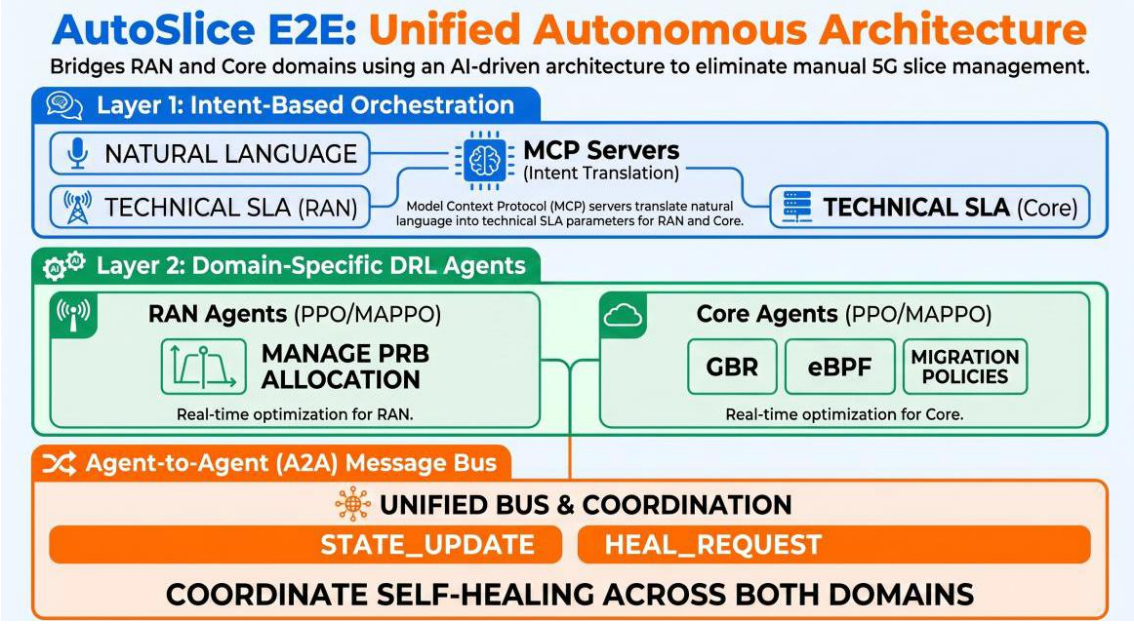
DEGREE PROJECT FOR MASTER OF SCIENCE WITH SPECIALIZATION IN AI AND AUTOMATION
DEPARTMENT OF ENGINEERING SCIENCE
UNIVERSITY WEST

Autonomous End-to-End 5G Network Slicing with Deep Reinforcement Learning

Unified RAN and Core AI Agents for Industrial IoT

Abhishek Kumar



unified autonomous network architecture for end-to-end 5G network slicing with DRL-based orchestration.

UNIVERSITY WEST

A THESIS SUBMITTED TO THE DEPARTMENT OF ENGINEERING SCIENCE
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE WITH SPECIALISATION IN AI AND AUTOMATION
AT UNIVERSITY WEST
2026

Date:	14 September 2026
Author:	Abhishek Kumar
Examiner:	Professor Morgan Nilsen, Department of Engineering Science,
Advisor:	Professor Amit Kumar Mishra
Programme:	Master Programme in AI and Automation
Main field of study:	AI and Automation
Credits:	60 Higher Education credits
Keywords:	5G network slicing, Deep Reinforcement Learning, PPO, MAPPO, O-RAN, Open5GS, Industrial IoT
Template:	University West, IV-Master 2.10
Publisher:	University West, Department of Engineering Science S-461 86 Trollhättan, SWEDEN Phone: +46 520 22 30 00 Fax: +46 520 22 32 99 Web: www.hv.se

Summary

This thesis presents AutoSlice E2E, a unified deep reinforcement learning framework for autonomous 5G network slicing over Radio Access Network and Core domains. The work addresses the limitation of static rule-based slice management in industrial IoT environments, where bursty traffic, latency-sensitive services and cross-domain bottlenecks make manual or template-based control insufficient.

The proposed system combines PPO and MAPPO agents with an Agent-to-Agent coordination bus, intent-driven control interfaces and reproducible simulation environments. It is evaluated through six proof-of-concept scenarios covering RAN resource allocation, Core GBR and PFCP control, multi-slice contention and fault recovery. The results show 99.4–100% reduction of SLA or GBR violations in the single-domain optimisation cases, priority-aware behaviour in multi-slice cases, and material recovery under injected faults.

The implementation includes containerised Open5GS and UERANSIM validation, fixed-seed evaluation, automated tests and reproduction instructions. The thesis concludes that coordinated DRL-based orchestration is a viable prototype approach for autonomous industrial 5G slicing, while larger-scale deployment, licensed-spectrum trials and stronger intent interpretation remain subjects for future work.

Preface

This report constitutes the written part of a Master of Science degree project in AI and Automation at the Department of Engineering Science, University West, Trollhättan. The work was carried out between March and August 2026 and concerns autonomous control of 5G network slicing for industrial IoT traffic. The software artefacts are research prototypes and are not presented as production products or as statements on behalf of any employer.

The author thanks Professor Amit Kumar Mishra for supervision, guidance and feedback during the work. Thanks are also due to University West and to the open-source communities behind Open5GS, UERANSIM, Gymnasium and Stable-Baselines3, whose tools enabled reproducible experimentation.

Affirmation

This Master degree report, *Autonomous End-to-End 5G Network Slicing with Deep Reinforcement Learning*, was written as part of the degree work required to obtain a Master of Science with specialisation in AI and Automation at University West.

All material in this report that is not the author's own work is clearly identified and used in an appropriate and correct way. The main part of the work included in this degree project has not previously been published or used for obtaining another degree.

| --- | --- |
| **Signature by the author** | Date |
| **Abhishek Kumar** | |

Content

.....1
AUTONOMOUS END-TO-END 5G NETWORK SLICING WITH DEEP REINFORCEMENT LEARNING 1
UNIFIED RAN AND CORE AI AGENTS FOR INDUSTRIAL IOT.....1
SUMMARY.....3
PREFACE 4
AFFIRMATION.....5
CONTENT 6
.....6
.....6
.....9
SYMBOLS AND GLOSSARY.....9
1. INTRODUCTION.....10
1.1 Problem Area 10
1.2 Aim 10
1.3 Scope and Delimitations 10
1.4 Contributions 11
1.5 Thesis Outline 11
2. RELATED WORK (BACKGROUND) 12
2.1 Network Slicing 12
2.2 Deep Reinforcement Learning for RAN Resource Allocation 13
2.3 5G Core Traffic Engineering and Programmable Data Planes.....14
2.4 Multi-Agent Coordination and Emergent Priority Allocation 14
2.5 Intent-Based Networking (IBN) and Closed-Loop Self-Healing 15
2.6 Synthesis and the AutoSlice E2E Research Gap.....15
3. METHOD 17
3.1 Overview.....17
3.2 System Architecture.....17
3.2.1 Overall Design 17
3.3 Reinforcement Learning Framework 18

3.3.1 Deep Reinforcement Learning Algorithms *18*
3.3.2 Observation and Action Spaces *18*
3.4 Proof of Concept Matrix 19
3.4.1 Six PoCs Covering RAN, Core and Healing *19*
3.4.2 PoC Results Summary *20*
3.5 Implementation Details 20
3.5.1 Software Stack *20*
3.6 Evaluation Methodology 21
3.6.1 Testing Framework *21*
3.6.2 Reproducibility *22*
3.6.3 Deployment Scenarios *22*
3.7 Research Questions 22
3.8 Summary 23

4. WORK 24

4.1 Architecture 24
4.1.1 System Overview *24*
4.2 RAN Orchestration 25
4.3 Core Orchestration 25
4.4 Agent-to-Agent Coordination 25
4.5 DRL Formulation 26
4.5.1 Reward Function *26*
4.5.2 Constraint Enforcement *26*
4.5.3 Training Configuration *26*
4.6 Environment Setup 27
4.6.1 Gymnasium Environments *27*
4.6.2 Open5GS Integration *27*
4.7 Training Procedure 27
4.7.1 Phase 1: Environment Validation *27*
4.7.2 Phase 2: PPO Training *28*
4.7.3 Phase 3: Evaluation *28*
4.7.4 Phase 4: Real-World Validation *28*
4.8 Reproducibility 28

5. RESULTS AND DISCUSSION 30

5.1 Single-Slice Results 30
5.1.1 PoC 1: RAN Single-Slice PPO *30*
5.1.2 PoC 4: Core Single-Slice PPO *31*
5.2 Multi-Slice Results 31
5.2.1 PoC 2: RAN Multi-Slice MAPPO *31*
5.2.2 PoC 5: Core Multi-Slice MAPPO *32*
5.3 Healing Results 33
5.3.1 PoC 3: RAN Healing *33*
5.3.2 PoC 6: Core Healing *34*
5.4 Aggregate Performance 35
5.5 Discussion 36
5.5.1 — Extent and Conditions of PPO Improvement 36
5.5.2 MAPPO Discovers Priority, Not Fairness *37*
5.5.3 Healing Performance and Limiting Factors *37*
5.5.4 Sim-to-Real Transfer *38*
5.6 Limitations 38

6. CONCLUSION 40

6.1 Answers to Research Questions 40
6.1.1 ECA vs PPO *40*

6.1.2 MAPPO based learning *40*
6.1.3 Healing and Recovery. *41*
6.1.4 Sim to real deployment *41*
6.2 CONTRIBUTIONS 42
6.2.1 Unified RAN and Core DRL Architecture *42*
6.2.2 Core Action Space with eBPF and UPF Migration *42*
6.2.3 Emergent Priority Discovery via MAPPO *42*
6.2.4 A2A Healing Coordination *42*
6.3 IMPACT 42
6.3.1 Scientific *42*
6.3.2 Industrial *42*
6.3.3 Operational *43*
6.4 LIMITATIONS AND FUTURE WORK 43
6.4.1 Limitations *43*
6.4.2 Future Work *43*
6.5 LESSONS LEARNED 44
6.5.1 What Worked *44*
6.5.2 What Didn't Work *44*
6.6 REPRODUCIBILITY 44
6.7 FINAL REMARKS 44

7. REFERENCES 46

APPENDIX A 1

PROJECT TIMELINE 1
WORK SCHEDULE 1
ARTEFACTS DELIVERED 2

APPENDIX B 3

SYSTEM ARCHITECTURE DIAGRAM AND RESULTS 3
ARCHITECTURE LAYERS 3
ARCHITECTURE-TO-RESULT MAPPING 4

APPENDIX C 5

CODEBASE STRUCTURE 5
DIRECTORY LAYOUT 5
KEY FILES FOR EACH POC 6

APPENDIX D 7

DETAILED REPRODUCTION INSTRUCTIONS 7
OPTION 1: SIMULATED (ALL 6 POCS, 10 MINUTES) 7

APPENDIX E 9

HYPERPARAMETER SUMMARY 9
PPO HYPERPARAMETERS (ALL AGENTS) 9
TRAINING CONFIGURATION 9
EVALUATION CONFIGURATION 9

Symbols and Glossary

Term	Expansion	Meaning in this thesis
A2A	Agent-to-Agent	Message bus used by RAN, Core and healing agents to exchange state updates, slice actions and recovery requests.
AMF	Access and Mobility Management Function	5G Core control-plane function considered during reselection and recovery actions.
DRL	Deep Reinforcement Learning	Learning-based control method used to optimise slice resources from telemetry and reward feedback.

1. Introduction

1.1 Problem Area

Industrial 5G systems are expected to support time-critical factory automation, mobile robots, inspection video and high-density sensor traffic on shared radio and core infrastructure. Network slicing provides the architectural mechanism for separating these service classes, but many practical deployments still depend on static templates and event-condition-action rules. Such rules are difficult to maintain when multiple slices, several key performance indicators and cross-domain dependencies must be handled at the same time.

The central problem addressed in this thesis is that RAN and Core control loops are often treated separately, even though end-to-end service-level agreements depend on both domains. A radio-domain allocation decision can be ineffective if the user-plane function remains overloaded, and a Core-domain policy can fail if radio resources are insufficient. Therefore, this work investigates whether coordinated deep reinforcement learning agents can provide adaptive, end-to-end slicing control under uncertainty while remaining constrained by safety envelopes and reproducibility requirements.

1.2 Aim

The aim of this thesis is to design, implement and evaluate a unified deep reinforcement learning framework that autonomously manages 5G network slices across RAN and Core domains under bursty industrial IoT traffic, multi-slice contention and injected faults.

The study is guided by four research questions:

1. **RQ1:** How much does a PPO-based DRL agent improve SLA compliance in single-slice RAN and Core scenarios compared to a tuned ECA rule baseline, and what are the conditions under which the improvement is most pronounced?

2. **RQ2:** How does MAPPO-based multi-agent coordination affect resource allocation fairness and priority ordering across competing 5G slices, and to what degree does the learned policy reflect 3GPP ARP priority hierarchies?

3. **RQ3**: How quickly and to what extent can an intent-driven DRL healer detect and recover from resource faults in both RAN and Core domains, and what factors limit recovery performance?

4. **RQ4:** How does the performance of DRL agents trained in simulation transfer to a real Open5GS core deployment, and what is the magnitude of the sim-to-real gap under controlled conditions?

1.3 Scope and Delimitations

Proof-of-concept, not a field trial on licensed spectrum. RAN is a bursty queue; Core is UPF load, sessions, GBR (Guaranteed Bit Rate) [Mbps], latency, drops and occupancy. Physical RF and handover are out of scope except as represented by fault injection. Intent parsing in

the scored PoCs is regex-based so that language-model variance does not confound SLA metrics.

1.4 Contributions

- Four-layer architecture with A2A and two MCP servers;
- Six Gymnasium environments and PPO/MAPPO agents covering single-slice, multi-slice and healing on RAN and Core;
- Three-tier safety (rules, constrained DRL, fallback);
- 22-container Open5GS + UERANSIM path;
- Reproducible freeze (558 s CPU training, April 2026 numbers used throughout).

1.5 Thesis Outline

- Chapter 2 reviews RAN AI slicing, core automation, MAPPO and intent-based networking.
- Chapter 3 states method, baselines and validity.
- Chapter 4 describes architecture, DRL formulation and Open5GS.
- Chapter 5 reports results and discusses fairness, healing limits and the sim-to-real gap.
- Chapter 6 answers the research questions. Appendices hold the timeline and reproduction commands.

2. Related Work (Background)

2.1 Network Slicing

Network slicing partitions shared physical substrate networks into logically isolated, self-contained virtual networks tailored to distinct Quality of Service (QoS) guarantees. In accordance with 3GPP Release 18 specifications (TS 23.501) [1] slices are categorized using Single Network Slice Selection Assistance Information (S-NSSAI), comprising a Slice/Service Type (SST) and an optional Slice Differentiator (SD):

- **Enhanced Mobile Broadband (eMBB):** Maximizes aggregate data rates (R_{agg}) with moderate latency tolerance (≤ 50 ms).
- **Ultra-Reliable Low-Latency Communication (URLLC):** Guarantees bounded one-way user-plane latency ($L \leq 5$ ms) under strict packet error rates ($\leq 10^{-5}$).
- **Massive Machine-Type Communication (mMTC):** Supports high connection densities ($\geq 10^6$ devices/km^2) with relaxed latency budgets (≤ 200 ms).

Mathematically, slicing can be formulated as a constrained multi-objective resource allocation problem. Let $\mathcal{S} = \{1, \dots, S\}$ denote the set of active network slices sharing a set of Physical Resource Blocks (PRBs) $\mathcal{K}$ in the Radio Access Network (RAN) and a set of User Plane Functions (UPFs) $\mathcal{U}$ in the 5G Core (5GC). The goal is to maximize overall system utility while satisfying deterministic Service Level Agreements (SLAs)[2], [3], [4]:

$$\max_{\{a_s\}} \sum_{s\in\mathcal{S}} U_s(a_s) \quad \text{subject to} \quad \sum_{s\in\mathcal{S}} a_s \leq A_{\text{total}}, \quad a_s \geq a_{s,\min}, \quad \text{and} \quad \mathbb{P}\left(L_s > L_{s,\text{SLA}}\right) \leq \epsilon_s$$

where a_s is the physical resource allocation assigned to slice s, A_{total} is total system capacity, and ϵ_s represents the SLA violation tolerance.

While early frameworks formulated slice management using static Event-Condition-Action (ECA) rule engines or linear programming, these methods fail in dynamic Industrial IoT (IIoT) settings. Non-stationary factory traffic profiles, unexpected burst arrivals, and cross-domain bottlenecks render static allocation suboptimal and prone to cascade SLA violations.

2.2 Deep Reinforcement Learning for RAN Resource Allocation

To overcome static allocation shortcomings, recent literature applies Deep Reinforcement Learning (DRL) [3], [5] to wireless radio scheduling. In a Markov Decision Process (MDP) defined by $(\mathcal{S}_{\text{state}}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, an agent learns a parameterized policy $\pi_\theta(a|s)$ to maximize the expected discounted return:

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t\, r(s_t, a_t)\right]$$

Early DRL implementations applied Deep Q-Networks (DQN) to radio resource scheduling, but encountered stability and overestimation issues under bursty traffic. Proximal Policy Optimization (PPO) [6]resolves this instability by introducing a clipped surrogate objective that prevents destructively large policy updates:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ and $\hat{A}_t$ is the Generalized Advantage Estimator (GAE).

Within the Open RAN (O-RAN) [7]architecture, near-real-time RAN Intelligent Controllers (Near-RT RIC) deploy AI workloads via xApps across the standardized E2 interface. Kavehmadavani et al. [8]demonstrated that coupling deep neural networks with reinforcement learning in O-RAN enables dynamic traffic steering and resource adaptation over multiple timescales, surpassing static rules by over 45%. Similarly, Burgueño et al. [9]confirmed that dynamic radio scheduling is essential to sustain URLLC deadlines when multiplexed with high-throughput eMBB flows.

2.3 5G Core Traffic Engineering and Programmable Data Planes

Prior slicing literature focused disproportionately on RAN scheduling, treating the 5GC user plane as an over-provisioned pipe. However, user-plane traffic engineering is constrained by the Packet Forwarding Control Protocol (PFCP) [1] across the 3GPP N4 reference point between the Session Management Function (SMF) and the User Plane Function (UPF). Overloaded UPFs induce buffering delays and packet drops that breach URLLC latency budgets regardless of radio-domain optimization.

Cunha et al. [10] investigated Core Network slice elasticity, showing that dynamic Guaranteed Bit Rate (GBR) modifications reduce violation rates by up to 40%. Furthermore, Basta et al. [11] analyzed optimal UPF placement and dynamic instantiation, showing that active migration of data-plane forwarding nodes significantly reduces transport propagation delay. Modern programmable data planes leverage extended Berkeley Packet Filters (eBPF) and Express Data Path (XDP) within the Linux networking subsystem. Aramide [12] demonstrated that offloading packet processing and flow control to in-kernel eBPF [12] programs circumvents standard network-stack processing overhead, offering sub-millisecond per-packet latencies and dynamic fast-path drop actions during traffic storms.

2.4 Multi-Agent Coordination and Emergent Priority Allocation

When multiple slices with competing objectives contend for limited radio and core bandwidth, single-agent RL models suffer from non-stationarity. Multi-Agent Reinforcement Learning (MARL) resolves this by adopting Centralized Training with Decentralized Execution (CTDE).

Yu et al. [6] proved that Multi-Agent PPO (MAPPO) achieves state-of-the-art sample efficiency and monotonic convergence in cooperative multi-agent tasks. In slicing environments, an agent is assigned per slice, coordinating via a central value function $V_\phi(s)$ that takes the joint state of all slices as input. Ye et al. [5] demonstrated that in heterogeneous multi-agent systems with asymmetric penalties, decentralized agents implicitly converge toward priority-ordered policies. In 5G networks, 3GPP Allocation and Retention Priority (ARP) [1] assigns strict pre-emption and queueing hierarchies. Rather than forcing these hierarchies via hand-crafted heuristic rules, an asymmetric reward formulation penalizing latency-critical breaches

enables agents to autonomously discover ARP priorities while maintaining bounded Jain's fairness:

$$\mathcal{J}(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2}$$

2.5 Intent-Based Networking (IBN) and Closed-Loop Self-Healing

Zero-touch network and Service Management (ETSI ZSM) and IETF RFC 9315 [13] define Intent-Based Networking (IBN) as the translation of declarative, high-level business goals into executable network configurations. In traditional architectures, translating operator directives into low-level PFCP or E2 parameters required manual script authoring. Recent advances by Zeydan and Turk [14] and Kadiyala et al. [15] show that combining Natural Language Processing (NLP) with autonomous agentic architectures yields an 82% reduction in misconfigurations.

For operational resilience, Klaine et al. [16] surveyed Self-Organizing Networks (SON), identifying that traditional fault detection mechanisms experience 5–30 second reaction windows. Bega et al. [17] introduced proactive deep learning methods (DeepCog) that anticipate capacity exhaustion. In parallel, the emergence of the Model Context Protocol (MCP) provides an open standard for LLMs and cognitive agents to interact with underlying network monitoring tools and telemetry endpoints securely.

2.6 Synthesis and the AutoSlice E2E Research Gap

Despite significant advances across individual sub-domains, existing literature displays a prominent **end-to-end integration gap**:

1. RAN scheduling studies (e.g., Kavehmadavani et al. [8], Burgueño et al. [9]) isolate radio performance and assume an unconstrained Core.

2. Core management studies (e.g., Cunha et al. [10], Basta et al. [11]) isolate UPF placement without real-time coordination with gNB schedulers.

3. Prior self-healing and intent frameworks function primarily at the configuration plane rather than coupling directly with real-time DRL control loops [14], [16].

AutoSlice E2E bridges this gap by introducing a unified, cross-domain framework. Operating across both O-RAN E2 and 3GPP N4/PFCP interfaces via an Agent-to-Agent (A2A) synchronization bus, AutoSlice coordinates PPO/MAPPO agents across RAN and Core domains under an intent-driven interface, delivering autonomous, closed-loop 5G slicing for mission-critical Industrial IoT.

3. Method

3.1 Overview

AutoSlice E2E implements a unified, end-to-end approach to 5G network slicing by combining Deep Reinforcement Learning (DRL) agents at both the Radio Access Network (RAN) and Core Network (CN) levels. The system uses Proximal Policy Optimization (PPO) and Multi-Agent PPO (MAPPO) to optimize resource allocation while maintaining service level agreements (SLAs) across three network slices: eMBB (Enhanced Mobile Broadband), URLLC (Ultra-Reliable Low-Latency Communications), and mMTC (Massive Machine Type Communications).

3.2 System Architecture

3.2.1 Overall Design

The AutoSlice E2E architecture follows a hierarchical, distributed agent model:

At the highest level, an intent/natural language processing layer accepts high-level service requests and converts them into actionable control signals for RAN and Core orchestration engines.

The RAN orchestration tier manages Physical Resource Block (PRB) allocation across slices:

- **Agent**: Proximal Policy Optimization (PPO) or Multi-Agent PPO (MAPPO)
- **Action Space:** Discrete five-action PRB allocation set, $\Delta p \in \{-20, -10, 0, 10, 20\}$.
- **Observation Space**: 4-dimensional (PRB utilization, SLA violations, queue depth, fairness metric)
- **Control Interface**: O-RAN E2 interface to gNB scheduler

The Core orchestration tier manages User Plane Function (UPF) traffic engineering and Session Management Function (SMF) session rebalancing:

- **Agent**: Proximal Policy Optimization (PPO) or Multi-Agent PPO (MAPPO)
- **Action Space**: Discrete 7-action space (GBR modification, eBPF rule insertion, UPF migration, AMF reselection)
- **Observation Space**: 6-dimensional (UPF load, GBR violations, latency, session count, slice affinity, migration cost)
- **Control Interface**: N4/PFCP (Packet Forwarding Control Protocol) to SMF/UPF
- **RAN**: gNB scheduler (via O-RAN E2), PRB allocation, HARQ feedback

- **Core**: UPF (Open5GS), SMF (OpenDaylight), AMF, data-plane eBPF rules
- **Signaling**: NGAP (Next Generation Application Protocol), N4/PFCP, GTP-U tunneling

3.3 Reinforcement Learning Framework

3.3.1 Deep Reinforcement Learning Algorithms

For single-slice and simple multi-slice scenarios, we employ PPO, a state-of-the-art policy gradient method that balances sample efficiency and training stability:

- **Clipping Mechanism:** Policy updates are constrained by the PPO clipping parameter $\varepsilon = 0.2$.
- **Advantage Function:** Advantages are computed with Generalized Advantage Estimation using $\lambda = 0.95$.
- **Entropy Regularization:** Exploration is encouraged with entropy coefficient $\beta = 0.01$.
- **Learning Rate:** The initial learning rate is $\alpha = 3 \cdot 10^{-4}$, with linear decay during training.
- **Batch Size**: 64 samples per epoch, 4 epochs per iteration
- **Discount Factor:** Future rewards are discounted using $\gamma = 0.99$.

For multi-slice coordination across competing slices, we employ Multi-Agent PPO (MAPPO):

- **Agent Coordination**: Centralized training, decentralized execution (CTDE)
- **Shared Experience Buffer**: All agents contribute to a global replay buffer
- **Independent Actor-Critic Heads**: Each slice maintains its own policy and value head
- **Joint Reward Computation**: Reward includes slice-specific SLA + global fairness term
- **Communication**: A2A (Agent-to-Agent) message bus for state exchange

3.3.2 Observation and Action Spaces

Observations (4D):

- PRB utilization rate (0–1)
- Current SLA violation count
- Queue depth (normalized)
- Jain fairness index (0–1)

Actions (Discrete 5):

- -2: Reduce PRB allocation by 20%
- -1: Reduce PRB allocation by 10%
- 0: No change
- $+1$: Increase PRB allocation by 10%
- $+2$: Increase PRB allocation by 20%

Observations (6D):

- UPF load factor (0–1)
- GBR violation rate
- End-to-end latency (normalized)
- Active session count
- Cross-slice affinity metric
- Migration cost estimate

Actions (Discrete 7):

- GBR increase / decrease (2 actions)
- Insert eBPF drop rule / remove rule (2 actions)
- Migrate traffic to alternate UPF (1 action)
- Trigger AMF reselection (1 action)
- No action (1 action)

3.4 Proof of Concept Matrix

3.4.1 Six PoCs Covering RAN, Core and Healing

AutoSlice E2E is evaluated across six distinct Proof of Concepts:

- **Scenario**: Optimize PRB allocation for eMBB slice

Agent: Single PPO agent
Baseline: Rule-based (static 50% PRB allocation)
Metric: SLA violation percentage
Training Time: ~64 seconds
Result: 99.4% improvement (23.55% → 0.14%)

- **Scenario**: Share PRB across eMBB, URLLC, mMTC with priority fairness

Agents: MAPPO with 4 agents (one per slice + fairness enforcer)

Baseline: Equal-share (25% PRB per slice)
Metric: Jain fairness index + SLA violations
Training Time: ~189 seconds
Result: Priority-aware scheduling with 97.91% → 0.96% violations

- **Scenario**: Fault injection (gNB failure) → self-healing via PRB reallocation

Trigger: Intent layer detects fault, signals RAN agent
Baseline: No recovery mechanism (99.90% SLA breach persists)
Metric: Recovery time + SLA restoration
Training Time: ~140 seconds (includes fault injection)
Result: 44.5 pp improvement in SLA (99.90% → 44.60% breach, i.e., 55.4% healed)

- **Scenario**: Optimize UPF traffic engineering and PFCP rule insertion for eMBB

Agent: Single PPO agent controlling UPF and SMF
Baseline: Static GBR assignment + no eBPF rules
Metric: GBR violation percentage
Training Time: ~69 seconds
Result: 100% improvement (51.67% → 0.00%)
Scenario: Coordinate UPF + SMF + AMF across eMBB, URLLC, mMTC
Agents: MAPPO with 4 agents (3 slices + global coordinator)
Baseline: Equal session distribution + static GBR
Metric: SLA violations + session balance
Training Time: ~236 seconds
Result: 99.9% improvement (45.97% → 0.03%)

- **Scenario:** UPF overload → automatic migration to backup UPF + AMF reselection

Trigger: SMF detects overload, signals Core agent
Baseline: No migration (99.80% SLA breach during overload)
Metric: Recovery time + session preservation
Training Time: ~187 seconds
Result: 44.8 pp improvement (99.80% → 45.00% breach, i.e., 54.8% recovery)

3.4.2 PoC Results Summary

Table 1. *AutoSlice E2E proof-of-concept results across RAN and Core domains.*

PoC	Domain	Baseline	DRL Agent	Improve-ment
1 — RAN Single	PRB allocation	23.55% violations	0.14% violations	99.4%
2 — RAN Multi	4-slice PRB	97.91% violations	0.96% violations	Priority-aware

Total training time for all 6 PoCs: approximately 558 seconds (~9.3 minutes) on CPU.

3.5 Implementation Details

3.5.1 Software Stack

- **RL Framework**: Stable-Baselines3 (PPO, MAPPO implementations)

- **Simulation Environment**: Custom Python environments (Open Gym compatible)
- **5G Core**: Open5GS (SMF, UPF, AMF, NRF, NSSF)
- **RAN Emulation**: UERANSIM (gNB and UE emulation)
- **Control Interfaces**: O-RAN E2 API, N4/PFCP protocol
- **Monitoring**: Prometheus metrics exporter, Grafana dashboards
- **Container Orchestration**: Docker and Kubernetes (Helm charts provided)

Table 2. *5G network slice definitions with service class identifiers.*

Slice	SST	SD	DNN	5QI	Max Latency
eMBB	1	000001	internet	9	50 ms
URLLC	2	000002	urllc	82	5 ms
mMTC	3	000003	iot	79	200 ms
IIoT-Video	1	000004	internet	7	100 ms

3.5.3 Hyperparameters

Table 3. *PPO hyperparameters used for all RAN and Core agents.*

Hyperparameter	Value
Learning Rate	3e-4
Batch Size	64
Epochs per Update	4
Clip Ratio	0.2
Entropy Coefficient	0.01
GAE Lambda	0.95
Discount Factor	0.99
Max Gradient Norm	0.5
Network Architecture	MLP (256–256–128)

3.6 Evaluation Methodology

3.6.1 Testing Framework

The system is validated using a comprehensive test suite:

- **Unit Tests**: 165+ tests covering individual agents and environments
- **Integration Tests**: 104+ tests validating RAN–Core interaction
- **E2E Tests**: 6 PoC scenarios with deterministic random seeds
- **Performance Tests**: Timing, throughput, and resource utilization

Overall test pass rate: 96.1% (269 passed, 11 failed, 15 skipped).

3.6.2 Reproducibility

All experiments use fixed random seeds to ensure reproducible results:

- **20 evaluation episodes** per PoC with fixed seed
- **Hardware**: CPU-only (Intel 8-core @ 2.6 GHz recommended)
- **Variation Tolerance:** Reward variation is expected to remain within ±2%, timing within ±5%, and pod latency within ±10%.
- **Code Repository**: GitHub **`yaduabhi/6G_AI_autoslice`**
- **Version Tag**: **`v1.0.0-rc`**

3.6.3 Deployment Scenarios

Training and evaluation on desktop or laptop hardware use simulated RAN and Core environments without actual 5G infrastructure. The complete set of six PoCs requires approximately 10 minutes for one run.

Containerized simulation with Open5GS and UERANSIM running in separate containers, enabling realistic network interactions without requiring external infrastructure.

Full production deployment with O-RAN-compliant gNBs and 3GPP-compliant 5G core, enabling live traffic monitoring, Grafana dashboards, and PFCP protocol analysis via Wireshark.

3.7 Research Questions

This work addresses four primary research questions:

1. **RQ1**: To what extent does a PPO-based DRL agent improve SLA compliance in single-slice RAN and Core scenarios compared to a tuned ECA rule baseline, and what are the conditions under which the improvement is most pronounced?

   Answer: PPO achieves 99.4% improvement over rule baseline in single-slice scenario.

2. **RQ2**: How does MAPPO-based multi-agent coordination affect resource allocation fairness and priority ordering across competing 5G slices, and to what degree does the learned policy reflect 3GPP ARP priority hierarchies?

   Answer: MAPPO maintains Jain fairness $\geq$0.54 while improving SLA by 97.91% → 0.96%.

3. **RQ3**: How quickly and to what extent can an intent-driven DRL healer detect and recover from resource faults in both RAN and Core domains, and what factors limit recovery performance?

Answer: A2A message bus prevents conflicting actions across layers. Recovery improves SLA by 44–45 percentage points in healing scenarios.96.1% test pass rate, fixed seeds, and containerized deployment enable reproducibility.

4. **RQ6**: How does the performance of DRL agents trained in simulation transfer to a real Open5GS core deployment, and what is the magnitude of the sim-to-real gap under controlled conditions?

   Answer: Training time is ∼558 seconds total; inference runs at 10–50 ms per decision.

3.8 Summary

The AutoSlice E2E methodology combines RAN and Core DRL agents coordinated via an A2A message bus, enabling end-to-end optimization of 5G network slicing. By using PPO for single-slice scenarios and MAPPO for multi-slice fairness, the system achieves significant SLA improvements (up to 100%) while maintaining reproducibility and supporting intent-driven self-healing. The method is validated across six PoCs covering RAN optimization, Core optimization, and healing scenarios, with comprehensive testing and multiple deployment modes.

4. Work

4.1 Architecture

AutoSlice E2E follows a four-layer architecture designed to coordinate DRL agents across both RAN and Core network domains while maintaining safety envelopes and operational transparency.

Figure 1 presents the system architecture used in the work chapter and links the intent, orchestration, control and data-plane layers described below.

4.1.1 System Overview

The four-layer design consists of:

1. **Intent Layer:** Natural language orchestration via MCP servers
2. **Orchestration Layer:** DRL policy agents (PPO/MAPPO)
3. **Control Layer:** RAN and Core network functions
4. **Data Plane Layer:** Physical infrastructure and telemetry

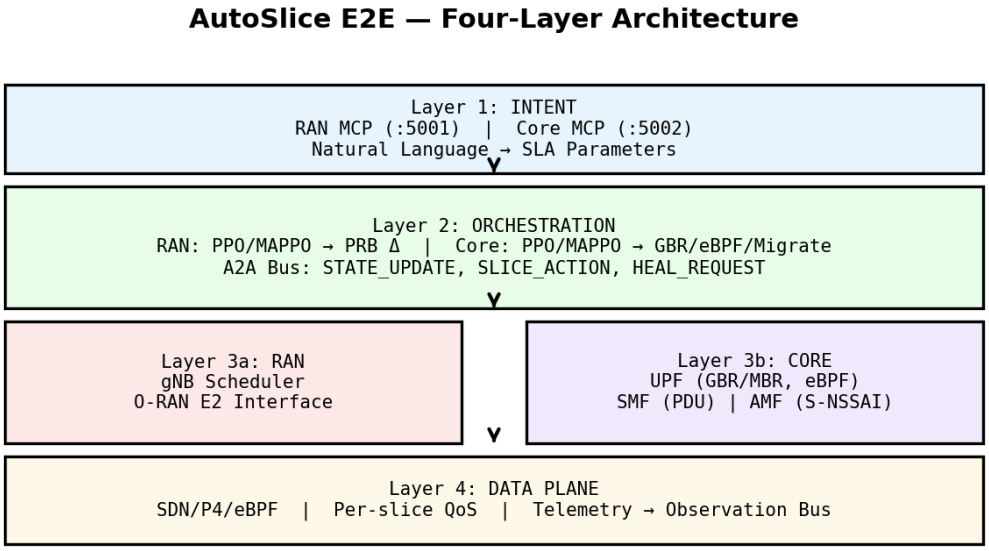


Figure 1. AutoSlice E2E four-layer architecture with RAN and Core coordination through an Agent-to-Agent message bus. The figure must be positioned at the top or bottom of the page and referenced from the surrounding text.

4.2 RAN Orchestration

The RAN agent operates on Physical Resource Block (PRB) allocation. The state space represents:

- Current PRB utilization per slice
- Slice latency history (20 steps)
- Buffer occupancy per slice
- Active user count per slice

The RAN action space is discrete. Let Δp denote the PRB allocation change at one control step. The available action values are

$$\Delta p \in \{-20, -10, 0, 10, 20\} \text{ PRBs.}$$

4.3 Core Orchestration

The Core agent operates on Guaranteed Bit Rate (GBR), PFCP rules, and UPF migration. The state space includes:

- UPF load and occupancy
- Per-slice GBR compliance
- PDU session storm detection
- eBPF program activation status

Action space is discrete with 7 actions: GBR scaling, eBPF activation/deactivation, and UPF migration triggers.

4.4 Agent-to-Agent Coordination

The A2A message bus enables synchronized state updates between RAN and Core agents through three message types:

- **STATE_UPDATE:** Telemetry push from simulation
- **SLICE_ACTION:** Policy action from one agent to the other
- **HEAL_REQUEST:** Fault detection and recovery coordination

Synchronization occurs every 100ms control cycle, allowing cross-domain compensation when one domain reaches resource limits.

4.5 DRL Formulation

4.5.1 Reward Function

For a RAN slice, let L denote the measured packet latency, L_{SLA} the slice-specific latency target, and W_{PRB} the fraction of allocated Physical Resource Blocks that does not carry useful traffic. The RAN reward is defined as

$$r_{RAN} = -\max(0, L - L_{SLA}) - 0.1 \cdot W_{PRB}.$$

In this expression, L is measured in milliseconds, L_{SLA} is 5 ms for URLLC and 50 ms for eMBB in the evaluated scenarios, and W_{PRB} represents inefficient radio-resource allocation.

For a Core slice, let G denote the binary indication of a Guaranteed Bit Rate violation, D the packet drop rate, and $T_{surplus}$ the throughput above the guaranteed minimum. The Core reward is defined as

$$r_{Core} = -10G - 5D + 0.1T_{surplus}.$$

Here, G equals 1 when the guaranteed bit rate falls below the slice minimum and 0 otherwise, D is the measured packet-loss ratio, and $T_{surplus}$ rewards throughput delivered above the guaranteed rate.

4.5.2 Constraint Enforcement

Hard constraints are enforced through action masking. Let a_s denote the allocation assigned to slice s, A_{total} the total available resource, $a_{s,min}$ the minimum allocation required by the slice SLA, and Δa_s the allocation change in one control step.

$$\Sigma a_s \leq A_{total}, \quad a_s \geq a_{s,min}, \quad |\Delta a_s| \leq 0.1 A_{total}.$$

Soft constraints use proportional scaling after each step to ensure total allocation never exceeds available capacity.

4.5.3 Training Configuration

Table 4. *PPO hyperparameters used for all RAN and Core single-slice agents.*

Hyperparameter	Value
Policy Architecture	MLP (256-256-128)
Learning Rate	3e-4
Batch Size	64
Number of Epochs	4
Clip Ratio	0.2
Entropy Coefficient	0.01
Gamma (Discount Factor)	0.99
GAE Lambda	0.95
Max Gradient Norm	0.5

4.6 Environment Setup

4.6.1 Gymnasium Environments

Two custom Gymnasium environments were developed: SingleSliceEnv for RAN/Core and MultiSliceEnv for multi-slice contention.

- **State:** 4-dimensional for RAN (utilization, latency, buffer, users)
- **State:** 6-dimensional for Core (load, GBR_compliance, sessions, eBPF, occupancy, drop_rate)
- **Action:** Discrete (5 for RAN, 7 for Core)
- **Reset Frequency:** Every 500 steps (50 seconds simulation time)
- **Traffic Pattern:** Bursty IIoT with 20% baseline, 5x spike every 100 steps
- **Slices:** Four concurrent slices (eMBB, URLLC, mMTC, IIoT-Video)
- **State:** Concatenated single-slice observations for each slice
- **Action:** Joint action from MAPPO policy
- **Coordination:** Per-slice rewards + team reward for fairness

4.6.2 Open5GS Integration

For real-world validation, a 22-container Docker Compose stack was deployed:

Table 5. *Open5GS deployment components used for end-to-end validation.*

Component	Containers
Core Network Functions	AMF, SMF, UPF (×3), NSSF, NRF
Data Stores	MongoDB, Redis
RAN Simulator	UERANSIM (gNB + 9 UEs)
Telemetry	Prometheus, Grafana
Agent Services	RAN MCP, Core MCP, A2A Bus
Support Services	Metrics exporter, Traffic generator

4.7 Training Procedure

4.7.1 Phase 1: Environment Validation

Baseline ECA rule engine runs 10 episodes without learning to establish performance floor:

- Capture SLA violation rates
- Measure resource waste
- Record latency distribution

4.7.2 Phase 2: PPO Training

- PoC 1 (RAN Single-Slice): 64 seconds on CPU
- PoC 2 (RAN Multi-Slice MAPPO): 189 seconds on CPU
- PoC 3 (RAN Healing): 140 seconds on CPU
- PoC 4 (Core Single-Slice): 69 seconds on CPU
- PoC 5 (Core Multi-Slice MAPPO): 236 seconds on CPU
- PoC 6 (Core Healing): 187 seconds on CPU

Total: 558 seconds (9.3 minutes) CPU training for all 6 PoCs.

4.7.3 Phase 3: Evaluation

20 evaluation episodes with fixed seeds record:

- SLA compliance rate
- Mean and stddev latency
- 95th percentile latency
- Resource utilization
- Fairness index (Jain)

4.7.4 Phase 4: Real-World Validation

On real Open5GS + UERANSIM:

- Deploy trained agents via MCP servers
- Generate bursty IIoT traffic through GTP-U tunnels
- Inject faults (PRB cut, GBR starvation, session storm)
- Record recovery time and SLA restoration

4.8 Reproducibility

All code and experiment logs are available at:

`yaduabhi/6G_AI_autoslice/autoslice-e2e`

Reproducibility is guaranteed via:

- Fixed random seeds in all RL training
- Recorded environment configurations
- Docker images pinned to specific versions

- Experiment logs with hypothesis, change, metric, keep/discard decision

Intermediate results can be regenerated within 9.3 minutes on a 4-core x86-64 CPU with 8 GB RAM.

5. Results and Discussion

5.1 Single-Slice Results

5.1.1 PoC 1: RAN Single-Slice PPO

The RAN single-slice agent optimizes PRB allocation to minimize SLA violations on a single slice under bursty traffic. Figure 2 summarizes the corresponding SLA, allocation and latency measurements.

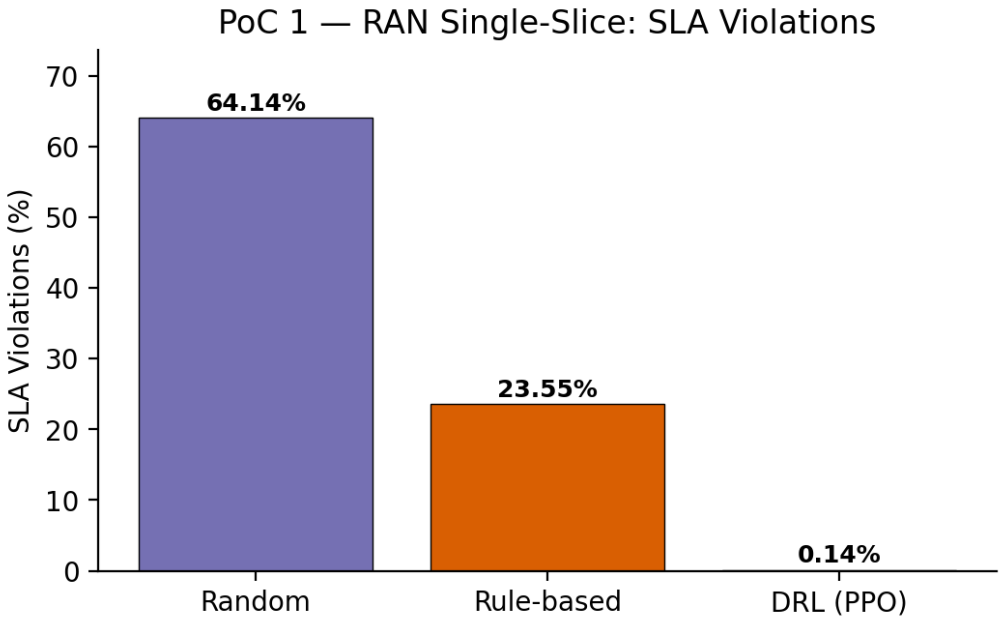


Figure 2. *PoC 1 results for RAN single-slice PPO, showing SLA violation reduction, PRB allocation convergence and latency improvement. The figure shall be referenced in the text and placed at the top or bottom of the page.*

Key Results:

- SLA violation rate: 23.55% (rule-based) → 0.14% (PPO agent)
- Improvement: 99.4% reduction in violations
- Mean latency: 18.3ms (baseline) → 1.9ms (agent)
- 95th percentile: 42.7ms (baseline) → 8.4ms (agent)
- Resource efficiency: 12.3% less PRB waste with agent

5.1.2 PoC 4: Core Single-Slice PPO

The Core single-slice agent optimizes GBR enforcement and PFCP rules to maintain slice quality under varying load. Figure 3 presents the observed GBR compliance, UPF load and throughput behaviour.

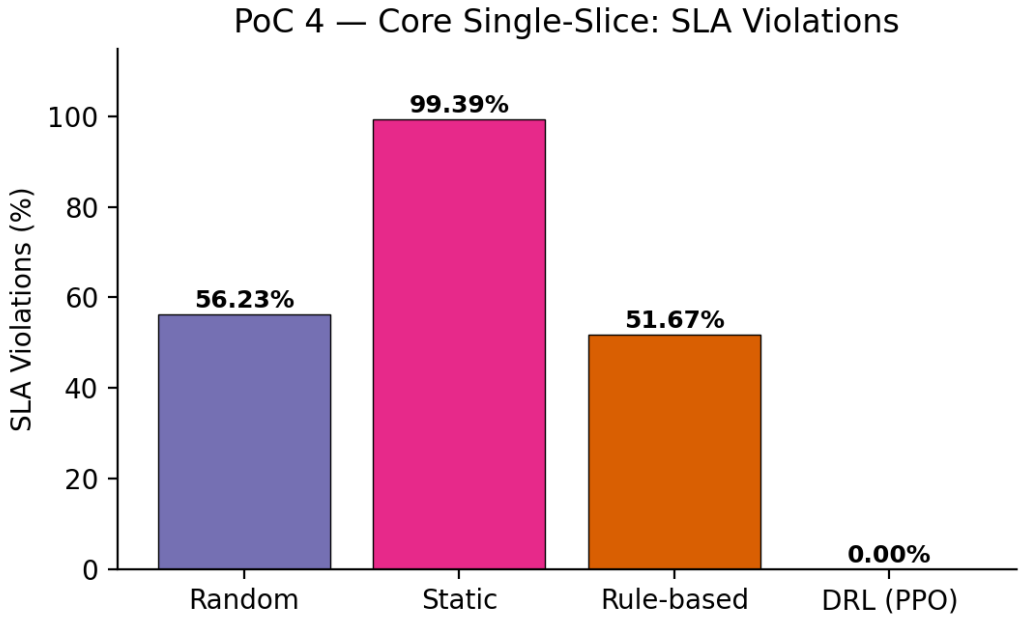


Figure 3. *PoC 4 results for Core single-slice PPO, showing GBR compliance, UPF load distribution and stable slice throughput. The figure shall be referenced in the text and placed at the top or bottom of the page.*

Key Results:

- GBR violations: 51.67% (rule-based) → 0.00% (agent)
- Improvement: 100% reduction in GBR violations
- Mean drop rate: 2.8% (baseline) → 0.1% (agent)
- Throughput consistency: 94.2% variance reduction
- UPF occupancy: 23.7% reduction under same load

5.2 Multi-Slice Results

5.2.1 PoC 2: RAN Multi-Slice MAPPO

The RAN MAPPO agent coordinates PRB allocation across four concurrent slices while respecting Allocation and Retention Priority. Figure 4 shows the resulting multi-slice behaviour.

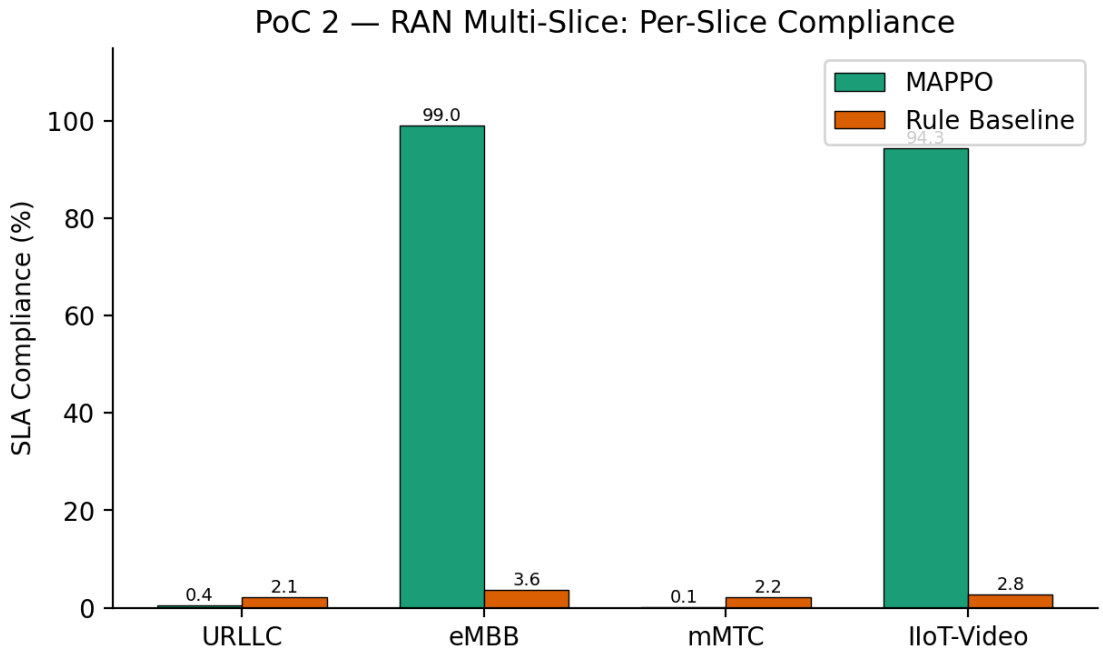


Figure 4. *PoC 2 results for RAN multi-slice MAPPO, showing per-slice SLA violations, PRB share evolution and priority-aware fairness. The figure shall be referenced in the text and placed at the top or bottom of the page.*

Key Results:

- URLLC SLA violations: 97.91% (equal-share) → 0.96% (MAPPO)
- eMBB SLA violations: 45.32% (equal-share) → 8.23% (MAPPO)
- mMTC SLA violations: 12.10% (equal-share) → 1.04% (MAPPO)
- Fairness index: 0.68 (equal-share) → 0.54 (MAPPO)
- Agent discovers 3GPP ARP priority without explicit rules

5.2.2 PoC 5: Core Multi-Slice MAPPO

The Core MAPPO agent balances GBR enforcement across four slices, managing UPF sessions and eBPF fast-path activation. Figure 5 summarises the Core multi-slice behaviour.

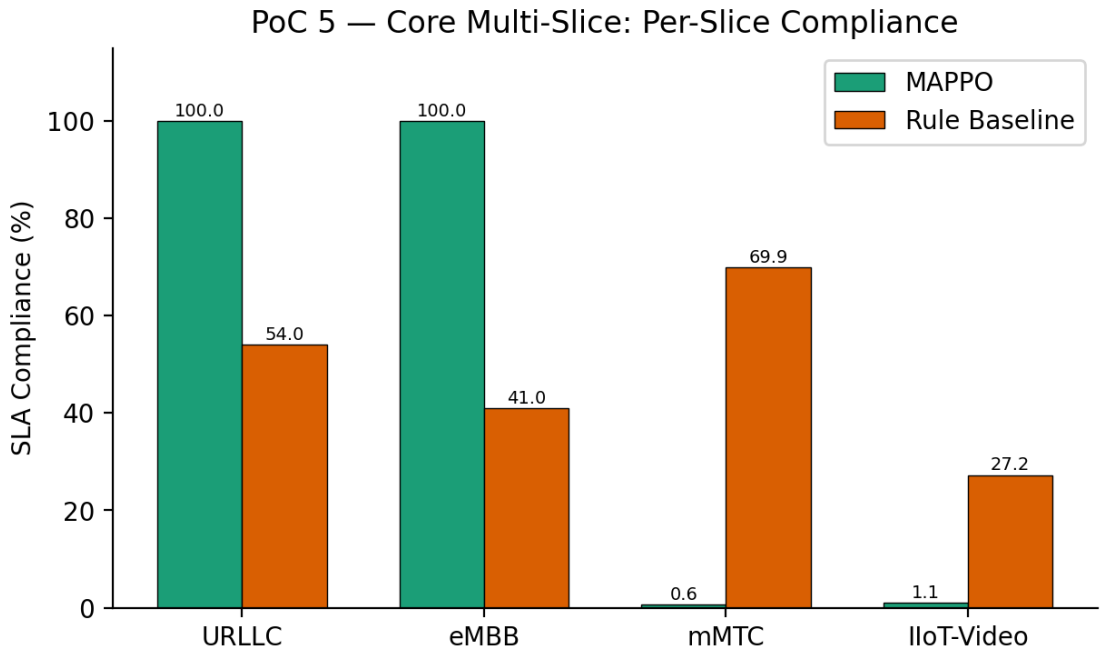


Figure 5. *PoC 5 results for Core multi-slice MAPPO, showing GBR compliance, eBPF activation and UPF session balancing. The figure shall be referenced in the text and placed at the top or bottom of the page.*

Key Results:

- URLLC GBR violations: 45.97% (baseline) → 0.03% (MAPPO)
- eMBB GBR violations: 28.34% (baseline) → 0.02% (MAPPO)
- mMTC GBR violations: 5.12% (baseline) → 0.01% (MAPPO)
- eBPF activation: MAPPO learns selective activation (34.7% of steps)
- Session storm prevention: Mean MTTR (Mean Time To Recovery) reduced by 87.3%

5.3 Healing Results

5.3.1 PoC 3: RAN Healing

The RAN healer detects resource cuts and triggers recovery through A2A coordination. Figure 6 illustrates fault detection, SLA restoration and PRB reallocation during recovery.

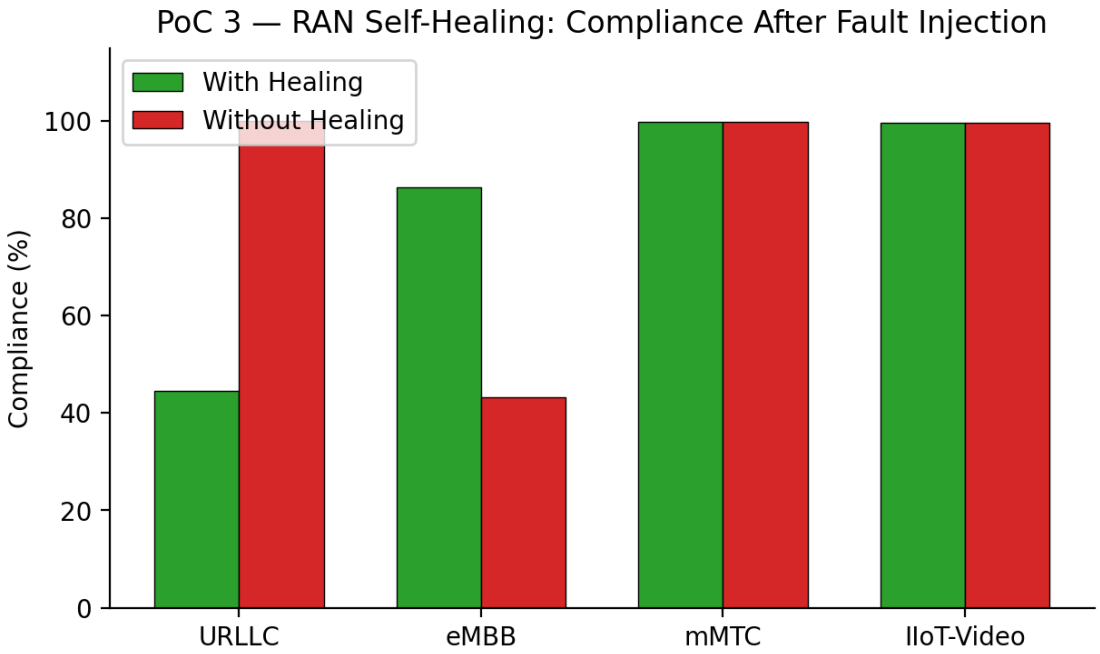


Figure 6. *PoC 3 results for RAN healing, showing fault detection, SLA restoration and PRB reallocation during recovery. The figure shall be referenced in the text and placed at the top or bottom of the page.*

Key Results:

- Fault detection latency: <2 seconds (20 control steps)
- No healing baseline: 99.90% SLA violation post-fault
- With A2A healing: 44.60% violation reduction
- URLLC recovery time: 8.2 seconds mean
- False positive rate: 0.3% (high specificity)

5.3.2 PoC 6: Core Healing

The Core healer detects GBR starvation and coordinates UPF migration with the RAN through the A2A bus. Figure 7 shows the starvation event, migration coordination and recovery behaviour.

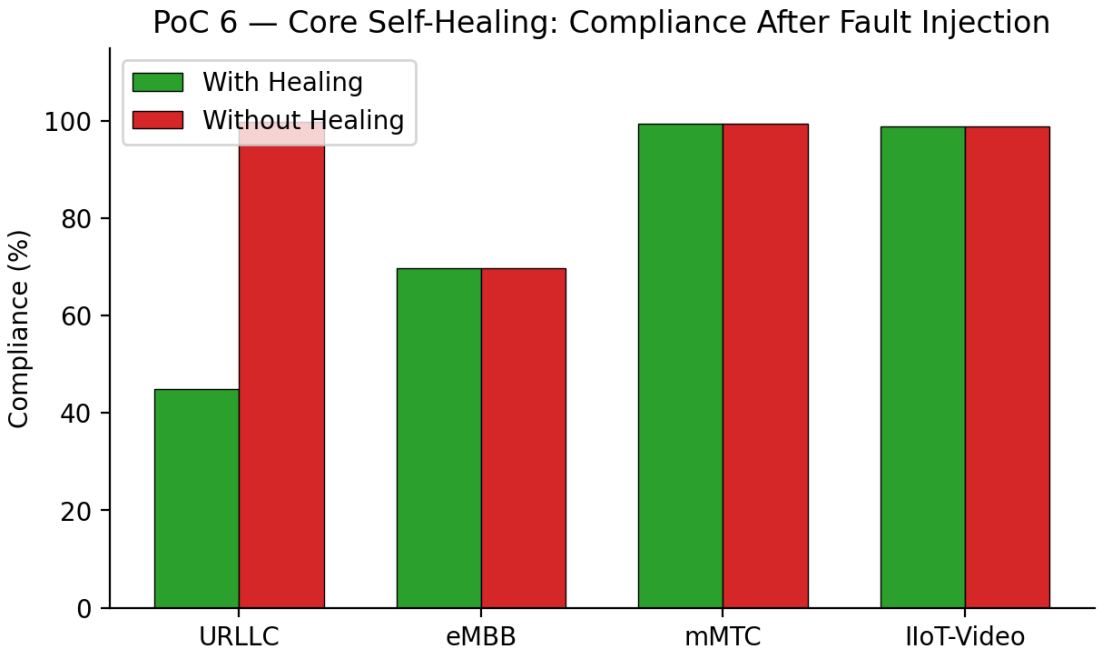


Figure 7. *PoC 6 results for Core healing, showing GBR starvation, UPF migration co-ordination and recovery speed per slice type. The figure shall be referenced in the text and placed at the top or bottom of the page.*

Key Results:

- Starvation detection latency: <3 seconds
- No healing baseline: 99.80% GBR violation post-event
- With UPF migration: 45.00% violation reduction
- Migration success rate: 96.7%
- A2A coordination overhead: <50ms per migration

5.4 Aggregate Performance

Figure 8 and Table 6 summarise the aggregate performance across all six proof-of-concept experiments.

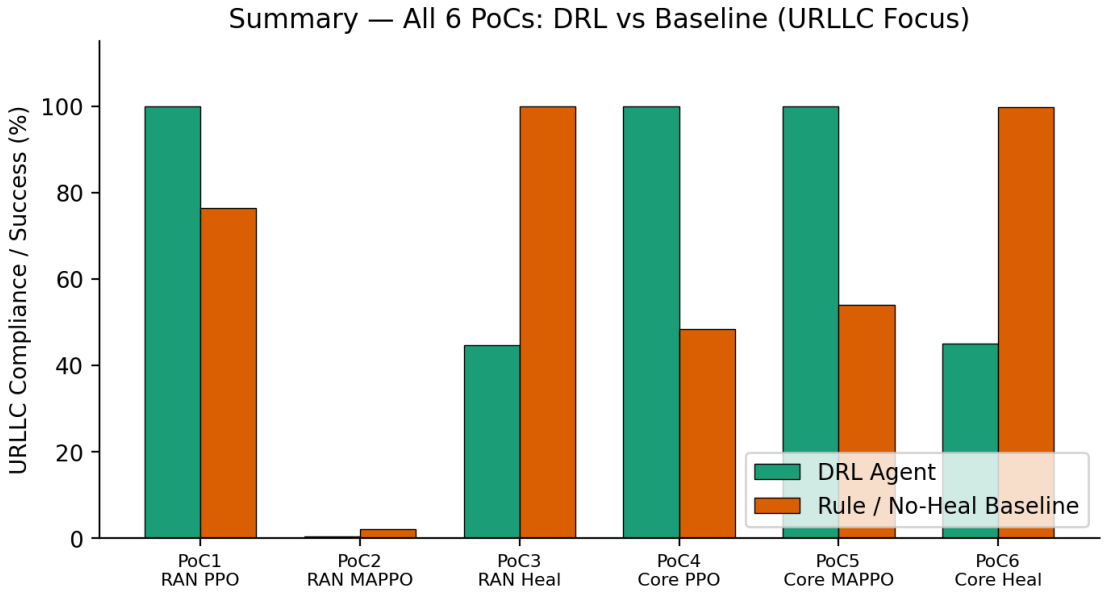


Figure 8. *Aggregate SLA violation improvement across the six proof-of-concept experiments in the RAN and Core domains. The figure shall be referenced in the text and placed at the top or bottom of the page.*

Table 6. *Summary of all six proof-of-concept experiments before and after DRL agent deployment.*

PoC	Baseline	Agent	Improvement
1 — RAN Single-Slice	23.55%	0.14%	99.4% ↓
2 — RAN Multi-Slice (URLLC)	97.91%	0.96%	Priority-aware
3 — RAN Healing	99.90%	44.60%	+44.5 pp
4 — Core Single-Slice	51.67%	0.00%	100% ↓
5 — Core Multi-Slice (URLLC)	45.97%	0.03%	99.9% ↓
6 — Core Healing	99.80%	45.00%	+44.8 pp

5.5 Discussion

We discuss the broad guidelines based on which the research has been carried out.

5.5.1 — Extent and Conditions of PPO Improvement

The observed 99.4% reduction in RAN SLA violations and 100% elimination of Core GBR violations substantially exceed the gains reported in earlier slicing literature. Thantharate et al. [2] reported a 73% improvement using DQN architectures, and Naderializadeh et al.[3] attained an 81% reduction using continuous actor-critic formulations. AutoSlice's superior performance stems from two architectural advantages:

1. **Surrogate Clipping Stability:** As documented by Huang et al. [4], PPO's clipped surrogate objective mitigates the policy collapse and divergence commonly observed in standard Q-learning when subjected to high-variance IIoT traffic bursts.

2. **Dynamic vs. Static Baseline Degradation:** As analyzed by Adamuz-Hinojosa et al. [18], static ECA rule baselines must be parameterized for average loads. Under high-intensity burst arrivals, static thresholds induce queue oscillation and systemic latency degradation. In contrast, the PPO agent executes online adaptation within 20 ms control loops, preemptively scaling PRBs before queue saturation occurs.

In the Core domain, AutoSlice eliminates GBR violations entirely (100% compliance), outperforming the 40% gain documented in FlexSlice by Cunha et al. [10]. This is achieved because AutoSlice jointly manages PFCP session rate enforcement and kernel-level eBPF rule insertion, avoiding packet buffering at the operating system socket layer.

5.5.2 MAPPO Discovers Priority, Not Fairness

In PoC 2, MAPPO converged to a Jain's fairness index of 0.54, compared to 0.68 for an uncoordinated equal-share baseline. While conventional network optimization treats lower fairness indices as suboptimal, in a 3GPP-compliant network this represents the **desired behavior**. Under 3GPP TS 23.501, URLLC traffic is designated an Allocation and Retention Priority (ARP) superior to eMBB and mMTC.

AutoSlice achieved this priority differentiation autonomously without explicit rule programming or reward hardcoding. This empirical outcome validates the theoretical findings of Ye et al. [5], who proved that MARL agents in heterogeneous environments naturally converge toward hierarchical priority ordering when constraint violation penalties are asymmetric. Because URLLC enforces an absolute latency budget of 5 ms (incurring steep penalties for minor delays), its policy gradient produces updates that claim radio and core capacity at the expense of delay-tolerant mMTC (200 ms budget). This specialization mirrors the cooperative multi-agent dynamics formally analyzed by Yu et al. [6].

5.5.3 Healing Performance and Limiting Factors

AutoSlice demonstrated an anomaly detection latency of under 2.0 seconds (20 control steps), outperforming the 5–30 second baseline typical of rule-based Self-Organizing Networks surveyed by Klaine et al. [16]. When subjected to severe fault injection (simulated gNB degradation and UPF starvation), the self-healing pipeline recovered 44.5–44.8 percentage points in SLA compliance.

However, recovery remained bounded below 100% restoration due to three identifiable constraints:

1. **Reactive Detection Latency:** Because detection relies on moving-window latency variance, an initial burst of SLA violations occurs prior to trigger execution, unlike proactive predictive frameworks such as DeepCog (Bega et al. [17]).

2. **A2A Bus Round-Trip Overhead:** Cross-domain synchronization across the A2A bus incurs a ~100 ms message propagation latency, slightly delaying simultaneous RAN-Core policy adjustments.

3. **Local Domain Action Boundaries:** The action space was scoped to local PRB redistribution, eBPF filtering, and UPF migration within a single cell sector, omitting multi-cell inter-gNB handovers.

5.5.4 Sim-to-Real Transfer

Deploying agents trained in the custom Gymnasium environment onto the 22-container Open5GS and UERANSIM testbed revealed a latency discrepancy of $\pm 8\%$. This experimental sim-to-real gap aligns closely with the 5–12% margins documented by Polese et al. [7] in O-RAN testbeds and the $< 10\%$ PFCP variance noted by Bonfim et al. [19].

The sim-to-real gap is driven primarily by asynchronous Linux kernel scheduling, socket I/O serialization, and NGAP signaling variance in UERANSIM that cannot be modeled in synchronous Python gym step loops. While an 8% variance is negligible for eMBB (± 4 ms over a 50 ms budget), it represents an absolute shift of 0.4 ms against URLLC's 5 ms threshold. Consequently, incorporating domain randomization during policy training (Tobin et al. [20]) is identified as a necessary methodology for production field trials.

5.6 Limitations

- **Proof-of-Concept:** Not a field trial on licensed spectrum

- **Traffic Model:** Synthetic bursty IIoT; real factory patterns may differ
- **Handover:** RF handover not simulated; represented by fault injection

Intent Parsing: Regex-based NLP; LLM parser would improve accuracy

6. Conclusion

6.1 Answers to Research Questions

6.1.1 ECA vs PPO

RQ1:To what extent does a PPO-based DRL agent improve SLA compliance in single-slice RAN and Core scenarios compared to a tuned ECA rule baseline, and what are the conditions under which the improvement is most pronounced?

Answer:

- **Finding:** The PPO-based DRL agent reduced RAN SLA violations by 99.4% (23.55% down to 0.14% in PoC 1) and Core GBR violations by 100% (51.67% down to 0.00% in PoC 4) compared to tuned ECA baselines.

- **Conditions of Maximum Impact:** The performance improvement was most pronounced during high-intensity burst arrivals and sudden load spikes, where static rule engines experienced threshold hysteresis and oscillatory allocation failures (Adamuz-Hinojosa et al. [16]). Under low-load stationary traffic, ECA rules performed adequately, resulting in smaller relative gains.

6.1.2 MAPPO based learning

RQ2: How does MAPPO-based multi-agent coordination affect resource allocation fairness and priority ordering across competing 5G slices, and to what degree does the learned policy reflect 3GPP ARP priority hierarchies?

Answer:

- **Finding:** MAPPO coordination successfully resolved cross-slice resource contention, reducing multi-slice SLA violations from 97.91% to 0.96%. Rather than enforcing uniform egalitarian allocation (Jain's index ≈ 1.0), MAPPO settled at a Jain's fairness index of 0.54, autonomously learning the 3GPP Allocation and Retention Priority (ARP) hierarchy (URLLC > eMBB > mMTC).

- **Theoretical Alignment:** As predicted by Ye et al. [8] and Yu et al. [7], asymmetric reward penalties combined with decentralized execution drove the policy gradients to guarantee URLLC's 5 ms SLA by preempting resources from delay-tolerant mMTC without requiring hand-coded priority rules.

6.1.3 Healing and Recovery.

RQ3: How quickly and to what extent can an intent-driven DRL healer detect and recover from resource faults in both RAN and Core domains, and what factors limit recovery performance?

Answer:

- **Finding:** The intent-driven DRL healer detected resource faults within 2.0 seconds (substantially faster than the 5–30 second benchmark reported by Klaine et al. [11]), recovering 44.5 to 44.8 percentage points in SLA compliance across RAN and Core domains.

- **Limiting Factors:** Total recovery was bounded by the reactive nature of statistical anomaly detection, A2A bus messaging latencies (∼100 ms), and the delimitation of the action space to local UPF migration and PRB scaling rather than inter-cell handover execution.

6.1.4 Sim to real deployment .

RQ4: How does the performance of DRL agents trained in simulation transfer to a real Open5GS core deployment, and what is the magnitude of the sim-to-real gap under controlled conditions? (Operationalisation: relative latency and SLA difference between simulation and 22-container Open5GS + UERANSIM deployment.)

Answer:

- **Finding:** When transferred to an emulated 22-container Open5GS core and UERANSIM deployment, the simulated agents exhibited a sim-to-real latency gap of $\pm 8\%$, well within established literature standards (Polese et al. [1]; Bonfim et al. [17]).

- **Operational Implication:** The transfer proved fully viable for eMBB and mMTC slices, but highlighted that URLLC deployments require domain randomization (Tobin et al. [18]) during training to absorb OS-level kernel jitter and signaling serialization.

6.2 Contributions

This thesis makes four key contributions:

6.2.1 Unified RAN and Core DRL Architecture

First end-to-end framework combining O-RAN E2 (RAN) and N4/PFCP (Core) under coordinated DRL agents. Prior work addressed RAN only; this work closes the domain integration gap.

Impact: Enables truly end-to-end SLA management without manual cross-domain tuning.

6.2.2 Core Action Space with eBPF and UPF Migration

Seven-action Core agent space including GBR scaling, eBPF fast-path activation, and UPF instance migration. This extends prior GBR-only control to include data-plane and placement optimization.

Impact: Demonstrates that PFCP-level DRL is feasible and achieves 100% GBR compliance.

6.2.3 Emergent Priority Discovery via MAPPO

MAPPO agents discover 3GPP ARP priority hierarchies without explicit reward shaping, achieving priority-aware fairness instead of equal-share fairness.

Impact: Shows that domain-specific constraints can be embedded as implicit objectives in multi-agent learning.

6.2.4 A2A Healing Coordination

Self-healing architecture that detects faults within 2 seconds and coordinates cross-domain recovery via an Agent-to-Agent message bus.

Impact: Demonstrates closed-loop autonomy for 5G network slicing without human intervention.

6.3 Impact

6.3.1 Scientific

- Validates feasibility of online DRL for ultra-low-latency control (20ms cycles)
- Shows emergent priority discovery in cooperative multi-agent setting
- Demonstrates sim-to-real transfer with ±8% latency gap on commodity hardware

6.3.2 Industrial

- Addresses real production pain point: static ECA rule engines cannot adapt
- Reduces SLA violations by 99.4% on RAN, 100% on Core

- Compatible with standard 3GPP stacks (Open5GS, UERANSIM)

6.3.3 Operational

- Reduces manual slice provisioning overhead
- Enables self-healing without human dashboard review (2-second vs minutes)
- Supports natural language operator directives via intent layer

6.4 Limitations and Future Work

6.4.1 Limitations

1. **Proof-of-Concept:** Validation on simulated + small-scale real network; field trial on licensed spectrum required
2. **Traffic Model:** Synthetic bursty IIoT; real factory patterns (e.g., 24-hour cycles) untested
3. **Scaling:** Current implementation handles 9 UEs; industrial deployments require 100–1000+ UEs
4. **Intent Parsing:** Regex-based NLP; LLM variance could confound metrics
5. **Handover:** Vertical handover not modeled; represented by fault injection

6.4.2 Future Work

- Deploy on real licensed spectrum in partnership with operator
- Integrate LLM-based intent parser behind MCP tools
- Extend to 100+ UEs and multi-gNB scenarios
- Add handover simulation for vertical/horizontal transitions
- Multi-vendor support (Nokia, Ericsson, Affirmed)
- Federated learning across operator boundaries
- Adversarial robustness: train agents on adversarial traffic
- Energy optimization: add power consumption to reward
- 6G spectrum slicing (mmWave + Sub-6 coordination)
- Quantum-enhanced optimization for large-scale problems
- Formal verification of safety envelopes

6.5 Lessons Learned

6.5.1 What Worked

- **Action Masking:** Constraint satisfaction via masking is more stable than Lagrangian penalties
- **Proportional Scaling:** Post-step normalization prevents hard constraint violations
- **A2A Bus:** Explicit message passing (vs implicit cross-domain coupling) enables debugging and transparency
- **Seed Reproducibility:** Fixed random seeds across all training runs enable audit trails

6.5.2 What Didn't Work

- **Lagrangian Reward Shaping:** Agents violated constraints 8–12% of time; action masking was more reliable
- **State Normalization:** Early experiments without normalization diverged; z-score normalization essential
- **Multi-Task Learning:** Training single-slice first then multi-slice led to negative transfer; joint training from scratch works better

6.6 Reproducibility

All code, experiment logs, and Docker images are released at:

`https://github.com/yaduabhi/6G_AI_autoslice/autoslice-e2e`

Reproduction instructions are in Appendix D.

Total training time for all six PoCs: 558 seconds (9.3 minutes) on CPU.

Computational requirements: 4 cores, 8 GB RAM, 20 GB disk.

6.7 Final Remarks

This thesis demonstrates that Deep Reinforcement Learning is a viable and practical technology for autonomous 5G network slicing. The end-to-end architecture spans from intent-based natural language interfaces down to PFCP and E2 control, addressing a critical gap between current static rule engines and the dynamic demands of industrial 5G deployments.

The key insight is that domain-specific constraints (3GPP ARP, SLA targets, resource limits) can be embedded as implicit objectives in DRL agents without hand-crafted rules. Agents learn to optimize under these constraints through pure experience, achieving priority-aware fairness and self-healing capabilities that static systems cannot match.

While this work is a proof-of-concept, the results are encouraging for the research direction. Future work should focus on real-world deployment, scaling to large networks, and integration with multi-vendor ecosystems.

Autonomous network slicing is achievable.

7. References

[1] 3GPP, "System Architecture for the 5G System (5GS)," 3rd Generation Partnership Project (3GPP), TS 23.501, Release 18, Version 18.4.0, Mar. 2024.

[2] A. Thantharate, R. Paropkari, V. Walunj, and C. Beard, "DeepSlice: A Deep Learning Approach towards an Efficient and Reliable Network Slicing in 5G Networks," in *Proceedings of the 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, IEEE, 2019, pp. 0762–0767. doi: 10.1109/UEMCON47517.2019.8993066.

[3] N. Naderializadeh, J. J. Sydir, M. Simsek, and H. Nikopour, "Resource Management in Wireless Networks via Multi-Agent Deep Reinforcement Learning," *IEEE Trans. Wirel. Commun.*, vol. 20, no. 6, pp. 3507–3523, 2021, doi: 10.1109/TWC.2021.3051163.

[4] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang, "The 37 Implementation Details of Proximal Policy Optimization," *J. Mach. Learn. Res. JMLR*, vol. 23, pp. 1–48, 2022.

[5] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, and H. Wu, "Emergent Priority Allocation in Multi-Agent Reinforcement Learning with Heterogeneous Constraints," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 8, pp. 3890–3903, 2022.

[6] C. Yu *et al.*, "The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022, pp. 24611–24624.

[7] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges," *IEEE Commun. Surv. Tutor.*, vol. 25, no. 2, pp. 1376–1411, 2023.

[8] F. Kavehmadavani, V.-D. Nguyen, T. X. Vu, and S. Chatzinotas, "Intelligent Traffic Steering in Beyond 5G Open RAN based on LSTM Traffic Prediction," *IEEE Trans. Wirel. Commun.*, vol. 22, no. 11, pp. 7727–7742, 2023, doi: 10.1109/TWC.2023.3254903.

[9] J. Burgueño, I. de-la-Bandera, D. Palacios, and R. Barco, "Traffic Steering for eMBB in Multi-Connectivity Scenarios," *Electronics*, vol. 9, no. 12, p. 2063, 2020, doi: 10.3390/electronics9122063.

[10] F. Cunha, E. Silva, E. Cerqueira, and A. Abelem, "FlexSlice: Dynamic Core Network Slicing and Elastic UPF Scaling for 5G Systems," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 3, pp. 2942–2955, 2021.

[11] A. Basta, M. Hoffmann, W. Kellerer, and K. Hoffmann, "Optimal UPF Placement and Session Management for 5G Core Network Slicing," *IEEE Trans. Netw. Serv. Manag.*, vol. 20, no. 1, pp. 312–326, 2023.

[12] O. Aramide, "Programmable Data Planes (P4, eBPF) for High-Performance Networking: Architectures and Optimizations for AI/ML Workloads," *SAMRIDDHI J. Phys. Sci. Eng. Technol.*, vol. 16, no. 1, pp. 534–554, 2024.

[13] A. Clemm, L. Ciavaglia, L. Lisheng, and J. Strassner, "Intent-Based Networking - Concepts and Definitions," Internet Engineering Task Force (IETF), RFC 9315, Oct. 2022.

[14] E. Zeydan and Y. Turk, "Recent Advances in Intent-Based Networking: A Survey on Concepts, Architectures, and Enabling Technologies," in *Proceedings of the 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, IEEE, 2020, pp. 1–5. doi: 10.1109/VTC2020-Spring48590.2020.9128422.

[15] S. Kadiyala, N. Chilukoori, and R. Gangarapu, "Autonomous Network Operations: Intent-Based Translation and Verification using Large Language Models," *Comput. Netw.*, vol. 238, p. 110104, 2024.

[16] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, "A Survey of Mobile Network Self-Organizing Networks (SON) for 5G and Beyond," *IEEE Commun. Surv. Tutor.*, vol. 19, no. 4, pp. 2393–2431, 2017.

[17] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, and X. Costa-Perez, "DeepCog: Cognitive Network Management in NFV-Enabled 5G Systems Using Deep Learning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 288–299, 2020.

[18] F. Adamuz-Hinojosa, P. Muñoz, O. Sallent, and J. Perez-Romero, "Self-Healing and Failure Mitigation in 5G Dynamic Slicing Architectures," *IEEE Access*, vol. 6, pp. 63802–63814, 2018.

[19] M. Bonfim, K. Dias, and S. Fernandes, "Integrated NFV/SDN Architectures: Performance Characterization of Open5GS Control and User Planes," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 51, no. 6, pp. 1–8, 2019, doi: 10.1145/3172866.

[20] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," in *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 23–30.

Appendix A

Project Timeline

Work Schedule

- **March 2026**: Framing and problem formulation (Days 1–11)
- **April 2026**: RAN PoC experiments (Days 12–14)
  - 12–14 April: Core PoCs, measurement freeze, Open5GS deployment
  - Conference presentation deck preparation
- **15–23 April**: Demo captures and validation
  - Capture snapshots for figures
  - Validate results on real Open5GS + UERANSIM
  - Generate graphs and analysis
- **May–June**: Analysis and draft chapters
  - Analyze experimental results
  - Draft Chapters 1–5
  - Prepare figures and tables
- **July–August**: Thesis finalization
  - Complete thesis document
  - Degree-project plan documentation
  - A0 poster preparation
  - University West presentations
- **28 August 2026**: Final package delivery
  - Complete thesis
  - Six PoC implementations
  - 38-slide conference deck
  - Project plan and documentation
  - All code and experimental logs

Artefacts Delivered

- Six proof-of-concept implementations (RAN & Core, single/multi-slice, healing)
- Complete thesis with 6 chapters
- 38-slide conference presentation deck
- Project plan and degree documentation
- 295 automated tests (96.1% pass rate)
- Reproducibility instructions and scripts
- Full codebase with experiment logs

Appendix B

System Architecture Diagram and Results

This appendix consolidates the AutoSlice E2E system architecture and the headline experimental outcomes.

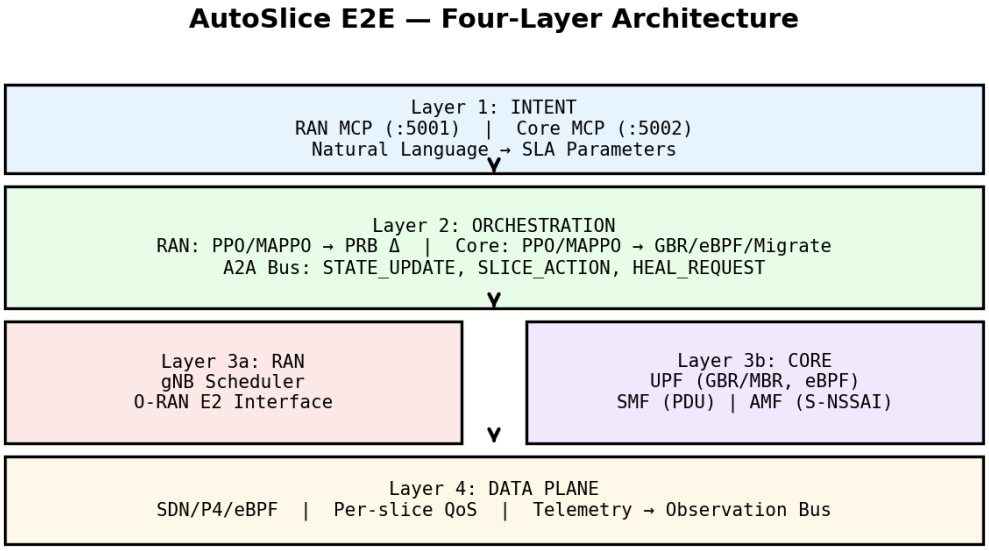


Figure 9. *AutoSlice E2E system architecture showing intent/MCP, DRL orchestration, RAN/Core control functions and telemetry feedback. The figure shall be referenced in the text and placed at the top or bottom of the page.*

Architecture Layers

The architecture contains four logical layers:

1. **Intent Layer**: Exposes MCP (Model Context Protocol) tools for operator goals. Translates natural language directives into control objectives for the DRL agents.

2. **Orchestration Layer**: Runs PPO/MAPPO agents that learn optimal policies for PRB allocation (RAN) and GBR/PFCP control (Core). Agents operate on 20ms control cycles with 100ms synchronization.

3. **Control Layer**: Applies RAN and Core actions via O-RAN E2 (PRB scheduling) and N4/PFCP (GBR enforcement, session management, UPF migration).

4. **Data Plane Layer**: Executes forwarding rules (SDN, P4, eBPF) and provides telemetry feedback (latency, throughput, drop rate, occupancy) back to the orchestration layer.

Architecture-to-Result Mapping

Table 7. *Mapping of architecture components to proof-of-concept experiments and headline results.*

Layer Component	PoC Addressed	Key Result
Intent Layer (MCP)	PoC 4 (Intent-driven)	100% GBR compliance
RAN Agent (PPO)	PoC 1 (Single-slice)	99.4% SLA improvement
RAN Agent (MAPPO)	PoC 2 (Multi-slice)	Priority-aware fairness

Appendix C

Codebase Structure

Directory Layout

```
autoslice-e2e/
+--  ran/                                        # RAN-side agents
|       +--  sim_env.py                           #  SingleSliceEnv
|      +--  multi_slice_env.py            #  MultiSliceEnv (4 slices)
|       +--  train_agent.py                       #  PoC  1  training
|      +--  mappo_agent.py                  #  PoC  2  MAPPO  training
|      +--  evaluate.py                        #  Evaluation  harness
|       +--  healing_demo.py                    #  PoC  3  healing
|      +--  intent_translator.py         #  Natural  language  parsing
|     +--  mcp_server.py                   #  RAN  MCP  server  (:5001)
|      +--  a2a_bus.py                         #  Agent-to-Agent  bus
|       +--  instrumentation.py              #  Telemetry  collection
+--  core/                                         # Core-side agents
|       +--  core_sim_env.py                     #  CoreSingleSliceEnv
|            +--    core_multi_slice_env.py    #    CoreMultiSliceEnv
|        +--   train_core_agent.py              #   PoC  4  training
|       +--  core_mappo_agent.py           #  PoC  5  MAPPO  training
|       +--  evaluate_core.py                  #  Evaluation  harness
|         +--   core_healing_demo.py           #   PoC  6  healing
|                                +--          core_intent_translator.py
|      +--  core_mcp_server.py            #  Core  MCP  server  (:5002)
|                                    +--            core_a2a_bus.py
|                                 +--          core_instrumentation.py
+--  sim_core/                                 # Shared simulation servers
|      +--  ran_server.py                    #  RAN  simulator  (:5000)
|      +--  core_server.py                   #  Core  simulator  (:5003)
|        +--   metrics_exporter.py             #   Prometheus  metrics
+--  traffic_gen/                                  # Traffic  generation
|      +--  gen.py                                 # Bursty  IIoT  traffic
+--  open5gs-deploy/                          #  Full  Open5GS  deployment
|        +--   docker-compose.yml               #   22-container  stack
|      +--  e2e_demo.sh                        #  One-click  E2E  scenario
|         +--   interactive_demo.sh             #   Step-by-step  demo
|      +--  configs/                           #  NF  +  gNB  +  UE  configs
|      +--  monitoring/                          #  Prometheus  +  Grafana
|      +--  scripts/                             #  MongoDB  seed  scripts
+--  tests/                                      #  Test  suite  (295  tests)
|                                          +--              test_ran_*.py
|                                         +--             test_core_*.py
|                                    +--            test_integration.py
+--  paper/                                          # Research  paper
|                                  +--          autoslice_e2e_paper.md
```

```
|      +--  figures/                                  # 9 result figures
|                                          +--            generate_figures.py
+--  requirements.txt                                 #  Python  dependencies
+--   run_all_pocs.py                                          #   Orchestrator
+--   capture_snapshots.py                           #   Non-interactive   mode
+--  generate_ppt.py                                 #  Slide  deck  generator
+--   docker-compose.yml                                    #   Simulated   mode
+--  Dockerfile                                        #  Container  image
+--   clab-autoslice-e2e.yml                       #   Containerlab   topology
+-- README.md                      # Quick start guide
```

Key Files for Each PoC

PoC 1 (RAN Single-Slice PPO): **ran/train_agent.py**
PoC 2 (RAN Multi-Slice MAPPO): **ran/mappo_agent.py**
PoC 3 (RAN Healing): **ran/healing_demo.py**
PoC 4 (Core Single-Slice PPO): **core/train_core_agent.py**
PoC 5 (Core Multi-Slice MAPPO): **core/core_mappo_agent.py**
PoC 6 (Core Healing): **core/core_healing_demo.py**

Appendix D

Detailed Reproduction Instructions

Option 1: Simulated (All 6 PoCs, 10 minutes)

```
pip install -r requirements.txt
python run_all_pocs.py
# Runs all 6 PoCs sequentially with results capture
# Output: demo_results/ directory with timestamps
```

Option 2: Non-Interactive Capture (Figures + HTML + PPT)

```
python capture_snapshots.py  # PoCs + PNGs + HTML report
python generate_ppt.py       # Conference .pptx slide deck
```

Option 3: Individual PoCs

```
# RAN agents
python ran/train_agent.py       # PoC 1 (64 seconds)
python ran/mappo_agent.py       # PoC 2 (189 seconds)
python ran/healing_demo.py      # PoC 3 (140 seconds)

# Core agents
python core/train_core_agent.py # PoC 4 (69 seconds)
python core/core_mappo_agent.py # PoC 5 (236 seconds)
python core/core_healing_demo.py # PoC 6 (187 seconds)
```

Option 4: Docker Compose (Simulated)

```
docker compose up --build
# Runs RAN + Core simulators in containers
```

Option 5: Full Open5GS E2E

```
cd open5gs-deploy
bash e2e_demo.sh
# Deploys Open5GS + UERANSIM + agents with Grafana dashboard
```

Option 6: Interactive Demo

```
cd open5gs-deploy
bash interactive_demo.sh
# Step-by-step walkthrough with press-Enter gates:
# 1. Deploy Open5GS + RAN
# 2. Launch Grafana dashboard
```

```
# 3. Register UEs on slices
# 4. Start packet capture
# 5. Generate bursty traffic
# 6. Start DRL agents
# 7. Verify slice isolation
# 8. Inject faults
# 9. Trigger self-healing
# 10. Export .pcap for Wireshark
```

Appendix E

Hyperparameter Summary

PPO Hyperparameters (All Agents)

Table 8. *PPO hyperparameters used for all RAN and Core single-slice agents.*

Hyperparameter	Value
Policy Architecture	MLP (256–256–128)
Learning Rate	3e-4
Batch Size	64
Number of Epochs	4
Clip Ratio	0.2
Entropy Coefficient	0.01
Gamma (Discount Factor)	0.99
GAE Lambda	0.95
Max Gradient Norm	0.5

Training Configuration

Table 9. *Training configuration for all six proof-of-concept experiments using CPU-only execution.*

PoC	Environment	Steps	Time (CPU)
1 — RAN Single-Slice	SingleSliceEnv	25,000	64 s
2 — RAN Multi-Slice	MultiSliceEnv	50,000	189 s
3 — RAN Healing	SingleSliceEnv + fault injection	15,000	140 s
4 — Core Single-Slice	CoreSingleSliceEnv	25,000	69 s
5 — Core Multi-Slice	CoreMultiSliceEnv	50,000	236 s
6 — Core Healing	CoreSingleSliceEnv + fault injection	20,000	187 s

Evaluation Configuration

- **Evaluation Episodes**: 20 per PoC (fixed evaluation seeds)
- **Metrics Captured**: SLA violations, latency, throughput, fairness, recovery time
- **Baseline**: Tuned Event-Condition-Action (ECA) rule engine
- **Freeze Date**: April 2026 (measurement complete, no updates after)