Cellular misconfiguration has been studied for over a decade. Li et al. [78] first formalized handover instability in 3G/4G distributed mobility management using a finite-state-machine model, showing that asymmetric neighbor-cell configurations can trap devices in persistent ping-

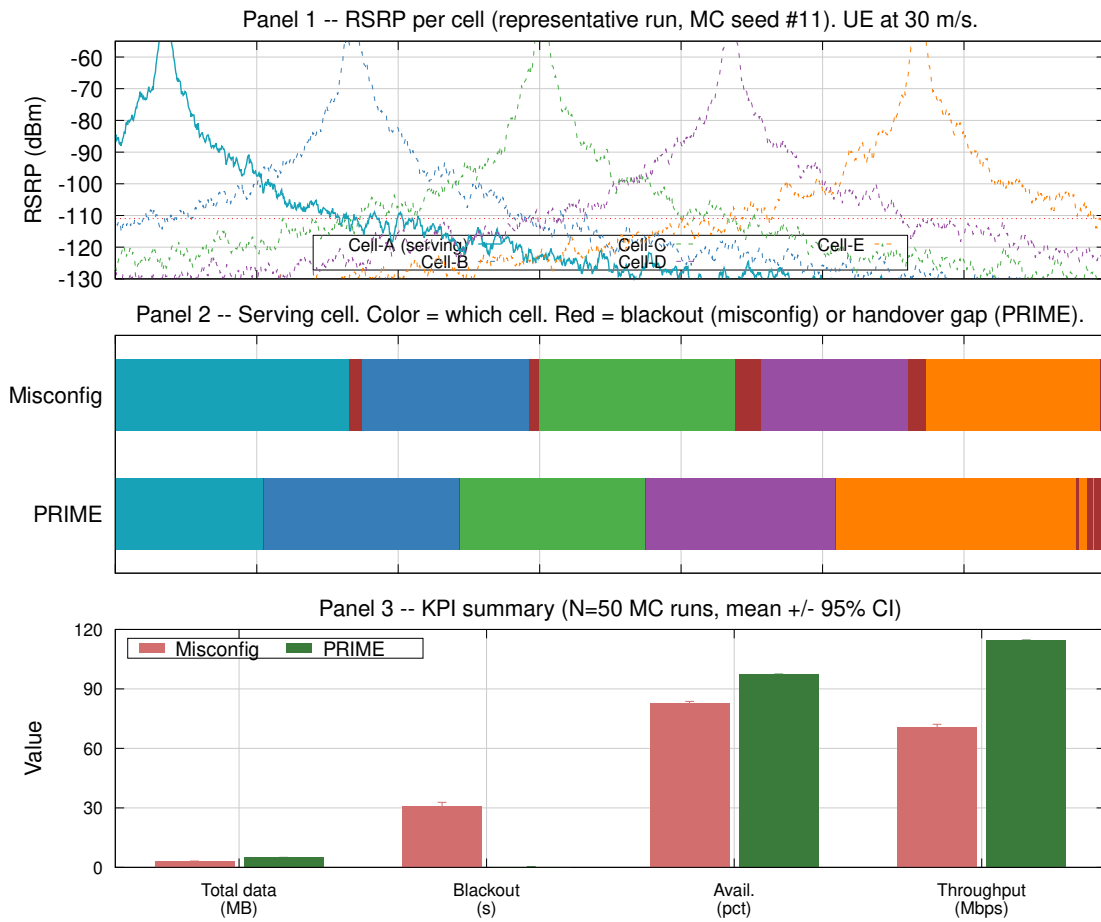


Figure 6.8. C5 (inter-frequency mobility). Without measGapConfig the UE cannot measure inter-frequency neighbors in CONNECTED, so each cell boundary triggers an RLF blackout; enforcing co-presence enables CONNECTED-mode handovers without service interruption.

pong loops. Subsequent work extended this line of investigation to femtocell deployments [120], undesired handovers in which a device converges to a measurably inferior cell [78, 120], inter-operator policy conflicts in multi-carrier access [184], instability patterns in operational Chinese networks [188], and broader measurement studies of mobility behavior and configuration practices [32, 177]. Most directly relevant to PRIME, Zhang et al. [187] examined dependent misconfigurations in 5G/4.5G RRC, formalizing configuration updates as a delta state machine and uncovering 644 instances of inter-state and intra-state dependency failures across three US operators and one Chinese operator. These studies demonstrate that configuration errors can persist even when individual parameters appear valid, because the failure often lies in the interaction among parameters, states, or cells.

However, existing studies are primarily measurement-driven and diagnostic. They identify misconfigurations after the problematic configuration has already been deployed, triggered under particular radio and mobility conditions, and observed in collected traces. As a result, dependencies that are present in the specification but not exercised during measurement remain hidden. PRIME complements this body of work by shifting the detection point from post-deployment diagnosis to pre-deployment analysis: it reasons directly over 3GPP specifications and flags dangerous inter-parameter combinations before they manifest in operational networks.

6.5.2 Conflict Detection and Mitigation in O-RAN

Multiple xApps operating concurrently in the near-RT RIC can issue competing control decisions over shared parameters, motivating research on conflict detection and mitigation in O-RAN. Recent work has explored architectural approaches [4], game-theoretic formulations [162], generalized conflict-management frameworks [44], and explainable-ML-based techniques [137] for detecting and resolving runtime conflicts. ORCA [72] introduced generative LLM agents with structured symbolic reasoning to estimate KPI sensitivity to control parameters in unseen environments and arbitrate xApp-requested changes accordingly.

PRIME shares ORCA’s use of LLM-agent reasoning, but it targets a fundamentally

different problem. O-RAN conflict-management systems focus on resolving conflicts among runtime control actions, typically by estimating their impact on KPIs or by arbitrating among competing xApp requests. PRIME instead focuses on specification-level configuration validity. It analyzes 3GPP specifications to extract implicit dependencies between parameters and produces a database of constraints that can be checked before a configuration is deployed. Thus, PRIME can flag dangerous configurations regardless of which xApp, controller, or manual process produces them.

6.5.3 LLM Applications in Telecom

A growing line of work applies LLMs to telecom tasks, including standards question answering and document understanding [12, 67], domain-specific benchmarks for telecom knowledge [94, 106], protocol and algorithm design with LLM agents [49, 71], and automated conformance testing against 3GPP and O-RAN specifications [40]. These works show that LLMs can help interpret technical specifications, generate network logic, and support validation workflows. However, they do not directly target configuration validity or hidden dependencies among configurable parameters.

The methodologically nearest work is AI5GTest [40], which also reads 3GPP and O-RAN specifications with LLMs. Its goal, however, is to generate procedural test flows and check whether observed signaling messages conform to those flows, thereby validating implementation behavior. PRIME addresses a different layer of correctness: whether a set of configuration parameters is safe when combined. Rather than testing protocol executions, PRIME extracts implicit inter-parameter dependencies from specification text and converts them into directly checkable constraints. This places PRIME in a distinct position within the broader LLM-in-telecom landscape, using generative agents not only to understand specifications but also to proactively prevent configuration-induced failures.

6.6 Conclusion

This chapter presented PRIME, a proactive framework for detecting implicit RRC misconfigurations from 3GPP specifications before deployment. PRIME combines a five-category dependency taxonomy, cross-reference-based semantic labeling, and category-guided LLM reasoning to extract directly checkable inter-parameter constraints. Evaluation on Release 17 LTE and NR specifications shows that PRIME recovers all in-scope oracle dependencies and extracts hundreds of constraints. Simulations further demonstrate that enforcing these constraints prevents representative failures such as oscillation, measurement conflicts, radio-link-failure errors, and mobility interruptions. These results show that specification-grounded generative agents can improve the safety of cellular configuration management as networks become increasingly automated.

This result completes the dissertation’s progression from autonomous design to autonomous management and safety. CP-AgentNet demonstrates that generative agents can design communication behavior, Auto-xApp shows that they can synthesize deployable O-RAN control applications, ORCA shows that they can reason about conflicts among those applications, and PRIME shows that they can extract reusable safety constraints from cellular specifications. The next chapter summarizes these findings and discusses future directions for autonomous wireless intelligence.

Chapter 6, in part, has been submitted for publication of the material as it may appear in Dae Cheol Kwon and Xinyu Zhang, “PRIME: Proactive Reasoning for Implicit Misconfiguration Elimination with Generative Agents,” submitted to *IEEE Conference on Computer Communications (INFOCOM)*, 2027. The dissertation author was the primary investigator and first author of this material.

Chapter 7

Conclusion and Future Directions

7.1 Conclusion

This dissertation studied how large-language-model-based generative agents can support autonomous wireless intelligence across communication protocol design and Open RAN management. Modern wireless systems are becoming increasingly heterogeneous, programmable, and complex. As a result, many design and management tasks require reasoning across high-level objectives, protocol semantics, runtime conditions, configuration constraints, and implementation details. Traditional approaches, including manual design, rule-based control, and conventional machine learning, remain valuable but often require substantial human expertise, task-specific engineering, large training datasets, or post-deployment observation. This dissertation explored a complementary direction: using generative agents as a reasoning and synthesis layer that connects human intent, wireless domain knowledge, technical specifications, and executable network behavior.

The first part of the dissertation focused on communication protocol design. Chapter 3 presented CP-AgentNet, a framework that uses generative agents as autonomous and explainable decision-makers for protocol design. CP-AgentNet showed that LLM-powered agents can reason over network states, generate strategies from limited demonstrations, adapt to heterogeneous environments, and provide human-readable explanations for their actions. Through multiple access and TCP use cases, this chapter demonstrated that generative agents can support protocol

design without relying solely on handcrafted rules or data-hungry deep reinforcement learning models.

The second part of the dissertation focused on Open RAN management. Chapter 4 presented Auto-xApp, a framework for synthesizing deployable O-RAN xApps from high-level user requirements. Auto-xApp addressed the gap between operator intent and standards-compliant xApp implementation by decomposing xApp generation into intent translation, semantic mapping, capability validation, and code synthesis. This showed that generative agents can help automate not only algorithmic design but also the practical engineering steps required to integrate xApps with O-RAN service models and near-real-time RIC platforms.

Chapter 5 presented ORCA, a framework for detecting and mitigating conflicts among independently developed xApps. As O-RAN enables multiple xApps to coexist in the same programmable RAN environment, conflict management becomes increasingly important. ORCA showed that generative agents can support fine-grained conflict reasoning by estimating how control parameters influence key performance indicators under unseen and dynamically changing environments. This chapter demonstrated the potential of combining feature-importance analysis, retrieval-augmented reasoning, and structured agentic workflows for adaptive xApp conflict mitigation.

Chapter 6 presented PRIME, a framework for proactive reasoning about implicit misconfigurations in cellular networks. PRIME addressed a deeper reliability problem: configuration parameters may be individually valid but unsafe in combination due to hidden dependencies across procedures, states, metrics, or stages. Rather than discovering such problems only after deployment, PRIME reasons over cellular specifications to extract implicit constraints and build a reusable constraint database. This chapter showed that generative agents can support specification-grounded reasoning for proactive misconfiguration prevention.

Together, these contributions support the central thesis of this dissertation: generative agents can provide a useful foundation for autonomous wireless intelligence by enabling reasoning, synthesis, explanation, and validation across multiple layers of wireless system design and

management. Across the four systems, generative agents were not used as standalone text generators. Instead, they were embedded in structured workflows that combine domain knowledge, memory, retrieval, deterministic validation, programmatic tools, and system-specific constraints. This design principle is essential because wireless systems require correctness, reliability, and interpretability, not merely plausible natural-language outputs.

7.2 Lessons Learned

Several lessons emerge from the systems developed in this dissertation.

First, generative agents are most effective when they are placed within structured workflows. Direct prompting alone is often insufficient for complex wireless tasks, especially when the output must satisfy protocol constraints, interface requirements, or configuration dependencies. CP-AgentNet uses role-specialized agents, memory, and self-reflection. Auto-xApp uses schema-guided intent translation, semantic mapping, and capability checking. ORCA uses structured feature-importance reasoning and retrieval. PRIME uses a taxonomy, semantic labeling, candidate-pair pruning, and category-specific constraint extraction. These designs show that agentic reasoning becomes more reliable when the search space, intermediate representation, and validation criteria are carefully defined.

Second, grounding is essential. Wireless systems are governed by standards, service models, protocol semantics, and platform-specific capabilities. An LLM may produce plausible but incorrect outputs if it relies only on pretrained knowledge. The systems in this dissertation therefore ground agent reasoning in demonstrations, network logs, O-RAN specifications, RAN capability information, 3GPP documents, or structured intermediate artifacts. This grounding reduces hallucination and improves the consistency of the generated designs, explanations, and constraints.

Third, generative agents are particularly useful for tasks that combine technical reasoning with semantic interpretation. Many wireless tasks are not purely numerical optimization problems.

They require interpreting human intent, understanding protocol descriptions, mapping informal terms to standardized identifiers, identifying hidden dependencies, and explaining why a decision is reasonable. These are precisely the types of tasks where language-based reasoning can complement conventional optimization, learning, and rule-based methods.

Finally, the results suggest that autonomy in wireless systems should be viewed as a spectrum rather than a single capability. Some tasks require autonomous design, such as protocol strategy generation. Others require autonomous synthesis, such as xApp generation. Others require autonomous analysis, such as conflict reasoning or misconfiguration detection. By studying these different forms of autonomy, this dissertation shows how generative agents can contribute to multiple stages of the wireless system lifecycle.

7.3 Limitations

Although the results are promising, several limitations remain.

First, generative agents can still produce inconsistent or incorrect outputs. The frameworks in this dissertation mitigate this risk through retrieval, validation, self-reflection, structured prompts, and deterministic processing, but they do not eliminate it completely. Safety-critical network control will require stronger verification mechanisms before agent-generated decisions or configurations can be deployed in operational networks.

Second, the systems are evaluated on representative but necessarily bounded scenarios. CP-AgentNet evaluates protocol design on selected MAC and TCP use cases. Auto-xApp evaluates xApp synthesis on representative rule-based, learning-based, and DRL-based applications. ORCA evaluates conflict reasoning across a designed set of O-RAN environments. PRIME focuses on implicit dependencies derived from RRC configuration semantics. Broader deployment will require testing across more technologies, larger-scale environments, additional vendors, and real operational traces.

Third, agentic systems introduce their own engineering complexity. Although they re-

duce manual effort in some parts of the workflow, they require careful construction of prompts, schemas, knowledge bases, validation modules, and system interfaces. Designing these components remains an important part of building reliable generative-agent-based network systems.

Fourth, latency and cost remain practical concerns. Some agentic reasoning tasks are suitable for offline or non-real-time operation, such as specification analysis or xApp synthesis. Others may require near-real-time decisions, where LLM inference latency can be a bottleneck. Future systems must carefully decide which tasks should be handled by generative agents online, which should be performed offline, and which should be delegated to lightweight deterministic controllers after agent-assisted design.

7.4 Future Directions

This dissertation opens several directions for future research.

A first direction is stronger verification for generative-agent-based wireless systems. Future frameworks should combine LLM reasoning with formal methods, program analysis, model checking, digital twins, and runtime monitors. Such integration would help ensure that agent-generated protocols, xApps, and configurations satisfy safety and performance constraints before deployment.

A second direction is closed-loop autonomous network management. The systems in this dissertation study design, synthesis, conflict reasoning, and misconfiguration prevention. A natural next step is to integrate these capabilities into a continuous lifecycle in which agents generate control applications, monitor their behavior, detect conflicts or configuration risks, revise policies, and update deployments under operator supervision.

A third direction is multi-agent coordination across the O-RAN ecosystem. Future RAN environments may include many xApps, rApps, operators, vendors, and AI models. Generative agents could support negotiation, policy arbitration, conflict explanation, and coordinated decision-making among independently developed applications. This will require mechanisms

for trust, accountability, and policy compliance.

A fourth direction is practical deployment and human-in-the-loop operation. Fully autonomous network control may not be immediately acceptable in operational environments. A more practical path is to use generative agents as assistants that propose designs, explain risks, generate candidate configurations, and ask for operator approval before deployment. This human-in-the-loop model can provide a safer transition toward higher levels of autonomy.

7.5 Final Remarks

Wireless networks are entering an era in which heterogeneity, programmability, and automation will continue to increase. These trends create new opportunities for innovation, but they also make design and management increasingly difficult. This dissertation has shown that generative agents can help address this complexity by providing reasoning and synthesis capabilities that complement conventional wireless optimization and machine learning. By applying generative agents to protocol design, xApp synthesis, conflict management, and misconfiguration prevention, this dissertation takes a step toward autonomous wireless intelligence that is not only adaptive, but also explainable, programmable, and reliable.

Bibliography

- [1] O-ran conflict mitigation. Technical Report O-RAN.WG3.TR.ConMit-R004-v01.00, O-RAN Alliance, Working Group 3, February 2024. Technical Report.
- [2] 3GPP. Evolved universal terrestrial radio access (E-UTRA); radio resource control (RRC) protocol specification. Technical Report TS 36.331, 3rd Generation Partnership Project, TODO. VERIFY release and version.
- [3] 3GPP. NR; radio resource control (RRC) protocol specification. Technical Report TS 38.331, 3rd Generation Partnership Project, TODO. VERIFY release and version.
- [4] Cezary Adamczyk and Adrian Kliks. Conflict mitigation framework and conflict detection in O-RAN near-RT RIC. *IEEE Communications Magazine*, 61(12):199–205, 2023.
- [5] Cezary Adamczyk and Adrian Kliks. Conflict mitigation framework and conflict detection in o-ran near-rt ric. *IEEE Communications Magazine*, 61(12):199–205, 2023.
- [6] Cezary Adamczyk and Adrian Kliks. Detection and mitigation of indirect conflicts between xApps in open radio access networks. *arXiv preprint arXiv:2305.13464*, 2023.
- [7] Ahmed Al-Tahmeesschi, Yi Chu, Gurdeep Singh, Charles Turyagyenda, Dritan Kaleshi, David Grace, and Hamed Ahmadi. Maximising energy efficiency in large-scale open ran: Hybrid xapps and digital twin integration. *arXiv preprint arXiv:2509.10097*, 2025.
- [8] Devanshu Anand and Gabriel-Miro Muntean. Mitigating interferences in 5g o-ran hetnets through ml-driven xapp to enhance users’ qos. *IEEE Transactions on Network and Service Management*, 2026.
- [9] Joss Armstrong, Enda Fallon, and Sheila Fallon. Pre-emptive conflict detection architecture for o-ran service management and orchestration. In *2024 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*, pages 335–340. IEEE, 2024.
- [10] Lina Bariah, Qiyang Zhao, Hang Zou, Yu Tian, Faouzi Bader, and Merouane Debbah. Large generative AI models for telecom: The next big thing? *IEEE Communications Magazine*, 62(11):84–90, 2024.

- [11] Lina Bariah, Hang Zou, Qiyang Zhao, Belkacem Mouhouche, Faouzi Bader, and Merouane Debbah. Understanding telecom language through large language models. In *GLOBECOM 2023-2023 IEEE Global Communications Conference*, pages 6542–6547. IEEE, 2023.
- [12] Lina Bariah, Hang Zou, Qiyang Zhao, Belkacem Mouhouche, Faouzi Bader, and Merouane Debbah. Understanding telecom language through large language models. In *GLOBECOM 2023-2023 IEEE Global Communications Conference*, pages 6542–6547. IEEE, 2023.
- [13] Ryan Barker, Alireza Ebrahimi Dorcheh, Tolunay Seyfi, and Fatemeh Afghah. Real: Reinforcement learning-enabled xapps for experimental closed-loop optimization in o-ran with osc ric and srsran. In *2025 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 389–395. IEEE, 2025.
- [14] Leonardo Bonati, Salvatore D’Oro, Michele Polese, Stefano Basagni, and Tommaso Melodia. Intelligence and learning in O-RAN for data-driven NextG cellular networks. *IEEE Communications Magazine*, 59(10):21–27, 2021.
- [15] Leonardo Bonati, Michele Polese, Salvatore D’Oro, Stefano Basagni, and Tommaso Melodia. Open, programmable, and virtualized 5G networks: State-of-the-art and the road ahead. *Computer Networks*, 182:1–28, 2020.
- [16] Leonardo Bonati, Michele Polese, Salvatore D’Oro, Stefano Basagni, and Tommaso Melodia. OpenRAN Gym: AI/ML development, testing, and integration of O-RAN in open RAN. *TODO, TODO. VERIFY venue – multiple OpenRAN Gym papers exist.*
- [17] Pietro Brach del Prever, Salvatore D’Oro, Leonardo Bonati, Michele Polese, Maria Tsampazi, Heiko Lehmann, and Tommaso Melodia. PACIFISTA: Conflict evaluation and management in open RAN. *arXiv preprint arXiv:2405.04395*, 2024.
- [18] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 93–104, 2000.
- [19] Ethan Brooks, Logan A Walls, Richard Lewis, and Satinder Singh. In-context policy iteration. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.
- [20] Tom B. Brown, Benjamin Mann, Nick Ryder, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [21] Łukasz Budzisz, Rade Stanojevic, Arie Schlote, Fred Baker, and Robert Shorten. On the fair coexistence of loss-and delay-based tcp. *IEEE/ACM transactions on networking*, 19(6):1811–1824, 2011.
- [22] Haoran Cai and Sijie Wu. Tkg: Telecom knowledge governance framework for llm application. 2023.

- [23] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Communications of the ACM*, 60(2):58–66, 2017.
- [24] Gouranga Charan, Muhammad Alrabeiah, and Ahmed Alkhateeb. Vision-aided 6g wireless communications: Blockage prediction and proactive handoff. *IEEE Transactions on Vehicular Technology*, 70(10):10193–10208, 2021.
- [25] Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- [26] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.
- [27] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*, 2023.
- [28] Hai Cheng, Salvatore D’Oro, Rajeev Gangula, Sakthivel Velumani, Davide Villa, Leonardo Bonati, Michele Polese, Tommaso Melodia, Gabriel Arrobo, and Christian Maciocco. Oranslice: An open source 5g network slicing platform for o-ran. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 2297–2302, 2024.
- [29] Idris Cinemre and Toktam Mahmoodi. xapp conflict mitigation with scheduler. *arXiv preprint arXiv:2504.06867*, 2025.
- [30] Pietro Brach del Prever, Salvatore D’Oro, Leonardo Bonati, Michele Polese, Maria Tsampazi, Heiko Lehmann, and Tommaso Melodia. Pacifista: Conflict evaluation and management in open ran. *IEEE Transactions on Mobile Computing*, 2025.
- [31] Ailin Deng and Bryan Hooi. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 4027–4035, 2021.
- [32] Haotian Deng, Chunyi Peng, Ans Fida, Jiayi Meng, and Y Charlie Hu. Mobility support in cellular networks: A measurement study on its configurations and implications. In *Proceedings of the Internet Measurement Conference 2018*, pages 147–160, 2018.
- [33] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, P. Brighten Godfrey, and Michael Schapira. PCC Vivace: Online-learning congestion control. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.
- [34] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. Self-collaboration code generation via chatgpt. *arXiv preprint arXiv:2304.07590*, 2023.

- [35] Salvatore D’Oro, Leonardo Bonati, Michele Polese, and Tommaso Melodia. OrchestRAN: Network automation through orchestrated intelligence in the open RAN. In *Proc. IEEE INFOCOM*, 2022.
- [36] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.
- [37] Nouha Dziri, Ximing Lu, Melanie Sclar, et al. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [38] Hakan Erdol, Xiaoyang Wang, Robert Piechocki, George Oikonomou, and Arjun Parekh. xapp distillation: Ai-based conflict mitigation in b5g o-ran. *Computer Networks*, page 111848, 2025.
- [39] Angelo Feraudo, Stefano Maxenti, Andrea Lacava, Paolo Bellavista, Michele Polese, and Tommaso Melodia. xdevsm: Streamlining xapp development with a flexible framework for o-ran e2 service models. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 1954–1961, 2024.
- [40] Abiodun Ganiyu, Pranshav Gajjar, and Vijay K Shah. Ai5gtest: Ai-driven specification-aware automated testing and validation of 5g o-ran components. In *18th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 53–64, 2025.
- [41] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proc. International Conference on Machine Learning (ICML)*, 2023.
- [42] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.
- [43] Anastasios Giannopoulos, Sotirios Spantideas, Maria Lamprini Bartsioka, and Panagiotis Trakadas. Interoperable rapp/xapp control over o-ran for mobility-aware dynamic spectrum allocation. *arXiv preprint arXiv:2601.13769*, 2026.
- [44] Anastasios Giannopoulos, Sotirios Spantideas, George Levis, Alexandros Kalafatelis, and Panagiotis Trakadas. Comix: Generalized conflict management in o-ran xapps-architecture, workflow, and a power control case. *IEEE Access*, 2025.
- [45] Ziyang Guo, Zhenyu Chen, Peng Liu, Jianjun Luo, Xun Yang, and Xinghua Sun. Multi-agent reinforcement learning-based distributed channel access for next generation wireless networks. *IEEE Journal on Selected Areas in Communications*, 40(5):1587–1599, 2022.
- [46] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

- [47] Md Arafat Habib, Hao Zhou, Pedro Enrique Iturria-Rivera, Yigit Ozcan, Medhat Elsayed, Majid Bavand, Raimundas Gaigalas, and Melike Erol-Kantarci. Machine learning-enabled traffic steering in o-ran: A case study on hierarchical learning approach. *IEEE Communications Magazine*, 63(1):100–107, 2024.
- [48] Zhiyuan He, Aashish Gottipati, Lili Qiu, Xufang Luo, Kenuo Xu, Yuqing Yang, and Francis Y Yan. Designing network algorithms via large language models. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*, pages 205–212, 2024.
- [49] Zhiyuan He, Aashish Gottipati, Lili Qiu, Xufang Luo, Kenuo Xu, Yuqing Yang, and Francis Y Yan. Designing network algorithms via large language models. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*, pages 205–212, 2024.
- [50] Alexandre Heuillet, Fabien Couthouis, and Natalia. Explainability in deep reinforcement learning. *Knowledge-Based Systems*, 214:106685, 2021.
- [51] Marcin Hoffmann, Salim Janji, Adam Samorzewski, Łukasz Kułacz, Cezary Adamczyk, Marcin Dryjański, Paweł Kryszkiewicz, Adrian Kliks, and Hanna Bogucka. Open RAN xApps design and evaluation: Lessons learnt and identified challenges. *IEEE Journal on Selected Areas in Communications*, 42(2):473–486, 2024.
- [52] Henrik Holm. Bidirectional encoder representations from transformers (bert) for question answering in the telecom domain.: Adapting a bert-like language model to the telecom domain using the electra pre-training approach, 2021.
- [53] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.
- [54] Sirui Hong, Mingchen Zhuge, Jonathan Chen, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In *Proc. International Conference on Learning Representations (ICLR)*, 2024.
- [55] John J Horton. Large language models as simulated economic agents: What can we learn from homo silicus? Technical report, National Bureau of Economic Research, 2023.
- [56] Bin Hu, Chenyang Zhao, Pu Zhang, Zihao Zhou, Yuanhang Yang, Zenglin Xu, and Bin Liu. Enabling intelligent interactions between an agent and an llm: A reinforcement learning approach. *arXiv preprint arXiv:2306.03604*, 2023.
- [57] Bin Hu, Chenyang Zhao, Pu Zhang, Zihao Zhou, Yuanhang Yang, Zenglin Xu, and Bin Liu. Enabling intelligent interactions between an agent and an llm: A reinforcement learning approach. *arXiv preprint arXiv:2306.03604*, 2023.
- [58] Jian Hu, Li Tao, June Yang, and Chandler Zhou. Aligning language models with offline reinforcement learning from human feedback. *arXiv preprint arXiv:2308.12050*, 2023.

- [59] Dong Huang, Qingwen Bu, Jie M Zhang, Michael Luck, and Heming Cui. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*, 2023.
- [60] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *Proc. International Conference on Learning Representations (ICLR)*, 2024.
- [61] Yoichi Ishibashi and Yoshimasa Nishimura. Self-organized agents: A llm multi-agent framework toward ultra large-scale code generation and optimization. *arXiv preprint arXiv:2404.02183*, 2024.
- [62] Pedro Enrique Iturria-Rivera, Han Zhang, Hao Zhou, Shahram Mollahasani, and Melike Erol-Kantarci. Multi-agent team learning in virtualized open radio access networks (O-RAN). *Sensors*, 22(14):5375, 2022.
- [63] Nathan Jay, Noga H. Rotman, P. Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *Proc. International Conference on Machine Learning (ICML)*, 2019.
- [64] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.
- [65] Michail Kalntis, George Iosifidis, José Suárez-Varela, Andra Lutu, and Fernando A Kuipers. Meta-learning-based handover management in nextg o-ran. *IEEE Journal on Selected Areas in Communications*, 2026.
- [66] Athanasios Karapantelakis, Mukesh Shakur, Alexandros Nikou, Farnaz Moradi, Christian Orlog, Fitsum Gaim, Henrik Holm, Doumitrou Daniil Nimara, and Vincent Huang. Using large language models to understand telecom standards. *arXiv preprint arXiv:2404.02929*, 2024.
- [67] Athanasios Karapantelakis, Mukesh Thakur, Alexandros Nikou, Farnaz Moradi, Christian Olrog, Fitsum Gaim, Henrik Holm, Doumitrou Daniil Nimara, and Vincent Huang. Using large language models to understand telecom standards. In *2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, pages 440–446. IEEE, 2024.
- [68] Caner Kilinc, Mahesh K Marina, Muhammad Usama, Salih Ergut, Jon Crowcroft, Tugrul Gundogdu, and Ilhan Akinci. Jade: Data-driven automated jammer detection framework for operational mobile networks. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 1139–1148. IEEE, 2022.
- [69] Mohammadreza Kouchaki and Vuk Marojevic. Actor-critic network for o-ran resource allocation: xapp design, deployment, and analysis. In *2022 IEEE Globecom Workshops (GC Wkshps)*, pages 968–973. IEEE, 2022.

- [70] Dae Cheol Kwon and Xinyu Zhang. CP-AgentNet: Autonomous and Explainable Communication Protocol Design Using Generative Agents. In *Proceedings of IEEE ICNP*, 2025.
- [71] Dae Cheol Kwon and Xinyu Zhang. Cp-agentnet: Autonomous and explainable communication protocol design using generative agents. In *2025 IEEE 33rd International Conference on Network Protocols (ICNP)*, pages 1–12. IEEE, 2025.
- [72] Dae Cheol Kwon and Xinyu Zhang. Open ran conflict agents: Detecting and mitigating xapp conflicts with generative agents. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2026.
- [73] Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. *arXiv preprint arXiv:2303.00001*, 2023.
- [74] Patrick Lewis, Ethan Perez, Aleksandra Piktus, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [75] Dan Li, Dacheng Chen, Baihong Jin, Lei Shi, Jonathan Goh, and See-Kiong Ng. Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks. In *International conference on artificial neural networks*, pages 703–716. Springer, 2019.
- [76] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for “mind” exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [77] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459*, 2024.
- [78] Yuanjie Li, Haotian Deng, Jiayao Li, Chunyi Peng, and Songwu Lu. Instability in distributed mobility management: Revisiting configuration management in 3G/4G mobile networks. In *Proc. ACM SIGMETRICS*, pages 261–272, 2016.
- [79] Yuanjie Li, Chunyi Peng, Zengwen Yuan, Jiayao Li, Haotian Deng, and Tao Wang. MobileInsight: Extracting and analyzing cellular network information on smartphones. In *Proc. ACM MobiCom*, 2016.
- [80] Yuanjie Li, Jiaqi Xu, Chunyi Peng, and Songwu Lu. A first look at unstable mobility management in cellular networks. In *Proc. ACM International Workshop on Mobile Computing Systems and Applications (HotMobile)*, pages 15–20, 2016.
- [81] X. Liang, A. Al-Tahmeesschi, Q. Wang, S. Chetty, C. Sun, and H. Ahmadi. Enhancing energy efficiency in O-RAN through intelligent xapps deployment. In *Proc. IEEE WINCOM*, 2024.

- [82] Xuanyu Liang, Ahmed Al-Tahmeesschi, Qiao Wang, Swarna Chetty, Chenrui Sun, and Hamed Ahmadi. Enhancing energy efficiency in o-ran through intelligent xapps deployment. In *2024 11th International Conference on Wireless Networks and Mobile Communications (WINCOM)*, pages 1–6. IEEE, 2024.
- [83] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- [84] Shuyu Lin, Ronald Clark, Robert Birke, Sandro Schönborn, Niki Trigoni, and Stephen Roberts. Anomaly detection for time series using vae-lstm hybrid model. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4322–4326. Ieee, 2020.
- [85] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.
- [86] Yanbing Liu, Gunpeng Guo, and Chunyi Peng. Demystifying secondary radio access failures in 5G. In *Proc. ACM International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2024.
- [87] Yanbing Liu and Chunyi Peng. A close look at 5G in the wild: Unrealized potentials and implications. In *Proc. IEEE INFOCOM*, 2023.
- [88] Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*, 2023.
- [89] Fatemeh Lotfi and Fatemeh Afghah. Open ran lstm traffic prediction and slice management using deep reinforcement learning. In *2023 57th Asilomar Conference on Signals, Systems, and Computers*, pages 646–650. IEEE, 2023.
- [90] Jie Lu, Peihao Yan, and Huacheng Zeng. Eexapp: Gnn-based reinforcement learning for radio unit energy optimization in 5g o-ran. *arXiv preprint arXiv:2602.09206*, 2026.
- [91] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [92] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.
- [93] Ali Maatouk, Fadhel Ayed, Nicola Piovesan, Antonio De Domenico, Merouane Debbah, and Zhi-Quan Luo. Teleqna: A benchmark dataset to assess large language models telecommunications knowledge. *arXiv preprint arXiv:2310.15051*, 2023.

- [94] Ali Maatouk, Fadhel Ayed, Nicola Piovesan, Antonio De Domenico, Merouane Debbah, and Zhi-Quan Luo. Teleqna: A benchmark dataset to assess large language models telecommunications knowledge. *IEEE Network*, 2025.
- [95] Ali Maatouk, Nicola Piovesan, Fadhel Ayed, Antonio De Domenico, and Merouane Debbah. Large language models for telecom: Forthcoming impact on the industry. *IEEE Communications Magazine*, 2024. VERIFY volume/number/pages.
- [96] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding BGP misconfiguration. *ACM SIGCOMM Computer Communication Review*, 32(4):3–16, 2002.
- [97] Ajay Mahimkar, Zihui Ge, Xuan Liu, Yusef Shaqalle, Yu Xiang, Jennifer Yates, Shomik Pathak, and Rick Reichel. Aurora: Conformity-based configuration recommendation to improve LTE/5G service. In *Proc. ACM Internet Measurement Conference (IMC)*, 2022.
- [98] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proc. ACM Workshop on Hot Topics in Networks (HotNets)*, 2016.
- [99] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with Pensieve. In *Proc. ACM SIGCOMM*, 2017.
- [100] Misaki Matsuura, Young Kyun Jung, and Ser Nam Lim. Visual-llm zero-shot classification. 2023.
- [101] Abdelkader Mekrache, Adlen Ksentini, and Christos Verikoukis. Intent-based management of next-generation networks: An LLM-centric approach. *IEEE Network*, 38(5):29–36, 2024.
- [102] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [103] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.
- [104] Reiichiro Nakano, William Hilton, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.
- [105] Explainability and auditability in ml: Definitions, techniques, and tools. <https://neptune.ai/blog/explainability-auditability-ml-definitions-techniques-tools>, 2023.
- [106] Rasoul Nikbakht, Mohamed Benzaghta, and Giovanni Geraci. Tspec-llm: An open-source dataset for llm understanding of 3gpp specifications. In *2024 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2024.
- [107] O-RAN Alliance. AI/ML Workflow Description and Requirements. Technical Report O-RAN.WG2.AIML-v01.03, O-RAN Working Group 2, 2021.
- [108] O-RAN Alliance. O-RAN Architecture Description, v10.00, 2023. <https://www.o-ran.org/specifications>.

- [109] O-RAN Alliance. O-RAN WG3 Technical Specification: Near-Real-Time RAN Intelligent Controller E2 Service Model (E2SM) – RAN Control. Technical Specification O-RAN.WG3.TS.E2SM-RC-R004-v07.00, O-RAN Alliance Working Group 3, 2025. Compatibility entry for dissertation citation key; verify version if v06 is required.
- [110] O-RAN Alliance. O-RAN WG3 Technical Specification: Near-Real-Time RAN Intelligent Controller E2 Service Model (E2SM) – RAN Control. Technical Specification O-RAN.WG3.TS.E2SM-RC-R004-v07.00, O-RAN Alliance Working Group 3, 2025.
- [111] O-RAN Alliance Working Group 1. O-RAN architecture description. Technical report, O-RAN Alliance, TODO. VERIFY document number and version.
- [112] O-RAN Alliance Working Group 2. Non-real-time RAN intelligent controller functional architecture. Technical report, O-RAN Alliance, 2021. VERIFY document number and version.
- [113] O-RAN Alliance Working Group 3. Near-real-time RAN intelligent controller architecture. Technical report, O-RAN Alliance, 2022. VERIFY document number and version.
- [114] O-RAN Alliance Working Group 3. Conflict mitigation. Technical Report O-RAN.WG3.TR.ConMit-R004-v01.00, O-RAN Alliance, 2024.
- [115] O-RAN Alliance Working Group 3. E2 service model (E2SM), KPM. Technical report, O-RAN Alliance, TODO. VERIFY document number and version.
- [116] O-RAN Alliance Working Group 3. E2 service model (E2SM), RAN control. Technical report, O-RAN Alliance, TODO. VERIFY document number and version.
- [117] OpenAI. gpt-4o-mini. <https://platform.openai.com>, 2025. Accessed via OpenAI API.
- [118] Joon Sung Park, Joseph O’Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, 2023.
- [119] Joon Sung Park, Joseph C. O’Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proc. ACM Symposium on User Interface Software and Technology (UIST)*, 2023.
- [120] Chunyi Peng, Yuanjie Li, Zhuoran Li, Jie Zhao, and Jiaqi Xu. Understanding and diagnosing real-world femtocell performance problems. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.
- [121] Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, pages 745–750, 2007.

- [122] Nicola Piovesan, Antonio De Domenico, and Fadhel Ayed. Telecom language models: Must they be large? *arXiv preprint arXiv:2403.04666*, 2024.
- [123] Michele Polese, Leonardo Bonati, Salvatore D’Oro, Stefano Basagni, and Tommaso Melodia. Colo-ran: Developing machine learning-based xapps for open ran closed-loop control on programmable experimental platforms. *IEEE Transactions on Mobile Computing*, 22(10):5787–5800, 2022.
- [124] Michele Polese, Leonardo Bonati, Salvatore D’Oro, Stefano Basagni, and Tommaso Melodia. CoLO-RAN: Developing machine learning-based xApps for open RAN closed-loop control on programmable experimental platforms. *IEEE Transactions on Mobile Computing*, 2023. VERIFY volume/number/pages.
- [125] Michele Polese, Leonardo Bonati, Salvatore D’Oro, Stefano Basagni, and Tommaso Melodia. Understanding O-RAN: Architecture, interfaces, algorithms, security, and research challenges. *IEEE Communications Surveys & Tutorials*, 25(2):1376–1411, 2023.
- [126] Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.
- [127] Yujia Qin and Liang. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.
- [128] Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, et al. Large language models are effective text rankers with pairwise ranking prompting. *arXiv preprint arXiv:2306.17563*, 2023.
- [129] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of machine learning research*, 22(268):1–8, 2021.
- [130] Raoul Raftopoulos, Salvatore D’Oro, Tommaso Melodia, and Giovanni Schembra. Drl-based latency-aware network slicing in o-ran with time-varying slas. *arXiv preprint arXiv:2401.05042*, 2024.
- [131] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.
- [132] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “why should I trust you?”: Explaining the predictions of any classifier. In *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016.
- [133] Mustafa Safdari, Greg Serapio-García, Clément Crepy, Stephen Fitz, Peter Romero, Luning Sun, Marwa Abdulhai, Aleksandra Faust, and Maja Matarić. Personality traits in large language models. *arXiv preprint arXiv:2307.00184*, 2023.

- [134] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [135] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [136] Lloyd S Shapley et al. A value for n-person games. 1953.
- [137] Pragya Sharma, Shihua Sun, Shachi Deshpande, Angelos Stavrou, and Haining Wang. Towards xapp conflict evaluation with explainable machine learning and causal inference in o-ran. *arXiv preprint arXiv:2510.13031*, 2025.
- [138] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [139] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.
- [140] Software Radio Systems. ORAN-SC-RIC: Simplified O-RAN SC near-RT RIC with Docker deployment. <https://github.com/srsran/oran-sc-ric>, 2024.
- [141] Software Radio Systems. srsRAN project. https://github.com/srsran/srsRAN_Project_TODO.
- [142] Jiayang Song, Zhehua Zhou, Jiawei Liu, Chunrong Fang, Zhan Shu, and Lei Ma. Self-refined large language model as automated reward function designer for deep reinforcement learning in robotics. *arXiv preprint arXiv:2309.06687*, 2023.
- [143] Paweł Sroka, Łukasz Kułacz, Salim Janji, Marcin Dryjański, and Adrian Kliks. Policy-based traffic steering and load balancing in o-ran-based vehicle-to-network communications. *IEEE Transactions on Vehicular Technology*, 73(7):9356–9369, 2024.
- [144] AK Subudhi, Alex Piccioni, Venkateswarlu Gudepu, Andrea Marotta, Fabio Graziosi, RM Hegde, and Koteswararao Kondepudi. Performance evaluation of scheduling scheme in o-ran 5g network using ns-3. In *2024 IEEE Future Networks World Forum (FNWF)*, pages 590–595. IEEE, 2024.
- [145] Chuanhao Sun, Ujjwal Pawar, Molham Khoja, Xenofon Foukas, Mahesh K. Marina, and Bozidar Radunovic. Spotlight: Accurate, explainable and efficient anomaly detection for open ran. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 923–937, 2024.

- [146] Chuanhao Sun, Ujjwal Pawar, Molham Khoja, Xenofon Foukas, Mahesh K. Marina, and Bozidar Radunovic. Spotlight: Accurate, explainable and efficient anomaly detection for open ran. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 923–937, 2024.
- [147] Chuanhao Sun, Ujjwal Pawar, Molham Khoja, Xenofon Foukas, Mahesh K Marina, and Bozidar Radunovic. Spotlight: Accurate, explainable and efficient anomaly detection for open ran. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 923–937, 2024.
- [148] Hao Sun. Offline prompt evaluation and optimization with inverse reinforcement learning. *arXiv preprint arXiv:2309.06553*, 2023.
- [149] James Surowiecki. *The wisdom of crowds*. Anchor, 2005.
- [150] sysmocom. sysmoISIM-SJA2 Programmable USIM Card. Product documentation. Bibliographic details should be verified before final submission.
- [151] Andrew Szot, Max Schwarzer, Harsh Agrawal, Bogdan Mazoure, Rin Metcalf, Walter Talbott, Natalie Mackraz, R Devon Hjelm, and Alexander T Toshev. Large language models as generalizable policies for embodied tasks. In *The Twelfth International Conference on Learning Representations*, 2023.
- [152] Sönke Tenckhoff, Mario Koddenbrock, and Erik Rodner. Llmstructbench: Benchmarking large language model structured data extraction. *arXiv preprint arXiv:2602.14743*, 2026.
- [153] TODO. Vivisecting mobility management in 5G cellular networks. In *Proc. ACM SIGCOMM*, 2022. VERIFY author list.
- [154] TODO. ORAN-Bench-13K: An open source benchmark for assessing LLMs in open radio access networks. *arXiv preprint arXiv:2407.06245*, 2024. VERIFY author list.
- [155] Hoang Duy Trinh, Engin Zeydan, Lorenza Giupponi, and Paolo Dini. Detecting mobile traffic anomalies through physical control channel fingerprinting: A deep semi-supervised approach. *IEEE Access*, 7:152187–152201, 2019.
- [156] Shreshth Tuli, Giuliano Casale, and Nicholas R Jennings. Tranad: Deep transformer networks for anomaly detection in multivariate time series data. *arXiv preprint arXiv:2201.07284*, 2022.
- [157] Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don’t always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965, 2023.
- [158] Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language models don’t always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

- [159] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [160] Alessio Vicario, Antonino Pagano, Alessandra Dino, Daniele Croce, and Ilenia Tinnirello. Towards intelligent mobility management: Customized handover in 5g o-ran networks. In *Proceedings of the ACM Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization*, pages 65–72, 2025.
- [161] Abdul Wadud, Fatemeh Golpayegani, and Nima Afraz. Conflict management in the near-rt-ric of open ran: A game theoretic approach. In *2023 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 479–486. IEEE, 2023.
- [162] Abdul Wadud, Fatemeh Golpayegani, and Nima Afraz. Conflict management in the near-rt-ric of open ran: A game theoretic approach. In *2023 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 479–486. IEEE, 2023.
- [163] Abdul Wadud, Fatemeh Golpayegani, and Nima Afraz. QACM: QoS-aware xApp conflict mitigation in open RAN. *arXiv preprint arXiv:2405.07324*, 2024.
- [164] Abdul Wadud, Fatemeh Golpayegani, and Nima Afraz. Qacm: Qos-aware xapp conflict mitigation in open ran. *IEEE Transactions on Green Communications and Networking*, 2024.
- [165] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.
- [166] Jianing Wang, Qiushi Sun, Xiang Li, and Ming Gao. Boosting language models reasoning with chain-of-knowledge prompting. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4958–4981, 2024.
- [167] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *Proc. International Conference on Learning Representations (ICLR)*, 2023.
- [168] Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. Unleashing cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *arXiv preprint arXiv:2307.05300*, 1(2):3, 2023.

- [169] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
- [170] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V. Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [171] Keith Winstein and Hari Balakrishnan. TCP ex machina: Computer-generated congestion control. In *Proc. ACM SIGCOMM*, 2013.
- [172] Duo Wu, Xianda Wang, Yaqi Qiao, Zhi Wang, Junchen Jiang, Shuguang Cui, and Fangxin Wang. NetLLM: Adapting large language models for networking. In *Proc. ACM SIGCOMM*, pages 661–678, 2024.
- [173] Qingyun Wu, Gagan Bansal, Jieyu Zhang, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023. VERIFY venue.
- [174] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.
- [175] Daliang Xu, Wangsong Yin, Xin Jin, Ying Zhang, Shiyun Wei, Mengwei Xu, and Xuanzhe Liu. Llmcad: Fast and scalable on-device large language model inference. *arXiv preprint arXiv:2309.04255*, 2023.
- [176] Derong Xu, Wei Chen, Wenjun Peng, Chao Zhang, Tong Xu, Xiangyu Zhao, Xian Wu, Yefeng Zheng, Yang Wang, and Enhong Chen. Large language models for generative information extraction: A survey. *Frontiers of Computer Science*, 18(6):186357, 2024.
- [177] Shichang Xu, Ashkan Nikraves, and Z Morley Mao. Leveraging context-triggered measurements to characterize lte handover performance. In *International Conference on Passive and Active Network Measurement*, pages 3–17. Springer, 2019.
- [178] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proc. International Conference on Learning Representations (ICLR)*, 2023.
- [179] Y. Yu, T. Wang, and S. C. Liew. Deep-reinforcement learning multiple access for heterogeneous wireless networks. Technical Report Tech. Rep. 1, Chinese University of Hong Kong, Hong Kong, 2020. Available online: <https://github.com/YidingYu/DLMA/blob/master/DLMA-benchmark.pdf>.
- [180] Yiding Yu, Soung Chang Liew, and Taotao Wang. Multi-agent deep reinforcement learning multiple access for heterogeneous wireless networks with imperfect channels. *IEEE Transactions on Mobile Computing*, 21(10):3718–3730, 2021.

- [181] Yiding Yu, Soung Chang Liew, and Taotao Wang. Non-uniform time-step deep Q-network for carrier-sense multiple access in heterogeneous wireless networks. *IEEE Transactions on Mobile Computing*, 20(9):2848–2861, 2021.
- [182] Yiding Yu, Taotao Wang, and Soung Chang Liew. Deep-reinforcement learning multiple access for heterogeneous wireless networks. *IEEE Journal on Selected Areas in Communications*, 37(6):1277–1290, 2019.
- [183] Yiding Yu, Taotao Wang, and Soung Chang Liew. Deep-reinforcement learning multiple access for heterogeneous wireless networks. *IEEE journal on selected areas in communications*, 37(6):1277–1290, 2019.
- [184] Zengwen Yuan, Qianru Li, Yuanjie Li, Songwu Lu, Chunyi Peng, and George Varghese. Resolving policy conflicts in multi-carrier cellular access. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 147–162, 2018.
- [185] Noe M Yungaicela-Naula, Vishal Sharma, and Sandra Scott-Hayward. Rslaq-a robust sla-driven 6g o-ran qos xapp using deep reinforcement learning. *IEEE Transactions on Mobile Computing*, 2026.
- [186] Han Zhang, Hao Zhou, and Melike Erol-Kantarci. Team learning-based resource allocation for open radio access network (O-RAN). In *Proc. IEEE International Conference on Communications (ICC)*, pages 4938–4943, 2022.
- [187] Zhehui Zhang, Yanbing Liu, Qianru Li, Zizheng Liu, Chunyi Peng, and Songwu Lu. Dependent misconfigurations in 5G/4.5G radio resource control. *Proceedings of the ACM on Networking*, 1(1):1–20, 2023. Article 2.
- [188] Xiaohui Zhao, Hanyang Ma, Yuan Jin, and Jianguo Yao. Measuring instability of mobility management in cellular networks. *IEEE Network*, 32(5):138–144, 2018.
- [189] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*, 2023.
- [190] Caleb Ziems, William Held, Omar Shaikh, Jiaao Chen, Zhehao Zhang, and Diyi Yang. Can large language models transform computational social science? *Computational Linguistics*, 50(1):237–291, 2024.