IOSR Journal of Dental and Medical Sciences (IOSR-JDMS)
e-ISSN: 2279-0853, p-ISSN: 2279-0861.Volume 25, Issue 3 Ser. 5 (March. 2026), PP 30-35
www.iosrjournals.org

AI-Driven End-To-End Network Automation And Orchestration In Open RAN (O-RAN)

Tanaver Karimkhan
Visvesvaraya Technological University, Belagavi, India

Abstract
Open RAN (O-RAN) breaks the radio access net- work into smaller pieces with open interfaces so operators can mix vendors and move functions into the cloud. As networks grow and carry more slices and services, manually keeping all of this tuned is not realistic any more.
We describe an AI-driven orchestration framework that starts from operator intents at the Service Management and Orches- tration (SMO) layer and drives coordinated actions in the RAN, core and, where possible, the transport network. The framework chooses which rApps/xApps to run and where to place them on O- Cloud resources using a simple three-stage heuristic. We outline the overall architecture, the orchestration logic, a practical way to implement it on current O-RAN and telco-cloud platforms, and a small simulation that compares a static setup, a RAN-only policy and an end-to-end policy.
Index Terms: *Open RAN, O-RAN, RIC, SMO, xApps, rApps, AI/ML, network automation, orchestration, 5G.*
Date of Submission: 20-03-2026 Date of Acceptance: 30-03-2026

I. Introduction

5G networks carry very different types of traffic on the same infrastructure: broadband users who mainly care about throughput, control traffic that has strict delay targets, and large numbers of simple IoT devices. At the same time, operators want more choice of vendors and more flexibility in where network functions run.

O-RAN addresses this by splitting the base station into O- RU, O-DU and O-CU parts and by defining open interfaces such as O1, A1 and E2. The RAN Intelligent Controller (RIC) sits next to these functions and runs xApps and rApps that can implement AI-based control at different time scales. In theory this makes automation easier, but in practice most deployments and research have still focused on the RAN side alone [4], [6], [7], [9].

In this work we focus on how to use these building blocks together to automate the network end to end. The key question is how to turn a high-level intent at the SMO (for example, enabling a new URLLC slice in one region) into concrete actions in the RAN, in the core and, when needed, in the transport network, without introducing a very heavy optimisation framework that is hard to run or explain.

Contributions: The main contributions are:

1) We sketch an end-to-end automation setup for O-RAN, starting from intents at the SMO and ending with concrete changes in RAN, core and transport.
2) We give a simple three-step heuristic that decides which AI models to use, where to run them and how to avoid obvious clashes between rApps and xApps.
3) We show how this fits on top of today's O-RAN and Kubernetes-based telco-cloud platforms, using ideas from OrchestRAN and AutoRAN [1]–[3].
4) We use a small simulation to compare a static config- uration, a RAN-only policy and an end-to-end policy, to show what kind of gains we can expect even from a basic implementation.

II. Background And Related Work

O-RAN and the RIC

In O-RAN the base station is split into an O-RU close to the antenna, an O-DU handling most of the lower-layer processing, and an O-CU that runs higher-layer functions. These elements are managed over O1 and controlled over E2, and a non-RT and near-RT RIC host rApps and xApps that use these interfaces [4], [12]. For the rest of this paper, we mainly rely on this standard picture of the architecture rather than going into all O-RAN options in detail.

Automation platforms

Commercial SMO and RIC products already offer many of the features we need: they can onboard rApps and xApps, collect KPIs, and trigger closed loops for optimisation and assurance. Vendor documentation shows that these platforms are being used today for tasks such as energy saving, slice life- cycle management and multi-vendor RAN coordination [6]– [8]. In this work we do not try to replace those systems; instead, we assume something similar is available and only add the orchestration logic on top of it.

Research prototypes

Two pieces of prior work are particularly relevant. Orches- tRAN looks at how to place AI models in the RAN and shows that the exact optimisation problem is NP-hard, but that simple heuristics can still give good results and reduce E2 overhead compared to naive placements [1], [2]. AutoRAN focuses on automating the deployment of Open RAN-based private 5G systems and demonstrates that a GitOps-style workflow can bring up an end-to-end network quickly [3]. We use the ideas of flexible model placement from OrchestRAN and the deployment style of AutoRAN as building blocks for our design. We also draw on reports about AI-native management and cross-domain AI in O-RAN [10], [12].

III. System Model And Problem Statement

We consider a mobile network operator that deploys an O- RAN-compliant RAN (O-RU/O-DU/O-CU), a 5G core, and a programmable transport network over distributed O-Cloud infrastructure [4], [11]. In a realistic medium-sized deployment this might correspond to tens of sites grouped into a few geographic regions, with a mix of edge and central cloud resources.

The control and management plane includes an SMO with non-RT RIC functionality and one or more near-RT RIC instances. The SMO has global visibility of inventory, configu- ration, performance metrics, and service definitions, while the near-RT RICs have fine-grained control over RAN behavior via E2 [4], [6].

The operator interacts with the SMO through intents, such as “deploy a URLLC slice in region X” or “reduce RAN energy consumption by 15% during off-peak hours without breaking coverage constraints.” These intents must be trans- lated into concrete control tasks and AI-driven policies realized by rApps and xApps across RAN, core, and transport [10], [12].

At a high level, the orchestration problem is to decide:

- Which AI models and control applications (rapps/xapps) to activate;
- Where to run them (central vs. Edge O-Cloud sites, which RIC instance);
- How to allocate O-Cloud resources to these applications; and
- How to configure telemetry collection and policy distri- bution

So that operator intents are satisfied as well as possible while control overhead remains bounded and conflicts between applications are avoided [1], [2]. This generalises the RAN- focused OrchestRAN problem to multiple domains and adds the practical constraints that come from working with existing SMO and RIC implementations.

IV. AI-Driven Orchestration Architecture

Architecture overview

The architecture we use is close to the standard O-RAN picture. The SMO keeps an inventory of network functions, handles configuration over O1 and exposes northbound APIs where intents and slice requests arrive [4], [6]. Inside the SMO, the non-RT RIC runs an AI catalogue, an intent manager and the orchestration rApp. One or more near-RT RIC instances host xApps and talk to O-DU/O-CU nodes over E2, while separate orchestrators manage the 5G core and, if needed, the transport network [6], [7].

Telemetry and data pipeline

For the orchestration to work, the SMO and non-RT RIC must see enough telemetry. In our design, O-DU and O-CU nodes report radio and scheduler KPIs over O1 and E2, while the core and transport domains export basic QoS and flow statistics. These streams are stored as time series and used by both long-term models (for example, forecasting) and short- term models (for example, anomaly detection) [9], [12].

One obvious question is where to run the heavy analytics. Putting everything in the SMO would be simpler to manage but would also increase control traffic and delay. Here we assume a mixed setup: some models run centrally, others close to the data at edge O-Cloud sites, as long as their lifecycle can still be controlled from the non-RT RIC.

V. Orchestration Algorithm Design

The orchestration logic did not start out as the three-step heuristic shown later. We first tried to write it as a mixed- integer linear programme (MILP), following the spirit of OrchestRAN but adding core and transport decisions [1], [2]. Using a standard solver (Gurobi) on a laptop with a 12-core CPU, instances with around 20 models and 10 possible sites already needed tens of seconds to a few minutes to solve. This is too slow for the type of online reaction we have in mind.

Because of this, we switched to a simpler approach that can be implemented as an rApp in the non-RT RIC. The current design has three stages: (i) break an intent down into smaller objectives and score candidate models and locations, (ii) greedily pick model placements that respect capacity and delay limits, and (iii) check for obvious conflicts between applications that try to control the same parameters.

Stage 1: Intent decomposition and scoring

When a new intent is received, or when network conditions change significantly, Stage 1 decomposes the intent into a set of domain-specific objectives and required functionalities. For example, a URLLC-focused intent typically implies the need for short-term traffic and mobility prediction, slice- aware admission control, and stricter buffering and scheduling policies in the RAN.

For each required functionality, the orchestration rApp queries the AI catalog to retrieve candidate models. For each (model, location) pair it keeps track of: (i) a performance estimate (e.g., accuracy or mean error from past evaluations), (ii) a resource footprint (CPU, memory, optional accelera- tors), and (iii) an approximate latency budget and telemetry requirements [10]. A simple utility score combines these aspects. In our prototype we use a linear combination of normalised performance, placement suitability, and estimated control overhead. We tuned the weights empirically in small- scale simulations; configurations that penalised overhead too heavily tended to push models aggressively to the edge, some- times making it difficult to aggregate enough data centrally.

Stage 2: Greedy selection and placement

Stage 2 receives a list of candidate (model, functionality, location) triplets sorted by utility score. It then walks this list and greedily selects triplets that pass three checks: (1) the target O-Cloud site has enough residual capacity, (2) the resulting control loop respects the latency budget for the targeted slice or service, and (3) the selected model does not introduce a conflict with already chosen applications (checked in Stage 3). Selected triplets are added to the deployment plan and capacities are updated. If a required functionality remains uncovered after the first pass, the algorithm tries an "outsourcing" step, looking for nearby sites that can host the model while still meeting telemetry and timing constraints, similar in spirit to OrchestRAN's placement choices [2]. If this also fails, the intent is marked as only partially satisfiable and a small report is generated so that an operator or higher-level policy can decide how to react.

Stage 3: Conflict detection and resolution

To avoid incompatible control actions, every rApp and xApp declares the parameters it intends to modify (for example, per-cell transmit power, per-slice bandwidth reservations) and the timescale of its control loop. The orchestration rApp maintains a simple parameter-ownership graph where nodes are parameters and edges correspond to applications writing them [8], [12]. When Stage 2 proposes a new application, Stage 3 checks this graph. If no existing application controls the same parameter at a similar timescale, the new applica- tion is accepted. Otherwise, a small set of rules is applied: applications that enforce hard SLAs or safety constraints have priority, while long-term optimisers such as energy savers may be downgraded to advisory mode or rejected for that cycle. This rule set is intentionally straightforward; the goal is to match what operators already do manually rather than to design a fully automated conflict-resolution framework in a single step.

VI. Implementation Blueprint

The framework is intentionally built around components that are already available in the O-RAN and telco-cloud ecosystem.

RAN, core, and O-Cloud

On the RAN side, an open-source stack such as srsRAN or OAI can provide O-DU/O-CU functionality, with SDR-based or commercial O-RUs connected over standard fronthaul [5], [11]. In a typical lab setup, these run on commodity x86 servers with 8–16 CPU cores and 32–64 GB of RAM. The 5G core can be realised using Open5GS or a CNF-based commercial core.

The SMO/non-RT RIC and near-RT RIC services run as microservices on a Kubernetes-based O-Cloud cluster. For a medium-sized testbed, we envision two or three edge clusters plus one central cluster,

totalling around 8–10 worker nodes. Existing SMO platforms and O-RAN SC components provide most of the required functionality, and the orchestration logic is added as an rApp that talks to them through their APIs [6]–[8].

Automation and CI/CD

To keep operations manageable, a GitOps-style workflow similar to AutoRAN is used [3]. Desired network, slice, and application states are described declaratively in version- controlled manifests. A controller such as Argo CD or Flux reconciles these manifests with the actual Kubernetes cluster state. Early dry-run experiments with this approach (on a small development cluster) already showed that it reduces configuration drift and makes it easier to roll back changes when a new xApp behaves unexpectedly.

Table I
Summary Of 60 S Simulation Results For The Three Configurations.

Mode	Mean eMBB lat. [ms]	Mean URLLC lat. [ms]	URLLC SLA viol. [%]
Static	2903.97	10.0	0.0
RAN-only	145.68	10.0	0.0
End-to-end	23.41	10.0	0.0

VII. Evaluation Methodology

The evaluation is designed around what can realistically be built and measured on a medium-sized lab testbed rather than a nationwide network. At the time of writing, the physical testbed is under construction, so this section describes the simplified simulator and scenarios used for initial experiments. We implement a discrete-time simulator in Python that models a single shared 10 Mbit/s link with Poisson eMBB and URLLC arrivals, FIFO queues, and strict priority for URLLC packets. Traffic is simulated for 60 s with a 10 ms step size. The middle third of the simulation experiences a 2× traffic surge. The three configurations differ only in admission control and effective capacity during surges:

- **Static baseline:** all arrivals are admitted; capacity is constant.
- **RAN-only orchestration:** during the surge, eMBB pack- ets are probabilistically dropped to emulate a RAN-side admission policy.
- **End-to-end orchestration:** combines the same admission policy with a 50% increase in effective capacity during the surge, emulating coordinated core/transport adjust- ments.

For each configuration we record mean, 95th- and 99th- percentile latencies for eMBB and URLLC traffic, the per- centage of URLLC packets exceeding a 10 ms latency target, and the number of packets served.

VIII. Results

Table I lists the numbers from the 60 s simulation, and Figs. 1 and 2 plot the URLLC-related metrics. The table reports mean latencies and URLLC SLA violations over a 60 s simulation.

Impact on URLLC performance

Across all configurations, the mean URLLC latency is essentially pinned at the 10 ms target and no URLLC packets violate the SLA in this particular setup. This is expected because the simulator gives URLLC strict priority at the shared link and the offered URLLC load remains well within the available capacity. This means we cannot really compare the three configurations based on URLLC in this simple setup, but at least it confirms that none of them breaks the 10 ms target under the traffic we modelled.

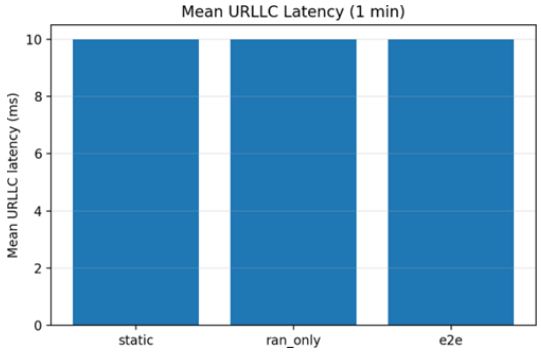

Fig. 1. URLLC SLA violation rate over a 60 s simulation for static, RAN- only and end-to-end orchestration.

Impact on eMBB latency

The main differences appear for eMBB traffic. In the static baseline, the mean eMBB latency is roughly 2.9 s, with 95th- and 99th-percentile latencies on the order of several seconds, reflecting the effect of high load and strict priority given to URLLC. When RAN-only AI orchestration is enabled, the mean eMBB latency drops by more than an order of magnitude to about 146 ms, because the admission policy reduces eMBB load during the surge period.

End-to-end orchestration improves the situation further. By combining the same RAN-side admission control with a tem- porary capacity boost that emulates coordinated core/transport adjustments, the mean eMBB latency falls to approximately 23 ms, and the 95th- and 99th-percentiles shrink to below 80 ms. This behaviour matches the intuition that end-to- end coordination should be able to recover some of the performance lost when the RAN alone throttles eMBB traffic.

Discussion

The model used here is very simple on purpose: a single shared link, Poisson arrivals and abstract capacity changes during the surge. Nevertheless, the numbers already show two useful points. First, even basic AI-inspired policies such as adaptive admission control can dramatically change eMBB la- tency without harming URLLC. Second, adding cross-domain levers—here represented by a capacity increase during the surge—allows the network to protect URLLC while also restoring much of the eMBB performance that would other- wise be sacrificed under RAN-only control.

IX. Open Challenges

While working through the design, several open challenges became apparent. First, portability of rApps and xApps across different SMO and RIC implementations is still limited in practice, even when they follow the same specifications [6], [7], [12]. Small differences in APIs and data models make it harder than expected to move applications between platforms. Second, operators need to trust AI-driven decisions. It is not enough that a model performs well on average; engineers in the network operations centre need at least a basic explanation of why certain actions are taken [9], [10].

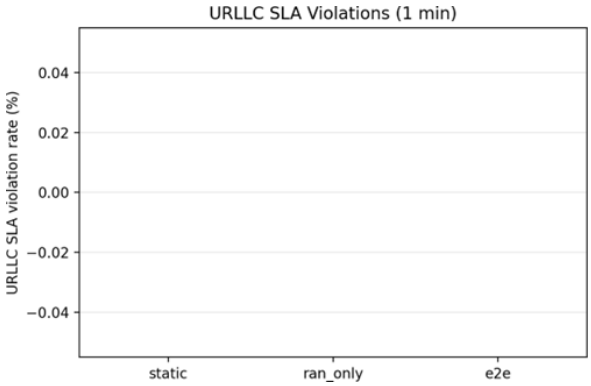

Fig. 2. Mean URLLC latency over a 60 s simulation for the three configu- rations.

Third, conflict management across multiple vendors and domains is far from solved. The simple rule-based system out- lined in this paper is only a starting point; more sophisticated schemes that combine priorities, learning, and formal safety constraints will likely be needed. Finally, the security of AI- enabled control loops themselves is a concern: compromised models or telemetry streams could have disproportionate im- pact on network behaviour [10], [12].

X. Conclusion

This paper has described an AI-driven, end-to-end orches- tration framework for O-RAN that starts from intents at the SMO, uses a lightweight three-stage heuristic in the non-RT RIC to select and place AI models, and coordinates rApps and xApps with multi-timescale control across RAN, core, and transport [1], [3], [6], [7]. The focus has been on a design that can be implemented with existing O-RAN and telco-cloud components, rather than on an idealised but impractical global optimiser.

Initial simulation results on a simple model show that RAN- only orchestration can already reduce eMBB latency by more than an order of magnitude compared to a static baseline, and that adding end-to-end coordination can recover much of the performance sacrificed under RAN-only control without degrading URLLC. Our goal is to move from this simple model to a real O-RAN testbed and see how much of this behaviour carries over under realistic traffic mixes and failure scenarios.

REFERENCES

[1]. S. D'oro, L. Bonati, M. Polese, And T. Melodia, "Orchestran: Net- Work Automation Through Orchestrated Intelligence In The Open Ran," Arxiv:2201.05632.
[2]. S. D'oro, L. Bonati, M. Polese, And T. Melodia, "Orchestrating Net- Work Intelligence In The Open Ran," [Online]. Available: Https://Ece. Northeastern.Edu/Wineslab/Papers/Doro2024orchestran.Pdf.
[3]. S. Maxenti Et Al., "Autoran: Automated And Zero-Touch Open Ran Systems," Arxiv:2504.11233, Preprint.
[4]. O-Ran Software Community, "O-Ran Architecture Overview," Documentation. [Online]. Available: Https://Docs.O-Ran-Sc.Org/En/Latest/ Architecture/Architecture.Html.
[5]. "Development Of Open Radio Access Networks (O-Ran) For Real-Time…," Nature Communications.
[6]. Vmware, "Orchestrate, Automate, And Assure O-Ran With Smo," White Paper.
[7]. Juniper Networks, "Solving The Complexity Of Orchestration, Automa- Tion And Service Assurance."
[8]. Juniper Networks, "Enabling Multi-Vendor O-Ran Ric Xapps/Rapps Coordination With Juniper Ran Intelligent Controller."
[9]. Ericsson, "Telecom Network Automation: Powering 5g Future."
[10]. "Ai-Native Management And Orchestration In O-Ran," Ieee Xplore.
[11]. Srsran Project, "O-Ran Gnb Overview," Documentation.
[12]. O-Ran Alliance, "Research Report On Cross-Domain Ai."