RAN Simulator Is NOT What You Need: O-RAN Reinforcement Learning for the Wireless Factory

Ta Dang Khoa Le
ta-dang-khoa.le@eurecom.fr
EURECOM
Biot, France

Navid Nikaein
navid.nikaein@eurecom.fr
EURECOM
Biot, France

ABSTRACT

As modern manufacturing lines embrace greater modularity and flexibility, the need to transition factory networks from wired to wireless grows. Yet the mission-critical nature of factory networks poses a key challenge - connecting numerous diverse machines with **high QoS predictability**. After formulating this challenge as predictable RAN optimization via Reinforcement Learning (RL), we highlight a major-yet-overlooked modeling issue: *matching the packet handling mechanics of a production/real RAN software*. In this paper, we show that these mismatches inside RAN simulators can cause non-trivial QoS gaps in production. Then, we present **Twin5G**, a novel training solution that brings scalable and near-discrete-time emulations to real RAN software, removing the need for RAN simulators. In a RAN Slicing example, Twin5G-trained policy outperforms simulator-trained and standard RL-trained policies in both QoS achieved (**+16%**) and predictability (**+19%**) during tests.

ACM Reference Format:
Ta Dang Khoa Le and Navid Nikaein. 2023. RAN Simulator Is NOT What You Need: O-RAN Reinforcement Learning for the Wireless Factory. In *The 29th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '23), October 2–6, 2023, Madrid, Spain.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3570361.3615758

1 INTRODUCTION

To meet increasingly customized and smaller manufacturing orders, modern manufacturing lines are designed to be highly modular and reconfigurable. This necessitates the use of mobile machinery at minimal rewiring costs, i.e., wireless networking [1]. However, transitioning mission-critical factory networks from wired to wireless is a challenge; one needs to connect as many UEs as possible, despite much limited resources, while guaranteeing each UE's QoS *before* actual deployment (i.e., QoS predictability), despite diverse requirements and stochastic workload variations.

In this paper, we formulate this wireless factory challenge as (a) per-UE predictable QoS optimization solved with (b) RL-based RAN control. (a) is crucial because we want to maximize the value of each used resource unit. Yet, optimizing system-level or multi-UE objectives cannot guarantee predictable network conditions for any UE [2]. (b) is necessary because, due to radio scarcity, RAN remains the only bottleneck in modern network deployments [3]. Yet, optimizing it requires a sequential stochastic control approach to handle adaptive traffic profiles and partially-observable channel variations. Under this formulation, multiple per-UE RL policies simultaneously control the RAN, with each policy's expected total reward being the prediction for its UE's in-production QoS.

Still, as Deep RL is sensitive to modeling errors [4], high QoS predictability requires accurate emulation of production traffic, channel, and RAN-stack dynamics. In this paper, we leave the well-known traffic-channel modeling errors to future work, and focus on the overlooked RAN-stack factor: *how a production RAN software handles its UEs' packets*. In Section-3, we show that even a minor difference between RAN simulators and production/real RAN software can cause non-trivial QoS gaps in production, yet training on real RAN software is restrictive due to scalability and non-discrete-time transitions. While the small-scale limit is well-known, the non-discrete-time limit is because classical Markov Decision Processes assume that states do not evolve during action selection and/or policy updates [5]; this is impossible as real RAN software is not pausable. To our knowledge, both limits are *not yet* addressed in the 5G literature.

In the next section, we present **Twin5G**, a novel training approach that overcomes both limits of real RAN software-based emulations. Twin5G is implemented as an O-RAN-compliant xApp (using the FlexApp library [6]), so it should apply to all O-RAN-compliant real RAN software.

ACM MobiCom '23, October 2–6, 2023, Madrid, Spain

ACM ISBN 978-1-4503-9990-6/23/10…$15.00
https://doi.org/10.1145/3570361.3615758

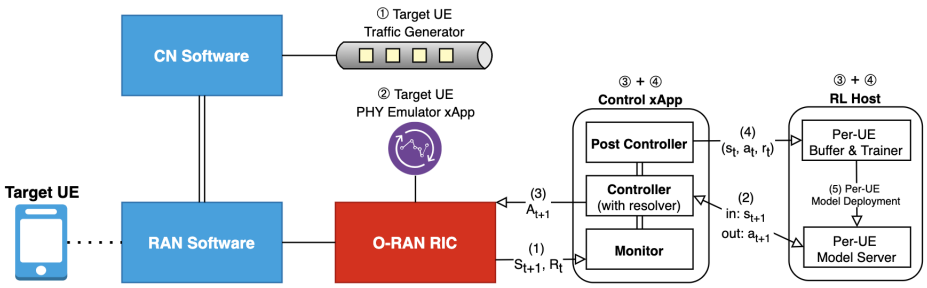

Figure 1: Twin5G's four elements train each target UE separately. In production, ① and ② are replaced with real workloads, while ③ and ④ have their Step-2 becomes *Multi-UE Batch Inference* and their Step-4 and -5 becomes *Multi-UE Parallel Updates*.

2 METHODOLOGY: TWIN5G

The key idea behind Twin5G is that, instead of training on real RAN software with multiple UEs connected, we separately train for each target UE (i.e., the UE we want to optimize) by emulating the interference other UEs may cause to its in-production traffic, channel, and RAN-stack dynamics; this helps execute RL for scenarios of *any* scale. Twin5G consists of four elements illustrated in Figure 1.

ONE. *Per-UE emulation of traffic profiles* - Since factory traffics are generated internally, RAN resources become the only bottleneck. Hence, UEs' RAN policies do not need to observe each other to fully control their own traffics. With this insight, one can utilize problem-specific knowledge to individually run each target UE's traffic profile (e.g., using iPerf, real apps, etc.) alongside real RAN software for training.

TWO. *Per-UE emulation of channel variations* - To emulate PHY-layer interference that other UEs may cause to each target UE, one can utilize problem-specific knowledge to conduct multi-UE ray-tracing analysis (e.g., with MATLAB) and run this UE's final propagation model alongside real RAN software for training. With this in mind, we wrap PHY models inside PHY emulation xApps to correctly get and set relevant PHY-related variables inside real RAN software.

THREE. *Training at fast control loop* - To achieve non-discrete-time transitions, we remark that the assumption of non-evolving states still holds if our control-loop time is negligible [5]. Since the minimum control interval of the near-real-time RIC is 10ms [7], we aim to reduce the total time from producing states to executing actions, i.e., the whole control loop, to under 2ms. To this end, we interpret O-RAN Machine Learning specifications [7] under the lens of Actor-based concurrency. Our approach "desynchronizes" the traditional rollout-then-update flow (of OpenAI's Gym) into two concurrent groups of actors. The first group, illustrated by Step-1 to Step-3 in Figure 1, runs the main control loop; the second group, illustrated by Step-4 and Step-5 in the same figure, runs the background training loop.

FOUR. *Multi-agent-aware Resource-constrained RL* - Simultaneously controlling the RAN, UEs' radio-related resource needs, e.g., physical resource blocks and/or transmission power, together may exceed system capacity. For reliable QoS guarantees, one must protect each target UE's optimal trajectory, separately derived during training, from possible "resource cuts" in production. We solve this in two generic steps: (i) At admission, each UE is guaranteed a "resource budget" per RL episode; (ii) During training, the RL agent seeks an optimal allocation of its budget along the transitions, accounting for possible resource cuts. Intuitively, episode-level admission control limits the need for resource cuts while still allowing each UE to use "just enough" per transition. Additionally, one can try multiple training runs to identify each UE's just-enough budget for a target QoS satisfaction ratio. Mathematically, for each target UE, we solve an instance of the following generic class of episodic RL problems:

$$\pi^*_{\mathcal{A}} = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \mathrm{QoS}(s_t, f(a_t))\right]$$

$$\text{s.t.} \sum_{t=0}^{T} \mathrm{cost}(f(a_t)) \leq B.$$

Where:

- $f(\cdot)$ is a UE-agnostic excess resolver that reduces UEs' resource requests to align with system capacity; optimally designed per problem instance.
- $\mathcal{A}$ is a policy search algorithm that accounts for external agents' actions and, therefore, possible resource cuts; also optimally designed per problem instance.
- $\mathrm{QoS}(\cdot)$, $\mathrm{cost}(\cdot)$, and B represent the QoS satisfaction function, the resource usage function, and the episode-level resource budget, respectively. They are defined per problem statements and UE requirements.

In the next section, we show how this class of problems is applied to a real use case: preparing a video streaming slice towards automated factory coordination.

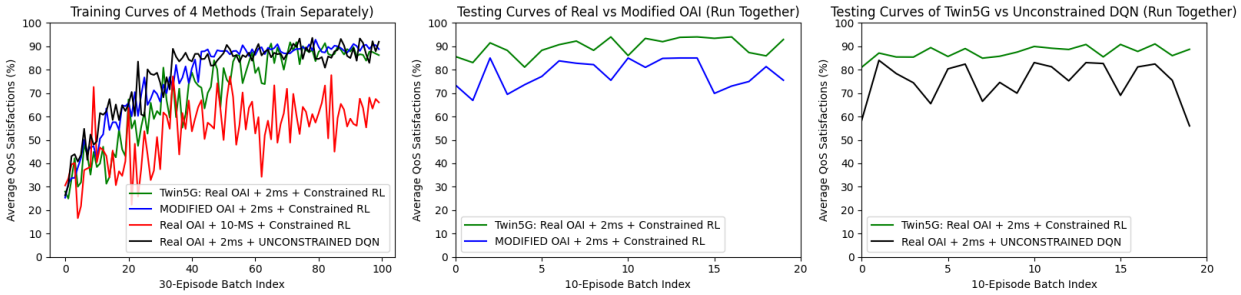

Figure 2: During training, only training at full control interval (10ms) performs badly. During testing, modified OAI-trained policy performs worse due to MCS-setting mismatch, while standard DQN-trained policy becomes unpredictable due to resource-cut unawareness.

3 DEMO: VIDEO STREAMING SLICES

The ultimate use case is automated factory coordination from the Edge, utilizing videos streamed from multiple Automated Guided Vehicles (AGVs). To train RL-based RAN policies that manage these AGVs' connections, let us consider the following per-UE uplink specifications, taken from a major manufacturing company, with emulation artifacts in Table 1.

Table 1: Per-UE Uplink Specifications

	Use Case Specs	Emulation Artifacts
Requirements	20mbps throughput & 50ms RTT (every 100ms) Evaluated every second with a 260-slot budget	
Traffic Profile	DASH on BBR (MSS: 1460 bytes)	BBR-based iPerf (MSS: 1460 bytes)
Channel Profile	Indoor Factory <10m/s	TR 38.901's InF Gauss-Markov Mobility

Our demo is conducted on top of FlexRIC and OpenAirInterface (OAI) [8], with Quectel-based UEs representing the AGVs. The 260-slot budget is identified after multiple runs targeting ~ 90% QoS satisfaction ratio. Algorithm-wise, there should be no differences as we use FlexRIC-OAI's downlink RAN Slicing capability. To protect each UE's optimal trajectory from possible resource cuts, we apply the following techniques during training:

- A Deep Q-Learning (DQN) policy with action space $+1 \vee 0 \vee -1$, signaling changes to the current number of slots per UE.
- A resolver that, once system capacity is exceeded, "cuts" all UEs' requests equally and just enough.

Together with episode-level admission control, these techniques help "flattening" our resource-cut upper-bound across 100 transitions (10ms each) of an episode (1 second each), minimally distorting each UE's optimal trajectory.

With the experiments in Figure 2, we show the impact of a minor mismatch inside RAN simulators and the two limits of real RAN software-based training on QoS predictability. We compare train versus test QoS satisfaction ratios (both use the same emulation artifacts) of four methods: (1) Twin5G; (2) Twin5G on a "simulator", built by modifying OAI in just one place: using 3GPP-based (instead OAI's BLER-based) MCS settings; (3) Twin5G at artificially-delayed 10ms control-loop time; and (4) Standard DQN without budget and action constraints. We remark that, due to bad training performance, the third method is not included in testing, where slices are run together to validate scalability.

In conclusion, to productionize RL policies: (i) *One should not use RAN simulators*, (ii) *Control-loop time optimization is needed*, and (iii) *Training on real RAN software should and can be scaled*; all are contrary to popular beliefs.

REFERENCES

[1] Bosch in Singapore. Industrial 5G. https://www.bosch.com.sg/news-and-stories/5g-in-action/, 2021.
[2] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient Surgery for Multi-task Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
[3] Amazon Web Services. Deploying DISH's 5G Network in AWS Cloud. https://aws.amazon.com/blogs/industries/telco-meets-aws-cloud-deploying-dishs-5g-network-in-aws-cloud/, 2022.
[4] Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. EPOpt: Learning Robust Neural Network Policies Using Model Ensembles. In *International Conference on Learning Representations (ICLR)*, 2017.
[5] Jaden B Travnik, Kory W Mathewson, Richard S Sutton, and Patrick M Pilarski. Reactive Reinforcement Learning in Asynchronous Environments. *Frontiers in Robotics and AI*, 2018.
[6] Chieh-Chun Chen, Mikel Irazabal, Chia-Yu Chang, Alireza Mohammadi, and Navid Nikaein. FlexApp: Flexible and Low-Latency xApp Framework for RAN Intelligent Controller. In *International Conference on Communications (ICC)*, 2023.
[7] O-RAN Working Group 2. O-RAN AI/ML Workflow Description and Requirements 1.03. Technical report, Open RAN, 2021.
[8] Robert Schmidt, Mikel Irazabal, and Navid Nikaein. FlexRIC: An SDK for Next-generation SD-RANs. In *International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2021.